Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра информационных](https://www.belstu.by/fakultety/fit/vm) систем и технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №10**

по дисциплине Информационная безопасность

Тема: Исследование ассиметричных шифров RSA и Эль-Гамаля

        Исполнитель:

Студент 3 курса группы 4

Станчик Максим Андреевич

Руководитель:

Ассистент Савельева М.Г.

Минск, 2025

**Лабораторная работа №10**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости ассиметричных шифров RSA и Эль-Гамаля.

2. Разработать приложение для реализации асимметричного зашифрования/ расшифрования на основе алгоритмов RSA и Эль-Гамаля.

3. Выполнить анализ криптостойкости ассиметричных шифров RSA и Эль-Гамаля.

4. Оценить скорость зашифрования/расшифрования реализованных шифров.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента

# Теоретические сведения

Асимметричная криптография основана на сложности решения некоторых математических задач. По существу, таких задач две:

• разложение больших чисел на простые сомножители (задача факторизации);

• вычисление дискретного логарифма в конечном поле, а также вычислительные операции над точками эллиптической кривой.

Задача дискретного логарифмирования формулируется так: для данных целых чисел *а* и *b*, 1 < *а*, *b* < *n*, найти логарифм – такое целое число *х*, что

*ax* ≡ *b* (mod *n*),

если такое число существует.

Китайская теорема об остатках. В общем случае если разложение числа *N* на простые множители представляет собой p1p2…pt (некоторые простые числа могут встречаться несколько раз), то система уравнений

(*x* mod *pi*) ≡ *ai*,

где *i* = 1, 2, …, *t*, имеет единственное решение: *x*, меньшее *N*.

Алгоритм RSA. Для генерации двух ключей: тайного и открытого (а по сути –двух взаимосвязанных частей одного ключа, т. е. ключа, принадлежащего одному физическому лицу (или группе лиц), либо одному юридическому лицу), используются два больших случайных простых числа *p* и *q*. Для максимальной большей криптостойкости нужно выбирать p и q равной длины. Рассчитывается произведение: n = pq. Это есть один из трех компонент ключа, состоящего из чисел *n*, *e*, *d*.

Затем случайным образом выбирается второй компонент ключа: открытый ключ или ключ зашифрования, e, такой что *e* и (*p* – 1)(*q* – 1) являются взаимно простыми числами; (*p* – 1)(*q* – 1) = *φ(n)* – функция Эйлера.

Наконец, расширенный алгоритм Евклида используется для вычисления третьего компонента ключа: ключа расшифрования d такого, что выполняется условие:

*ed* ≡ 1 (mod *φ(n)*).

Зашифрование. Если шифруется сообщение М, состоящее из r блоков: *m1*, *m2*, …, *mi*, …, *mr*, то шифртекст *С* будет состоять из такого же числа (*r*) блоков, представляемых числами:

*ci* ≡ (*mi*)*e* mod *n*.

Расшифрование. Для расшифрования каждого зашифрованного блока производится вычисление вида:

*mi* ≡ (*ci*)*d*mod *n*.

Алгоритм Эль-Гамаля. Безопасность алгоритма Эль-Гамаля, как и безопасность алгоритма Диффи – Хеллмана, основана на трудности вычисления дискретных логарифмов. Алгоритм Эль-Гамаля фактически использует схему Диффи – Хеллмана, чтобы сформировать общий секретный ключ для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ.

# Ход работы

Первое консольное приложение составляет табличную или графическую форму зависимости времени вычисления параметра *у*, функционально заданного выражением вида:

*у* ≡ *ax* mod *n*.

Массив xValues содержит заранее определенные значения (простые числа), которые используются в криптографических вычислениях, таких как шифрование и расчет параметров в алгоритмах RSA и Эль-Гамаля. При вычислении *y*, вместо *x* подставляется значение какого-либо объекта из этого массива. Массив инициализируется с использованием конструктора, который принимает строковое представление чисел. Код массива xValues представлен в листинге 2.1.

