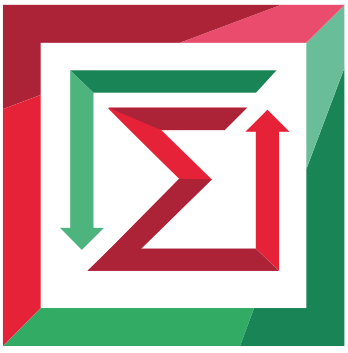
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 7
по дисциплине «Программные Средства Защиты Информации»
Однонаправленные хэш-функции. Электронная цифровая подпись



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П. К. Малышкина Е. Д. Слободчикова А. Э.
БРИГАДА:	2
ПРЕПОДАВАТЕЛЬ:	Авдеенко Т. В.

Новосибирск
2021

1. Цель работы

Изучить различные алгоритмы однонаправленного хэширования данных, которые основаны на симметричных блочных алгоритмах шифрования. Ознакомиться со схемами цифровой подписи и получить навыки создания и проверки подлинности электронной цифровой подписи.

2. Задание

1. Реализовать приложение, позволяющее вычислять и проверять ЭЦП, сформированную по алгоритмам RSA и Эль-Гамаля.
2. С помощью реализованных приложений выполнить следующие задания.
    - 2.1. Протестировать правильность работы разработанных приложений.
    - 2.2. Для заданных в варианте открытых ключей пользователя проверить подлинность подписанных по алгоритму RSA хэш-значений m некоторых сообщений M.
    - 2.3. Абоненты некоторой сети применяют подпись Эль-Гамаля с известными общими параметрами p и g. Для указанных в варианте секретных параметров абонентов найти открытый ключ и построить подпись для хэш-значения m некоторого сообщения M. Проверить правильность подписи.

3. Теоретическая часть используемых алгоритмов

3.1. Электронная подпись RSA

Для осуществления подписи сообщения $M = M_1M_2M_3 \dots M_n$ с помощью RSA необходимо вычислить хэш-функцию $m = h(M_1M_2M_3 \dots M_n)$, которая ставит в соответствие сообщению M число m. На следующем шаге достаточно снабдить подписью только число m, и эта подпись будет относиться ко всему сообщению M. Далее по алгоритму RSA вычисляются ключи (e, n) и (d, n).
Затем вычисляется $s = m^d mod\ n$ ($d - $ *секретная степень*).
Число s – это и есть цифровая подпись. Она просто добавляется к сообщению и получается подписанное сообщение ⟨M, s⟩.
Теперь каждый, кто знает параметры подписавшего сообщение (т.е. числа e и n), может проверить подлинность подписи.
Для этого необходимо проверить выполнение равенства
$h(M) = s^e mod\ n$.

3.2. Алгоритм Эль-Гамаля

Для генерации пары ключей сначала выбирается большое простое число p, один из его первообразных корней g и случайное число x (g < p, x < p).
Затем вычисляется $y = g^x mod\ p$.
Открытым ключом являются y, g и p. Закрытым ключом является x.
Чтобы подписать m, являющееся хэш-значением некоторого сообщения M, сначала выбирается секретное случайное число k, взаимно простое с p-1. Затем вычисляется $a = g^k mod\ p$.
Из соотношения $m = (x * a + k * b)\ mod\ (p - 1)$ определяется b.
Выполнив преобразования, получим $b = k^{-1} * (m - x * a)\ mod\ (p - 1)$, где – k^{-1} определяется из соотношения $k^{-1} * k \equiv 1(mod(p - 1))$.
В результате подписью будет пара (a, b).
Для проверки подписи нужно убедиться, что $y^a * a^b mod\ p = g^k mod\ p$.

