МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Исследование и реализация криптографических алгоритмов RSA

Обучающийся С. И. Смирнов (Инициалы Фамилия) _____ (личная подпись)

Руководитель К.ф.-м.н., доцент кафедры, Г. А. Тырыгина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант Старший преподаватель, Е. В. Косс
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

# Аннотация

Тема выпускной квалификационной работы: «Исследование и реализация криптографических алгоритмов RSA»

Цель данной работы: исследование работы алгоритма RSA и реализация программы, которая будет зашифровывать данные с помощью этого алгоритма.

Данная выпускная квалификационная работа посвящена исследованию и реализации алгоритма шифрования RSA и его актуальности в наше время.

Структура дипломной работы состоит из введения, трёх глав, заключения, списка используемой литературы и приложения.

Главным вопросом работы будет рассматриваться актуальность алгоритма RSA, возможные атаки на него, надёжность шифрования, положительные и отрицательные стороны алгоритма. Программный код написан на языке C++.

В первой главе рассматриваются основные понятия, связанные с алгоритмом RSA, а также сам алгоритм RSA, связанная с ним теория и описаны другие алгоритмы, которые применяются в RSA, также его достоинства и слабые стороны и возможные применения.

Во второй главе будет реализация алгоритма RSA на языке программирования C++

В третьей главе будет проводиться тестирование готовой библиотеки.

Готовый программный код представляет собой настраиваемую библиотеку для алгоритма RSA, который можно интегрировать в любой проект и использовать для зашифрования данных с использованием ключа любой длины.

# Abstract

The title of the graduation work is "Research and implementation of RSA cryptographic algorithms".

The aim of the graduation work is a study of the RSA algorithm and the implementation of a program that will encrypt data using this algorithm.

This graduation work is dedicated to the study and implementation of the RSA encryption algorithm and its relevance at present.

The thesis consists of an introduction, three chapters, a conclusion, a list of references and an appendix.

The main issue of the work is the relevance of the RSA algorithm, possible attacks on it, the reliability of encryption, positive and negative sides of the algorithm. The program code is written in C++.

In the first chapter the basic concepts associated with RSA algorithm are discussed, and the RSA algorithm itself, the theory associated with it. The chapter describes other algorithms that are used in RSA, its strengths and weaknesses, and possible applications.

The second chapter presents the implementation of RSA algorithm in C++ programming language.

In the third chapter the ready code is tested.

The programme code is a customized library for RSA algorithm, which can be integrated into any project and used to encrypt data using any key length.

# Оглавление

# Введение

С тех пор как люди начали записывать события из жизни, они нуждались в криптографии. Криптография – шифрование текста таким образом, что любой незнакомый с секретным ключом человек никогда не сможет понять сообщение, но нужный читатель сможет расшифровать шифр. У древних людей, как и у современных, всегда была нужда в секретности, потому что обычно это в интересе шифровщика и дешифровщика, чтобы нужная информация не стала широко известной. Во время войны было важно, чтобы враг не знал, что вы и ваши союзники планируете против него, потому что это может подорвать целую операцию и повлиять на дальнейший ход развития событий. Тем не менее, у всех криптосистем до RSA были определённые проблемы: все они предполагали, что для шифровки и расшифровки обе стороны должны знать метод шифрования и ключ шифрования. Проблема распределения ключей как была основной проблемой, для которой была придумана криптография: желаемый получатель должен знать ключ и метод, по которому было зашифровано сообщение, но как можно передать ключ так, чтобы никто другой его не получил? С помощью шифрования, конечно же. Но как тогда прислать ключ для зашифрованного ключа? Этот цикл можно продолжать вечность. Возможно, и немцы могли бы избежать многочисленных поражений от Союзников на море, если бы им не приходилось ежедневно печатать настройки для Энигмы и выдавать их всем своим лидерам. Одна из таких книг была перехвачена англичанами, что привело к поражению Германии на море, поскольку англичане смогли перехватывать сообщения от немецких подлодок и дешифровать их. Так проблема раздачи ключей и осталась актуальной вплоть до 20-го века.

Эта проблема была решена Вайтфилдом Диффи, работающим вместе с Мартином Хеллманом. Диффи нашёл нечто революционное, новый тип шифра: его шифр включал асимметричный ключ. Во всех других криптосистемах, расшифровка производится так же, как и шифрование (Тот

же самый алгоритм, но наоборот). В этих системах включен симметричный ключ, потому что шифровка и расшифровка симметричны. В ассиметричном шифре есть два разных ключа. Если человек, например, Иван, хочет послать сообщение другому человеку, Алисе, то всё, что ему нужно – использовать открытый ключ Алисы, чтобы зашифровать сообщение. И теперь единственный человек во вселенной, что сможет расшифровать это сообщение – Это Алиса, потому что у неё есть закрытый ключ для расшифровки. Иван зашифровывает сообщение, используя открытый ключ, но расшифровать его не может: шифровка – односторонняя функция, действия которой необратимы и могут быть расшифрованы лишь в том случае, если дешифровщик имеет закрытый ключ (который известен лишь одному лицу). Хоть Диффи и придумал общий концепт асимметричного шифрования, но у него не было такой функции, которая ему была нужна. Тем не менее, его доклад (опубликованный в 1975) показал, что решение раздачи ключей существует, чем зажёг интерес среди других математиков и физиков. Хоть он и пытался изо всех сил, но у него, его и его партнёров Хеллмана и Меркля, ничего не выходило. Функция была найдена другими тремя исследователями: Ривестом, Шамиром и Адлеманом. Позже она была названа RSA (Rivest, Shamir, Adelman).

Этот алгоритм был первой ассиметричной криптосистемой, которая основывалась на факторизации больших простых чисел. Ассиметричная криптография способствовала развитию многих прикладных областей. Например, Система электронной цифровой подписи. Алгоритм RSA широко используется и сейчас, его часто сочетают с симметричными алгоритмами из-за низкой скорости шифрования.

Таким образом, актуальность темы ВКР определяется широким распространением алгоритма и его использованием и сейчас.

Объект исследования – ассиметричный алгоритм шифрования данных RSA.

Предмет – реализация алгоритма RSA.

Цель работы – реализация криптографического алгоритма RSA.

Для достижения работы необходимо решить следующие задачи:

- Рассмотреть и изучить теоретическую часть алгоритма RSA, изучить его стойкость алгоритма;
- Программная реализация алгоритма RSA;
- Тестирование написанной программы, выявление положительных сторон и недостатков алгоритма.

# Глава 1. Теоретические аспекты алгоритма RSA

Сам алгоритм был создан в 1977 году группой учёных Массачусетского технологического университета, Р. Л. Ривест, А. Шамир и Л.М. Адлеман и был прорывом в криптографии. Основным преимуществом алгоритма было наличие такого компонента, без которого было невозможно расшифровать сообщение. Сам алгоритм опирается на факторизацию простых чисел и требует большого количества ресурсов. Хоть это и является недостатком, но делает алгоритм более стойким к атакам. Также это приводит к тому, что скорость шифрования у него значительно ниже, чем у других подобных алгоритмов.