BigInteger[] xValues = {

new BigInteger("1009"),

new BigInteger("1013"),

new BigInteger("1019"),

new BigInteger("1021"),

new BigInteger("1031")

};

Листинг 2.1 – Код массива xValues

Массив aValues содержит набор целых чисел, которые используются в криптографических вычислениях, таких как определение параметра *а* a в выражении *у* ≡ *ax* mod *n*. Код массива aValues представлен в листинге 2.2.

int[] aValues = {5, 10, 15, 20, 25, 30, 35};

Листинг 2.2 – Код массива aValues

Массив nValues содержит два значения, представляющих большие целые числа в двоичном виде, состоящие из 1024 и 2048 битов, которые подставляются вместо *n* в выражении *у* ≡ *ax* mod *n*. Код массива nValues представлен в листинге 2.3.

int[] aValues = {5, 10, 15, 20, 25, 30, 35};

Листинг 2.3 – Код массива nValues

Для подстановки из массивов xValues, aValues и nValues различных комбинаций параметров 𝑎,𝑥 и 𝑛 в выражение *у* ≡ *ax* mod *n* был разработан цикл. Цикл также измеряет время выполнения этих операций. Код цикла представлен в листинге 2.4.

for (int a : aValues) {

for (BigInteger x : xValues) {

for (BigInteger n : nValues) {

long startTime = System.nanoTime();

BigInteger y = BigInteger.valueOf(a).multiply(x).mod(n);

long endTime = System.nanoTime();

long duration = endTime - startTime;

String nBinary = n.toString(2);

if (n == nValues[0]) {

nBinary = "1024 bits";

}

else if (n == nValues[1]) {

nBinary = "2048 bits";

}

System.out.printf("%-10d %-20s %-20s %-20s %-15d\n", a, x, nBinary, y, duration);

}

}

}

Листинг 2.4 – Код цикла

Результат работы первого консольного приложения представлен на рисунке 2.1.

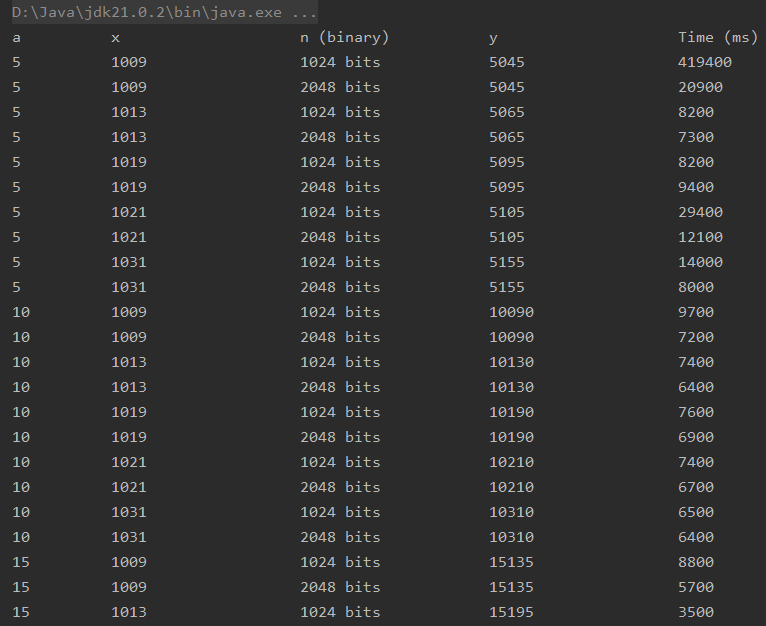


Рисунок 2.1 – Результат работы приложения

Далее было разработано второе консольное приложение, выполняющее шифрование и расшифрование с помощью алгоритмов RSA и Эль-Гамаля.

Для расшифрования каждого зашифрованного блока производится вычисление вида:

*mi* = (*ci*)*d*mod*n*

Конструктор CryptoApp выполняет основную функцию инициализации, создавая необходимые криптографические ключи для работы с данными. Это позволяет приложению использовать как RSA, так и Эль-Гамаль для шифрования и расшифрования текстовой информации. Код конструктора и представлен в листинге 2.5.

private KeyPair rsaKeyPair;

public CryptoApp() throws NoSuchAlgorithmException {

KeyPairGenerator rsaKeyGen = KeyPairGenerator.getInstance("RSA");

rsaKeyGen.initialize(2048);

rsaKeyPair = rsaKeyGen.generateKeyPair();

generateElGamalKeys(2048);

}

Листинг 2.5 – Код конструктора

Переменная rsaKeyPair используется для хранения пары ключей, используемых для алгоритма RSA. Публичный ключ используется для шифрования данных. Приватный ключ используется для расшифрования данных. Код переменной представлен в листинге 2.6.

private KeyPair rsaKeyPair;