4. Исследования

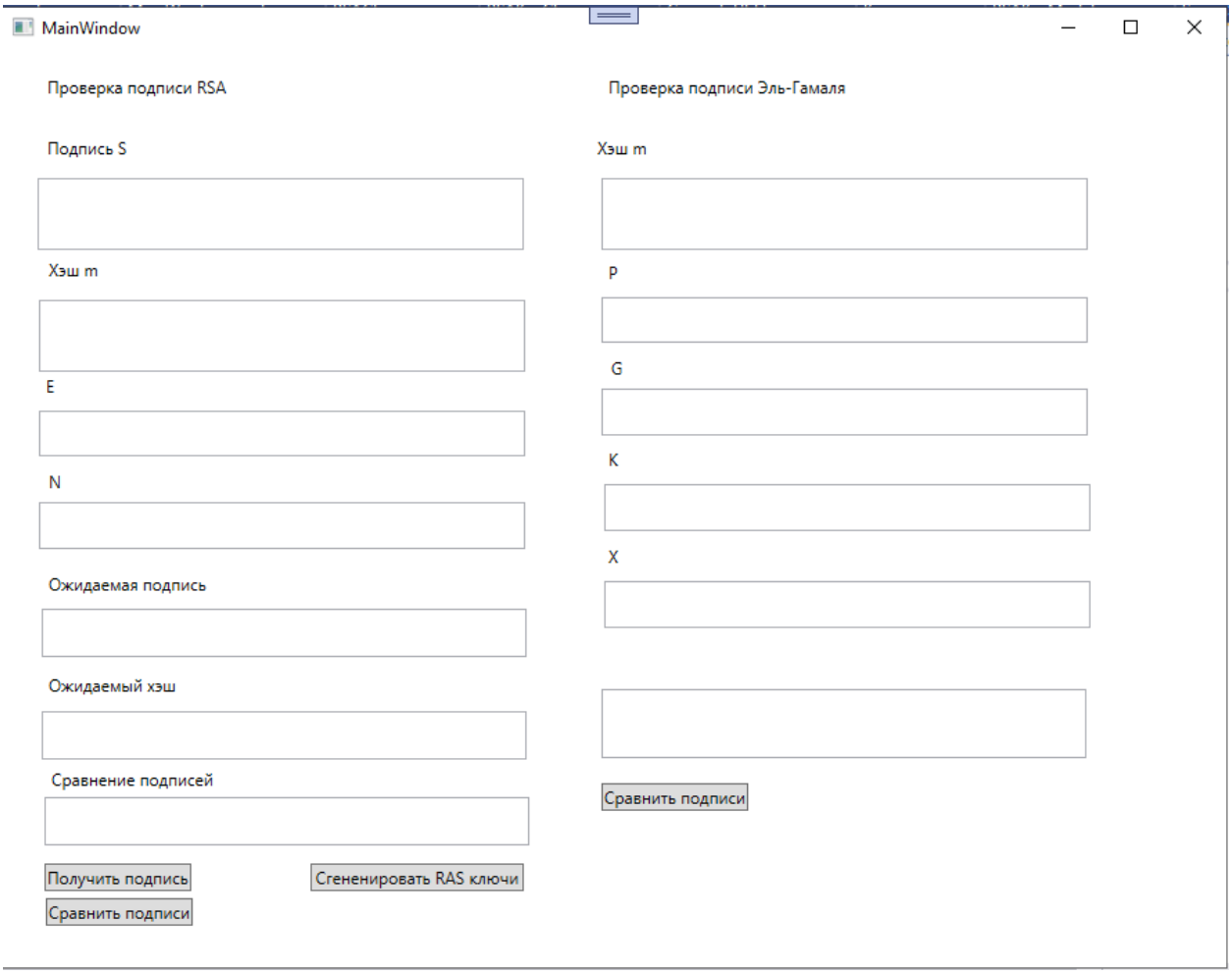


Рисунок 1. Демонстрация интерфейса приложения

Проведём исследования заданных пар хэш-значение – подпись для указанных параметров открытого RSA ключа.

Параметры исследования			
Открытый ключ	m – хэш-значение	s - подпись	Результаты проверки
n = 65 e = 5	10	30	Ожидаемый хэш 10 Сравнение подписей Подписи совпадают

	6	42	Ожидаемый хэш 22 Сравнение подписей Подписи не совпадают
	6	41	Ожидаемый хэш 6 Сравнение подписей Подписи совпадают

Проведём исследование цифровой подписи Эль-Гамаля для заданных в варианте параметров.

Открытые параметры: p = 23, g = 5; секретные параметры: x = 10, k =15; хэш-значение m = 5.

