

Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Основы криптографии
Лабораторная работа №4

Факультет: прикладной математики и информатики
Группа: ПМ-63
Студенты: Шепрут И.М.
 Кожекин М.В.
 Утюганов Д.С.
Преподаватель: Ступаков И.М.

Новосибирск

2018

1. Цель работы

Изучить работу с цифровой подписью и шифрование данных в языке C#.

2. Ход работы

Задача 1

Используя класс **RSA** сгенерировать открытый и закрытый ключ. Выложить открытый ключ в общую папку.

RSA Algorithm

To generate a key pair, you start by creating two large prime numbers named p and q . These numbers are multiplied and the result is called n . Because p and q are both prime numbers, the only factors of n are 1, p , q , and n .

If we consider only numbers that are less than n , the count of numbers that are relatively prime to n , that is, have no factors in common with n , equals $(p - 1)(q - 1)$.

Now you choose a number e , which is relatively prime to the value you calculated. The public key is now represented as $\{e, n\}$.

To create the private key, you must calculate d , which is a number such that $(d)(e) \bmod (p - 1)(q - 1) = 1$. In accordance with the Euclidean algorithm, the private key is now $\{d, n\}$.

Encryption of plaintext m to ciphertext c is defined as $c = (m^e) \bmod n$. Decryption would then be defined as $m = (c^d) \bmod n$.

GenerateRsaKeys.cs

```
using System;
using System.IO;
using System.Security.Cryptography;

class GenerateKeys
{
    static RSA rsa;
    static string to_string(byte[] data)
    {
        return BitConverter.ToString(data).Replace("-", "").ToLowerInvariant();
    }

    static void WriteKeysToFiles()
    {
        // Сначала сохраним XML ключи
        File.WriteAllText("private_key_XML.txt", rsa.ToXmlString(true));
        File.WriteAllText("public_key_XML.txt", rsa.ToXmlString(false));

        string Modulus, Exponent, d, p, q;

        var parameters = rsa.ExportParameters(true);
        Modulus = to_string(parameters.Modulus);
        Exponent = to_string(parameters.Exponent);
        d = to_string(parameters.D);
        p = to_string(parameters.P);
        q = to_string(parameters.Q);
    }
}
```

```

string privateKeyFile = "private_key.txt",
    publicKeyFile = "public_key.txt";

File.WriteAllText(privateKeyFile, "Private exponent (d):\n" + d);
File.AppendAllText(privateKeyFile, "\nModulus (n):\n" + Modulus);
File.AppendAllText(privateKeyFile, "\nP (p):\n" + p);
File.AppendAllText(privateKeyFile, "\nQ (q):\n" + q);

File.WriteAllText(publicKeyFile, "Exponent (e):\n" + Exponent);
File.AppendAllText(publicKeyFile, "\nModulus (n):\n" + Modulus);

}

static void Main()
{
    // Инициализация
    rsa = RSA.Create();

    WriteKeysToFiles();
}
}

```

private_key.txt

```

Private exponent (d):
66e67f6a3ed509b18f45a63c6ff835a949c659e4a1013a14f121d01a2cb703937ce45d6abd8ba5167ea3259df9
000a900adc97e2ab9c667f466ad85341ca85e536f41cf6a6d288bf1b8995089dbf987082d39ef3adfb0635f4be
094466efa984598bbb339581f9e37fc49aa3cd4e77834a1d886a80ee9a7e55a395481411693d
Modulus (n):
a072a4521173a2cd97c21a563f04cda82e242a30c17a473d04b357e53ae60c93088b5ec8e72561fb8d55601ff4
aef8e163f8a27f1c58c4634f1d06e3a5f71e28ec9c137f0943fbaac2875c6d1de2a6eac20f85e5546536b20ad7
334dd55f882a6ee8fbaf51ec3f709dca8b4514a550a1c0365f77c79121ced32805a25866ffe9
P (p):
c17039e756e7a260c37eb573d52b871fdf223616e563f62124d16f800ed33deafc0b1067377f426d9e71cbf5bd
d0c2ac8450b71a0ba58831a2a14a01314e60cb
Q (q):
d456f464ad062c6a25c447b9bc0c600b09e1f111b33f53f0d7168b77c5e6e95366ce8461fa7d125200a7d22885
f0c7b63bccfca7dcc45c63e7c2bd49ca838f9b

