Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК

Расчетно-графическое задание

по дисциплине

«Перспективные технологии защиты информации»

Выполнил:

студент гр. МГ-211 _____ / Бурдуковский И.А./

подпись

Проверил:

Профессор

кафедры ПМиК _____ / Фионов А.Н./

Новосибирск

2023 г.

ОГЛАВЛЕНИЕ

Задание

Реализовать на основе протокола Диффи-Хеллмана приложение для передачи сообщения в зашифрованном виде по TCP протоколу

Выполнение

Был реализован консольный чат с TCP протоколом передачи данных. Решение состоит из двух проектов. Для серверной части и для клиентской.

Клиентская часть состоит из одного класса. В нем реализованы методы принятия сообщения, отправки сообщения, помимо процедуры mail.

В серверном проекте реализованы три класса. Класс Program содержит процедуру Mail, в которой происходит в потоке запуск серверного класса ServerObject. В классе ServerObject реализованы методы для прослушивания входящих подключений, добавлении и удалении подключения, трансляции всем пользователям сообщений и отключения всех пользователей, при прекращении работы серверного класса. Класс ServerObject так же содержит список подключений, реализованный в классе ClientObject. В классе ClientObject реализована процедура получения первой информации от подключившегося пользователя (его имени) и в бесконечном цикле получение от пользователя сообщений. В виду утери после изменений кода, первоначального вида функций, вставки кода отсутствуют.

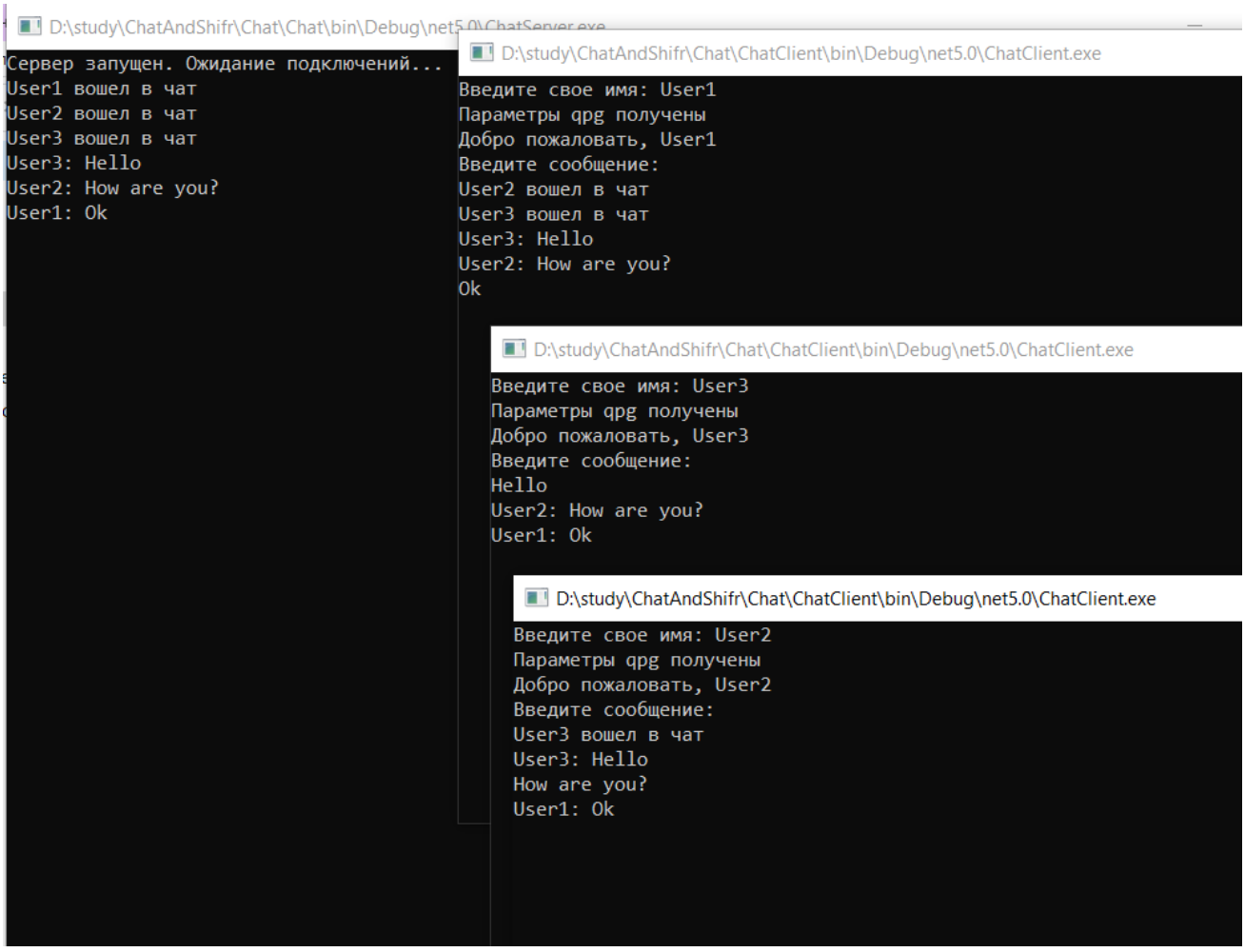
Класс для создания ключей по протоколу Диффи-Хеллмана был взят из первой лабораторной и немного преобразован на разные функции. Функция для генерации q, p, g осталась прежней.