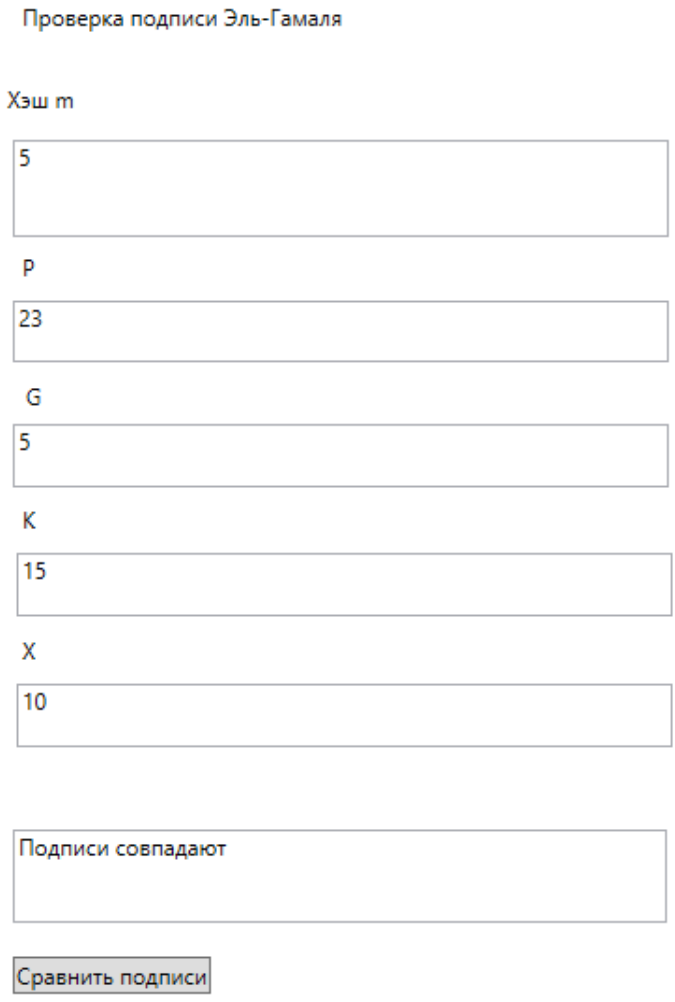


Рисунок 2. Результаты проверки подписи Эль-Гамаля для заданных параметров

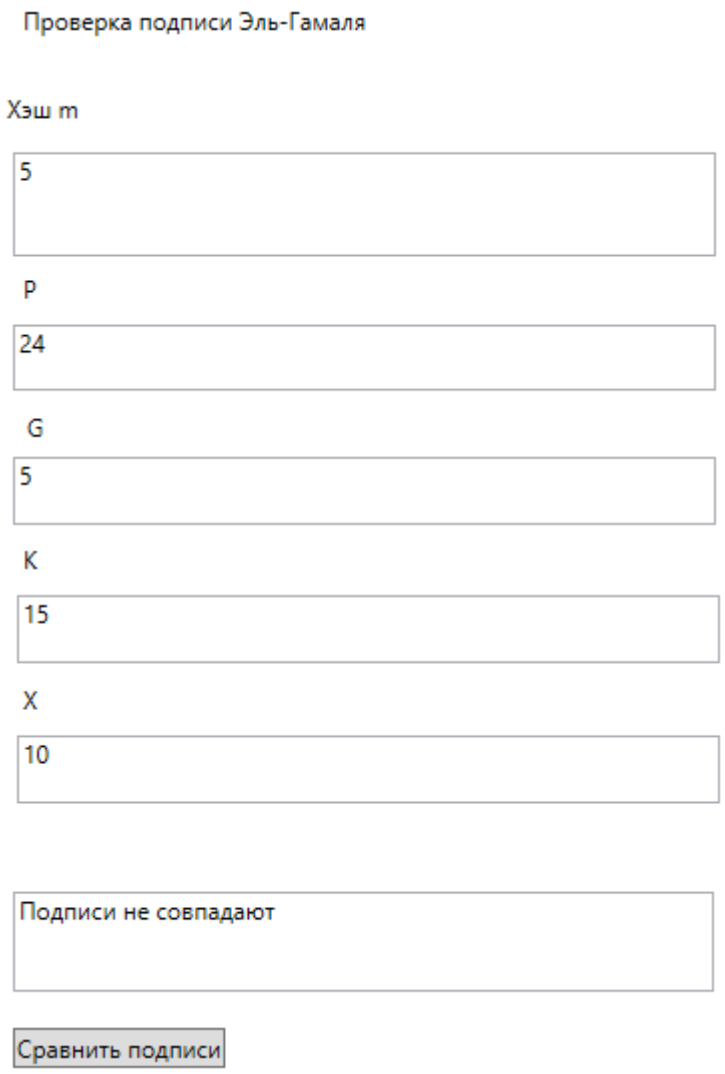


Рисунок 3. Результаты проверки подписи Эль-Гамаля для изменённых параметров (изменён открытый параметр p)

5. Выводы

В ходе работы разработано приложение, способное генерировать цифровую подпись по алгоритму RSA и Эль-Гамаль и проверять корректность каждой подписи.

