Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Информационная безопасность**

Студент: Козека Е. М.

ФИТ 3 курс 4 группа

Преподаватель: Нистюк О. А.

Минск 2025

**Лабораторная работа №10. Исследование ассиметричных шифров RSA и Эль-Гамаля**

**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.

Эти задачи объединяет то, что они используют операцию получения остатка от целочисленного деления. В силу этого практически все системы асимметричного зашифрования и расшифрования основаны либо на проблеме факторизации (среди них – RSA), либо на проблеме дискретного логарифмирования (среди них – Эль-Гамаля).

*Основная теорема арифметики*: всякое натуральное число *N*, кроме 1, можно представить как произведение простых множителей:

*N = p*1*p*2*p*3*...pz*, *z* > 1.

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 *p*1*p*2…*p*t (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 тайного и открытого ключа (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 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: *m*1, *m*2, …, *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***может быть использован для решения трех основных криптографических задач: для зашифрования/расшифрования данных, для формирования цифровой подписи и для согласования общего ключа. Кроме того, возможны модификации алгоритма для схем проверки пароля, доказательства идентичности сообщения и другие варианты*.*

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.

*Генерация ключевой информации*. 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:

*y* ≡ *gх* mod *р*.

Владельцу сформированной ключевой информации, состоящей из 4 чисел, может посылаться некоторый шифртекст, созданный с использованием открытого ключа получателя: *p*, *g*, *y*. Расшифрование шифртекста получатель производит своим тайным ключом: *p*, *g*, *х*.

**Ход работы**

**Задание 1.** С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра *у*, функционально заданного выражением вида:

*у* ≡ *ax* mod *n*,

от параметров: *а* (десятичные числа от 5 до 35; можно взять 1 или 2 числа), *х* (числа, желательно простые, из диапазона от 103 до 10100; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне), *n* (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

Код приложения представлен в листинге 10.1.

const bigInt = require('big-integer');

const xValues = [

    bigInt(1000),

    bigInt(3000),

    bigInt(5000),

    bigInt(10000),

    bigInt(15000)

];

const nValues = [

    bigInt(2).pow(1024),

    bigInt(2).pow(2048)

];

const aValues = [5, 10];

aValues.forEach(a => {

    nValues.forEach(n => {

        console.log("=================================");

        console.log(`a: ${a}, n: ${n}`);

        console.log("=================================");

        console.log("   x       | Время вычисления (мс)");

        console.log("=================================");

        xValues.forEach(x => {

            const timerStart = performance.now();

            const timerEnd = performance.now();

            const elapsedMilliseconds = timerEnd - timerStart;

            console.log(`${x}   |${elapsedMilliseconds}`);

        });

    });

});

Листинг 10.1 – Приложение task1

Результат работы приложения приведен на рисунке 10.1.



Рисунок 10.1 – Результат работы приложения task1

**Задание 2.** Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами. В основе вычислений – кодировочные таблицы Base64 и ASCII. Приложение должно реализовывать следующие операции:

• зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;

• определение времени выполнения операций.

Функция *rsaEncrypt* принимает на вход оригинальный текст и публичный ключ и осуществляет зашифрование с помощью алгоритма RSA.

static rsaEncrypt(data, publicKey) {

    const buffer = Buffer.from(data);

    const encryptedData = crypto.publicEncrypt(publicKey, buffer);

    return encryptedData.toString('base64');

}

Листинг 10.2 – Функция *rsaEncrypt*

Функция *rsaDecrypt* принимает на вход зашифрованный текст и приватный ключ и осуществляет расшифрование с помощью алгоритма RSA.

static rsaDecrypt(data, privateKey) {

    const buffer = Buffer.from(data, 'base64');

    const decryptedData = crypto.privateDecrypt(privateKey, buffer);

    return decryptedData.toString('utf-8');

}

Листинг 10.3 – Функция *rsaDecrypt*

Результат работы приложения с использованием алгоритма RSA и исходным сообщением «Kozeka Elizaveta Maksimovna» представлен на рисунке 10.2.



