**Лабораторная работа № 7**

**АСИММЕТРИЧНЫЕ АЛГОРИТМЫ В .NET**

**Вариант 9**

Цель работы. Изучить возможности реализации асимметричной криптосистемы средствами .NET

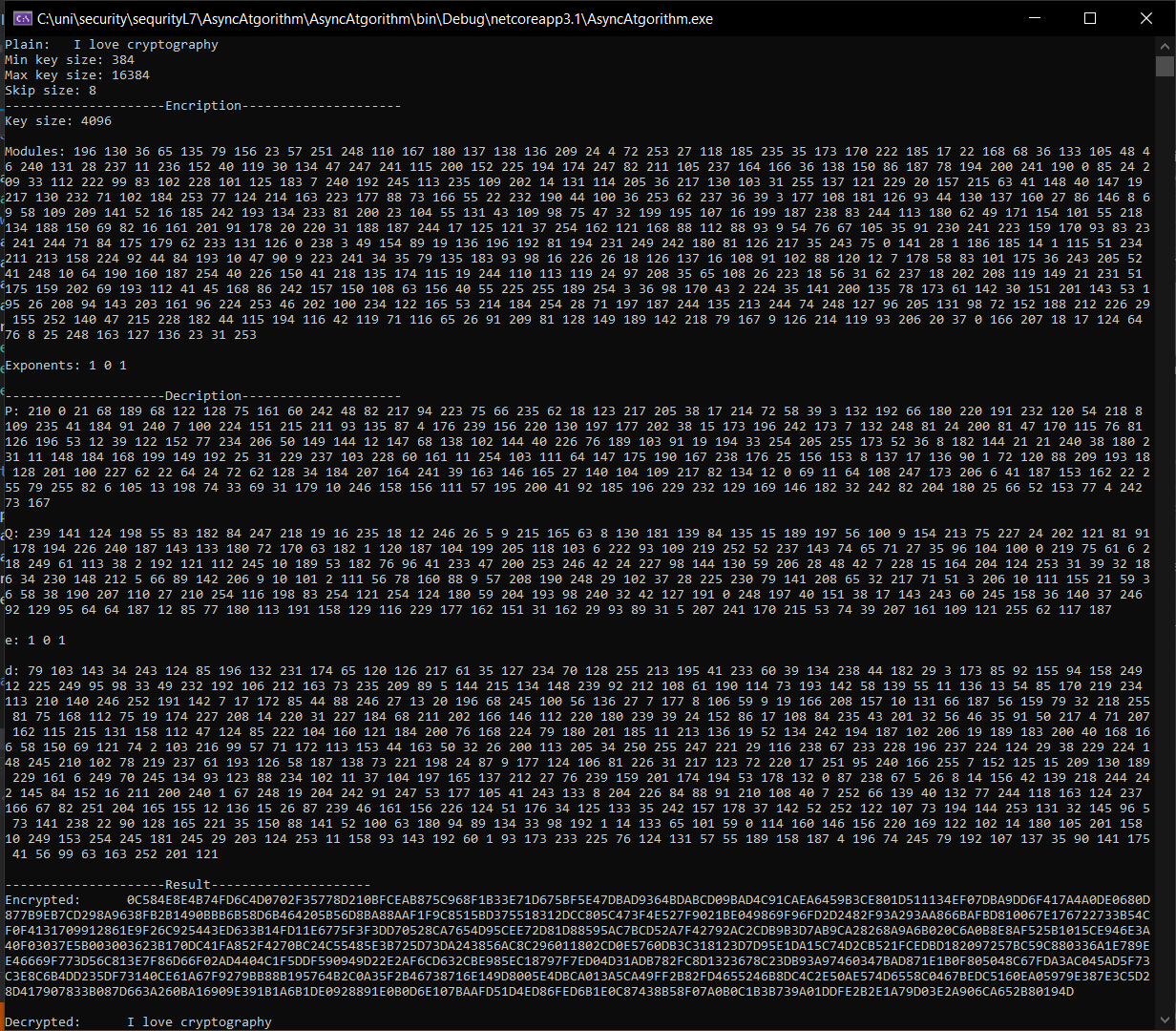
Воробей Иван Александрович 10701118

Размер ключа 4096.

