МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1—98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**Отчет**

по дисциплине «Основы информационной безопасности»

**Исполнитель**

студент(ка) 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов М.О.

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ржеутская Н. В.

должность, ученая степень, ученое звание подпись, дата

**Практическое занятие №4.2**

**Тема « Криптографическая защита информации »**

Цель: Овладение основными криптографическими алгоритмами асимметричного шифрования.

**Теоретическое введение**

# 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} |

# 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).

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.

# 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



З**адание №1**

Рассказать процесс работы алгоритма RSA.

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

using System;

using System.Security.Cryptography;

class Program

{

static void Main()

{

try

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

RSAParameters publicKey = rsa.ExportParameters(false);

RSAParameters privateKey = rsa.ExportParameters(true);

Console.Write("Введите текст для шифрования: ");

string message = Console.ReadLine();

byte[] encryptedData = Encrypt(message, publicKey);

string decryptedMessage = Decrypt(encryptedData, privateKey);

Console.WriteLine("\nОткрытый ключ:");

Console.WriteLine("Modulus (n): " + BitConverter.ToString(publicKey.Modulus));

Console.WriteLine("Exponent (e): " + BitConverter.ToString(publicKey.Exponent));

Console.WriteLine("\nЗакрытый ключ:");

Console.WriteLine("Modulus (n): " + BitConverter.ToString(privateKey.Modulus));

Console.WriteLine("Exponent (d): " + BitConverter.ToString(privateKey.D));

Console.WriteLine("\nЗашифрованное сообщение: " + BitConverter.ToString(encryptedData));

Console.WriteLine("Расшифрованное сообщение: " + decryptedMessage);

}

}

catch (CryptographicException e)

{

Console.WriteLine("Произошла ошибка при работе с RSA: " + e.Message);

}

}

static byte[] Encrypt(string message, RSAParameters publicKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(publicKey);

byte[] data = System.Text.Encoding.UTF8.GetBytes(message);

return rsa.Encrypt(data, false);

}

}

static string Decrypt(byte[] data, RSAParameters privateKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(privateKey);

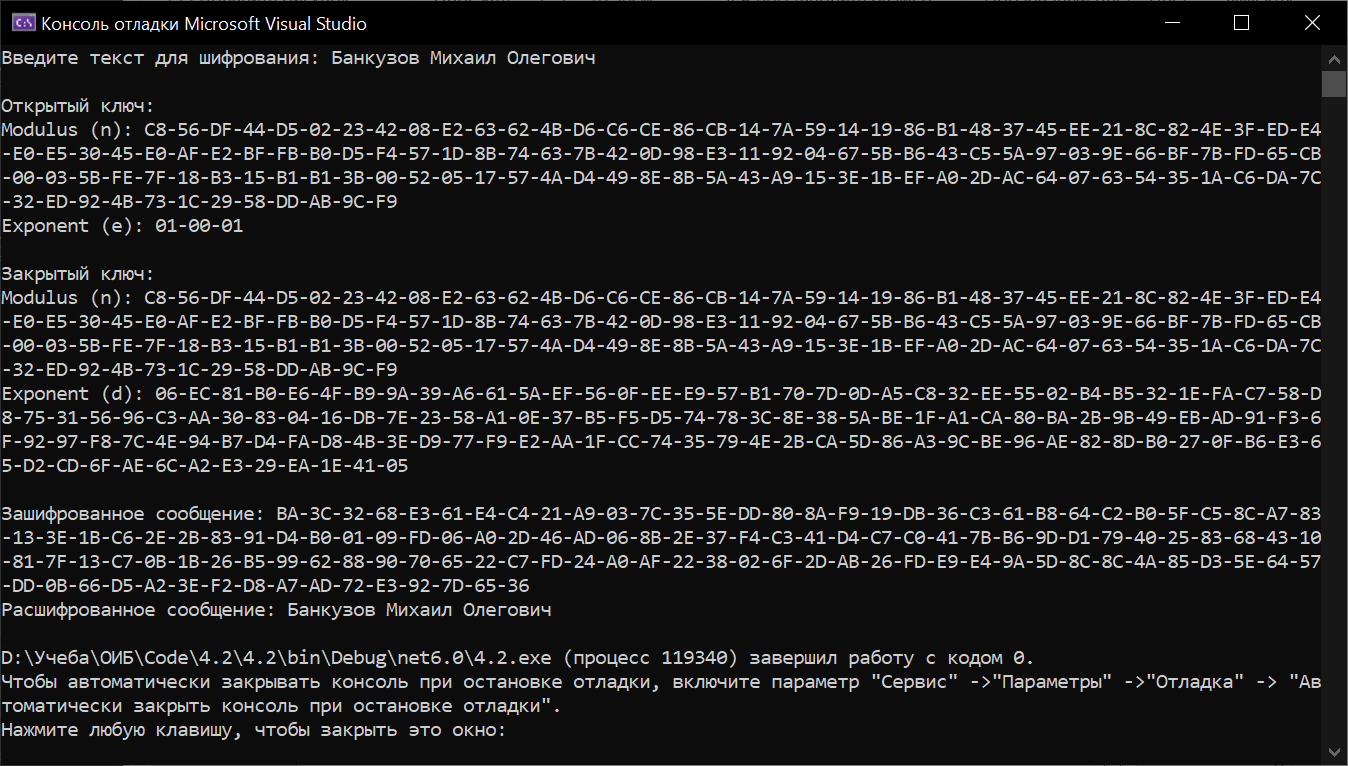
byte[] decryptedData = rsa.Decrypt(data, false);

return System.Text.Encoding.UTF8.GetString(decryptedData);