Для шифрования сообщений были изменены как серверный, так и клиентский проект. В клиентском проекте появилось шифрование данных при отправке сообщений и дешифрование при получении и методы для генерации закрытого ключа. Таким образом, серверу не известно, каким является закрытый ключ X и общий ключ Z.

Для работы приложения нужно запустить файлы ChatServer.exe и несколько ChatClient.exe. После этого будет открыты консоль для сервера и консоли для каждого клиента. После добавление каждого из клиентов, происходит рассылка открытых ключей всем остальным клиентам и получение всех открытых ключей уже добавленных клиентов этим клиентом. Для каждого из открытых ключей каждый пользователь создает общий ключ Z.

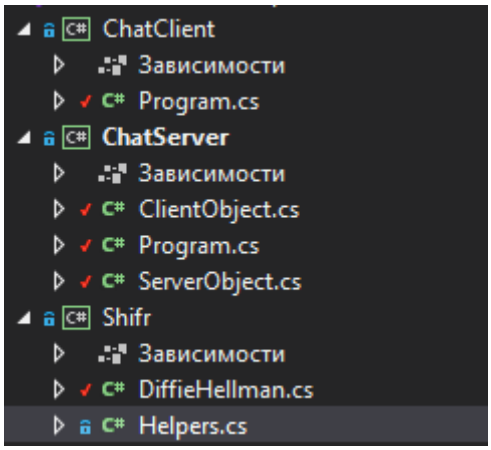
После написания сообщения первым клиентом, оно шифруется для каждого клиента согласно общему ключу и рассылается всем клиентам, где дешифруется каждым из них благодаря полученному в этом же сообщении открытому ключу клиента, отправившего сообщение. Таким образом, не происходит каждый раз вычисление общего ключа, что ускоряет время работы, а также сервер не знает значения секретных ключей пользователей.

Пример работы приложения:



Листинг

Структура:



ChatClient:

```
namespace ChatClient
{
    class Program
    {
        static string userName;
        private const string host = "127.0.0.1";
        private const int port = 8888;
        static TcpClient client;
        static NetworkStream stream;
        private static BigInteger X;
        private static BigInteger Y;
        private static string YString;
        private static CryptoInitializers Keys;
        private static List<Tuple<BigInteger, BigInteger, string>> UserList = new
List<Tuple<BigInteger, BigInteger, string>>(); //Y, Z, YString

        static void Main(string[] args)
        {
            Console.Write("Введите свое имя: ");
            userName = Console.ReadLine();
            client = new TcpClient();
            try
            {
                client.Connect(host, port);
                stream = client.GetStream();

                GetKeys();

                string message = userName + "," + YString;
                byte[] data = Encoding.ASCII.GetBytes(message);
                stream.Write(data, 0, data.Length);

                Thread receiveThread = new Thread(new ThreadStart(ReceiveMessage));
                receiveThread.Start();
                Console.WriteLine("Добро пожаловать, {0}", userName);
                SendMessage();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
```

```
            {
                Disconnect();
            }
        }

        private static void GetKeys()
        {
            try
            {
                byte[] data = new byte[64];
                StringBuilder builder = new StringBuilder();
                int bytes = 0;
                do
                {
                    bytes = stream.Read(data, 0, data.Length);
                    builder.Append(Encoding.ASCII.GetString(data, 0, bytes));
                } while (stream.DataAvailable);

                string message = builder.ToString();
                var list = message.Split("XXXKEYXXX");
                var keys = JsonSerializer.Deserialize<CryptoInitializersString>(list[0]);
                Keys = DiffieHellman.Converter(keys);
                X = DiffieHellman.GetCloseKeyX(Keys);
                Y = DiffieHellman.GetOpenKeyY(Keys, X);
                YString = Y.ToString();
                var listw = list[1].Split(",");
                for (int i = 1; i < listw.Length; i++)
                {
                    var y = new BigInteger(Encoding.ASCII.GetBytes(listw[i]));
                    var z = DiffieHellman.GetZ(y, X, Keys.P);
                    UserList.Add(new Tuple<BigInteger, BigInteger, string>(y, z,
y.ToString()));
                }
                // Console.WriteLine("Параметры qpg получены");
                return;
            }
            catch
            {
                Console.WriteLine("Подключение прервано! Попробуйте еще раз.");
                Console.ReadLine();
                Disconnect();
            }
        }

        static void SendMessage()
        {
            Console.WriteLine("Введите сообщение: ");

            while (true)
            {
                string message = Console.ReadLine();
                byte[] data = Encrypt(message);
                stream.Write(data, 0, data.Length);
            }
        }
        static byte[] Encrypt(string message)
        {
            message = String.Format("{0}: {1}", userName, message);
            var msgList = new List<string>();
            foreach (var user in UserList)
            {
                var msg = user.Item3 + "XXXKEYXXX";
                var shifrMsgByte = Encrypt(message, user.Item2.ToString());
                var shifrMsg = Encoding.ASCII.GetString(shifrMsgByte);
                msg += shifrMsg;
```

```
                msgList.Add(msg);
            }
            var shifrMessage = JsonSerializer.Serialize(msgList);
            return Encoding.ASCII.GetBytes(shifrMessage);
        }
        static void ReceiveMessage()
        {
            while (true)
            {
                try
                {
                    byte[] data = new byte[64];
                    StringBuilder builder = new StringBuilder();
                    int bytes = 0;
                    do
                    {
                        bytes = stream.Read(data, 0, data.Length);