Передача сообщений с помощью алгоритма выглядит следующим образом:

- Каждый пользователь генерирует пару ключей: один для шифрования, другой для дешифрования;
- Каждый пользователь публикует свой ключ шифрования, размещает его в открытом для всех доступе. Второй ключ, соответствующий открытому, сохраняется в секрете;
- Если пользователь *A* собирается послать сообщение пользователю *B*, он шифрует сообщение открытым ключом пользователя *B*;
- Когда пользователь *B* получает сообщение, он дешифрует его с помощью своего личного (секретного) ключа. Другой получатель не сможет дешифровать сообщение, поскольку личный ключ *B* знает только *B*. [1][2][5][14]

## 1.1 Понятие асимметричного алгоритма шифрования

Сама концепция асимметричных алгоритмов была выдвинута Уитфилдом Диффи и Мартином Хеллманом, а также независимо Ральфом Мерклом. Основной их идеей было то, что ключи можно было использовать парами – отдельные для шифрования и отдельные для дешифрования и что ключи эти невозможно было получить из друг друга. Впервые Диффи и

Хеллман представили эту идею на Национальной Компьютерной Конференции в 1976 году, но из-за отсутствия интереса в публике, первый вклад в асимметричное шифрование был внесён через два года, в 1978 году Ральфом Мерклом. На рисунке 1 представлен сам алгоритм Димми-Хеллмана. [22]

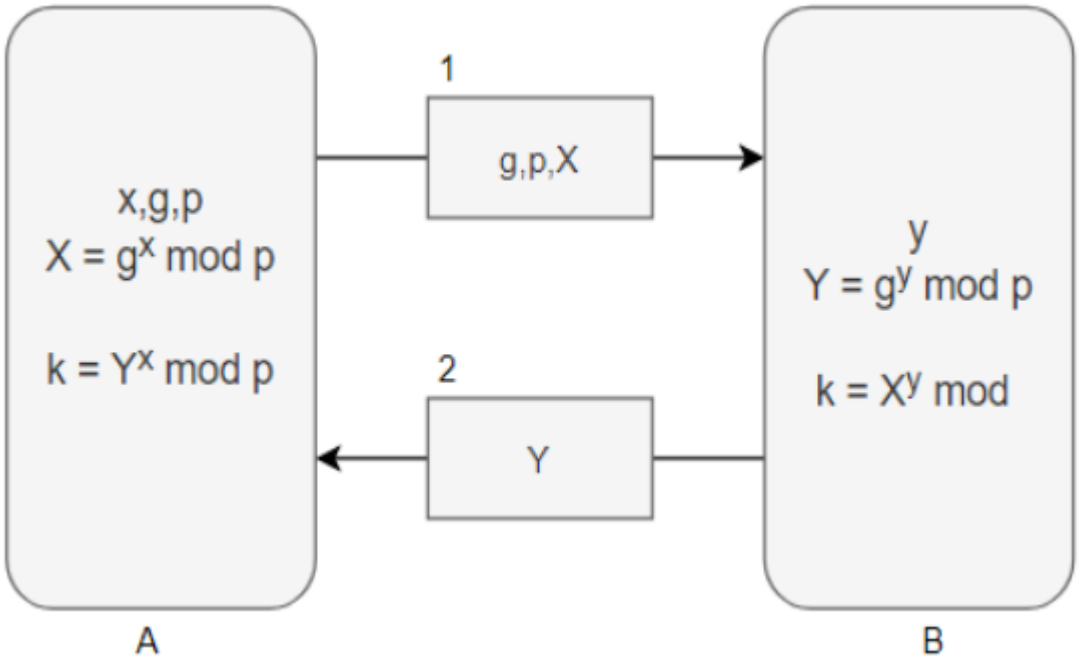


Рисунок 1 алгоритм Димми-Хелмана

В 1976 году было предложено очень много асимметричных алгоритмов, но многие из них являются небезопасными. Безопасные же были слишком сложны для реализации, либо же для них используется слишком большой ключ. В других размер зашифрованного текста превышал размер оригинального [11][17].

Лишь немногие алгоритмы являются практичными и в то же самое время безопасными. Многие из них не могут быть использованы универсально и в основном используются для разных сфер (электронная подпись, распределение ключей, шифрование текста). И лишь немногие работают одновременно хорошо для всего, одним из таких алгоритмов является алгоритм RSA. Но все эти алгоритмы очень медленно шифруют данные и зачастую не используются для шифрования большого объёма данных.

Но основным достоинством этих алгоритмов является их возможность пересылать сообщения по небезопасным каналам данных с сохранением их секретности. И для их расшфировки потребуется решать разные сложные математические проблемы, которые зачастую бывают трудновычислимыми. Например, RSA строится на факторизации чисел. И для решения этих проблем могут уйти столетия и большое количество ресурсов. [1][2][3][6]

### 1.2 Описание алгоритма RSA

Сперва исходный текст, который нужно зашифровать, переводится в числовую форму. Чаще всего выбирается ASCII-кодировка, но можно взять и свою (в этом случае нужно стараться избегать двусмысленности, чтобы, например, блок 12 не мог обозначать пару букв АБ, либо одну букву Л). Как результат, у нас выходит одно большое число, которое необходимо разбить на блоки. Желательно не выделять блоки, которые начинаются с нуля и нежелательно, чтобы блоки были больше, чем число $N - 1$ (о числе N будет написано ниже). Для каждого блока шифрование будет проходить одинаково.

“Дальше каждый пользователь генерирует свою пару ключей. Для этого генерируются два больших простых числа $p$ и $q$ и вычисляется $N = p \cdot q$. Затем генерируется случайное число $e$, которое будет взаимно простым со значением функции Эйлета от числа $N$, $\varphi(N) = (p-1) \cdot (q-1)$ и находит число $d$ из условия $e \cdot d = 1(mod\ \varphi(N))$. Так как $(e, \varphi(N)) = 1$, то такое число $d$ существует, и оно единственно. Пара $(N, e)$ объявляется открытым ключом, в то время как пара $(N, d)$” [7] объявляется закрытым. Для расшифровки сообщения пригодится лишь закрытый ключ, остальные же числа больше не пригодятся и от них можно избавиться.

Ключи сгенерированы, следующим шагом будет зашифровка сообщения, его передача и расшифровка. Иван хочет отправить сообщение Алисе. Для этого он берёт её открытый ключ и производит шифрование сообщения $x$. $y = x^e (mod\ N)$. Алиса получает зашифрованное сообщение $y$ и расшифровывает его $x = y^d (mod\ N)$

$x = y^d (mod\ N)$, так как $e \cdot d = \varphi(N) \cdot k + 1$, где $k$ – целое. Применив теорему Эйлера, получим следующее соотношение: $y^d \equiv (x^e)^d \equiv x^{ed} \equiv x^{\varphi(N)\cdot k+1} \equiv \left(x^{\varphi(N)}\right)^k \cdot x \equiv x (mod\ N)$.

Функция Эйлера показывает количество целых чисел на отрезке от 1 до $m$, которые взаимно простые с $m$ [13][18][19].

