МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Практическая работа №4**

по дисциплине «Основы информационной безопасности»

на тему: Криптографическая защита информации

Выполнил студент 2 курса 7 группы специальность ПОИБМС Володькин Никифор Дмитриевич

(Ф.И.О.)

Преподаватель Ржеутская Надежда Викентьевна

(Ф.И.О.)

**Цель: Овладение основными криптографическими алгоритмами асимметричного шифрования.**

**Выполнение работы**

# **Реализация элементов криптосистемы RSA**

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Весь алгоритм расписан в таблице:

|  |  |  |
| --- | --- | --- |
| Этап | Описание операции | Результат операции |
| Генерация ключей | Выбрать два простых различных числа | p=3557,  q=2579 |
| Вычислить модуль (произведение) | n = p \cdot q = 3557 \cdot 2579 = 9173503 |
| Вычислить функцию Эйлера | \varphi(n) = (p-1) (q-1) = 9167368 |
| Выбрать открытую экспоненту | e = 3 |
| Вычислить секретную экспоненту | d = e^{-1} \mod \varphi(n)  d = 6111579 |
| Опубликовать открытый ключ | \{e, n\} = \{3,9173503 \} |
| Сохранить закрытый ключ | \{d, n\} = \{6111579, 9173503 \} |
| Шифрование | Выбрать текст для зашифровки | m = 111111 |
| Вычислить шифротекст | \begin{align} c &= E(m) \\  &= m^e \mod n \\  &= 111111^3   \mod 9173503 \\  &= 4051753 \end{align} |
| Расшифрование | Вычислить исходное сообщение | \begin{align} m &= D(c) = \\   &= c^d \mod n \\   &= 4051753^{6111579} \mod 9173503 \\   &= 111111 \end{align} |

Было написано следующее приложение на языке C#, реализующее шифрование RSA с помощью класса **RSACryptoServiceProvider**:

using System;

using System.Security.Cryptography;

using System.Text;

namespace ConsoleApp1

{

class RSACSPSample

{

static void Main()

{

try

{

UnicodeEncoding ByteConverter = new UnicodeEncoding();

Console.Write("Введите сообщение: ");

byte[] dataToEncrypt = ByteConverter.GetBytes(Console.ReadLine());

byte[] encryptedData;

byte[] decryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

encryptedData = RSAEncrypt(dataToEncrypt, RSA.ExportParameters(false), false);

decryptedData = RSADecrypt(encryptedData, RSA.ExportParameters(true), false);

Console.WriteLine("Расшифрованное сообщение: {0}", ByteConverter.GetString(decryptedData));

}

}

catch (ArgumentNullException)

{

Console.WriteLine("Не удалось расшифровать сообщение.");

}

}

public static byte[] RSAEncrypt(byte[] DataToEncrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

{

try

{

byte[] encryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKeyInfo);

encryptedData = RSA.Encrypt(DataToEncrypt, DoOAEPPadding);

}

return encryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

return null;

}

}

public static byte[] RSADecrypt(byte[] DataToDecrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

{

try

{

byte[] decryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKeyInfo);

decryptedData = RSA.Decrypt(DataToDecrypt, DoOAEPPadding);

}

return decryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.ToString());

return null;