```

public_key.txt

```

Exponent (e):
010001
Modulus (n):
a072a4521173a2cd97c21a563f04cda82e242a30c17a473d04b357e53ae60c93088b5ec8e72561fb8d55601ff4
aef8e163f8a27f1c58c4634f1d06e3a5f71e28ec9c137f0943fbaac2875c6d1de2a6eac20f85e5546536b20ad7
334dd55f882a6ee8fbaf51ec3f709dca8b4514a550a1c0365f77c79121ced32805a25866ffe9

```

private_key_XML.txt

```

<RSAKeyValue><Modulus>oHKkUhFzos2XwhpWPwTNqC4kKjDBekc9BLNX5TrmDJMIi17I5yVh+41VYB/0rvjhY/ii
fxxYxGNPHQbjpfceK0ycE38JQ/uqwodcbR3ipurCD4XlVGU2sgrXM03VX4gqbu7r1HsP3CdyotFFKVQocA2X3fHkS
H00ygFolhm/+k=</Modulus><Exponent>AQAB</Exponent><P>wXA551bnomDDfrVz1SuHH98iNhblY/YhJNFvGA
7TPer8CxBn39CbZ5xy/W90MKshFC3GguliDGioUoBMU5gyw==</P><Q>1Fb0ZK0GLGolxEe5vAxcGwnh8RGzP1Pw1
xaLd8Xm6VNmzoRh+n0SUGCn0iiF8Me208z8p9zEXGPnwr1JyoOPmw==</Q><DP>kH+BAasNhWK7Jp/tU0QcwFYyfC1
GVhG6WIKKxCJ08mXiuXEbaX2K5dACOdXsYVHLBVw2KyKgGMjbl5jUozx1WQ==</DP><DQ>ZVS1vlgfr5RcP4XnCe1x
7/KgYzWKL790IIC+v8KuhkxptEYSp8IF+yIa9DyoZpY3zePt8oF1J435rMI+M6mC3w==</DQ><InverseQ>005eD0E
+/IXVao2VJ7pSvpV0ZMXtZLx4b08SyDsAgNEBeYWySeU048zqYGumcd0K7tnGOMq1Iamow+RthAa2QA==</Inverse
Q><D>ZuZ/aj7VCbGPRaY8b/g1qUnGWeShAToU8SHQGiy3A5N85F1qvYulFn6jJZ35AAQcQcTyX4qucZn9GathTQcqF5
Tb0HPam0oi/G4mVCJ2/mHCC057zrfSGNFs+CURm76mEWYw7M5WB+eN/xJqjzU53g0odiGqA7pp+VaOVSBQRaT0=</D
></RSAKeyValue>

```

public_key_XML.txt

```
<RSAKeyValue><Modulus>oHkKUhFzos2XwhpWPwTNqC4kKjDBekc9BLNX5TrmDJMi17I5yVh+41VYB/0rvjhY/ii  
fxxYxGNPHQbjpfceK0ycE38JQ/uqwodcbR3ipurCD4XlVGU2sgrXM03VX4gqbuj7r1HsP3CdyotFFKVQocA2X3fHkS  
H00ygFolhm/+k=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
```

Задача 2

Взять текстовый файл и используя **RSA.SignData** сгенерировать для него цифровую подпись.
Записать исходный файл + подпись в новый файл

Возьмём исходный файл:

Нечеткое множество называется конечным, если его носитель не является конечным множеством. Мощность такого множества является конечной и равна количеству элементов его носителя как обычного множества. Нечеткое множество называется бесконечным, если его носитель является конечным множеством. Счетное нечеткое множество имеет счетный носитель. Несчетным нечетким множеством называется нечеткое множество, имеющее носитель, имеющий мощность континуума.