### 1.3 Поиск нужных $p$ и $q$

Для работы алгоритма требуются простые числа. Самыми подходящими будут случайно генерируемые значения и проверка этих чисел на простоту. Существуют определённые тесты для проверки чисел на простоту. В случае, если число окажется составным, то алгоритм шифрования-расшифрования не будет работать. Также к случайно генерируемым числам есть определённые требования. Числа не должны быть в списках известных больших простых чисел[5]. Эти числа не должны стоять близко, иначе их можно будет вычислить с помощью теоремы Ферма. [22]

Для избежания факторизации чисел используется следующее требование, сформулированное Райвестом: все числа $p_1 = \frac{p-1}{2}, p_2 = \frac{p+1}{2}, q_1 = \frac{q-1}{2}, q_2 = \frac{q+1}{2}$ должны быть простыми. [5][18]

### 1.4 Про параметры $e$ и $d$

Параметры $e$ и $d$ являются важными параметрами, поскольку они напрямую влияют на время зашифрования и расшифрования сообщения. Для многих ситуаций эти параметры выбираются по-разному и есть определённые ситуации, где эти значения должны быть большими. Например, алгоритм RSA используется в электронных платежах, где значение $e$ для центрального компьютера является большим, в то время как значение $d$ у пользователя является малым. [5]

Тем не менее, использование малых значений является небезопасным и может привести к взлому шифра с помощью различных алгоритмов. Если малым будет значение $d$, то его можно найти обычным перебором. В случае,

когда малым является параметр $e$, то большое число сообщений можно будет зашифровать возведением в степень. В случае, если у нескольких абонентов $e$ совпадает, то шифр можно атаковать китайской теоремой об остатках. [10]

### 1.5 Подбор простых чисел, алгоритм поиска простых чисел

Одним из обязательных условий работы алгоритмы RSA является поиск простых чисел $p$ и $q$. Если хотя бы одно из этих чисел будет составным, то алгоритм не сможет работать. Поскольку в реальном применении в алгоритме используются очень большие простые числа, то нужно их правильно находить. Использование малых простых чисел, либо простых чисел, что уже известны, ставит безопасность использования такой системы под угрозу, поскольку её будет легко взломать обычным перебором простых чисел.

Один из возможных вариантов для поиска простых чисел, является решето Эратосфена, открытое древнегреческим учёным Эратосфеном Киренским. Вся суть метода заключается в том, чтобы взять таблицу с числами от 2 до $n$ и вычеркнуть сперва всё чётные числа, дальше все числа, что делятся на 3 (за исключением самой 3). Дальше зачёркивается каждое пятое число после 5 [4][9][12].

Тем не менее, есть более эффективные и сложные алгоритмы для поиска простых чисел. Например, тест AKS, названный так в честь разработавших его учёных (Аггравала, Каяла, Саксены). Это единственный универсальный тест, который не зависит от недоказанных гипотез. Сам тест может использоваться для проверки любых чисел, что делает его универсальным, и даёт гарантированный ответ, в то время, как другие предлагают лишь вероятностный.

Алгоритм теста AKS:

- Вводится $n \in \mathbb{N}, n > 1$;
- Найти наименьшее $r$ такое что $o_r(n) > \left(log_2(n)\right)^2$;
- Если НОД $(a, n) \neq 1$ для любого $a \leq r$, вернуть “составное”;

– Если для всех $a$ от 1 до $\lfloor \sqrt{r} log\, n \rfloor$ верно, что $(x+a)^n \equiv x^n + a\ (mod\ \varphi(x), n)$, вернуть “простое”;
– Иначе вернуть “составное”.

Таким образом, были рассмотрены основные алгоритма для поиска и теста простых чисел.

### 1.6 Поиск НОД, алгоритм Евклида

Основой алгоритма RSA является поиск наибольшего общего делителя. Без этого алгоритм не сможет полноценно работать. Эту проблему решает алгоритм Евклида. Он нужен для поиска наибольшего общего делителя двух чисел, был описан ещё в древности математиком Евклидом. Этот алгоритм, не смотря на свою старость, используется и в наше время. Алгоритм Евклида используется для поиска параметра $d$, то есть, для шифрования и расшифрования сообщения $m$ [4].

Работа алгоритма Евклида:

Пусть числа $a$, $b$ целые и не равны 0.

Тогда последовательность $a > b > r_1 > r_2 > r_3 \ldots > r_n$ определена тем, что каждое $r_k$ – это остаток от деления предыдущего числа на предыдущее, а последнее делится нацело, то есть [4]:

$a = bq_0 + r_1,$

$b = r_1 q_1 + r_2,$

$r_1 = r_2 q_2 + q_3,$

...

$r_{n-2} = r_{n-1} q_{n-1} + r_n,$

$r_{n-1} = r_n q_n.$

Сложность алгоритма Евклида оценивают в $O(h^2)$, где $h$ – среднее число цифр в двух начальных числах $a$ и $b$ в десятичной записи.

Этот алгоритм будет использоваться дальше для поиска взаимно простых чисел между числами $d$ и $m$ [8][10][15].

### 1.7 Зашифрование и расшифрование сообщения

После получения всех нужных числовых коэффициентов, мы берём уже переведённое в численный вид сообщение и можем приступить к его зашифровке. Функция, которая реализует схему RSA:

$$x \leftrightarrow x^e (mod\ m)$$

Шифрование происходит блоками, и в случае, если нужно зашифровать определённое количество блоков. То можно добавить дополнительные нули слева. Сама формула шифрования по блокам будет выглядеть так:

$$c_i = m_i^e\ mod\ N$$

Для расшифровки сообщения берётся зашифрованный блок и вычисляется по формуле

$$m_i = c_i^d\ mod\ n$$

Поскольку

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i;$$

все $(mod\ n)$

Эта формула восстанавливает сообщение из его зашифрованного вида [5].

### 1.8 Скорость работы алгоритма

По большей части алгоритм RSA представляет из себя возведение в степень, как во время шифрования и расшифровки, так и во время проверки и создания подписи на основе алгоритма RSA, что представляет собой ряд умножений чисел.

В практике для открытого ключа чаще всего выбираются небольшие числа $e$ и $N$ и очень часто разные группы пользователей могут выбирать один и тот же открытый показатель, но уже с разным $N$. Если же это число остаётся тем же самым, то тогда вводятся ограничения на главные делители модуля. Причём шифрование данных проходит быстрее расшифровки, как и проверка подписи проходит быстрее, чем, непосредственно, само подписание с помощью алгоритма RSA. [20]

Если $k$ – количество битов в модуле, то в обычно используемых для RSA алгоритмах количество шагов необходимых для выполнения операции с открытым (public) ключом пропорционально второй степени $k$, количество шагов для операций частного (private) ключа – третьей степени $k$, количество шагов для операции создания ключей – четвертой степени $k$.

Методы "быстрого умножения" – например, методы основанные на Быстром Преобразовании Фурье (FFT – Fast Fourier Transform) – выполняются меньшим количеством шагов; тем не менее они не получили широкого распространения из-за сложности программного обеспечения, а также потому, что с типичными размерами ключей они фактически работают медленнее. Однако производительность и эффективность приложений и оборудования реализующих алгоритм RSA быстро увеличиваются.

