

**Технически университет**

**София**

**Курсов проект по**

**„Криптографски методи за защита на информацията“**

**Тема: „RSA алгоритъм“**

**Изпълнен от:**

**Студент: Красимир Етов**

**Факултет: ФКСТ**

**Специалност: КСИ**

**Факултетен № 121216142**

**Група: 51**

**Дата: 26.05.2019 Проверил:**

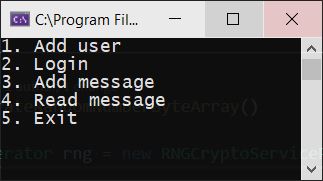
**Град: София**

**Задание**

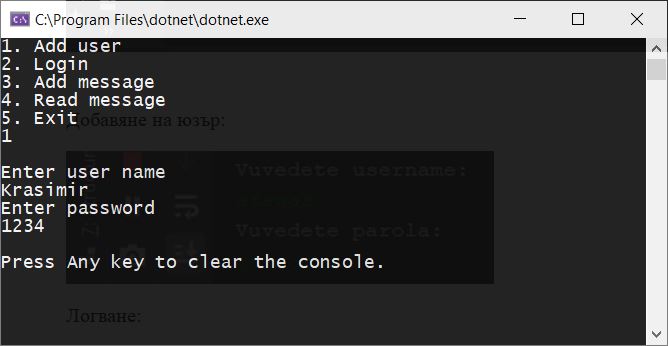
Да се реализира програмно RSA-алгоритъм. Генерираните прости числа да се подлагат на проверка с вероятностния метод на Rabin-Miller. Да се дефинират n потребители на системата и за всеки от тях да се определи парола, предназначена освен за контрол на достъпа, също и за осигуряване на еднозначното съответствие „потребител – секретен ключ“. Да се изчислят двойки криптографски ключове за всеки от потребители и да се запишат в два отделни файла – първият с публичните ключове, а вторият – със секретните. Да се осигури достъп на всички потребители до файла с публичните ключове, а секретните ключове от втория файл да бъдат защитени чрез шифроването му със симетричен алгоритъм. Да се представят тестови примери, които демонстрират последователно кодиране на текст, шифроване на текст, предназначен за конкретен потребител, както и резултатът от дешифрирането.

**Примери**

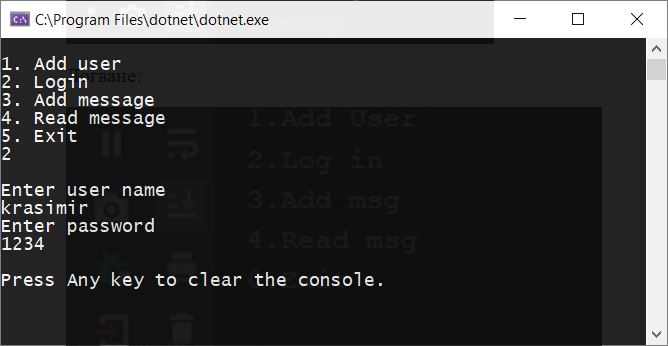
Начален екран:



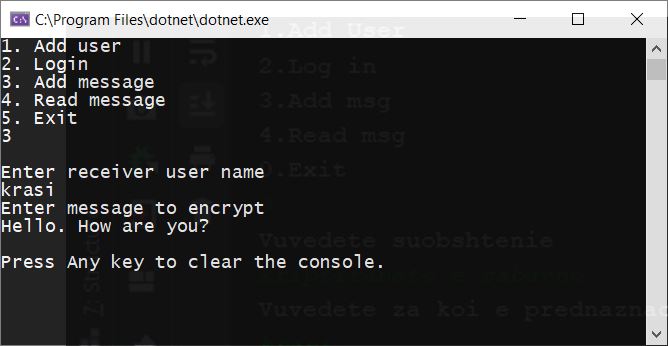
Добавяне на потребител:



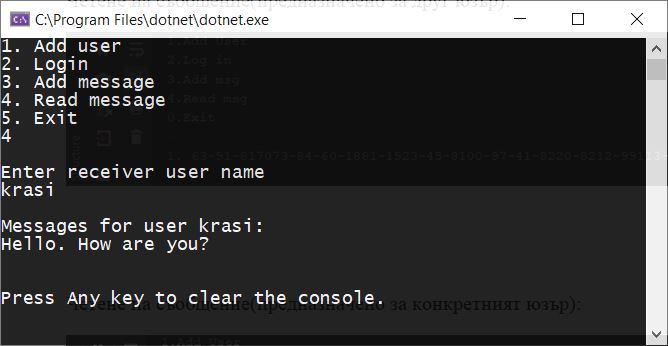
Вход на потребител:



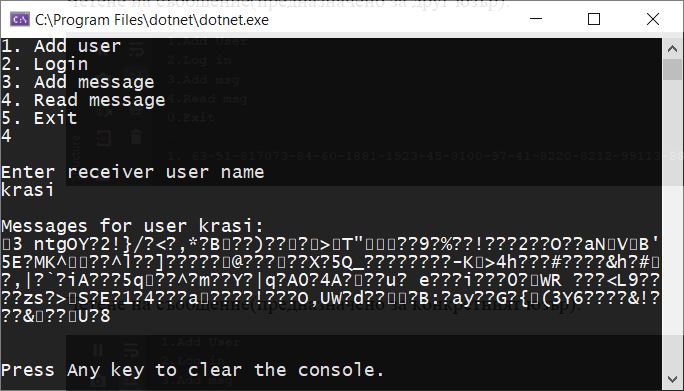
Добавяне на съобщение:



Четене на съобщение предназначето за правилния потребител:



Четене на съобщение предназначено за друг потребител:



**Програмна реализация - C#**

using System;

using System.Collections.Generic;

using System.IO;

using System.Security.Cryptography;

using System.Text;

using Org.BouncyCastle.Math;

namespace CourseProject

{

public class Program

{

public static Dictionary<string, List<BigInteger>> Messages { get; set; } = new Dictionary<string, List<BigInteger>>();

static void Main(string[] args)

{

while (true)

{

PrintMenu();

switch (Console.ReadLine())

{

case "1":

AddUser();

Console.WriteLine("\nPress Any key to clear the console.\n");

Console.ReadKey();

Console.Clear();

break;

case "2":

Login();

Console.WriteLine("\nPress Any key to clear the console.\n");

Console.ReadKey();

Console.Clear();

break;

case "3":

AddMessage();

Console.WriteLine("\nPress Any key to clear the console.\n");

Console.ReadKey();

Console.Clear();

break;

case "4":

ReadMessage();

Console.WriteLine("\nPress Any key to clear the console.\n");

Console.ReadKey();

Console.Clear();

break;

case "5":

return;

}

}

}

public static void PrintMenu()

{

Console.WriteLine("1. Add user");

Console.WriteLine("2. Login");

Console.WriteLine("3. Add message");

Console.WriteLine("4. Read message");

Console.WriteLine("5. Exit");

}

public static void AddUser()

{

Console.WriteLine();

Console.WriteLine("Enter user name");

var userName = Console.ReadLine();

Console.WriteLine("Enter password");

var password = Console.ReadLine();

var p = GetRandomNumber(); // random number that passes rabin miller test

var q = GetRandomNumber(); // random number that passes rabin miller test

var N = GenerateN(p, q); // not secret parameter

var fN = GenerateFn(p, q); // secret parameter - used later for keys

var publicKey = GetPublicKey(fN); // public key => 0 < k < fN, and gcd(k) = 1

var privateKey = GetPrivateKey(publicKey, fN); // multiplicative inverse

string usersInfo = userName + " " + password + " " + N + "\n";

string privateKeysInfo = userName + " " + privateKey.ToString() + "\n";

string publicKeysInfo = userName + " " + publicKey.ToString() + "\n";

File.AppendAllText(Environment.CurrentDirectory + "\\users.txt", usersInfo);

File.AppendAllText(Environment.CurrentDirectory + "\\privateKeys.txt", privateKeysInfo);

File.AppendAllText(Environment.CurrentDirectory + "\\publicKeys.txt", publicKeysInfo);

}

public static void Login()

{

Console.WriteLine();

Console.WriteLine("Enter user name");

var userName = Console.ReadLine();

Console.WriteLine("Enter password");

var password = Console.ReadLine();