}

}

}

****

**Задание №2**

Рассказать процесс работы алгоритма Диффи-Хеллмана.

using System;

using System.Numerics;

public class DiffieHellman

{

private static Random random = new Random();

public static void RunDiffieHellman()

{

// Генерация простого числа p и первообразного корня g

int p = GeneratePrimeNumber();

int g = GeneratePrimitiveRoot(p);

// Алиса генерирует закрытый ключ a

int a = random.Next(2, p - 1);

// Боб генерирует закрытый ключ b

int b = random.Next(2, p - 1);

// Алиса и Боб вычисляют открытые ключи

BigInteger A = BigInteger.ModPow(g, a, p);

BigInteger B = BigInteger.ModPow(g, b, p);

// Алиса и Боб обмениваются открытыми ключами

BigInteger sharedKeyA = BigInteger.ModPow(B, a, p);

BigInteger sharedKeyB = BigInteger.ModPow(A, b, p);

// Общий ключ у Алисы и Боба теперь одинаков

Console.WriteLine($"p: {p}");

Console.WriteLine($"g: {g}");

Console.WriteLine($"Открытый ключ участника 1 (A): {A}");

Console.WriteLine($"Открытый ключ участника 2 (B): {B}");

Console.WriteLine($"Общий ключ у А: {sharedKeyA}");

Console.WriteLine($"Общий ключ у В: {sharedKeyB}");

}

// Генерация простого числа

private static int GeneratePrimeNumber()

{

int candidate = random.Next(100, 200); // Начнем с небольшого диапазона для упрощения примера

while (!IsPrime(candidate))

{

candidate++;

}

return candidate;

}

// Проверка на простоту

private static bool IsPrime(int number)

{

if (number < 2)

return false;

for (int i = 2; i <= Math.Sqrt(number); i++)

{

if (number % i == 0)

return false;

}

return true;

}

// Генерация первообразного корня

private static int GeneratePrimitiveRoot(int p)

{

for (int g = 2; g < p; g++)

{

bool isPrimitiveRoot = true;

for (int i = 1; i < p - 1; i++)

{

if (BigInteger.ModPow(g, i, p) == 1)

{

isPrimitiveRoot = false;

break;

}

}

if (isPrimitiveRoot)

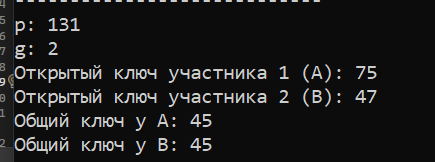
return g;

}

throw new Exception("Unable to find a primitive root.");

}

}



**Задание №3**

Рассказать процесс работы алгоритма Эль-Гамаля.

static int g\_main;

static BigInteger a;

public static bool Search\_g(int p, int g)

{

bool boolean = false;

List<BigInteger> array\_mod\_number = new List<BigInteger>();

BigInteger integer = ((BigInteger.Pow(g, 1)) % p);

array\_mod\_number.Add(integer);

for (int i = 2; i != p; i++)

{

integer = BigInteger.Pow(g, i) % p;

for (int j = 0; j != i - 1; j++)

{

if (array\_mod\_number[j] == integer)

{

g--;

array\_mod\_number.Clear();

i = 1;

integer = BigInteger.Pow(g, 1) % p;

array\_mod\_number.Add(integer);

break;

}

if ((j == i - 2) && (array\_mod\_number[j] != integer))

{

array\_mod\_number.Add(integer);

}

}

}

g\_main = g;

boolean = true;

return boolean;

}

public static int Search\_p()

{

Random random = new Random();

int p = 0;

Boolean boolean = false;

do

{

p = random.Next(2000, 2500);

for (int i = 2; i != p; i++)

{

if (i == p - 1)

{

boolean = Search\_g(p, p - 1);

break;

}

if (p % i == 0)

{

break;

}

}

}

while (boolean == false);

return p;

}

public static List<BigInteger> Cipher(string text, int p, BigInteger y)

{

List<BigInteger> array = new List<BigInteger>();

Random random = new Random();

int k = random.Next(1, p - 1);

for (int i = 0; i != text.Length; i++)

{

a = BigInteger.Pow(g\_main, k) % p;

array.Add((BigInteger.Pow(y, k) \* (int)text[i]) % p);

}

return array;

}

public static string Cipher\_RAZ(int length\_text, List<BigInteger> array\_number, int x, int p)

{

string save\_text = "";

BigInteger integer;

for (int i = 0; i != length\_text; i++)

{

integer = (array\_number[i] \* (BigInteger.Pow(a, p - 1 - x))) % p;

save\_text += (char)integer;

}

return save\_text;

}

int p = 0;

Random random = new Random();

p = Search\_p();

int x = random.Next(1, p - 1); //Генерирую закрытый ключ

BigInteger y = BigInteger.Pow(g\_main, x) % p; //Нахожу открытый ключ

Console.Write($"Числа [{p}, {g\_main}, {y}] ---> [p, g, y]\n");

Console.Write($"Открытый ключ - это <y>\n");

Console.Write($"Закрытый ключ: {x}\n");

Console.Write($"Введите текст: ");

string text = Console.ReadLine();

Console.WriteLine();

List<BigInteger> array\_cipher\_text = new List<BigInteger>();

array\_cipher\_text = Cipher(text, p, y);

Console.WriteLine("Зашифрованное сообщение: ");

for (int i = 0; i != text.Length; i++)

{

Console.WriteLine($"{i}:[{a}, {array\_cipher\_text[i]}] |--| ");

}

Console.WriteLine();

Console.Write($"\nРасшифрованное сообщение: {text = Cipher\_RAZ(text.Length, array\_cipher\_text, x, p)}");

Console.ReadKey();