Алгоритм RSA сам по себе значительно медленнее чем DES или любые другие симметричные алгоритмы блокового шифрования. Реализация алгоритма DES зашифровывает и расшифровывает данные по крайней мере в 100 раз, в некоторых случаях даже больше, быстрее алгоритма RSA – в аппаратной реализации. Скорее всего, в будущем аппаратное обеспечение станет лучше и работа алгоритма RSA ускорится, но точно так же ускорятся и симметричные алгоритмы. На рисунке 2 показана скорость RSA для различных длин модулей при 8-битном открытом ключе [16].

**Табл. 19-4.**
**Скорости RSA для различных длин модулей при 8-битовом открытом ключе (на SPARC II)**

| | 512 битов | 768 битов | 1024 бита |
|---|---|---|---|
| Шифрование | 0.03 с | 0.05 с | 0.08 с |
| Дешифрирование | 0.16 с | 0.48 с | 0.93 с |
| Подпись | 0.16 с | 0.52 с | 0.97 с |
| Проверка | 0.02 с | 0.07 с | 0.08 с |

Рисунок 2 Скорости RSA

Таким образом можно увидеть, что расшифрование представляет собой более ресурсоёмкий процесс и занимает значительно больше времени, когда как зашифрование является процессом более простым.

### 1.9 Использование алгоритма для цифровых подписей

Цифровая подпись – копия открытого ключа удостоверения подлинности сообщения, их синтаксис и гарантии безопасности – аналоги. Алгоритм, который пользователь выбрал для зашифровки сообщения, называется mac, а результат работы алгоритма называется tag. Алгоритм и tag, которые используются получателем для подтверждения действительности, называется vrfy.

Работа шифрования цифровых подписей:

Для этого владелец подписи сначала вычисляет хеш-значение. Дальше происходит шифрование полученного хеш-значения. После этого открытые ключи выкладываются в общественный доступ, где любой пользователь может расшифровать хеш-значение документа и проверить его подлинность.

Для цифровых подписей обычно используются модификации RSA, которые используют дополнительную криптосистему для избежания взлома сообщения путём метода бесключевого чтения, суть которого заключается в посылании поддельной подписи. Также есть и куда более опасная атака, которая подделывает $m$ с помощью открытого ключа. Злоумышленник выбирает произвольные $m_1$и $m_2$ такие, что $m = m_1 \cdot m_2 mod\ N$. Дальше он получает mac на оба сообщение одновременно и вычисляет mac от оригинального $m$ [16][18].

### 1.10 Пример работы алгоритма

В качестве работы алгоритма будет приведён очень простой пример, результаты которого можно будет получить самостоятельно на листке бумаги, либо с помощью калькулятора.

- Выбрать простые числа $p = 11$ и $q = 3$;
- $N = p \cdot q = 33$;

$\varphi(N) = (p-1)\cdot(q-1) = 10\cdot 2 = 20;$

– Выбрать простое число $e = 3;$

Проверить НОД$(e, p-1)$ = НОД$(3, 10) = 1$ (Т.к. у 3 и 10 нет общих делителей, кроме 1)

и проверить НОД$(e, q-1)$ = НОД$(3, 2) = 1$

таким образом НОД$\big(e, \varphi(N)\big)$= НОД$(3, 20) = 1;$

– Найти $d$ такое, что $e \cdot d \equiv 1\big(mod\ \varphi(N)\big)$

Посчитать $d = \left(\frac{1}{e}\right)\ mod\ \varphi(N) = \left(\frac{1}{3}\right) mod\ 20$

Найти такое значение $d$ при котором $\varphi(N)$ делится на $(e \cdot d - 1)$

Посчитаем, тогда $d = 7;$

– Открытый ключ $= (N, e) = (33{,}3)$

Закрытый ключ $= (N, d) = (33{,}7).$

Это самое малое возможное значение для $N$, с которым алгоритм RSA может работать.

Теперь попробуем зашфировать сообщение $m = 7$

$$c = m^e mod\ n = \ 7^3 mod\ 33 = 343 mod\ 33 = 13$$

Таким образом, шифротекст $c = 13$

Теперь расшифруем это сообщение

$$m' = c^d mod\ n = \ 13^7 mod\ 33 = 7$$

На рисунке 3 можно наглядно увидеть ход работы алгоритма.

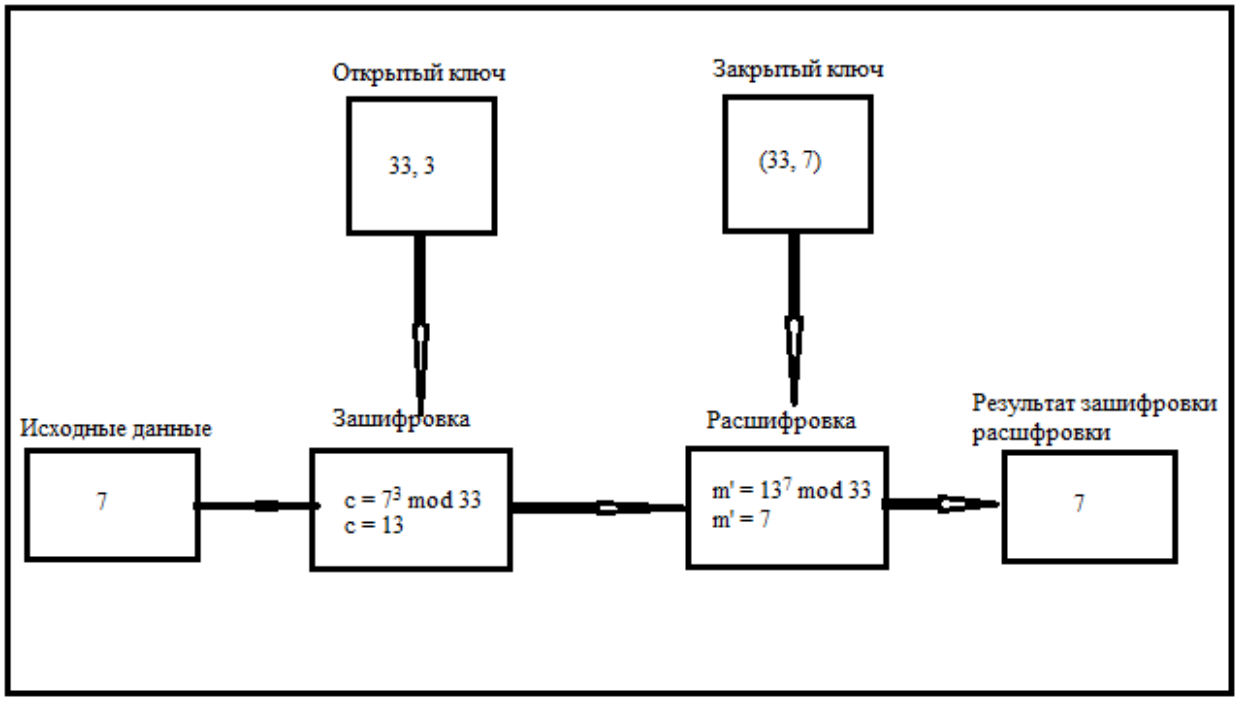


Рисунок 3 работа алгоритма

В итоге, был показан пример, который на очень простом примере позволяет рассмотреть работу алгоритма RSA, шифрование и дешифрование, генерирование ключей.

