Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Основы информационной безопасности»

Отчёт по практическому занятию №4.2

**Криптографическая защита информации**

Студент: Жук С.С.

ФИТ 2 курс 2 группа

Преподаватель: Ржеутская Н.В.

**Практическое занятие №4.2**

**«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" \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.

# 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



**Задание для выполнения.**

**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.

Для начала нужно сгенерировать публичный и приватный ключ.

* Выбираю два простых числа. Пусть это будет p=3 и q=7.
* Вычисляю модуль — произведение p и q: n=p\*q=3\*7=21.
* Вычисляю функциюЭйлера: φ=(p-1)\*(q-1)=2\*6=12.
* Выбираю число e.

Теперь пара чисел {e, n} — это мой открытый ключ. Я отправляю его кому-то, чтобы этот кто-то зашифровал своё сообщение. Но мне еще нужно получить закрытый ключ.

Мне нужно вычислить число d, обратное е по модулю φ. То есть остаток от деления по модулю φ произведения d\*e должен быть равен 1. d может быть равен 17. Пара {d, n} — это секретный ключ, его я оставляю у себя. Его нельзя сообщать никому. Только обладатель секретного ключа может расшифровать то, что было зашифровано открытым ключом.

Теперь зашифрую какое-нибудь сообщение(число).

Пусть это будет число 19. Обозначу его P=19. Также имеется открытый ключ: {e, n} = {5, 21}. Шифрование выполняется по следующему алгоритму:

* Выбранное число возводится в степень e по модулю n. То есть, вычисляется 19 в степени 5 (2476099) и берётся остаток от деления на 21. Получается 10 — это закодированные данные.

Полученные данные E=10 отправляются назад отправителю.

Однако сообщение P=19 не должно быть больше n=21, иначе ничего не получится.

Теперь расшифрую то, что было зашифровано.

Я получил данные (E=10), и у меня имеется закрытый ключ  
{d, n} = {17, 21}.

Суть в том, что открытый ключ не может расшифровать сообщение, а закрытый ключ я никому не сообщал.

*Начинаю расшифровывать:*

Я делаю операцию, похожую на шифровку, но вместо e использую d. Возвожу E в степень d: получаю 10 в степени 17. Вычисляю остаток от деления на 21 и получаю 19 — исходное сообщение.

Никто, кроме обладателя закрытого ключа не сможет расшифровать сообщение (ну почти никто).

# **Преимущество** — при помощи открытого ключа алгоритма шифрования невозможно прочитать видоизмененное сообщение. Для этого требуется закрытый ключ, который есть только у адресата.

**Недостаток** — медленное шифрование из-за громоздкости вычислительных операций. Невозможность получить доступ ко всем сообщениям, которые шифруются одним ключом.

Шифр RSA:

using System;

using System.Numerics;

class RSA

{

static void Main()

{

// Генерация ключей

BigInteger p = 3557;

BigInteger q = 2579;

BigInteger n = p \* q;

BigInteger phi = (p - 1) \* (q - 1);

BigInteger e = GeneratePublicKey(phi);

BigInteger d = GeneratePrivateKey(e, phi);

Console.WriteLine("Публичный ключ (e, n): (" + e + ", " + n + ")");

Console.WriteLine("Приватный ключ (d, n): (" + d + ", " + n + ")");

//Шифрование и расшифрование сообщения

BigInteger m = 111111;

Console.WriteLine("Исходное сообщение: " + m);

BigInteger encryptedMessage = Encrypt(m, e, n);

Console.WriteLine("Зашифрованное сообщение: " + encryptedMessage);

BigInteger decryptedMessage = Decrypt(encryptedMessage, d, n);

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

}

static BigInteger GeneratePublicKey(BigInteger phi)

{

//Выбираем открытую экспоненту e

BigInteger e = 3;

return e;

}

static BigInteger GeneratePrivateKey(BigInteger e, BigInteger phi)

{

//Вычисляем закрытый ключ d с использованием расширенного алгоритма Евклида

BigInteger d = ModularInverse(e, phi);

return d;

}

static BigInteger ModularInverse(BigInteger a, BigInteger m)

{

BigInteger m0 = m;

BigInteger x0 = 0;

BigInteger x1 = 1;

while (a > 1)

{

BigInteger q = a / m;

BigInteger t = m;

m = a % m;

a = t;

t = x0;

x0 = x1 - q \* x0;

x1 = t;

}

if (x1 < 0)

{

x1 += m0;

}

return x1;

}

static BigInteger Encrypt(BigInteger m, BigInteger e, BigInteger n)

{

return BigInteger.ModPow(m, e, n);