Храню ключи в пользовательском контейнере.

Программа генерирует ключи, затем сохраняет в пользовательско контейнере. Затем получает сохраненные ключи по имени контейнера, шифрует введенный текст. Расшифровывает шифротекст. Затем удаляет контейнер.

Необязательно создавать ключи при каждом запуске, достаточно создать однажды и они будут сохранены.



**Листинг**

namespace AsyncAtgorithm

{

class Program

{

static int KEY\_SIZE\_4096 = 1 << 12;

static string CONTAINER\_NAME = "MyContainerName";

static void Main(string[] args)

{

Console.Write("Plain:\t ");

string message = Console.ReadLine();

PersistKeys();

Console.WriteLine("---------------------Encription---------------------");

byte[] encrypted = Encrypt(Encoding.UTF8.GetBytes(message));

Console.WriteLine("---------------------Decription---------------------");

byte[] decrypted = Decrypt(encrypted);

Console.WriteLine("---------------------Result---------------------");

DeleteKeyInCSP();

Console.WriteLine("Encrypted:\t" + BitConverter.ToString(encrypted).Replace("-", "") + "\n");

Console.WriteLine("Decrypted:\t" + Encoding.UTF8.GetString(decrypted));

Console.ReadLine();

}

static void PersistKeys()

{

int rsa\_provider = 1;

CspParameters cspParameters = new CspParameters(rsa\_provider);

cspParameters.KeyContainerName = CONTAINER\_NAME;

cspParameters.Flags = CspProviderFlags.UseMachineKeyStore;

cspParameters.ProviderName = "Microsoft Strong Cryptographic Provider";

var rsa = new RSACryptoServiceProvider(cspParameters);

rsa.PersistKeyInCsp = true;

Console.WriteLine("Min key size: " + rsa.LegalKeySizes[0].MinSize);

Console.WriteLine("Max key size: " + rsa.LegalKeySizes[0].MaxSize);

Console.WriteLine("Skip size: " + rsa.LegalKeySizes[0].SkipSize);

}

public static void DeleteKeyInCSP()

{

var cspParams = new CspParameters();

cspParams.KeyContainerName = CONTAINER\_NAME;

var rsa = new RSACryptoServiceProvider(cspParams);

rsa.PersistKeyInCsp = false;

rsa.Clear();

}

private static byte[] Encrypt(byte[] plain)

{

byte[] encrypted;

int rsa\_provider = 1;

CspParameters cspParameters = new CspParameters(rsa\_provider);

cspParameters.KeyContainerName = CONTAINER\_NAME;

var rsa = new RSACryptoServiceProvider(KEY\_SIZE\_4096, cspParameters);

RSAParameters rSAParameters = rsa.ExportParameters(true);

string modules = "";

foreach(Byte b in rSAParameters.Modulus) { modules += b + " "; }

string exps = "";

foreach (Byte b in rSAParameters.Exponent) { exps += b + " "; }

Console.WriteLine("Key size: " + rsa.KeySize + "\n");

Console.WriteLine("Modules: " + modules + "\n");

Console.WriteLine("Exponents: " + exps + "\n");

encrypted = rsa.Encrypt(plain, true);

return encrypted;

}

private static byte[] Decrypt(byte[] encrypted)

{

byte[] decrypted;

CspParameters cspParameters = new CspParameters();

cspParameters.KeyContainerName = CONTAINER\_NAME;

var rsa = new RSACryptoServiceProvider(KEY\_SIZE\_4096, cspParameters);

RSAParameters rSAParameters = rsa.ExportParameters(true);

Console.WriteLine("P: " + ArrayToStr(rSAParameters.P) + "\n");

Console.WriteLine("Q: " + ArrayToStr(rSAParameters.Q) + "\n");

Console.WriteLine("e: " + ArrayToStr(rSAParameters.Exponent) + "\n");

Console.WriteLine("d: " + ArrayToStr(rSAParameters.D) + "\n");

decrypted = rsa.Decrypt(encrypted, true);

return decrypted;

}

private static string ArrayToStr(Byte[] array)

{

string modules = "";

foreach (Byte b in array) { modules += b + " "; }

return modules;

}

}

}