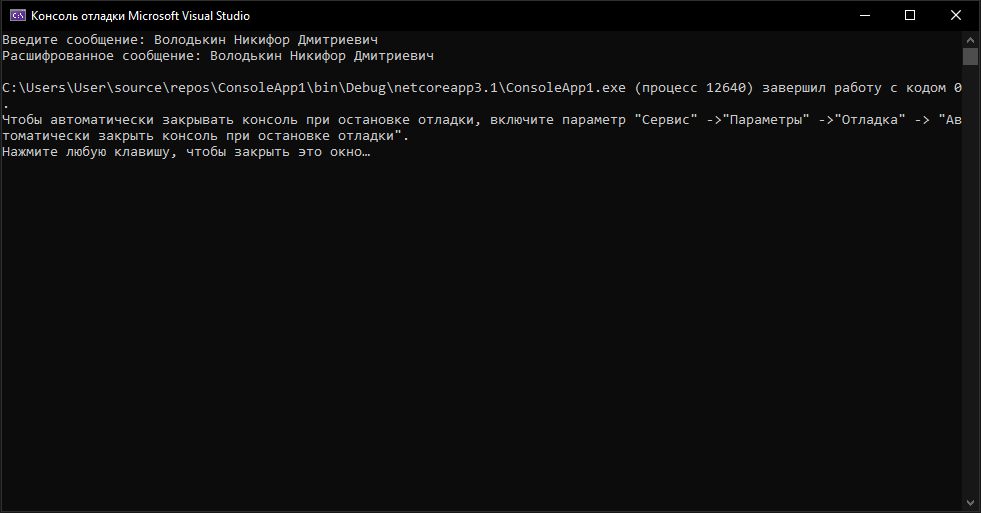
}

}

}

}

Результат выполнения программы:



# **Реализация элементов схемы шифрования Дифи-Хеллмана**

## **Генерация ключей**

В 1976 году после публичной критики алгоритма DES и указания на сложность обработки секретных ключей Уитфилд Диффи (Whitfield Diffie) и Мартин Хеллман (Martin Hellman) опубликовали свой алгоритм обмена ключами. Это была первая публикация на тему криптографии с открытым ключом и, возможно, самый большой шаг вперед в области криптографии, сделанный когда‑либо.

Из‑за невысокого быстродействия, свойственного асимметричным алгоритмам, алгоритм Диффи‑Хеллмана не предназначен для шифрования данных. Он был ориентирован на передачу секретных ключей DES, ARS или других подобных алгоритмов через небезопасную среду. В большинстве случаев алгоритм Диффи‑Хеллмана не используется для шифрования сообщений, потому что он, в зависимости от реализации, от 10 до 1000 раз медленнее алгоритма DES.

До алгоритма Диффи‑Хеллмана было сложно совместно использовать зашифрованные данные из‑за проблем хранения ключей и передачи информации. В большинстве случаев передача информации по каналам связи небезопасна, потому что сообщение может пройти десятки систем, прежде чем оно достигнет потенциального адресата, и нет никаких гарантий, что по пути никто не сможет взломать секретный ключ. Уитфилд Диффи и Мартин Хеллман предложили зашифровывать секретный ключ DES по алгоритму Диффи‑Хеллмана на передающей стороне и пересылать его вместе с сообщением, зашифрованным с использованием DES. Тогда на другом конце его сможет расшифровать только получатель сообщения.

На практике **обмен ключами** по алгоритму Диффи‑Хеллмана происходит по следующей схеме.

1. Два участника обмена договариваются о двух числах. Один выбирает большое простое число, а другой – целое число, меньшее числа первого участника. Переговоры они могут вести открыто, и это никак не отразится на безопасности.
2. Каждый из двух участников, независимо друг от друга, генерирует другое число, которое они будут хранить в тайне. Эти числа выполняют роль секретного ключа. Далее в вычислениях используются секретный ключ и два предыдущих целых числа. Результат вычислений посылается участнику обмена, и он играет роль открытого ключа.
3. Участники обмена обмениваются открытыми ключами. Далее они, используя собственный секретный ключ и открытый ключ партнера, конфиденциально вычисляют ключ сессии. Каждый партер вычисляет один и тот же ключ сессии.
4. Ключ сессии может использоваться как секретный ключ для другого алгоритма шифрования, например DES. Никакое третье лицо, контролирующее обмен, не сможет вычислить ключ сессии, не зная один из секретных ключей.

**Самое сложное в алгоритме** Диффи‑Хеллмана обмена ключами – это понять, что в нем фактически два различных независимых цикла шифрования. Алгоритм Диффи‑Хеллмана применяется для обработки небольших сообщений от отправителя получателю. Но в этом маленьком сообщении передается секретный ключ для расшифровки большого сообщения.

**Сильная сторона алгоритма** - никто не сможет скомпрометировать секретное сообщение, зная один или даже два открытых ключа получателя и отправителя. В качестве секретных и открытых ключей используются очень большие целые числа. Алгоритм Диффи‑Хеллмана основан на полезных для криптографии свойствах дискретных логарифмов.

## **Пример**

Ева — криптоаналитик. Она читает пересылку Боба и Алисы, но не изменяет содержимого их сообщений.