                        builder.Append(Encoding.ASCII.GetString(data, 0, bytes));
                    }
                    while (stream.DataAvailable);

                    string messageShift = builder.ToString();
                    if (messageShift.Contains("XXXNEWUSER"))
                    {
                        try
                        {
                            var str = messageShift.Replace("XXXNEWUSER", "");
                            var y = new BigInteger(Encoding.ASCII.GetBytes(str));
                            var tw = y.ToString();
                            var z = DiffieHellman.GetZ(y, X, Keys.P);
                            UserList.Add(new Tuple<BigInteger, BigInteger, string>(y, z,
str));
                            return;
                        }
                        catch
                        {
                            Console.WriteLine("Новый пользователь не добавлен!");
                            Console.ReadLine();
                        }
                    }
                    var message = messageShift.Contains("XXXKEYXXX") ?
DecoderMessage(messageShift) : messageShift;
                    Console.WriteLine(message);
                }
                catch
                {
                    Console.WriteLine("Подключение прервано!");
                    Console.ReadLine();
                    Disconnect();
                }

            }
        }

        private static string DecoderMessage(string messageShift)
        {
            try
            {
                var strM = messageShift.Split("XXXKEYXXX");
                var y = new BigInteger(Encoding.ASCII.GetBytes(strM[0]));
                var msgS = strM[1];
                var message = string.Empty;
                var z = UserList.FirstOrDefault(x => x.Item1.CompareTo(y) == 0)?.Item3;
                var msg = Decrypt(Encoding.ASCII.GetBytes(msgS), z);
```

```
                return msg;
            }
            catch
            {
                Console.WriteLine("Не удалось декодировать прервано!");
                Console.ReadLine();
                return "";
            }
        }

        static void Disconnect()
        {
            if (stream != null)
                stream.Close();
            client.Close();
            Environment.Exit(0);
        }
        private static byte[] Encrypt(string clearText, string EncryptionKey)
        {
            var EncryptionKeyByte = Encoding.ASCII.GetBytes(EncryptionKey);
            byte[] clearBytes = Encoding.ASCII.GetBytes(clearText);
            byte[] encrypted;
            using (Aes encryptor = Aes.Create())
            {
                encryptor.Key = EncryptionKeyByte.SkipLast(EncryptionKeyByte.Length -
32).ToArray();
                encryptor.IV = EncryptionKeyByte.SkipLast(EncryptionKeyByte.Length -
16).ToArray();
                using (MemoryStream ms = new MemoryStream())
                {
                    using (CryptoStream cs = new CryptoStream(ms,
encryptor.CreateEncryptor(), CryptoStreamMode.Write))
                    {
                        cs.Write(clearBytes, 0, clearBytes.Length);
                        cs.Close();
                    }
                    encrypted = ms.ToArray();
                }
            }
            return encrypted;
        }

        private static string Decrypt(byte[] cipherBytes, string EncryptionKey)
        {
            var EncryptionKeyByte = Encoding.ASCII.GetBytes(EncryptionKey);
            string cipherText = "";
            using (Aes encryptor = Aes.Create())
            {
                encryptor.Key = EncryptionKeyByte.SkipLast(EncryptionKeyByte.Length -
32).ToArray();
                encryptor.IV = EncryptionKeyByte.SkipLast(EncryptionKeyByte.Length -
16).ToArray();
                using (MemoryStream ms = new MemoryStream())
                {
                    using (CryptoStream cs = new CryptoStream(ms,
encryptor.CreateDecryptor(), CryptoStreamMode.Write))
                    {
                        cs.Write(cipherBytes, 0, cipherBytes.Length);
                        cs.Close();
                    }
                    cipherText = Encoding.ASCII.GetString(ms.ToArray());
                }
            }
            return cipherText;
        }
```

```
    }
}
```

