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

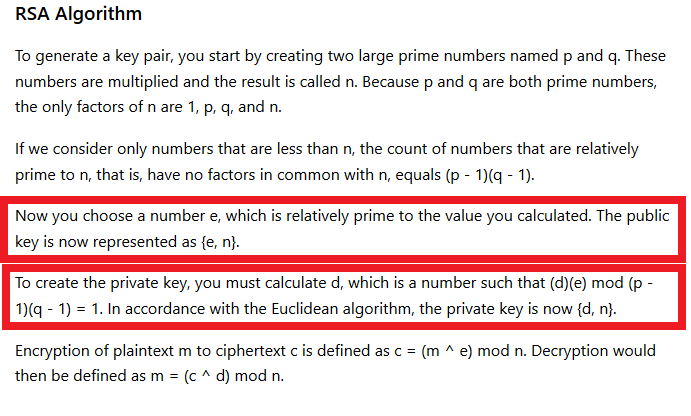
1. Цель работы

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

1. Ход работы

**Задача 1**

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



**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):  66e67f6a3ed509b18f45a63c6ff835a949c659e4a1013a14f121d01a2cb703937ce45d6abd8ba5167ea3259df9000a900adc97e2ab9c667f466ad85341ca85e536f41cf6a6d288bf1b8995089dbf987082d39ef3adfb0635f4be094466efa984598bbb339581f9e37fc49aa3cd4e77834a1d886a80ee9a7e55a395481411693d  Modulus (n):  a072a4521173a2cd97c21a563f04cda82e242a30c17a473d04b357e53ae60c93088b5ec8e72561fb8d55601ff4aef8e163f8a27f1c58c4634f1d06e3a5f71e28ec9c137f0943fbaac2875c6d1de2a6eac20f85e5546536b20ad7334dd55f882a6ee8fbaf51ec3f709dca8b4514a550a1c0365f77c79121ced32805a25866ffe9  P (p):  c17039e756e7a260c37eb573d52b871fdf223616e563f62124d16f800ed33deafc0b1067377f426d9e71cbf5bdd0c2ac8450b71a0ba58831a2a14a01314e60cb  Q (q):  d456f464ad062c6a25c447b9bc0c600b09e1f111b33f53f0d7168b77c5e6e95366ce8461fa7d125200a7d22885f0c7b63bccfca7dcc45c63e7c2bd49ca838f9b |

**public\_key.txt**

|  |
| --- |
| Exponent (e):  010001  Modulus (n):  a072a4521173a2cd97c21a563f04cda82e242a30c17a473d04b357e53ae60c93088b5ec8e72561fb8d55601ff4aef8e163f8a27f1c58c4634f1d06e3a5f71e28ec9c137f0943fbaac2875c6d1de2a6eac20f85e5546536b20ad7334dd55f882a6ee8fbaf51ec3f709dca8b4514a550a1c0365f77c79121ced32805a25866ffe9 |

**private\_key\_XML.txt**

|  |
| --- |
| <RSAKeyValue><Modulus>oHKkUhFzos2XwhpWPwTNqC4kKjDBekc9BLNX5TrmDJMIi17I5yVh+41VYB/0rvjhY/iifxxYxGNPHQbjpfceKOycE38JQ/uqwodcbR3ipurCD4XlVGU2sgrXM03VX4gqbuj7r1HsP3CdyotFFKVQocA2X3fHkSHO0ygFolhm/+k=</Modulus><Exponent>AQAB</Exponent><P>wXA551bnomDDfrVz1SuHH98iNhblY/YhJNFvgA7TPer8CxBnN39CbZ5xy/W90MKshFC3GguliDGioUoBMU5gyw==</P><Q>1Fb0ZK0GLGolxEe5vAxgCwnh8RGzP1Pw1xaLd8Xm6VNmzoRh+n0SUgCn0iiF8Me2O8z8p9zEXGPnwr1JyoOPmw==</Q><DP>kH+BAasNhWK7Jp/tU0QcwFYyfC1GVhG6WIKKxCJO8mXiuXEbaX2K5dACOdXsYVHLBVw2KyKgGMjbL5jUozxlWQ==</DP><DQ>ZVS1vlGfr5RcP4XnCelx7/KgYzWKL790IIC+v8KuhkxptEYSp8IF+yIa9DyoZpY3zePt8oF1J435rMI+M6mC3w==</DQ><InverseQ>OO5eD0E+/IXVao2VJ7pSvpV0ZMXTZLx4bO8SyDsAgNEBeYWySeUO48zqYGumcd0K7tnGOMqlIamoW+RthAa2QA==</InverseQ><D>ZuZ/aj7VCbGPRaY8b/g1qUnGWeShAToU8SHQGiy3A5N85F1qvYulFn6jJZ35AAqQCtyX4qucZn9GathTQcqF5Tb0HPam0oi/G4mVCJ2/mHCC057zrfsGNfS+CURm76mEWYu7M5WB+eN/xJqjzU53g0odiGqA7pp+VaOVSBQRaT0=</D></RSAKeyValue> |