Листинг 2.6 – Код пары ключей

Метод rsaEncrypt предназначен для шифрования текстовой строки с использованием алгоритма RSA. В качестве параметров принимает исходный текст, который необходимо зашифровать, возвращает зашифрованный текст в формате Base64. Для шифрования включает режим шифрования и использует публичный ключ. Код метода представлен в листинге 2.7.

public String rsaEncrypt(String plaintext) throws Exception {

Cipher cipher = Cipher.getInstance("RSA");

cipher.init(Cipher.ENCRYPT\_MODE, rsaKeyPair.getPublic());

byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes());

return Base64.getEncoder().encodeToString(encryptedBytes);

}

Листинг 2.7 – Код метода для шифрования текста с помощью RSA

Метод rsaDecrypt предназначен для расшифрования зашифрованной строки, используя алгоритм RSA. Принимает в качестве параметров зашифрованный текст в формате Base64, который необходимо расшифровать с использованием приватного ключа. Метод возвращает расшифрованный текст. Код метода представлен в листинге 2.8.

public String rsaDecrypt(String ciphertext) throws Exception {

Cipher cipher = Cipher.getInstance("RSA");

cipher.init(Cipher.DECRYPT\_MODE, rsaKeyPair.getPrivate());

byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

return new String(decryptedBytes);

}

Листинг 2.8 – Код метода для расшифрования текста с помощью RSA

Результат шифрования и расшифрования текста «Maxim Stanchik Andreevich» с помощью RSA представлен на рисунке 2.2.

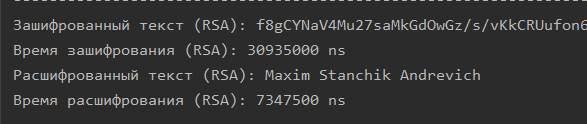


Рисунок 2.2 – Результат шифрования и расшифрования с помощью RSA

В алгоритме Эль-Гамаля выбирается простое число *р*. Выбирается число (*g*, *g* < *p*), являющееся первообразным корнем числа *р* – очень важный элемент с точки зрения безопасности алгоритма. Далее выбирается число *х* (*х* < *p*) и вычисляется последний компонент ключевой информации:

*mi* ≡ (*ci*)*d*mod *n*.

private BigInteger p хранит большое простое число, используемое в алгоритме Эль-Гамаля. Это число служит модулем для криптографических операций. private BigInteger g хранит генератор (основание) для группы, использующейся в алгоритме Эль-Гамаля. Обычно это малое простое число. private BigInteger хранит приватный ключ для алгоритма Эль-Гамаля. Это случайное число, выбранное из диапазона от 1 до *p*−1. private BigInteger *y* хранит публичный ключ для алгоритма Эль-Гамаля, вычисляемый как *y* ≡ *gx* mod*p*. Код переменных представлен в листинге 2.9.

private BigInteger p;

private BigInteger g;

private BigInteger x;

private BigInteger y;

Листинг 2.9 – Код метода для расшифрования текста с помощью RSA

Метод generateElGamalKeys создает ключи для алгоритма Эль-Гамаля, включая публичный и приватный ключи. В качестве параметров принимает длину ключа в битах, определяющая уровень безопасности. Код метода представлен в листинге 2.10.

private void generateElGamalKeys(int bitLength) {

SecureRandom random = new SecureRandom();

p = BigInteger.probablePrime(bitLength, random);

g = BigInteger.valueOf(2);

x = new BigInteger(bitLength, random).mod(p.subtract(BigInteger.ONE));

y = g.modPow(x, p);

};

Листинг 2.10 – Код метода для создания ключей для алгоритма Эль-Гамаля

Метод elGamalEncrypt предназначен для шифрования текстовой строки с использованием алгоритма Эль-Гамаля. Принимает в качестве параметров исходный текст, который необходимо зашифровать. Метод возвращает массив, состоящий из первого компонента зашифрованного сообщения, второго компонента зашифрованного сообщения. Код метода представлен в листинге 2.11.

public BigInteger[] elGamalEncrypt(String plaintext) {

BigInteger m = new BigInteger(1, plaintext.getBytes());

SecureRandom random = new SecureRandom();

BigInteger k = new BigInteger(2048, random).mod(p.subtract(BigInteger.ONE));

BigInteger c1 = g.modPow(k, p);

BigInteger c2 = m.multiply(y.modPow(k, p)).mod(p);

return new BigInteger[]{c1, c2};

};

Листинг 2.11 – Код метода для шифрования текстовой строки с использованием алгоритма Эль-Гамаля