ChatServer:

```
namespace ChatServer
{
    namespace ChatServer
    {
        public class ClientObject
        {
            protected internal string Id { get; private set; }
            protected internal NetworkStream Stream { get; private set; }
            string userName;
            TcpClient client;
            ServerObject server;
            private BigInteger Y;
            protected internal string YString;

            public ClientObject(TcpClient tcpClient, ServerObject serverObject)
            {
                Id = Guid.NewGuid().ToString();
                client = tcpClient;
                server = serverObject;
                serverObject.AddConnection(this);
            }

            public void Process()
            {
                try
                {
                    Stream = client.GetStream();
                    server.SendKey(this.Id);
                    string message = GetNameAndKey();
                    server.BroadcastMessage("XXXNEWUSER" + YString, this.Id);
                    userName = message;

                    message = userName + " вошел в чат";
                    server.BroadcastMessage(message, this.Id);
                    Console.WriteLine(message);
                    while (true)
                    {
                        try
                        {
                            message = GetMessage();
                            Console.WriteLine(message);
                            var model =
JsonSerializer.Deserialize<List<string>>(message);
                            foreach (var str in model)
                            {
                                var list = str.Split("XXXKEYXXX");

                                server.BroadcastMessageToUser(YString + "XXXKEYXXX" +
list[1], this.Id, list[0]);
                            }
                        }
                        catch
                        {
                            message = String.Format("{0}: покинул чат", userName);
                            Console.WriteLine(message);
                            server.BroadcastMessage(message, this.Id);
                            break;
                        }
```

```
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                server.RemoveConnection(this.Id);
                Close();
            }
        }

        private string GetNameAndKey()
        {
            byte[] data = new byte[64];
            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            do
            {
                bytes = Stream.Read(data, 0, data.Length);
                builder.Append(Encoding.ASCII.GetString(data, 0, bytes));
            } while (Stream.DataAvailable);

            var str = builder.ToString();
            var strMas = str.Split(",");
            var name = strMas[0];
            YString = strMas[1];
            Console.WriteLine(YString);

            return name;
        }

        private string GetMessage()
        {
            byte[] data = new byte[64];
            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            do
            {
                bytes = Stream.Read(data, 0, data.Length);
                builder.Append(Encoding.ASCII.GetString(data, 0, bytes));
            } while (Stream.DataAvailable);

            return builder.ToString();
        }

        protected internal void Close()
        {
            if (Stream != null)
                Stream.Close();
            if (client != null)
                client.Close();
        }
    }
}

namespace ChatServer
{
    class Program
    {
        static ServerObject server;
        static Thread listenThread;
        static void Main(string[] args)
```

```
        {
            try
            {
                server = new ServerObject();
                listenThread = new Thread(new ThreadStart(server.Listen));
                listenThread.Start();
            }
            catch (Exception ex)
            {
                server.Disconnect();
                Console.WriteLine(ex.Message);
            }
        }
    }
}

namespace ChatServer
{
    public class ServerObject
    {
        static TcpListener tcpListener; // сервер для прослушивания
        List<ClientObject> clients = new List<ClientObject>(); // все подключения
        public CryptoInitializers Keys;
        public string JsonKeys;

        public ServerObject()
        {
            Keys = DiffieHellman.GetOpenParametersAll();
            var stringModel = DiffieHellman.Converter(Keys);
            JsonKeys = JsonSerializer.Serialize(stringModel);
        }
        protected internal void AddConnection(ClientObject clientObject)
        {
            clients.Add(clientObject);
        }
        protected internal void RemoveConnection(string id)
        {
            ClientObject client = clients.FirstOrDefault(c => c.Id == id);
            if (client != null)
                clients.Remove(client);
        }
        protected internal void Listen()
        {
            try
            {
                tcpListener = new TcpListener(IPAddress.Any, 8888);
                tcpListener.Start();
                Console.WriteLine("Сервер запущен. Ожидание подключений...");

                while (true)
                {
                    TcpClient tcpClient = tcpListener.AcceptTcpClient();

                    ClientObject clientObject = new ClientObject(tcpClient, this);
                    Thread clientThread = new Thread(new
ThreadStart(clientObject.Process));
                    clientThread.Start();
                    Task.Delay(3);

                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Disconnect();
```

```
            }
        }

        protected internal void BroadcastMessage(string message, string id)
        {
            byte[] data = Encoding.ASCII.GetBytes(message);
            for (int i = 0; i < clients.Count; i++)
            {
                if (clients[i].Id != id)
                {
                    clients[i].Stream.Write(data, 0, data.Length);
                }
            }
        }
        protected internal void BroadcastMessageToUser(string message, string id, string
yOpen)
        {
            byte[] data = Encoding.ASCII.GetBytes(message);
            var index = clients.FindIndex(x => x.Equals(yOpen));
            if (index >= 0)
            {
                clients[index].Stream.Write(data, 0, data.Length);
            }
        }
        protected internal void Disconnect()
        {
            tcpListener.Stop();

            for (int i = 0; i < clients.Count; i++)
            {
                clients[i].Close();
            }
            Environment.Exit(0);
        }

        protected internal void SendKey(string id)
        {
            var msg = JsonKeys + "XXXKEYXXX";
            foreach (var item in clients.Where(x=> x.Id != id).ToList())
            {
                msg += "," + item.YString;
            }
            byte[] data = Encoding.ASCII.GetBytes(msg);
            var index = clients.FindIndex(x => x.Id == id);
            clients[index].Stream.Write(data, 0, data.Length);
        }
    }
}
```