**public\_key\_XML.txt**

|  |
| --- |
| <RSAKeyValue><Modulus>oHKkUhFzos2XwhpWPwTNqC4kKjDBekc9BLNX5TrmDJMIi17I5yVh+41VYB/0rvjhY/iifxxYxGNPHQbjpfceKOycE38JQ/uqwodcbR3ipurCD4XlVGU2sgrXM03VX4gqbuj7r1HsP3CdyotFFKVQocA2X3fHkSHO0ygFolhm/+k=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue> |

**Задача 2**

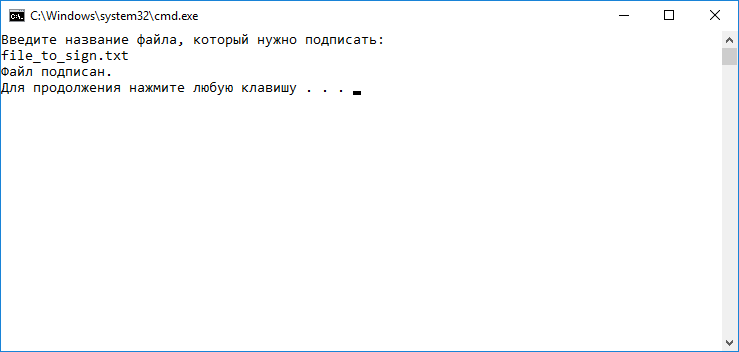
Взять текстовый файл и используя​ **RSA.SignData**​ сгенерировать для него цифровую подпись. Записать исходный файл + подпись в новый файл

**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();  }  } |



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.r‚‡-;‚± \_Ѓс\_Зf\_,\_€\_=бDЩ‚x0нРо{Њ-ЮtФиk\_{КЄEХµФUТс\_\_±Ај\_я\_\_K{~ЃЪ/%еk­‡\_|Ш)р д=~УlЎ\_єиІhTЇ.©bэ™о€\_Ё¦(Um¬©А\њ~•Л:a\_шџ3\_\_ЕQа\_\\_ТЇ\_\_ |

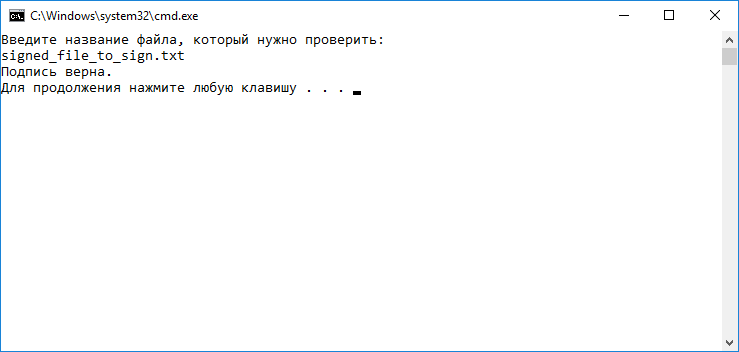
**Задача 3**

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:**



**Задача 4**

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

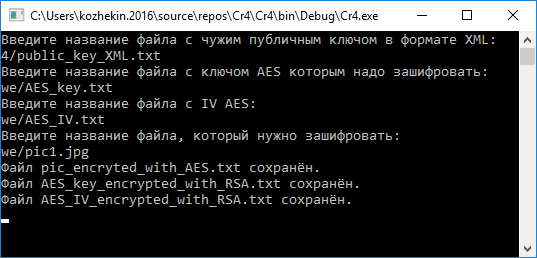
**Возьмём фото и зашифруем его:**



**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\_encryted\_with\_AES.txt", encryptedPicture);  Console.WriteLine("Файл pic\_encryted\_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:**



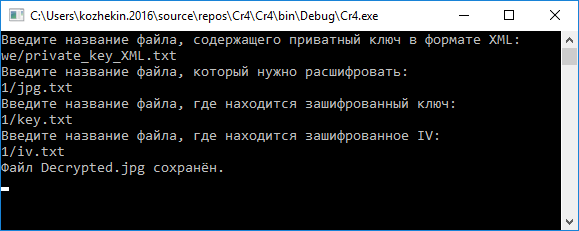
**Задача 5**

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:**





1. Выводы

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