6. Код программы

MainWindows.xaml.cs

```
using Lab1_Gamming_Srammbling.CryptoClass;
using System;
using System.Numerics;
using System.Windows;

namespace Lab1_Gamming_Srammbling
{

    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

        }

        private RSAKeyClass PU = null, PV = null;
        private byte[] Sk = null;

        private void CiphButton_Click(object sender, RoutedEventArgs e)
        {
            if (RSA_E.Text == "" || RSA_N.Text == "" || Text.Text == "")
            {
                if (PU == null && PV == null)
                    MessageBox.Show("Пожалуйста, сгенерируйте RSA ключи");
                else
                {
                    if (TextM.Text == "")
                        MessageBox.Show("Пожалуйста, введите шифруемый хэш");
                    else
                    {
                        Sk = RSAClass.Encrypt(BitConverter.GetBytes(Convert.ToInt32(TextM.Text)), PV);
                        byte[] tt = new byte[4];
                        tt[0] = Sk[0];
                        Chiphrtext.Text = BitConverter.ToInt32(tt, 0).ToString();
                    }
                }
            }
        }

        private void ElGamVerification_Click(object sender, RoutedEventArgs e)
        {
            if (ElGamX.Text == "" || ElGamG.Text == "" || ElGamP.Text == "" || ElGamK.Text == "" || ElGamHesh.Text == "")
                MessageBox.Show("Пожалуйста, введите параметры G, P, X, K, а также шифруемый хэш");
            else
            {
                var elGamKey = ElGamalSignature.GenModKey(new BigInteger(Convert.ToInt32(ElGamX.Text)),
                                                          new BigInteger(Convert.ToInt32(ElGamG.Text)),
                                                          new BigInteger(Convert.ToInt32(ElGamP.Text)));
                var signature = ElGamalSignature.CreateSignature(BitConverter.GetBytes(Convert.ToInt32(ElGamHesh.Text)),
elGamKey,
                                                                 new BigInteger(Convert.ToInt32(ElGamK.Text)));

                if (ElGamalSignature.VerifySignature(BitConverter.GetBytes(Convert.ToInt32(ElGamHesh.Text)), signature,
elGamKey))
                    ElGamRes.Text = "Подписи совпадают";
                else
                    ElGamRes.Text = "Подписи не совпадают";
            }
        }

        private void VerifSign_Click(object sender, RoutedEventArgs e)
        {
            if (Text.Text == "" || TextM.Text == "")
            {
                MessageBox.Show("Пожалуйста, введите хэш, подпись");
            }
            else
            {
                if (PU == null && PV == null && (RSA_E.Text == "" || RSA_N.Text == ""))
                    MessageBox.Show("Пожалуйста, введите параметры открытого RSA ключа, либо сгенерируйте ключи заново");
                else
                {
                    if (RSA_E.Text == "" || RSA_N.Text == "")
                    {
                        if (PU == null && PV == null)
                            MessageBox.Show("Пожалуйста, введите параметры открытого RSA ключа");
                    }
                    else
                    {
                        PU = new RSAKeyClass(new System.Numerics.BigInteger(Convert.ToInt32(RSA_E.Text)),
```

```
                        new System.Numerics.BigInteger(Convert.ToInt32(RSA_N.Text)));
                    Sk = RSAClass.Encrypt(BitConverter.GetBytes(Convert.ToInt32(Text.Text)), PU);
                    byte[] tt = new byte[4];
                    tt[0] = Sk[0];
                    ExpectedSign.Text = BitConverter.ToInt32(tt, 0).ToString();
                    if (BitConverter.ToInt32(tt, 0) == Convert.ToInt32(TextM.Text))
                        SignVerRes.Text = "Подписи совпадают";
                    else
                        SignVerRes.Text = "Подписи не совпадают";
                }

                if (PU == null && PV == null)
                    MessageBox.Show("Пожалуйста, сгенерируйте ключи заново");
                else
                {
                    Sk = RSAClass.Encrypt(BitConverter.GetBytes(Convert.ToInt32(Text.Text)), PU);
                    byte[] tt = new byte[4];
                    tt[0] = Sk[0];
                    ExpectedSign.Text = BitConverter.ToInt32(tt, 0).ToString();
                    if (BitConverter.ToInt32(tt, 0) == Convert.ToInt32(TextM.Text))
                        SignVerRes.Text = "Подписи совпадают";
                    else
                        SignVerRes.Text = "Подписи не совпадают";
                }
            }
        }
    }

    private void RSAGen_Click(object sender, RoutedEventArgs e)
    {
        var tt = RSAClass.GenerateKeyPair();
        PU = tt.PublicKey;
        PV = tt.PrivateKey;
    }
  }
}
```