PutASignature.cs

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

class PutASignature
{
    static RSA rsa;

    static void ReadKeysFromFiles()
    {
        rsa.FromXmlString(File.ReadAllText("private_key_XML.txt"));
    }

    static void SignFile()
    {
        Console.WriteLine("Введите название файла, который нужно подписать:");
        string fileName = Console.ReadLine();
        byte[] fileToSign = File.ReadAllBytes(fileName);
        byte[] signature = rsa.SignData(fileToSign, HashAlgorithmName.SHA256,
        RSASignaturePadding.Pkcs1);
        byte[] signedFile = new byte[fileToSign.Length + signature.Length];
        fileToSign.CopyTo(signedFile, 0);
        signature.CopyTo(signedFile, fileToSign.Length);

        File.WriteAllBytes("sign_for_" + fileName, signature); // Отдельно подпись
        File.WriteAllBytes("signed_" + fileName, signedFile); // И файл с подписью в
        конце

        Console.WriteLine("Файл подписан.");
    }

    static void Main(string[] args)
    {
        // Инициализация
        rsa = RSA.Create();
    }
}
```

```

        ReadKeysFromFiles();

        SignFile();
    }
}

```

```

C:\Windows\system32\cmd.exe
Введите название файла, который нужно подписать:
file_to_sign.txt
Файл подписан.
Для продолжения нажмите любую клавишу . . .

```

Откроем файл **signed_file_to_sign**

Нечеткое множество называется конечным, если его носитель не является конечным множеством. Мощность такого множества является конечной и равна количеству элементов его носителя как обычного множества. Нечеткое множество называется бесконечным, если его носитель является конечным множеством. Счетное нечеткое множество имеет счетный носитель. Несчетным нечетким множеством называется нечеткое множество, имеющее носитель, имеющий мощность континуума.

р, ±-; , ± _f c_3f_, _€_ =6DЩ, x0HPO{Ъ-
ЮтФик_{KEEXμUTc__±Aj_я_K{~fЪ/%ek-±_|Ш)р д=±~y1Ÿ_еиIhTİ. @бэ™o€_Ě! (Um-@A\ъ~•Л: a_шц3__EQa_
_Tİ__

Задача 3

Взять файл с подписью и используя **RSA.VerifyData** проверить подпись.

Validate.cs

```

using System;
using System.IO;
using System.Linq;
using System.Security.Cryptography;

class Validate
{
    static RSA rsa;

    static void ReadKeysFromFiles()
    {
        rsa.FromXmlString(File.ReadAllText("private_key_XML.txt"));
    }

    static void ValidateFile()
    {
        int signatureSize = 128;
        Console.WriteLine("Введите название файла, который нужно проверить:");
        string sourceFileName = Console.ReadLine();
        byte[] sourceFile = File.ReadAllBytes(sourceFileName);
        byte[] signature = new byte[signatureSize];

        Array.Copy(sourceFile, sourceFile.Length - signatureSize, signature, 0,
signatureSize);
        Array.Resize(ref sourceFile, sourceFile.Length - signatureSize);

        if (rsa.VerifyData(sourceFile, signature, HashAlgorithmName.SHA256,
RSASignaturePadding.Pkcs1))
            Console.WriteLine("Подпись верна.");
        else
            Console.WriteLine("Подпись не совпала");
    }
}

```

```

static void Main(string[] args)
{
    // Инициализация
    rsa = RSA.Create();

    ReadKeysFromFiles();

    ValidateFile();
}
}

```

Результат:

```

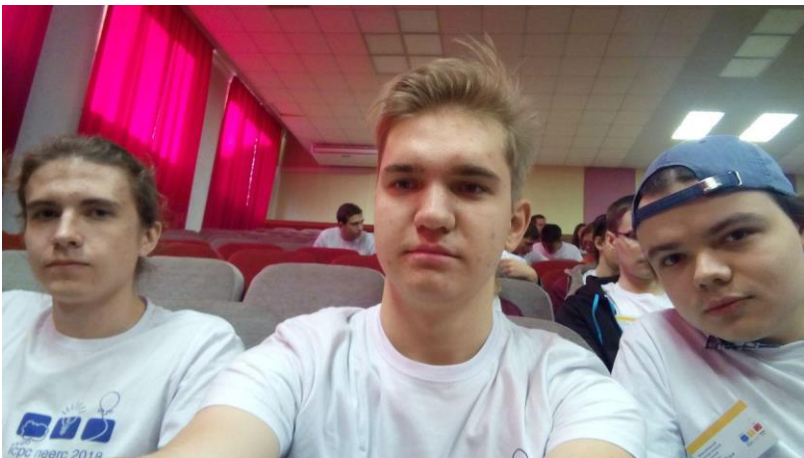
C:\Windows\system32\cmd.exe
Введите название файла, который нужно проверить:
signed_file_to_sign.txt
Подпись верна.
Для продолжения нажмите любую клавишу . . .

```

Задача 4

Сделать селфи бригады и зашифровать его с помощью класса **Aes**. Ключ **AES** зашифровать с помощью **RSA.Encrypt** (с публичным ключом следующей бригады) и добавить в конец зашифрованного файла. Передать файл следующей бригаде.

Возьмём фото и зашифруем его:



Encrypt.cs

```

using System;
using System.IO;
using System.Security.Cryptography;

class Encrypt
{
    static RSA rsa;
    static Aes aes;

    // Шифруем фото при помощи AES, а ключи AES при помощи RSA
    static void EncryptPhoto()
    {
        Console.WriteLine("Введите название файла, который нужно зашифровать:");
        string fileName = Console.ReadLine();
    }
}

```

```

byte[] pictureToEncrypt = File.ReadAllBytes(fileName);
byte[] encryptedPicture;

// Обработка ошибок
if (pictureToEncrypt == null || pictureToEncrypt.Length <= 0)
    throw new ArgumentNullException("pictureToEncrypt");
if (aes.Key == null || aes.Key.Length <= 0)
    throw new ArgumentNullException("Key");
if (aes.IV == null || aes.IV.Length <= 0)
    throw new ArgumentNullException("IV");

var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);
using (MemoryStream msEncrypt = new MemoryStream())
{
    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor,
CryptoStreamMode.Write))
    {
        csEncrypt.Write(pictureToEncrypt, 0, pictureToEncrypt.Length);
        csEncrypt.FlushFinalBlock();
        encryptedPicture = msEncrypt.ToArray();
    }
}

byte[] aesEncryptedKey = rsa.Encrypt(aes.Key, RSAEncryptionPadding.Pkcs1);
byte[] aesEncryptedIV = rsa.Encrypt(aes.IV, RSAEncryptionPadding.Pkcs1);

File.WriteAllBytes("pic_encrypted_with_AES.txt", encryptedPicture);
Console.WriteLine("Файл pic_encrypted_with_AES.txt сохранён.");
File.WriteAllBytes("AES_key_encrypted_with_RSA.txt", aesEncryptedKey);
Console.WriteLine("Файл AES_key_encrypted_with_RSA.txt сохранён.");
File.WriteAllBytes("AES_IV_encrypted_with_RSA.txt", aesEncryptedIV);
Console.WriteLine("Файл AES_IV_encrypted_with_RSA.txt сохранён.");
}

static void Main(string[] args)
{
    // Инициализация
    rsa = RSA.Create();
    aes = Aes.Create();

    //aes.GenerateKey();
    //aes.GenerateIV();
    //File.WriteAllBytes("AES_key.txt", aes.Key);
    //File.WriteAllBytes("AES_IV.txt", aes.IV);

    Console.WriteLine("Введите название файла с чужим публичным ключом в формате
XML:");
    string another_public_key = Console.ReadLine();

    Console.WriteLine("Введите название файла с ключом AES которым надо
зашифровать:");
    string aes_key = Console.ReadLine();