}

static BigInteger Decrypt(BigInteger encryptedMessage, BigInteger d, BigInteger n)

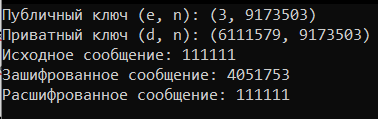
{

return BigInteger.ModPow(encryptedMessage, d, n);

}

}

Вывод программы:



**Эль-Гамаль** разработал один из вариантов алгоритма Диффи-Хеллмана. Он усовершенствовал систему Диффи-Хеллмана и получил два алгоритма, которые использовались для шифрования и для обеспечения аутентификации. В отличие от RSA алгоритм Эль-Гамаля не был запатентован и, поэтому, стал более дешевой альтернативой, так как не требовалась оплата взносов за лицензию. Считается, что алгоритм попадает под действие патента Диффи-Хеллмана.

Пусть имеются абоненты А, В, С, ..., которые хотят передавать друг другу зашифрованные сообщения, не имея никаких защищенных каналов связи. Фактически здесь используется схема Диффи-Хеллмана, чтобы сформировать общий секретный ключ для двух абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ. Для каждого следующего сообщения секретный ключ вычисляется заново.

*Шифрование*

Допустим, что нужно зашифровать сообщение М = 5.  
1. Произведем генерацию ключей: пусть p = 11, g = 2. Выберем x = 8 - случайное целое число x такое, что 1 < х < (р - 1).

Вычислим y = g^x mod p = 2^8 mod 11 = 3.

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

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

3.Вычисляем число A = g^k mod p = 2^9 mod 11 = 512 mod 11 = 6.

4.Вычисляем число B = y^k\*M mod p = 3^9\*5 mod 11 = 19683\*5 mod 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 mod p = 9\*(6^8)^-1 mod 11 = 5.

Получили исходное сообщение M = 5.

**Преимущество** — вероятностный характер шифрования — большая стойкость.

**Недостаток** — удвоение длины зашифрованного текста по сравнению с начальным текстом. Само сообщение и ключ не определяют шифртекст однозначно.

Шифр Эль- Гамаля:

using System;

using System.Collections.Generic;

namespace Lab\_4

{

public class ElGamal

{

public int p;

public int x;

public int g;

public int y;

public ElGamal()

{

GenerateKeys();

Console.WriteLine($"Открытые ключи: p = {p}, g = {g}, y = {y}");

Console.WriteLine($"Закрытый ключ: x = {x}");

}

//Генерация ключей

public void GenerateKeys()

{

p = 11;

g = 2;

x = 8;

y = Power(g, x, p);

}

//Возведение числа a в степень b по модулю n

public int Power(int a, int b, int n)

{

int result = 1;

for (int i = 0; i < b; i++)

{

result = (result \* a) % n;

}

return result;

}

//Умножение чисел a и b по модулю n

public int Mul(int a, int b, int n)

{

return (a \* b % n);

}

//Шифрование текста

public string Encode(int number)

{

int k = 9;

int a = Power(g, k, p);

int b = Mul(Power(y, k, p), number, p);

return $"({a},{b})";

}

//Расшифрование текста

public int Decode(string input)

{

string[] nums = input.Trim('(', ')').Split(',');

int a = int.Parse(nums[0]);

int b = int.Parse(nums[1]);

int dM = Mul(b, Power(a, p - 1 - x, p), p);

return dM;

}

static void Main(string[] args)

{

ElGamal elGamal = new ElGamal();

int numberToEncrypt = 5; //Здесь можно указать число, которое нужно зашифровать

string encrypted = elGamal.Encode(numberToEncrypt);

Console.WriteLine($"Зашифрованное число ({numberToEncrypt}): {encrypted}");

int decryptedNumber = elGamal.Decode(encrypted);

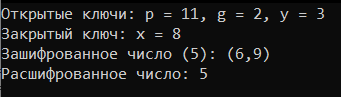
Console.WriteLine($"Расшифрованное число: {decryptedNumber}");

}

}

}

Вывод программы:



**Диффи-Хеллман**. Пусть есть 2 числа – g и p. Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют большие случайные числа: первый абонент — число a, второй абонент — число b. Затем первый абонент вычисляет значение A = g^a mod p и пересылает его второму, а второй вычисляет B = g^b mod p и передаёт первому. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (т.е. у него нет возможности вмешаться в процесс передачи). На втором этапе первый абонент на основе имеющегося у него a и полученного по сети B вычисляет значение, B^a mod p = g^(a\*b) mod p, а второй абонент на основе имеющегося у него b и полученного по сети A вычисляет значение A^b mod p = g^(a\*b) mod p. Как нетрудно видеть, у обоих абонентов получилось одно и то же число: K = g^(a\*b) mod p. Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления g^(a\*b) mod p по перехваченным g^a mod p и g^b mod p, если числа p, a, b выбраны достаточно большими.