RSAClass.cs

```
using System.Linq;
using System.Numerics;
using System.Security.Cryptography;

namespace Lab1_Gamming_Srammbling.CryptoClass
{
    public static class RSAClass
    {
        public static System.Numerics.BigInteger RandomBigIntInRange(System.Numerics.BigInteger min,
System.Numerics.BigInteger max)
        {
            RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

            if (min > max)
            {
                System.Numerics.BigInteger temp = min;
                min = max;
                max = temp;
            }

            System.Numerics.BigInteger offset = -min;
            min = 0;
            max += offset;

            System.Numerics.BigInteger value = RandomBigIntFromZero(rng, max) - offset;
            return value;
        }

        public static System.Numerics.BigInteger RandomBigIntFromZero(RandomNumberGenerator rng,
System.Numerics.BigInteger max)
        {
            System.Numerics.BigInteger value;
            byte[] bytes = max.ToByteArray();
            byte ZeroBitsMask = 0b00000000;
            byte MostSignificantByte = bytes[bytes.Length - 1];

            for (int i = 7; i >= 0; i--)
            {
                if ((MostSignificantByte & (0b1 << i)) != 0)
                {
                    int ZeroBits = 7 - i;
                    ZeroBitsMask = (byte)(0b11111111 >> ZeroBits);
                    break;
                }
            }
```

```
    do
    {
        rng.GetBytes(bytes);
        bytes[bytes.Length - 1] &= ZeroBitsMask;
        value = new System.Numerics.BigInteger(bytes);

    }
    while (value > max);

    return value;
}

public static bool MillerRabinTest(System.Numerics.BigInteger N, System.Numerics.BigInteger D)
{
    System.Numerics.BigInteger a = RandomBigIntInRange(2, N - 2);

    System.Numerics.BigInteger x = System.Numerics.BigInteger.ModPow(a, D, N);

    if (x == 1 || x == N - 1)
        return true;
    else
        return false;
}

public static bool IsPrime(System.Numerics.BigInteger N)
{
    if (N < 2)
        return false;

    if (N == 2 || N == 3)
        return true;

    if (N % 2 == 0)
        return false;

    System.Numerics.BigInteger D = N - 1;
    while (D % 2 == 0)
        D /= 2;

    for (int k = 0; k < 64; k++)
    {
        if (!MillerRabinTest(N, D))
            return false;
    }

    return true;
}

public static System.Numerics.BigInteger GetFirstPrime(System.Numerics.BigInteger N)
{
    int Limit = 10000000;
    while (Limit-- > 0)
    {
        if (IsPrime(N))
            return N;
        N++;
    }
    return 2;
}

public static System.Numerics.BigInteger GetLargeRandomPrime()
{
    byte[] max = Enumerable.Repeat((byte)0xFF, 128).ToArray();

    max[max.Length - 1] &= 0x7F;

    System.Numerics.BigInteger Bmax = new System.Numerics.BigInteger(max);

    System.Numerics.BigInteger N = RandomBigIntInRange(Bmax / 8, Bmax);

    if (IsPrime(N))
        return N;

    else
        return GetFirstPrime(N);
}

public static System.Numerics.BigInteger GCD(System.Numerics.BigInteger a, System.Numerics.BigInteger b)
{
    while (a != 0 && b != 0)
    {
        if (a > b)
            a %= b;
        else
            b %= a;
    }
    return a == 0 ? b : a;
}
```

```
        public static System.Numerics.BigInteger ModInverse(System.Numerics.BigInteger a, System.Numerics.BigInteger n)
        {
            System.Numerics.BigInteger i = n, v = 0, d = 1;
            while (a > 0)
            {
                System.Numerics.BigInteger t = i / a, x = a;
                a = i % x;
                i = x;
                x = d;
                d = v - t * x;
                v = x;
            }
            v %= n;
            if (v < 0) v = (v + n) % n;
            return v;
        }

        public static (RSAKeyClass PublicKey, RSAKeyClass PrivateKey) GenerateKeyPair()
        {
            System.Numerics.BigInteger P = GetLargeRandomPrime();
            System.Numerics.BigInteger Q = GetLargeRandomPrime();
            System.Numerics.BigInteger N = P * Q;
            System.Numerics.BigInteger Phi = (P - 1) * (Q - 1);
            System.Numerics.BigInteger e;
            e = 65537;

            while (GCD(e, Phi) != 1)
            {
                e = GetFirstPrime(e);
            }

            System.Numerics.BigInteger d = ModInverse(e, Phi);

            var PublicKey = new RSAKeyClass(e, N);
            var PrivateKey = new RSAKeyClass(d, N);

            return (PublicKey, PrivateKey);
        }

        public static byte[] Encrypt(byte[] M, RSAKeyClass EncryptionKey) => System.Numerics.BigInteger.ModPow(new
System.Numerics.BigInteger(M), EncryptionKey.Key, EncryptionKey.N).ToByteArray();

    }
}
```