Shifr:

```
namespace Shifr
{
    public record CryptoInitializers
    {
        public BigInteger P { get; set; }
        public BigInteger Q { get; set; }
        public BigInteger G { get; set; }
    }
    public record CryptoInitializersString
    {
        public string P { get; set; }
        public string Q { get; set; }
        public string G { get; set; }
    }
```

```
    public static class DiffieHellman
    {
        private static CryptoInitializers CyclingSubgroupPowerOfQ(Random random, int
keySizeQ, int keySizeP)
        {
            BigInteger q;
            q = ((Func<BigInteger>)(() =>
            {
                var buffer = new byte[keySizeQ];
                while (true)
                {
                    random.NextBytes(buffer);
                    buffer[^1] |= 0x80;

                    var value = new BigInteger(buffer, true);

                    if (value.IsProbablyPrime(random))
                        return value;
                }
            }))();

            BigInteger p;
            p = ((Func<BigInteger>)(() =>
            {
                var buffer = new byte[keySizeP - keySizeQ];
                while (true)
                {
                    random.NextBytes(buffer);
                    buffer[^1] |= 0x80;

                    var value = (new BigInteger(buffer, true) * q + 1);

                    if (value.IsProbablyPrime(random))
                        return value;
                }
            }))();

            var g = ((Func<BigInteger>)(() =>
            {
                var buffer = new byte[keySizeP - keySizeQ];
                while (true)
                {
                    random.NextBytes(buffer);

                    var r = new BigInteger(buffer, true);
                    var g = BigInteger.ModPow(r, (p - 1) / q, p);

                    if (g > 1 && BigInteger.ModPow(g, q, p) == 1)
                        return g;
                }
            }))();

            return new CryptoInitializers
            {
                P = p,
                Q = q,
                G = g
            };
        }

        public static CryptoInitializers GetOpenParametersAll()
        {
            var rand = new Random();
            var init = CyclingSubgroupPowerOfQ(rand, 32, 128);
            return init;
```

```
        }
        public static BigInteger GetCloseKeyX(CryptoInitializers init)
        {
            var rand = new Random();
            var a = ((Func<BigInteger>)(() =>
            {
                var buffer = new byte[(init.P - 1).GetByteCount(true)];
                while (true)
                {
                    rand.NextBytes(buffer);

                    var value = new BigInteger(buffer, true);
                    if (value < init.P - 1 && BigInteger.GreatestCommonDivisor(value,
init.P - 1) == 1)
                        return value;
                }
            }))();

            return a;
        }

        public static CryptoInitializersString Converter(CryptoInitializers model)
        {
            var modelString = new CryptoInitializersString()
            {
                G = model.G.ToString(),
                P = model.P.ToString(),
                Q = ""
            };
            return modelString;
        }
        public static CryptoInitializers Converter(CryptoInitializersString model)
        {
            var item = new CryptoInitializers()
            {
                G = new BigInteger(Encoding.Default.GetBytes(model.G)),
                P = new BigInteger(Encoding.Default.GetBytes(model.P)),
                Q = new BigInteger()
            };
            return item;
        }

        public static BigInteger GetOpenKeyY(CryptoInitializers init, BigInteger x)
        {
            var A = BigInteger.ModPow(init.G, x, init.P);
            return A;
        }
        public static BigInteger GetZ(BigInteger y, BigInteger x, BigInteger p)
        {
            var Zab = BigInteger.ModPow(y, x, p);
            return Zab;
        }
    }
}

namespace Shifr
{
    public static class Helpers
    {
        public static bool IsProbablyPrime(this BigInteger value, Random rand, int
witnesses = 10)
        {
            if (value <= 1)
                return false;
```

```
            if (witnesses <= 0)
                witnesses = 10;

            var d = value - 1;
            var s = 0;

            while (d % 2 == 0)
            {
                d /= 2;
                s += 1;
            }

            var bytes = new byte[value.ToByteArray().LongLength];

            for (var i = 0; i < witnesses; i++)
            {
                BigInteger a;
                do
                {
                    rand.NextBytes(bytes);

                    a = new BigInteger(bytes, true);
                } while (a < 2 || a >= value - 2);

                var x = BigInteger.ModPow(a, d, value);
                if (x == 1 || x == value - 1)
                    continue;

                for (var r = 1; r < s; r++)
                {
                    x = BigInteger.ModPow(x, 2, value);

                    if (x == 1)
                        return false;
                    if (x == value - 1)
                        break;
                }

                if (x != value - 1)
                    return false;
            }

            return true;
        }
    }
}
```