### 1.11 Криптоанализ алгоритма

“Криптография – наука о шифрах – долгое время была засекречена, так как применялась, в основном, для защиты и государственных секретов. В настоящее время методы и средства криптографии используются для обеспечения не только государства, но и частных лиц, и организаций.”[3]

Безопасность самого алгоритма состоит в трудоёмкости разложения на множители больших чисел. Одним из примеров того, насколько безопасен алгоритм, может послужить зашифрованное сообщение “THE MAGIC WORD ARE SQUEAMISH OSSIFRAGE”. На дешифровку этого сообщения ушло около полугода у 600 добровольцев, которые пожертвовали процессорное время 1600 машин. В качестве открытых параметров системы были использованы числа из 129 десятичных знаков, также известный как RSA-129.

Хоть алгоритм и является очень устойчивым, для его корректной работы требуется правильное задание параметров. Среди возможных атак на алгоритм являются: атака повторным шифрованием (работает лишь при слишком близких значениях $p$ и $q$), обычным подбором (что маловероятно при больших цифрах) и бесключевое чтение (при условии наличия общих модулей $N$). “Поэтому выбор параметров криптосистемы является очень ответственной задачей и выбирать их требуется в соответствии с требованиями. Тем не менее, при их соблюдении атака на алгоритм или криптосистему может быть проведена лишь в случае неправильного подбора параметров.” [12]

Даже в случае неправильно подобранных алгоритмов, в случаях, когда $p$ и $q$ являются составными числами, то алгоритм попросту не будет работать.

Как уже было сказано, алгоритм RSA является очень требовательным и для его работы требуется $O(\ln e)$ операций умножения по модулю.

Основными ограничения по RSA были введены Джудит Мур на основании всевозможных атак:

- Знание одной пары показателей шифрования/дешифрования для модуля позволит взломщику разложить модуль на множители;
- Знание одной пары показателей шифрования/дешифрования для модуля позволяет взломщику вычислить другие пары показателей, не раскладывая модуль на множители;
- В протоколах сетей связи, применяющих RSA, не должен использоваться общий модуль;
- Для предотвращения вскрытия малого показателя шифрования сообщения должны быть дополнены случайными значениями;
- Показатель дешифрования должен быть большим. [10]

Для безопасности криптосистемы недостаточно использовать один лишь алгоритм. Следует обеспечить безопасность крипосистемы, криптопротокола и криптоалгоритма. Слабое место любой из трёх

составляющих может привести к тому, что вся система будет небезопасной. [6][10]

### 1.12 Достоинства и недостатки алгоритма

Криптографическая система RSA – одна из надёжных систем защиты информации при условии соблюдения всех правил. RSA BSEFE используется пользователями по всему миру для зашифрования и передачи данных.

Алгоритм RSA очень прост для понимания, также он обладает более простым шифрованием и проверкой, по сравнению с алгоритмом ECC.

Одно из достоинств системы – её асимметричность. Система позволяет шифровать сообщения и отправлять их другим пользователям, и никто, кроме получателя, не сможет расшифровать сообщение. Также RSA используется в программе шифрования PGP, которая занимается шифровкой сообщений и для цифровых подписей.

Благодаря асимметричности алгоритма заменять ключи шифрования можно спустя долгое время, либо в случае их утечки. Для симметричных алгоритмов требуется постоянная перезапись этих ключей для повышения безопасности.

Тем не менее, у RSA есть и недостатки. Одним из основных недостатков является производительность криптосистемы. Алгоритм RSA уступает в скорости шифрования алгоритму DES и другим симметричным алгоритмам. В то же самое время алгоритм ECC использует ключи куда меньшей длины и обеспечивает точно такую же безопасность (AES-256 ~ ECC-512 ~ RSA-15424) [16][18]

Кроме того, недостатком является и более низкая скорость шифрования данных. Поэтому чаще всего используются гибридные алгоритмы. Часто данные шифруются с помощью симметричного алгоритма, в то время как ключи для этого алгоритма шифруются и передаются с помощью асимметричного алгоритма.

Выводы по первой главе:

В ходе написания первой главы данной выпускной квалификационной работы были рассмотрены основные теоретические аспекты алгоритма RSA, которые понадобятся для дальнейшего хода работы ВКР.

В данной главе были рассмотрены следующие понятия:

- Асимметричные криптоалгоритмы;
- алгоритм RSA;
- алгоритм Евклида;
- алгоритмы поиска простых чисел;
- безопасность RSA;
- эффективность RSA;
- плюсы и минусы RSA.

Также было подготовлено задание для тестирования программы алгоритма RSA, выявлены основные достоинства и недостатки, разобраны разные возможные атаки на данную криптосистему.

# Глава 2. Реализация алгоритма RSA

Для реализации алгоритма RSA был выбран язык программирования C++. «Язык C++ предназначен для разработки высокопроизводительного программного обеспечения и чрезвычайно популярен среди программистов. При этом он обеспечивает концептуальный фундамент (синтаксис и стиль), на который опираются другие языки программирования. Более того, C++ можно назвать универсальным языком программирования, поскольку практически все профессиональные программисты на том или ином уровне знакомы с C++» [15, с. 17].

Средой разработки был выбран Code::Blocks, поскольку она очень удобна для работы и хорошо подходит для программирования на языке C++.

В качестве компилятора был выбран идущий вместе с IDE MinGW.

В ходе реализации алгоритма RSA будет написан программный код, который будет самостоятельно генерировать числа $p$ и $q$. Дальше программа самостоятельно рассчитывать все нужные параметры, генерировать открытый и закрытый ключи и шифровать случайную строку из цифр. Дальше программа будет подсчитывать время работы программы: время генерации ключа, зашифровку и расшифровку сообщения. Пример реализации кода находится в Приложении А.