* s = секретный ключ. s = 2
* g = простое число меньшее p. g = 5
* p = открытое простое число. p = 23
* a = секретный ключ Алисы. a = 6
* A = открытый ключ Алисы. A = ga mod p = 8
* b = секретный ключ Боба. b = 15
* B = открытый ключ Боба. B = gb mod p = 19



Было написано следующее приложение на языке C#, реализующее шифрование RSA с помощью класса **ECDiffieHellmanCng** :

using System;

using System.Security.Cryptography;

using System.Text;

using System.IO;

namespace ConsoleApp1

{

class Sender

{

public static byte[] alicePublicKey;

static void Main()

{

using (ECDiffieHellmanCng sender = new ECDiffieHellmanCng())

{

sender.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

sender.HashAlgorithm = CngAlgorithm.Sha256;

alicePublicKey = sender.PublicKey.ToByteArray();

Receiver bob = new Receiver();

CngKey bobKey = CngKey.Import(bob.bobPublicKey, CngKeyBlobFormat.EccPublicBlob);

byte[] aliceKey = sender.DeriveKeyMaterial(bobKey);

byte[] encryptedMessage = null;

byte[] iv = null;

Console.Write("Введите сообщение: ");

Send(aliceKey, Console.ReadLine(), out encryptedMessage, out iv);

bob.Receive(encryptedMessage, iv);

}

}

private static void Send(byte[] key, string secretMessage, out byte[] encryptedMessage, out byte[] iv)

{

using (Aes aes = new AesCryptoServiceProvider())

{

aes.Key = key;

iv = aes.IV;

using (MemoryStream ciphertext = new MemoryStream())

using (CryptoStream cs = new CryptoStream(ciphertext, aes.CreateEncryptor(), CryptoStreamMode.Write))

{

byte[] plaintextMessage = Encoding.UTF8.GetBytes(secretMessage);

cs.Write(plaintextMessage, 0, plaintextMessage.Length);

cs.Close();

encryptedMessage = ciphertext.ToArray();

}

}

}

}

public class Receiver

{

public byte[] bobPublicKey;

private byte[] bobKey;

public Receiver()

{

using (ECDiffieHellmanCng receiver = new ECDiffieHellmanCng())

{

receiver.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

receiver.HashAlgorithm = CngAlgorithm.Sha256;

bobPublicKey = receiver.PublicKey.ToByteArray();

bobKey = receiver.DeriveKeyMaterial(CngKey.Import(Sender.alicePublicKey, CngKeyBlobFormat.EccPublicBlob));

}

}

public void Receive(byte[] encryptedMessage, byte[] iv)

{

using (Aes aes = new AesCryptoServiceProvider())

{

aes.Key = bobKey;

aes.IV = iv;

using (MemoryStream plaintext = new MemoryStream())

{

using (CryptoStream cs = new CryptoStream(plaintext, aes.CreateDecryptor(), CryptoStreamMode.Write))

{

cs.Write(encryptedMessage, 0, encryptedMessage.Length);

cs.Close();

string message = Encoding.UTF8.GetString(plaintext.ToArray());

Console.WriteLine($"Полученное сообщение: {message}");

}

}

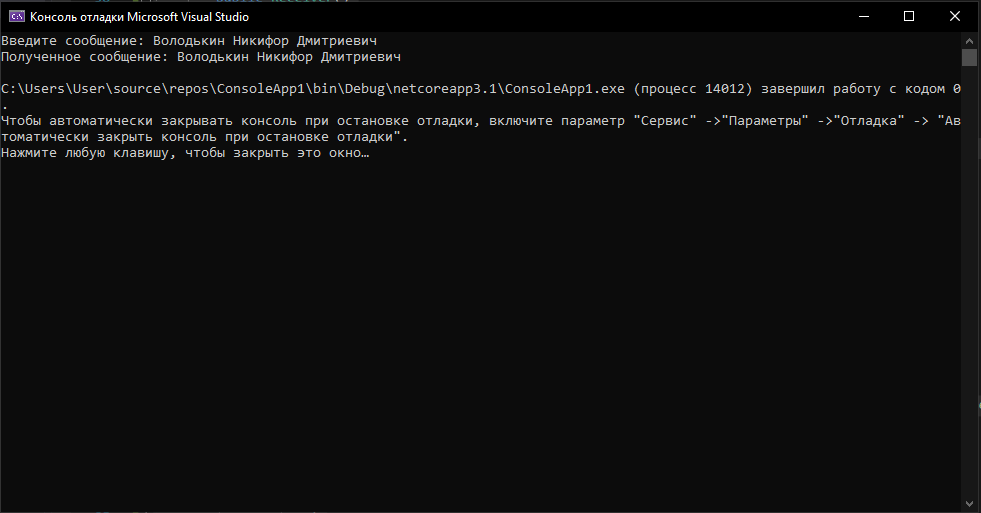
}

}

}

}

Результат выполнения программы:



**Реализация элементов схемы шифрования Эль-Гамаля**

## **Генерация ключей**

1. Генерируется случайное простое число ~p длины ~n [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82).
2. Выбирается случайный примитивный элемент ~g.
3. Выбирается случайное целое число ~x такое, что ~1 < x < p-1.
4. Вычисляется ~y = g^x\,\bmod\,p.
5. Открытым ключом является тройка \left( p,g,y \right), закрытым ключом — число ~x.

## **Шифрование**

Сообщение ~M шифруется следующим образом:

1. Выбирается сессионный [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) — случайное целое число ~k такое, что ~1 < k < p - 1
2. Вычисляются числа a = g^k\,\bmod\,p и b = y^k M\,\bmod\,p.
3. Пара чисел \left( a, b \right) является [шифротекстом](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст).

Нетрудно видеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

## **Расшифрование**

Зная закрытый ключ ~x, исходное сообщение можно вычислить из шифротекста \left( a, b \right) по формуле:

M = b(a^x)^{-1}\,\bmod\,p.

При этом нетрудно проверить, что

~(a^x)^{-1}\equiv g^{-kx}\pmod{p}

и поэтому

~b(a^x)^{-1}\equiv (y^kM)g^{-xk}\equiv (g^{xk}M) g^{-xk}\equiv M \pmod{p}.

Для практических вычислений больше подходит следующая формула:

M = b(a^x)^{-1}\,\bmod\,p = b \cdot a^{(p-1-x)}\,\bmod\,p 

## **Пример:**

**Шифрование**

Допустим, что нужно зашифровать сообщение ~M=5.

Произведем генерацию ключей :

пусть ~p=11, g=2. Выберем ~x=8 - случайное целое число ~x такое, что ~1 < x < p.

Вычислим ~y= g^x\bmod{p}=2^8\bmod{11}=3.

Итак, открытым является тройка ~(p,g,y)=(11,2,3), а закрытым ключом является число ~x=8.

Выбираем случайное целое число ~k такое, что 1 < k < (p − 1). Пусть ~k=9.

Вычисляем число ~a=g^k\bmod{p}=2^9 \bmod{11}=512 \bmod{11}=6.

Вычисляем число ~b=y^k M\bmod{p}=3^9 5 \bmod{11}=19683 \cdot 5 \bmod{11}=9.

Полученная пара ~(a,b)=(6,9) является шифротекстом.

**Расшифрование**

Необходимо получить сообщение ~M=5 по известному шифротексту ~(a,b)=(6,9) и закрытому ключу ~x=8.

Вычисляем M по формуле : ~M=b(a^x)^{-1}\bmod{p}=9(6^8)^{-1}\mod{11}=5

Получили исходное сообщение ~M=5.

Было написано следующее приложение на языке C#, реализующее шифрование Эль-Гамаля с помощью библиотеки **Safester**:

using Org.BouncyCastle.Bcpg.OpenPgp;

using Safester.CryptoLibrary.Api;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

public class ElGamalExample

{

public static string clearText = "Володькин Никифор Дмитриевич";

public const string identity = "myemail@domain.com";

public static string passphrase = "password";

public static PublicKeyAlgorithm pgpAsymAlgo = PublicKeyAlgorithm.DSA\_ELGAMAL;

public static PublicKeyLength pgpAsymKeyLength = PublicKeyLength.BITS\_2048;

static void Main()

{

Console.WriteLine(DateTime.Now + " Generate key pair...");

PgpKeyPairGeneratorSample pgpKeyPairGeneratorSample = new PgpKeyPairGeneratorSample();

pgpKeyPairGeneratorSample.Generate(identity, passphrase, pgpAsymAlgo, pgpAsymKeyLength);