Рисунок 10.2 – Результат работы приложения (алгоритм RSA)

Как видим, зашифрование выполнилось за 1,07 секунды, в то время как расшифрование заняло гораздо больше времени – почти 4 секунды.

Функция *encryptElGamal* принимает на вход оригинальный текст и параметры публичного ключа и осуществляет зашифрование с помощью алгоритма Эль-Гамаля.

static encryptElGamal(p, g, x, originalString) {

    let result = "";

    const y = this.exponentiation(g, x, p);

    for (const char of originalString) {

        const code = char.charCodeAt(0);

        if (code > 0) {

            const k = Math.floor(Math.random() \* (p - 2)) + 1;

            const a = this.exponentiation(g, k, p);

            const b = this.multiplication(

this.exponentiation(y, k, p), code, p

);

            result += `${a} ${b} `;

        }

    }

    return result;

}

Листинг 10.4 – Функция *encryptElGamal*

Функция *decryptElGamal* принимает на вход зашифрованный текст и приватный параметр x и осуществляет расшифрование с помощью алгоритма Эль-Гамаля.

static decryptElGamal(p, x, encryptedText) {

    let result = "";

    const arr = encryptedText.split(' ').filter(xx => xx !== "");

    for (let i = 0; i < arr.length; i += 2) {

        const a = parseInt(arr[i]);

        const b = parseInt(arr[i + 1]);

        if (a !== 0 && b !== 0) {

            const deM = this.multiplication(

b, this.exponentiation(a, p - 1 - x, p), p

);

            const m = String.fromCharCode(deM);

            result += m;

        }

    }

    return result;

}

Листинг 10.5 – Функция *decryptElGamal*

Алгоритм Эль-Гамаля использует несколько параметров, которые являются ключевыми. Для начала было выбрано число *p* = 241. Далее было выбрано второе число *g* = 56, меньшее числа *p* и являющееся его первообразным корнем. Затем было выбрано число *x* = 9. Число *x* должно быть меньше, чем число *p*. Компонент *y* вычисляется внутри функции шифрования.

Результат работы приложения с использованием алгоритма Эль-Гамаля и исходным сообщением «Kozeka Elizaveta Maksimovna» приведен на рисунке 10.3.

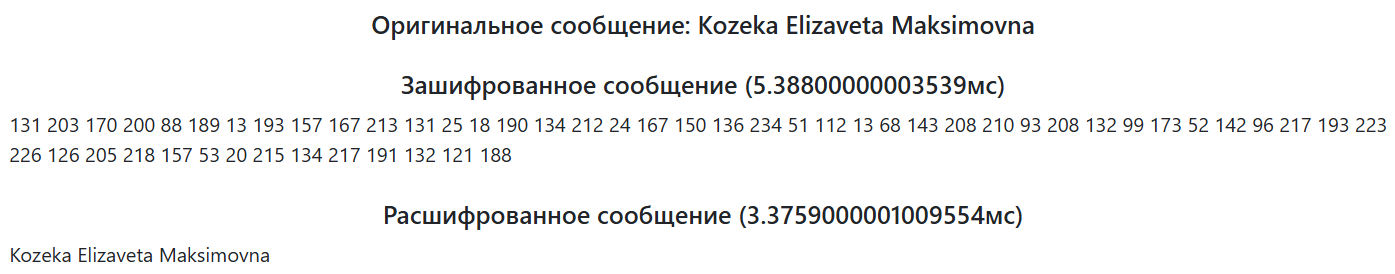


Рисунок 10.3 – Результат работы приложения (алгоритм Эль-Гамаля)

Как видим, сообщение было зашифровано за 5.39 секунды. Расшифрование с помощью алгоритма Эль-Гамаля заняло немного меньше времени по сравнению с алгоритмом RSA и составило 3.38 секунды.

**Задание 3**. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов).

Попробуем зашифровать и расшифровать более длинное сообщение с помощью алгоритма RSA.