Для реализации алгоритма была составлена блок-схема

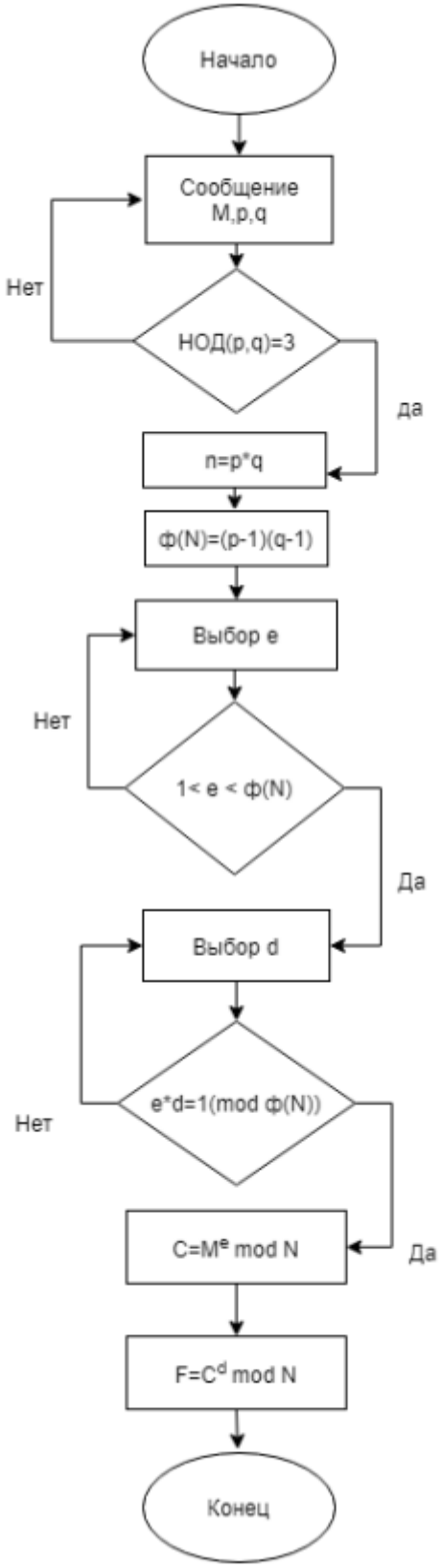


Рисунок 4 Блок-схема алгоритма

Входными данными для алгоритма является сперва ввод двух простых чисел $p$ и $q$, генерирующиеся случайно программой. После этого вычисляется их сумма и функция Эйлера. После этого подсчитывается число $e$, взаимно простое со значением функции Эйлера. Дальше вычисляется число $d$, которое мультипликативно обратное к числу $e$ по модулю $\varphi(N)$, то есть, удовлетворяющая сравнению $de \equiv 1 \left(mod\varphi(N)\right)$. В результате получаются открытый и закрытый ключи, которые можно использовать для зашифровки сообщений. После этого программа автоматически генерирует случайно сообщение, которое она сперва зашифровывает с помощью открытого ключа, а после расшифровывает при помощи закрытого.

Сперва для реализации алгоритма были подключены следующие библиотеки, показанные на рисунке 5:

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <exception>
#include <string>
#include <memory>
#include "BigInteger.h"
#include "RSABigInteger.h"
#include <ctime>
```

Рисунок 5 используемые библиотеки

Эти библиотеки позволят работать с выводом и вводом текста, с математическими формулами и прочими математическими функциями, которые нужны для реализации RSA, а также со строками, подсчётом времени, двумя заголовочными файлами, которые содержат все нужные для работы RSA функции и два новых класса BigInteger и RSABigInteger.

В программе для упрощения работы с ней работа будет проводиться с числами. Это нужно для упрощения написания самой программы и для того, чтобы избавиться от необходимости создания отдельного алфавита для работы с программой. Пример того, как это выглядит, представлен на рисунке 6:

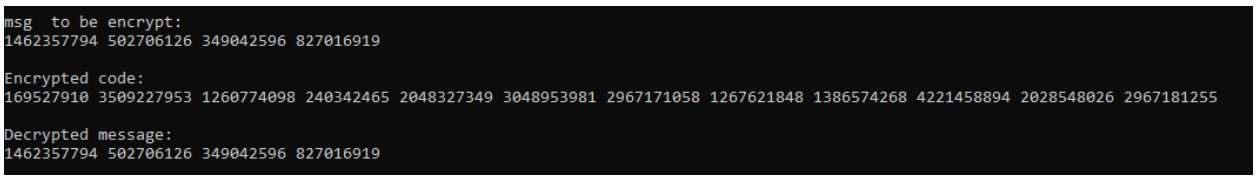


Рисунок 6 Пример вывода зашифрованного сообщения

Таким образом, не придётся создавать конвертер для перевода цифр в буквы, что несколько ускорит работу всей программы. Но для создания больших чисел нужно будет два класса для работы с большими числами. Для этого было реализовано два класса BigInteger и RSABigInteger, представленные на рисунках 7, 8.

```
#ifndef BIGINTEGER_H
#define BIGINTEGER_H


class BigInteger
{
    public:
        int nSize;
        unsigned int *digit;
        BigInteger(){}
        BigInteger(int n);
        BigInteger(const BigInteger &obj);
        ~BigInteger();


        void addBigInteger(BigInteger& a, BigInteger& b);
        void subBigInteger(BigInteger& a, BigInteger& b);
        void multBigInteger(BigInteger& a, BigInteger& b);
        void copyBigInteger(BigInteger& a, int index);
        void expoModNBigInteger(BigInteger& x, BigInteger& y, BigInteger& N, BigInteger& result);
        void setSize(int n);
        int msbBigInteger();
        void clearBigInteger();
        void showDigits();
        void setDigits(int index);
};

int Compare(BigInteger& first, BigInteger& second);
int divBigInteger(BigInteger& u, BigInteger& v, BigInteger& q, BigInteger& r);
void gcdBigInteger(BigInteger& a, BigInteger& b, BigInteger& result);

#endif // BIGINTEGER_H
```

Рисунок 7 класс BigInteger

```
#ifndef RSABIGINTEGER_H
#define RSABIGINTEGER_H
#include "BigInteger.h"

class RSABigInteger
{
    public:
        RSABigInteger(int nSize);
        virtual ~RSABigInteger();

        void CalculateD(BigInteger& e, BigInteger& phi, BigInteger& d);
        void init(BigInteger&p, BigInteger& q);
        void eGeneration(BigInteger& phi,BigInteger& result);
        void encryption(BigInteger& msg, BigInteger& code);
        void decryption(BigInteger& code, BigInteger& msg);
        void randomNGeneration(BigInteger& randResult, int n);
        void primeNumberGeneration(BigInteger& randPrime, int n);

    private:
        int SIZE;
        BigInteger N;
        BigInteger d;
        BigInteger e;
};

#endif // RSABIGINTEGER_H
```

Рисунок 8 RSABigInteger

Все функции, которые будут использоваться в программе перечислены в начале самой программы, представлены на рисунке 5 и 6.

Сперва происходит генерация двух простых чисел $p$ и $q$. Для этого в классе RSABigInteger запускается функция primeNumberGeneration, которая генерирует случайное большое число и проверяет его на простоту. Сначала генерируется несколько объектов класса BigInteger, часть из которых используется временно. Фрагмент кода находится на рисунке 9.

```
void RSABigInteger::primeNumberGeneration(BigInteger& randPrime, int n)
{
    BigInteger valueOne(randPrime.nSize);
    valueOne.digit[0] = 1;
    BigInteger valueTwo(randPrime.nSize);
    valueTwo.digit[0] = 2;
    BigInteger valueThree(randPrime.nSize);
    valueThree.digit[0] = 3;

    BigInteger tempPrime(randPrime.nSize);
    BigInteger tempExpoDummy(randPrime.nSize);
    BigInteger ExpoDummy(randPrime.nSize);

    BigInteger tempRemainder(randPrime.nSize);
    BigInteger tempPrimeMinusOne(randPrime.nSize);
    randomNGeneration(tempPrime,n);

    while(1)
    {
        tempPrimeMinusOne.clearBigInteger();
        tempRemainder.clearBigInteger();
        ExpoDummy.clearBigInteger();
        tempExpoDummy.clearBigInteger();
        tempPrimeMinusOne.subBigInteger(tempPrime, valueOne);

        ExpoDummy.expoModNBigInteger(valueTwo, tempPrimeMinusOne, tempPrime, tempRemainder);

        if(tempRemainder.msbBigInteger()==0 && tempRemainder.digit[0]==1)
        {
            cout<<"Prime Number Generating ... Wait a little bit\n";
            tempRemainder.clearBigInteger();
            tempExpoDummy.expoModNBigInteger(valueThree, tempPrimeMinusOne, tempPrime, tempRemainder);
            if(tempRemainder.msbBigInteger()==0 && tempRemainder.digit[0]==1)
            {
                break;
            }
        }
        tempPrime.clearBigInteger();
        randomNGeneration(tempPrime,n);
    }
    randPrime.copyBigInteger(tempPrime,0);
}
```

Рисунок 9 генерация случайного большого числа

Далее программа будет выводить это числа на экран. Пример кода находится на рисунке 10.