PgpKeyPairHolder pgpPairKeyring = pgpKeyPairGeneratorSample.PgpKeyPairHolder;

TextEncryptionSample textEncryptionSample = new TextEncryptionSample(pgpPairKeyring.PublicKeyRing);

string encryptedText = textEncryptionSample.Encrypt(clearText);

Console.WriteLine("Зашифрованный текст:");

Console.WriteLine(encryptedText);

Decryptor decryptor = new Decryptor(pgpPairKeyring.PrivateKeyRing, passphrase.ToCharArray());

string clearText\_2 = decryptor.Decrypt(encryptedText);

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(clearText\_2);

}

public class TextEncryptionSample

{

private string publicKeyRing = null;

private const bool armor = true;

private bool withIntegrityCheck = true;

public TextEncryptionSample(string publicKeyRing)

{

this.publicKeyRing = publicKeyRing;

}

public string Encrypt(string inText)

{

//Console.WriteLine("Encrypting " + inText + " using string keyring.");

List<PgpPublicKey> encKeys = new List<PgpPublicKey>();

encKeys.Add(PgpPublicKeyGetter.ReadPublicKey(this.publicKeyRing));

Encryptor encryptor = new Encryptor(armor, withIntegrityCheck);

string outText = encryptor.Encrypt(encKeys, inText);

Console.WriteLine("Encryption done.");

return outText;

}

}

public class PgpKeyPairGeneratorSample

{

private PgpKeyPairHolder pgpKeyPairHolder = null;

public PgpKeyPairHolder PgpKeyPairHolder { get => pgpKeyPairHolder; }

public void Generate(string identity, string passphrase, PublicKeyAlgorithm publicKeyAlgorithm, PublicKeyLength publicKeyLength)

{

Console.WriteLine(DateTime.Now + " Starting generation...");

PgpKeyPairGenerator pgpKeyPairGenerator = new PgpKeyPairGenerator(identity, passphrase.ToCharArray(), publicKeyAlgorithm, publicKeyLength);

pgpKeyPairHolder = pgpKeyPairGenerator.Generate();

Console.WriteLine(DateTime.Now + " Done");

Console.WriteLine();

Console.WriteLine(PgpKeyPairHolder.PrivateKeyRing);

Console.WriteLine();

Console.WriteLine(PgpKeyPairHolder.PublicKeyRing);

Console.WriteLine();

Console.WriteLine("Press enter to close....");

Console.ReadLine();

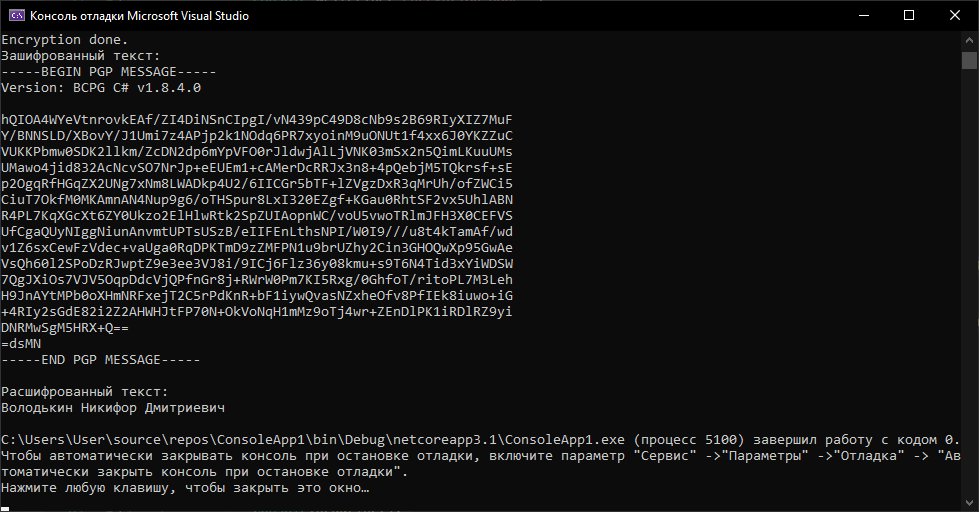
}

}

}

}

Результат выполнения программы:



**Вывод: Изучены основные криптографические алгоритмы асимметричного шифрования.**