Метод elGamalDecrypt предназначен для расшифрования сообщения, зашифрованного с использованием алгоритма Эль-Гамаля. В качестве параметров принимает первый компонент зашифрованного сообщения, второй компонент зашифрованного сообщения. Метод возвращает расшифрованный текст. Код метода представлен в листинге 2.12.

public String elGamalDecrypt(BigInteger c1, BigInteger c2) {

BigInteger s = c1.modPow(x, p);

BigInteger m = c2.multiply(s.modInverse(p)).mod(p);

return new String(m.toByteArray());

};

Листинг 2.12 – Код метода для расшифрования текстовой строки с использованием алгоритма Эль-Гамаля

Алгоритм Эль-Гамаля использует несколько параметров, которые являются ключевыми. Для начало было выбрано простое число *p*. Далее было выбрано число *g* = 56, меньшее числа *p* и являющееся первообразным корнем числа *p*. Затем было выбрано число *x*. Число *x* должно быть меньше, чем число *p*. Компонент *y* вычисляется внутри метода шифрования. Результат работы приложения с исходным текстом «Maxim Stanchik Andreevich» представлен на рисунке 2.8.

Результат работы зашифрования и расшифрования показан на рисунке 2.3.

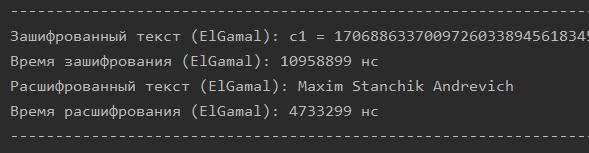


Рисунок 2.3 – Результат шифрования и расшифрования с помощью алгоритма Эль-Гамаля

# Оценка времени выполнения

Необходимо вычислить скорость вычисления параметра *y* в данном соотношении:

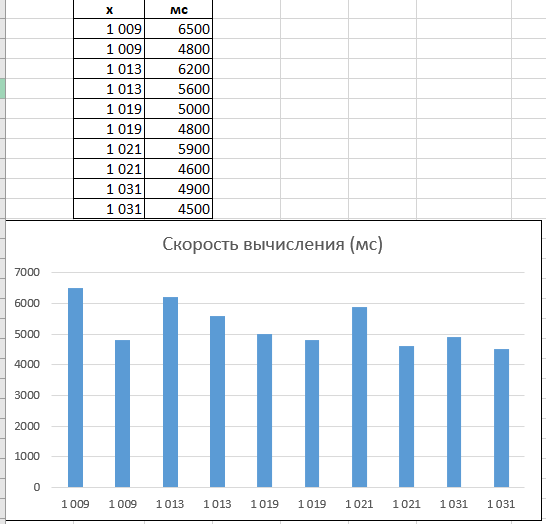


Рисунок 3.1 – График скорости вычисления *y*

На рисунке 3.2. представлен график сравнения скорости зашифрования и расшифрования в RSA и алгоритме Эль-Гамаля при примерно одинаковых ключах и при длине сообщения 24 символов. Время измеряется в наносекундах.