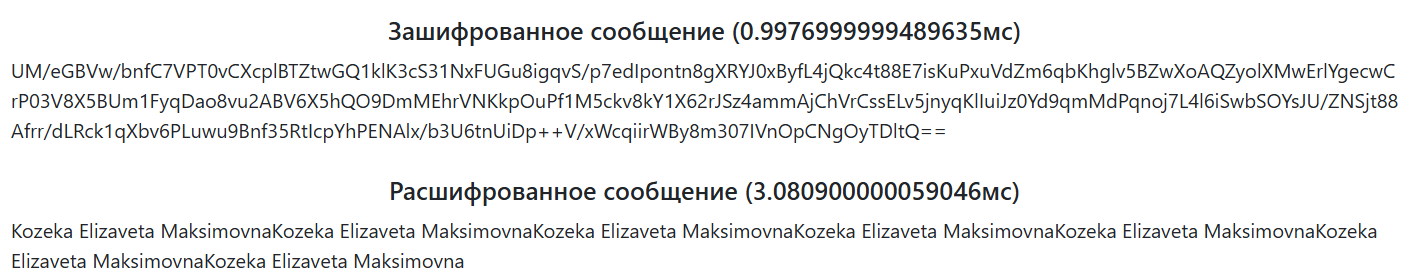


Рисунок 10.4 – Результат работы приложения с более длинным сообщением (алгоритм Эль-Гамаля)

Как видим, скорость шифрования и расшифрования немного уменьшилась, а объем шифртекста остался таким же.

Теперь зашифруем и расшифруем более длинное сообщение с помощью алгоритма Эль-Гамаля.

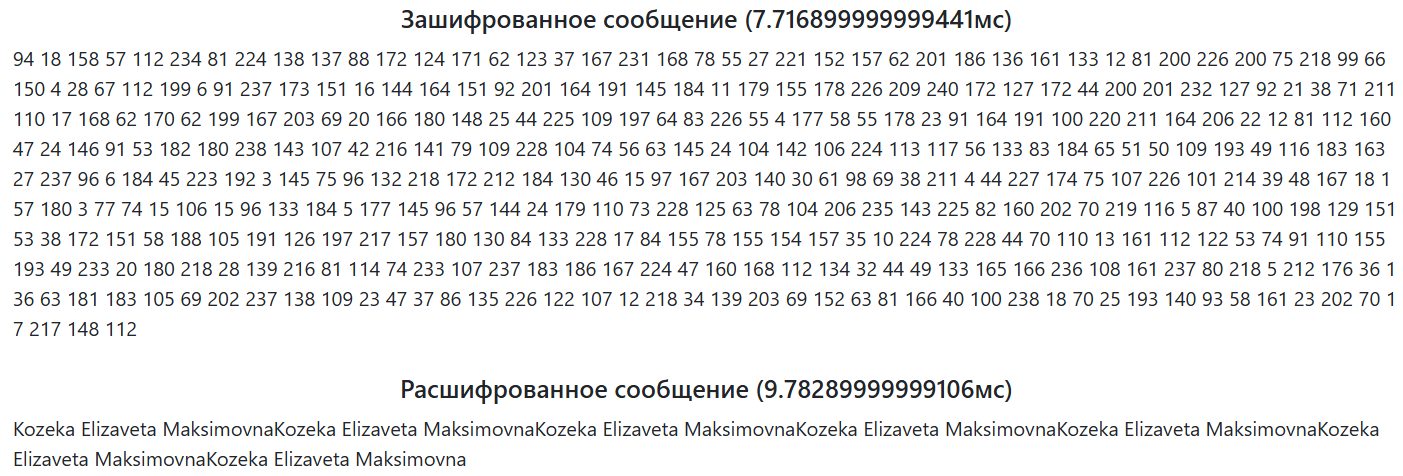


Рисунок 10.4 – Результат работы приложения с более длинным сообщением (алгоритм Эль-Гамаля)

Как видим, скорость шифрования и расшифрования увеличилась, как и объем шифртекста.

**Вывод:** в ходе выполнения лабораторной работы был изучен принцип реализации асимметричных шифров RSA и Эль-Гамаля. Также было разработано приложение, выполняющее шифрование и расшифрование с помощью шифра, основанного на алгоритме об укладке ранца. Была оценена скорость шифрования и дешифрования сообщений.