```
void BigInteger::showDigits()
{
    int nonZeroDigitFlag =0;
    for(int i=(nSize-1);i>=0; i--)
    {
        if(digit[i]!=0)
        {
            nonZeroDigitFlag=1;
        }
        if(nonZeroDigitFlag==1)
        {
            printf("%u ",digit[i]);
        }
    }

    if(nonZeroDigitFlag ==0)
    {
        printf("0");
    }
    cout << endl;
}
```

Рисунок 10 Вывод большого числа

Как только программа заканчивает работу с генерацией больших простых чисел, она сразу же приступает к высчитыванию всех основных данных для работы алгоритма RSA, такие как $\varphi(N), N, d, e$. Фрагмент кода предоставлен на рисунке 11.

```
void RSABigInteger::init(BigInteger&p, BigInteger& q)
{
    N.multBigInteger(p,q);
    BigInteger one(p.nSize);
    one.digit[0] = 1;
    BigInteger phi(p.nSize);

    BigInteger tempP(p.nSize), tempQ(p.nSize);

    tempP.subBigInteger(p,one);
    tempQ.subBigInteger(q,one);

    phi.multBigInteger(tempP,tempQ);
    cout<<"\nGenerated 'phi' :"<<endl;
    phi.showDigits();
    eGeneration(phi,e);
    cout<<"\nGenerated 'e' :"<<endl;
    e.showDigits();
    CalculateD(e,phi,d);
    cout<<"\nGenerated 'd' :"<<endl;
    d.showDigits();
}
```

Рисунок 11 генерация и вывод чисел

Но для генерации $d$ и $e$ программой используются специально сделанные для этого функции EGeneration и CalculateD. EGeneration ищет наибольший общий делитель функции Эйлера. Для этого в функцию сообщается $\varphi(N)$ и результат, в которую будет заноситься $e$. Фрагмент кода EGeneration представлен на рисунке 10.

```
void RSABigInteger::init(BigInteger&p, BigInteger& q)
{
    N.multBigInteger(p,q);
    BigInteger one(p.nSize);
    one.digit[0] = 1;
    BigInteger phi(p.nSize);

    BigInteger tempP(p.nSize), tempQ(p.nSize);

    tempP.subBigInteger(p,one);
    tempQ.subBigInteger(q,one);

    phi.multBigInteger(tempP,tempQ);
    cout<<"\nGenerated 'phi' :"<<endl;
    phi.showDigits();
    eGeneration(phi,e);
    cout<<"\nGenerated 'e' :"<<endl;
    e.showDigits();
    CalculateD(e,phi,d);
    cout<<"\nGenerated 'd' :"<<endl;
    d.showDigits();
}
```

Рисунок 12 поиск $e$

Для вычисления числа $d$ в функцию CalculateD вводится значение функции Эйлера, а также число $e$. Далее из них высчитывается значение $d$ с помощью алгоритма Евклида, описанном в первой главе. После всех подсчётов функция копирует значение $d$ в переданный в неё заранее BigInteger $d$. Фрагмент кода генерации $d$ представлен на рисунке 11.

```
void RSABigInteger::CalculateD(BigInteger& e, BigInteger& phi, BigInteger& d)
{
    int i =0;
    BigInteger temp1(phi.nSize),temp2(phi.nSize),quotient(phi.nSize),remainder(phi.nSize);
    BigInteger one(phi.nSize);
    one.digit[0] = 1;
    BigInteger k(phi.nSize);
    while(true)
    {
        i++;
        k.digit[0] = i;
        temp1.multBigInteger(k,phi);
        temp2.addBigInteger(temp1,one);
        divBigInteger(temp2,e,quotient,remainder);
        if(remainder.msbBigInteger() == 0 && remainder.digit[0] == 0)
        {
            d.copyBigInteger(quotient,0);
            break;
        }
    }
}
```

Рисунок 13 генерация d

После создания и генерации всех нужных переменных, программа генерирует случайную переменную RSABigInteger, состоящую из случайной последовательности цифр. Эта последовательность цифр затем будет зашифрована и расшифрована с применением алгоритма RSA. Для этого использовалась. Для этого вызываются функции encryption и decryption, которые используют функцию expoModNBigInteger для зашифровки и расшифровки сообщений. Показаны на рисунке 12 и 13.

```
void RSABigInteger::encryption(BigInteger& msg, BigInteger& code)
{
    BigInteger temp(SIZE);
    temp.expoModNBigInteger(msg, e, N, code);
}

void RSABigInteger::decryption(BigInteger& code, BigInteger& msg)
{
    BigInteger temp(SIZE);
    temp.expoModNBigInteger(code, d, N, msg);

}
```

Рисунок 14 функции encryption и decryption

```
void BigInteger::expoModNBigInteger(BigInteger& x, BigInteger& y, BigInteger& N, BigInteger& result)
{
    if (y.msbBigInteger() == 0 && y.digit[0] ==0)
    {
        result.digit[0] = 1;
        for (int i = 1; i < result.nSize; i++)
        {
            result.digit[i] = 0;
        }
    }
    else
    {
        BigInteger temp(nSize);
        BigInteger reminder(nSize);

        BigInteger value2(nSize);
        value2.digit[0]=2;
        divBigInteger(y,value2, *this, reminder);

        temp.copyBigInteger(*this, 0);
        expoModNBigInteger(x, temp,N, result);

        multBigInteger(result, result);

        temp.copyBigInteger(*this, 0);

        if (y.digit[0] % 2 != 0)
        {
            multBigInteger(temp, x);
            temp.copyBigInteger(*this, 0);
        }
        BigInteger q(nSize), r(nSize);
        divBigInteger(temp, N,q,r);
        result.copyBigInteger(r,0);
    }