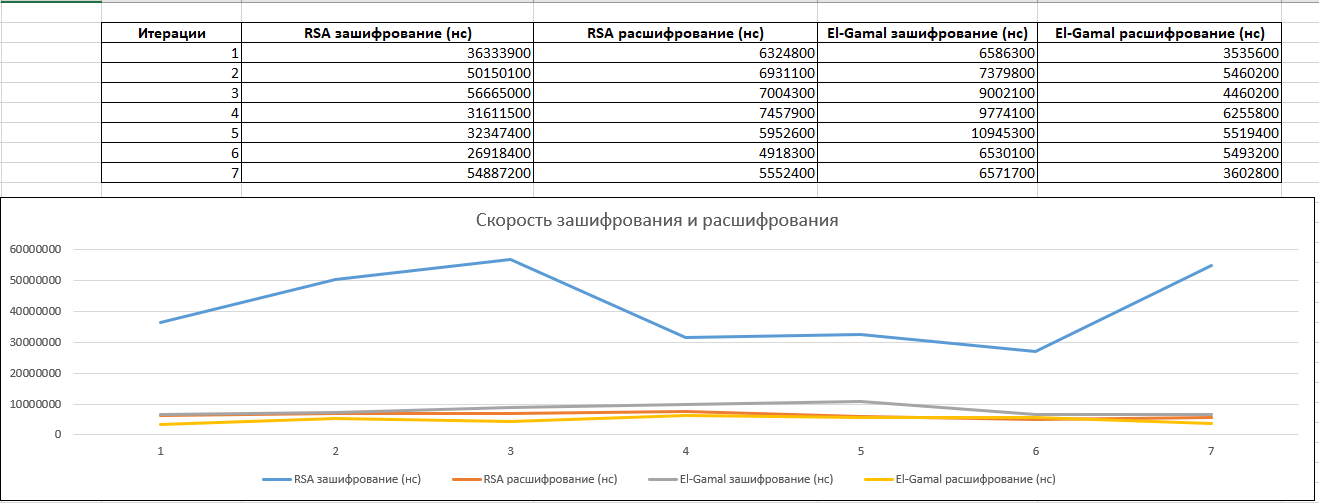


Рисунок 3.2 – График сравнения скорости алгоритмов

Исходя из информации на графике, можно сделать вывод, что RSA — более затратен по времени при шифровании, но может быть эффективнее при расшифровании, особенно если оптимизирован (например, с помощью CRT). El-Gamal — быстрее и сбалансированнее в обеих операциях, что делает его потенциально предпочтительным для систем, где важно равномерное распределение вычислительной нагрузки между шифрованием и расшифрованием.

График сравнения объемов шифротекстов RSA и Эль-Гамаля представлен на рисунке 3.3.

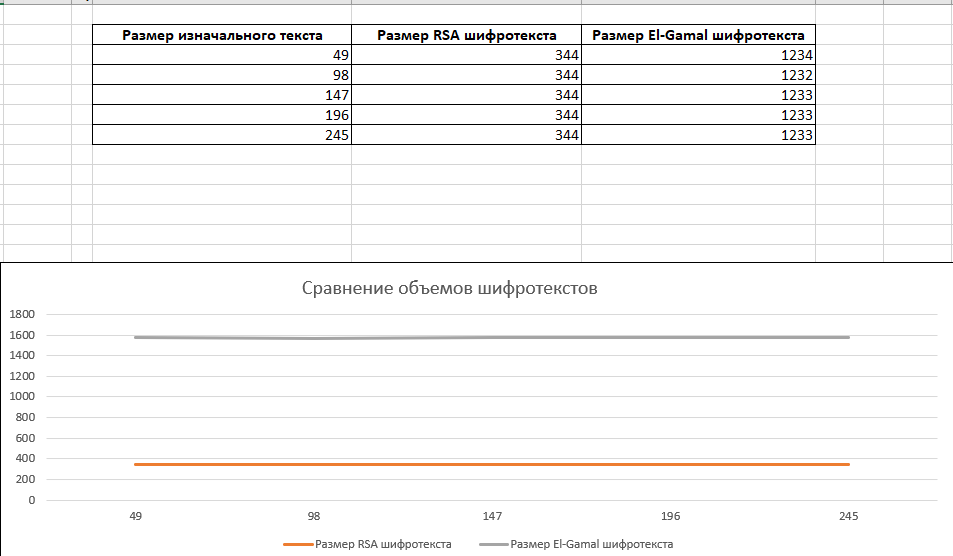


Рисунок 3.3 – Сравнение объемов шифротекстов

Исходя из данных на графике, можно сделать вывод, что RSA шифрование показывает более компактные размеры шифротекстов по сравнению с Эль-Гамалем. Для изначальных текстов небольших размеров объем шифротекста не увеличивается по мере увеличения объема изначального текста. При выборе алгоритма шифрования важно учитывать не только безопасность, но и объем шифротекстов, особенно для систем с ограниченными ресурсами. Размер шифротекста у Эль-Гамаля в несколько раз больше, чем у RSA при одинаковой длине ключа *p* ≈ *n*, потому что Эль-Гамаль, особенно в реализации для длинных сообщений, шифрует каждый блок или байт отдельно, и для каждого создаёт новую пару чисел (*c₁*, *c₂*), что приводит к огромному увеличению общего объёма.

**Вывод:** в ходе выполнения лабораторной работы был изучен принцип реализации асимметричных шифров RSA и Эль-Гамаля. Также было разработано два консольных приложения, выполняющих шифрование и расшифрование с помощью шифра, основанного на алгоритме об укладке ранца. Была оценена скорость шифрования и дешифрования сообщений. Алгоритмы RSA и Эль-Гамаля предоставляют надежную защиту данных при асимметричном шифровании, но несмотря на это, для максимальной безопасности необходимо правильно выбирать параметры шифра и следить за их обновлением в соответствии с рекомендациями безопасности. Также стоит обратить внимание на скорости работы обоих алгоритмов.