    Console.WriteLine("Введите название файла с IV AES:");
    string iv_key = Console.ReadLine();

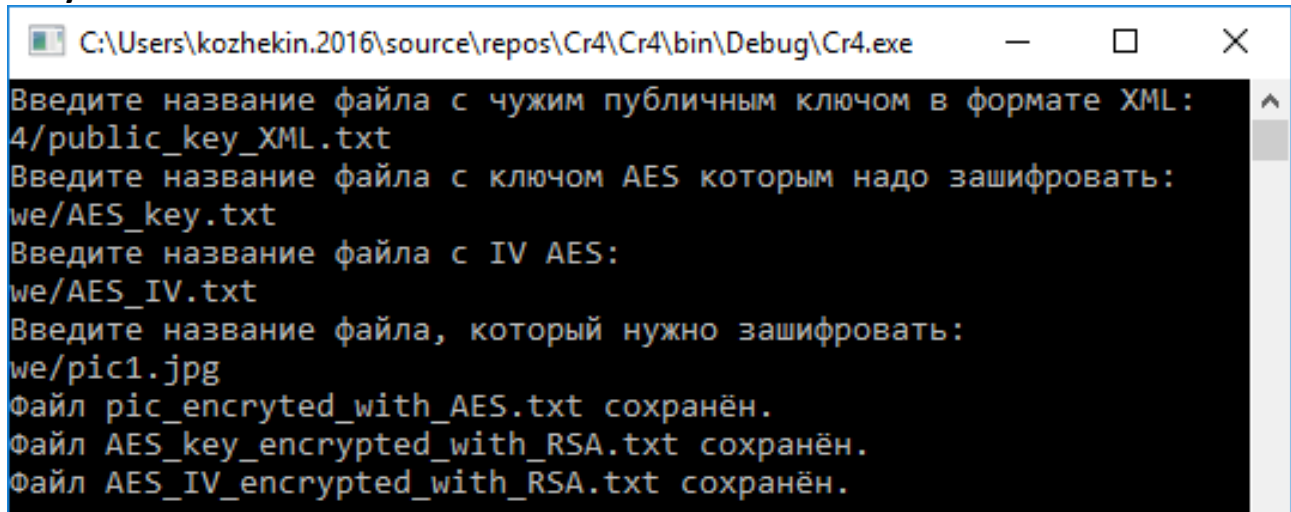
    rsa.FromXmlString(File.ReadAllText(another_public_key));
    aes.Key = File.ReadAllBytes(aes_key);
    aes.IV = File.ReadAllBytes(iv_key);

    EncryptPhoto();
}

```

```
        Console.ReadKey();  
    }  
}
```

Результат:



```
C:\Users\kozhekin.2016\source\repos\Cr4\Cr4\bin\Debug\Cr4.exe  
Введите название файла с чужим публичным ключом в формате XML:  
4/public_key_XML.txt  
Введите название файла с ключом AES которым надо зашифровать:  
we/AES_key.txt  
Введите название файла с IV AES:  
we/AES_IV.txt  
Введите название файла, который нужно зашифровать:  
we/pic1.jpg  
Файл pic_encrypted_with_AES.txt сохранён.  
Файл AES_key_encrypted_with_RSA.txt сохранён.  
Файл AES_IV_encrypted_with_RSA.txt сохранён.
```

Задача 5

Расшифровать файл от предыдущей бригады.

Decrypt.cs

```
using System;  
using System.IO;  
using System.Security.Cryptography;  
  
class Decrypt  
{  
    static RSA rsa;  
    static Aes aes;  
  
    static void DecryptPhoto()  
    {  
        Console.WriteLine("Введите название файла, содержащего приватный ключ в формате XML:");  
        string private_key = Console.ReadLine();  
  
        Console.WriteLine("Введите название файла, который нужно расшифровать:");  
        string pic_file = Console.ReadLine();  
  
        Console.WriteLine("Введите название файла, где находится зашифрованный ключ:");  
        string key_file = Console.ReadLine();  
  