RSAKeyClass.cs

```
using System.Numerics;

namespace Lab1_Gamming_Srammbling.CryptoClass
{
    public class RSAKeyClass
    {
        public System.Numerics.BigInteger Key;
        public System.Numerics.BigInteger N;

        public RSAKeyClass(System.Numerics.BigInteger Key, System.Numerics.BigInteger N)
        {
            this.Key = Key;
            this.N = N;
        }
    }
}
```

ElGamalKeyStruct.cs

```
namespace Lab1_Gamming_Srammbling.CryptoClass
{
    public struct ElGamalKeyStruct
    {
        public BigInteger P;
        public BigInteger G;
        public BigInteger Y;
        public BigInteger X;
    }
}
```

ElGamalSignature.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1_Gamming_Srammbling.CryptoClass
{
    public static class ElGamalSignature
    {
        public static BigInteger module(BigInteger number, BigInteger moduleValue)
        {
            BigInteger result = number % moduleValue;
            if (result < 0)
            {
                result += moduleValue;
            }
            return result;
        }


        public static ElGamalKeyStruct GenModKey(BigInteger X, BigInteger G, BigInteger P)
        {
            var out_key = new ElGamalKeyStruct();
            out_key.X = X;
            out_key.G = G;
            out_key.P = P;
            out_key.Y = G.modPow(X, P);

            return out_key;
        }

        public static byte[] CreateSignature(byte[] pData, ElGamalKeyStruct keyStruct, BigInteger K_in = null)
        {
            IList<byte> data = pData;
            BigInteger KValuesRange = keyStruct.P - 1;
            BigInteger K;
            K = K_in;

            BigInteger A = keyStruct.G.modPow(K, keyStruct.P);
            BigInteger B = module(K.modInverse(KValuesRange) * (new BigInteger(data) - (keyStruct.X * A)), KValuesRange);

            byte[] aInBytes = A.getBytes();
            byte[] bInBytes = B.getBytes();

            int size = (((keyStruct.P.bitCount() + 7) / 8) * 2);
            byte[] result = new byte[size];

            Array.Copy(aInBytes, 0, result, size / 2 - aInBytes.Length, aInBytes.Length);
            Array.Copy(bInBytes, 0, result, size - bInBytes.Length, bInBytes.Length);


            return result;
        }
        public static bool VerifySignature(byte[] data, byte[] signature, ElGamalKeyStruct keyStruct)
        {
            int size = signature.Length / 2;
            byte[] aInBytes = new byte[size];
            Array.Copy(signature, 0, aInBytes, 0, aInBytes.Length);

            byte[] bInBytes = new byte[size];
            Array.Copy(signature, size, bInBytes, 0, bInBytes.Length);

            BigInteger A = new BigInteger(aInBytes);
            BigInteger B = new BigInteger(bInBytes);

            BigInteger e1 = module(keyStruct.Y.modPow(A, keyStruct.P) * A.modPow(B, keyStruct.P), keyStruct.P);

            BigInteger e2 = keyStruct.G.modPow(new BigInteger(data), keyStruct.P);

            return e1 == e2;
        }
    }
}
```