**Преимущество** — никто не сможет скомпрометировать секретное сообщение. Полученный ключ может быть использован для шифрования по любым доступным сторонам алгоритма.

**Недостаток** — два различных независимых цикла шифрования. Отсутствие взаимной аутентификации сторон.

Шифр Диффи-Хеллмана:

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

namespace Lab\_4

{

public static class Power

{

//Метод для вычисления a^b mod n

public static uint power(uint a, uint b, uint n)

{

uint result = 1;

for (int i = 0; i < b; i++)

{

result = (result \* a) % n;

}

return result;

}

}

class Alice

{

public static uint primeNumber;

public static uint aliceNumber;

private static uint secretKey;

public static uint openKey;

private static uint generalSecretKey;

public static void GenerateNumber()

{

//Генерация случайного числа меньшего primeNumber и передача его Бобу (g = 5)

aliceNumber = 5;

Bob.aliceNumber = aliceNumber;

}

public static void GenerateSecretKey()

{

// Генерация секретного ключа для Алисы (а=6)

secretKey = 6;

}

public static void GenerateOpenKey()

{

// Вычисление открытого ключа Алисы (g^b mod p = 5 ^ 6 mod 23 = 8)

openKey = Power.power(aliceNumber, secretKey, primeNumber);

}

public static void GenerateGeneralSecretKey()

{

//Вычисление общего секретного ключа, полученного Алисой (s= 19^6 mod 23 = 2)

generalSecretKey = Power.power(Bob.openKey, secretKey, primeNumber);

}

public static void Output()

{

//Вывод информации о числах и ключах Алисы

Console.WriteLine($"Случайное число меньшее {primeNumber}: {aliceNumber}");

Console.WriteLine($"Секретный ключ Алисы {secretKey}");

Console.WriteLine($"Открытый ключ Алисы {openKey}");

Console.WriteLine($"Общий секретный ключ, который получила Алиса {generalSecretKey}");

}

}

public static class Bob

{

public static uint primeNumber;

public static uint aliceNumber;

private static uint secretKey;

public static uint openKey;

private static uint generalSecretKey;

public static void GeneratePrimeNumber()

{

//Генерация простого числа Бобом и передача его Алисе (p = 23)

primeNumber = 23;

Alice.primeNumber = primeNumber;

}

public static void GenerateSecretKey()

{

//Генерация секретного ключа для Боба (b=15)

secretKey = 15;

}

public static void GenerateOpenKey()

{

//Вычисление открытого ключа Боба (5^15 mod 23 = 19)

openKey = Power.power(aliceNumber, secretKey, primeNumber);

}

public static void GenerateGeneralSecretKey()

{

//Вычисление общего секретного ключа, полученного Бобом (8^15 mod 23 = 2)

generalSecretKey = Power.power(Alice.openKey, secretKey, primeNumber);

}

public static void Output()

{

//Вывод информации о числах и ключах Боба

Console.WriteLine($"Простое число: {primeNumber}");

Console.WriteLine($"Секретный ключ Боба: {secretKey}");

Console.WriteLine($"Открытый ключ Боба: {openKey}");

Console.WriteLine($"Общий секретный ключ, который получил Боб: {generalSecretKey}");

}

}

public class DiffieHellman

{

public DiffieHellman()

{

Bob.GeneratePrimeNumber();

Alice.GenerateNumber();

Bob.GenerateSecretKey();

Alice.GenerateSecretKey();

Bob.GenerateOpenKey();

Alice.GenerateOpenKey();

Bob.GenerateGeneralSecretKey();

Alice.GenerateGeneralSecretKey();

Bob.Output();

Alice.Output();

}

}

class Program

{

static void Main(string[] args)

{

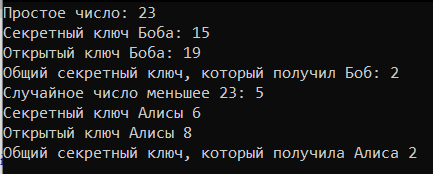
DiffieHellman diffieHellman = new DiffieHellman();

}

}

}

Вывод программы:



**Вывод**:

Овладение основными криптографическими алгоритмами асимметричного шифрования (RSA, Диффи-Хеллмана, Эль-Гамаля). Они являются более громоздкими и затратными по ресурсам.