        Console.WriteLine("Введите название файла, где находится зашифрованное IV:");  
        string iv_file = Console.ReadLine();  
  
        rsa.FromXmlString(File.ReadAllText(private_key));  
  
        byte[] pictureToDecrypt = File.ReadAllBytes(pic_file);  
        byte[] decryptedPicture;  
  
        aes.Key = rsa.Decrypt(File.ReadAllBytes(key_file), RSAEncryptionPadding.Pkcs1);  
        aes.IV = rsa.Decrypt(File.ReadAllBytes(iv_file), RSAEncryptionPadding.Pkcs1);  
    }  
}
```



```

// Обработка ошибок
if (pictureToDecrypt == null || pictureToDecrypt.Length <= 0)
    throw new ArgumentNullException("pictureToDecrypt");
if (aes.Key == null || aes.Key.Length <= 0)
    throw new ArgumentNullException("Key");
if (aes.IV == null || aes.IV.Length <= 0)
    throw new ArgumentNullException("IV");

var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);
using (MemoryStream msDecrypt = new MemoryStream())
{
    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor,
CryptoStreamMode.Write))
    {
        csDecrypt.Write(pictureToDecrypt, 0, pictureToDecrypt.Length);
        csDecrypt.FlushFinalBlock();
        decryptedPicture = msDecrypt.ToArray();
    }
}

File.WriteAllBytes("Decrypted.jpg", decryptedPicture);
Console.WriteLine("Файл Decrypted.jpg сохранён.");
}

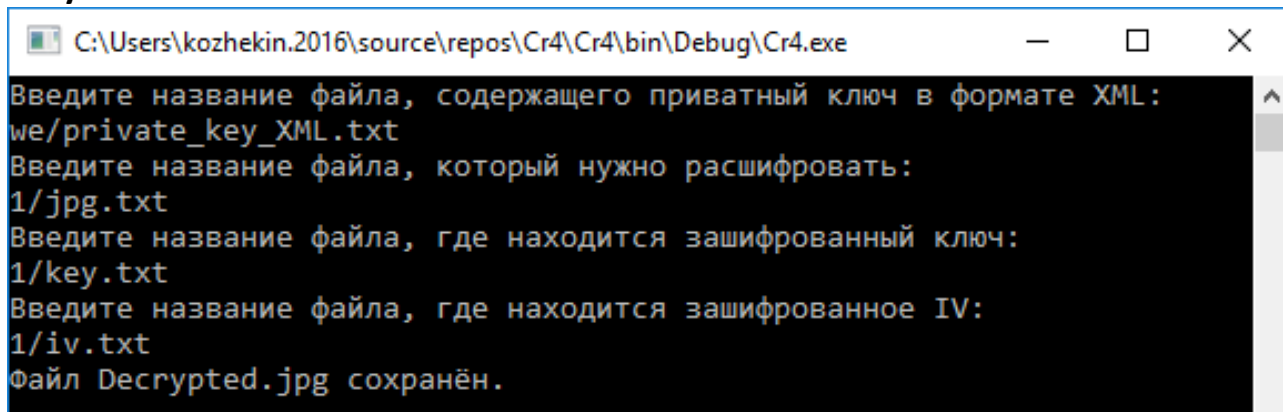
static void Main(string[] args)
{
    // Инициализация
    rsa = RSA.Create();
    aes = Aes.Create();

    DecryptPhoto();

    Console.ReadKey();
}
}

```

Результат:



```

C:\Users\kozhekin.2016\source\repos\Cr4\Cr4\bin\Debug\Cr4.exe
Введите название файла, содержащего приватный ключ в формате XML:
we/private_key_XML.txt
Введите название файла, который нужно расшифровать:
1/jpg.txt
Введите название файла, где находится зашифрованный ключ:
1/key.txt
Введите название файла, где находится зашифрованное IV:
1/iv.txt
Файл Decrypted.jpg сохранён.

```



3. Выводы

В ходе выполнения 4 лабораторной работы было изучена работа с цифровой подписью и шифрование данных в языке C#.