GetCurrentUser(userName, password);

}

public static void AddMessage()

{

Console.WriteLine();

Console.WriteLine("Enter receiver user name");

var userName = Console.ReadLine();

Console.WriteLine("Enter message to encrypt");

var messageInput = Console.ReadLine();

GetCurrentUser(userName);

var message = EncodeMessage(messageInput, CurrentUser.PublicKey, CurrentUser.N);

if (Messages.ContainsKey(userName))

{

Messages[userName].Add(message);

}

else

{

var messages = new List<BigInteger>() { message };

Messages.Add(userName, messages);

}

}

public static void ReadMessage()

{

Console.WriteLine();

StringBuilder decodedMessages = new StringBuilder();

Console.WriteLine("Enter receiver user name");

var userName = Console.ReadLine();

if (!Messages.ContainsKey(userName))

{

Console.WriteLine("No messages for user {0}", userName);

return;

}

var userMessages = Messages[userName];

foreach (var msg in userMessages)

{

decodedMessages.AppendLine(DecodeMessage(msg, CurrentUser.PrivateKey, CurrentUser.N));

}

Console.WriteLine();

Console.WriteLine("Messages for user {0}:", userName);

Console.WriteLine(decodedMessages);

}

private static void GetCurrentUser(string userName, string password = null)

{

string[] userFile = File.ReadAllLines(Environment.CurrentDirectory + "\\users.txt");

foreach (var userLine in userFile)

{

var userInfo = userLine.Split();

if (userInfo[0] == userName)

{

CurrentUser.UserName = userInfo[0];

if (!string.IsNullOrEmpty(password))

{

CurrentUser.Password = userInfo[1];

}

CurrentUser.N = new BigInteger(userInfo[2]);

}

}

string[] privateKeysFile = File.ReadAllLines(Environment.CurrentDirectory + "\\privateKeys.txt");

foreach (var privateKeyLine in privateKeysFile)

{

var userInfo = privateKeyLine.Split();

if (userInfo[0] == userName)

{

CurrentUser.PrivateKey = new BigInteger(userInfo[1]);

}

}

string[] publicKeysFile = File.ReadAllLines(Environment.CurrentDirectory + "\\publicKeys.txt");

foreach (var publicKeysLine in publicKeysFile)

{

var userInfo = publicKeysLine.Split();

if (userInfo[0] == userName)

{

CurrentUser.PublicKey = new BigInteger(userInfo[1]);

}

}

}

private static BigInteger EncodeMessage(string message, BigInteger publicKey, BigInteger N)

{

byte[] messageToByteArray = Encoding.ASCII.GetBytes(message);

BigInteger encoded = new BigInteger(messageToByteArray);

return encoded.ModPow(publicKey, N);

}

private static string DecodeMessage(BigInteger encodedMessage, BigInteger privateKey, BigInteger N)

{

var decodedMessage = encodedMessage.ModPow(privateKey, N);

return Encoding.ASCII.GetString(decodedMessage.ToByteArray());

}

private static BigInteger GetRandomNumber()

{

while (true)

{

Random random = new Random();

var number = new BigInteger(1024, random);

if (number.RabinMillerTest(10, random))

{

return number;

}

}

}

private static byte[] GenerateRandomNumberByteArray()

{

using (RandomNumberGenerator rng = new RNGCryptoServiceProvider())

{

byte[] bytes = new byte[1024];

rng.GetBytes(bytes);

return bytes;

}

}

private static BigInteger GenerateN(BigInteger p, BigInteger q)

{

return p.Multiply(q);

}

private static BigInteger GenerateFn(BigInteger p, BigInteger q)

{

return p.Subtract(BigInteger.One).Multiply(q.Subtract(BigInteger.One));

}

private static BigInteger GetPublicKey(BigInteger fN)

{

BigInteger K = GetRandomNumber();

while (fN.Gcd(K).CompareTo(BigInteger.One) > 0 && K.CompareTo(fN) < 0)

{

K = K.Add(BigInteger.One);

}

return K;

}

private static BigInteger GetPrivateKey(BigInteger publicKey, BigInteger fN)

{

return publicKey.ModInverse(fN);

}

}

}