    return;
}
```

Рисунок 15 функция expoModNBigInteger

В функцию expoModNBigInteger сообщается оригинальное сообщение, ключи и пустой BigInteger для записи в него результата. Также программа использует другие функции класса BigInteger, такую как divBigInteger. Фрагмент кода divBigInteger представлен на рисунке 14.

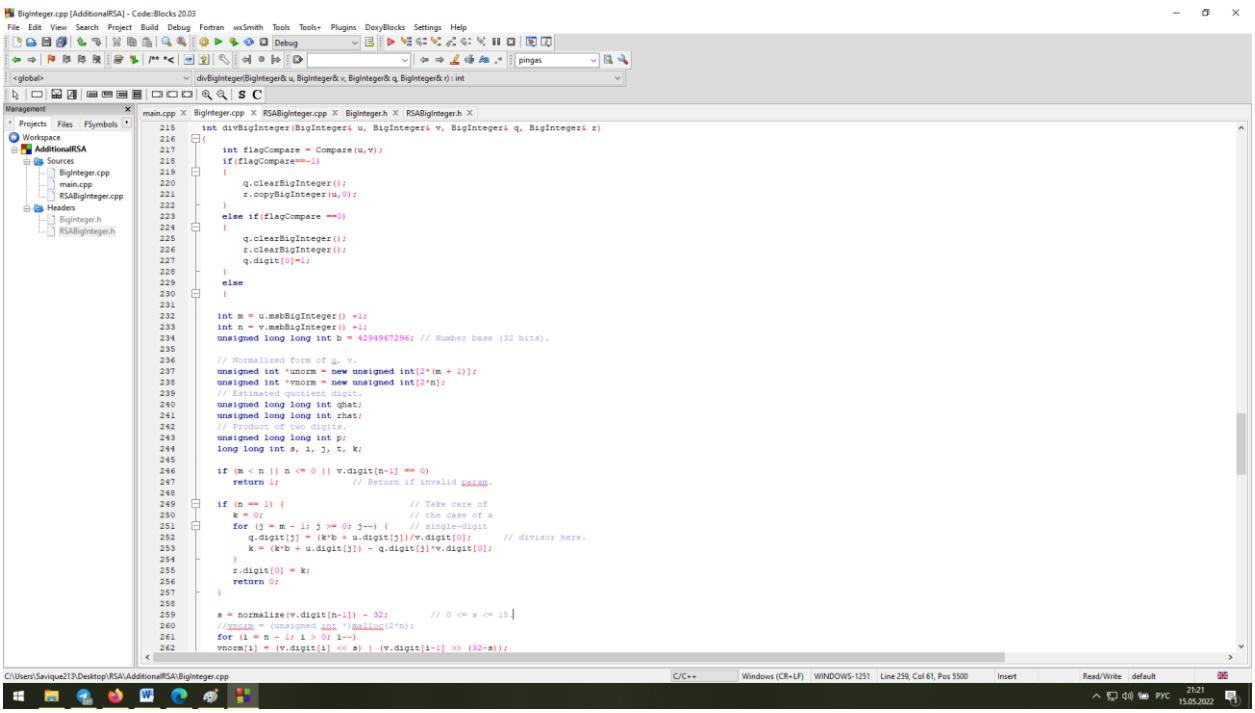

Рисунок 16 bigDivInteger

Выводы по второй главе:

Во второй главе выпускной квалификационной работы был реализован криптоалгоритм RSA на языке программирования C++. Также была построена блок-схема и разобран сам алгоритм и необходимые для его работы элементы.

Данный программный модуль может выполнять следующие шаги:

– Автоматическая генерация простых чисел;
– генерация открытого и закрытого ключей;
– зашифровка и расшифровка сообщений.

Данный алгоритм является учебным, поэтому расшифровка и зашифровка проходят в нём одновременно. Но основное тело алгоритма выполняет свои главные функции. Сам код при небольших изменениях можно встраивать в другие программные модули.

# Глава 3. Тестирование программной реализации

Для тестирования этой программы был выбран персональный компьютер со следующими характеристиками, представленными на рисунке 17.

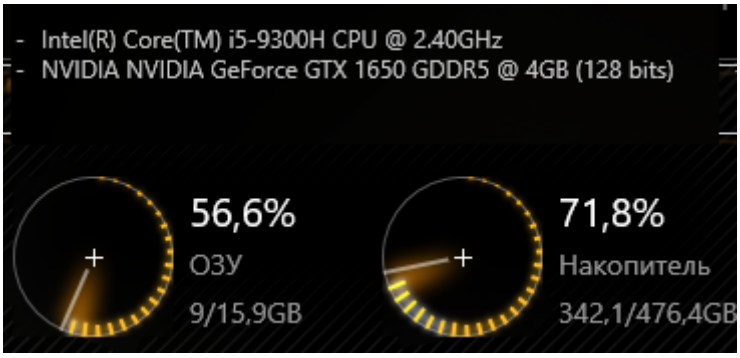


Рисунок 17 основные характеристики

Основное окно для работы с программой – системная консоль. Пользователю не нужно будет вводить какие-либо данные, поскольку написанная программа автоматически генерирует данные, простые числа и ключи. На рисунке 18 показано окно инициализации программы.

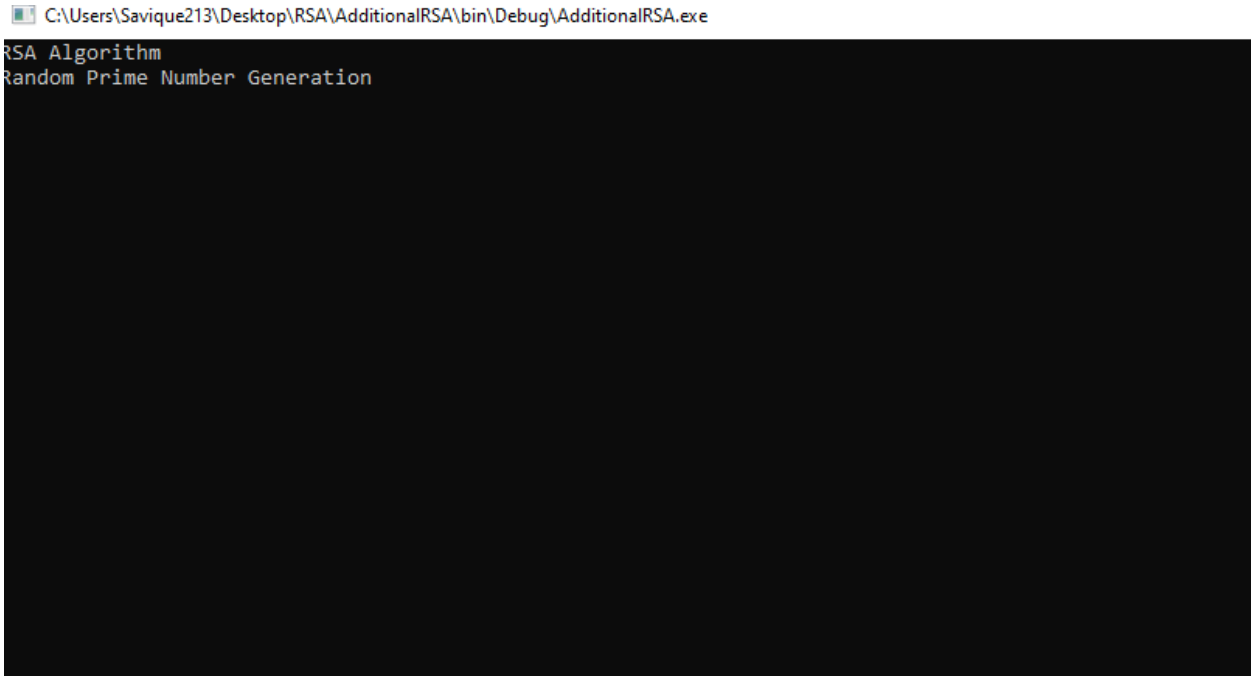


Рисунок 18 окно с инициализацией программы

Программа полностью рабочая и генерирует простые числа, ключи и текст для шифрования. (рисунок 19)

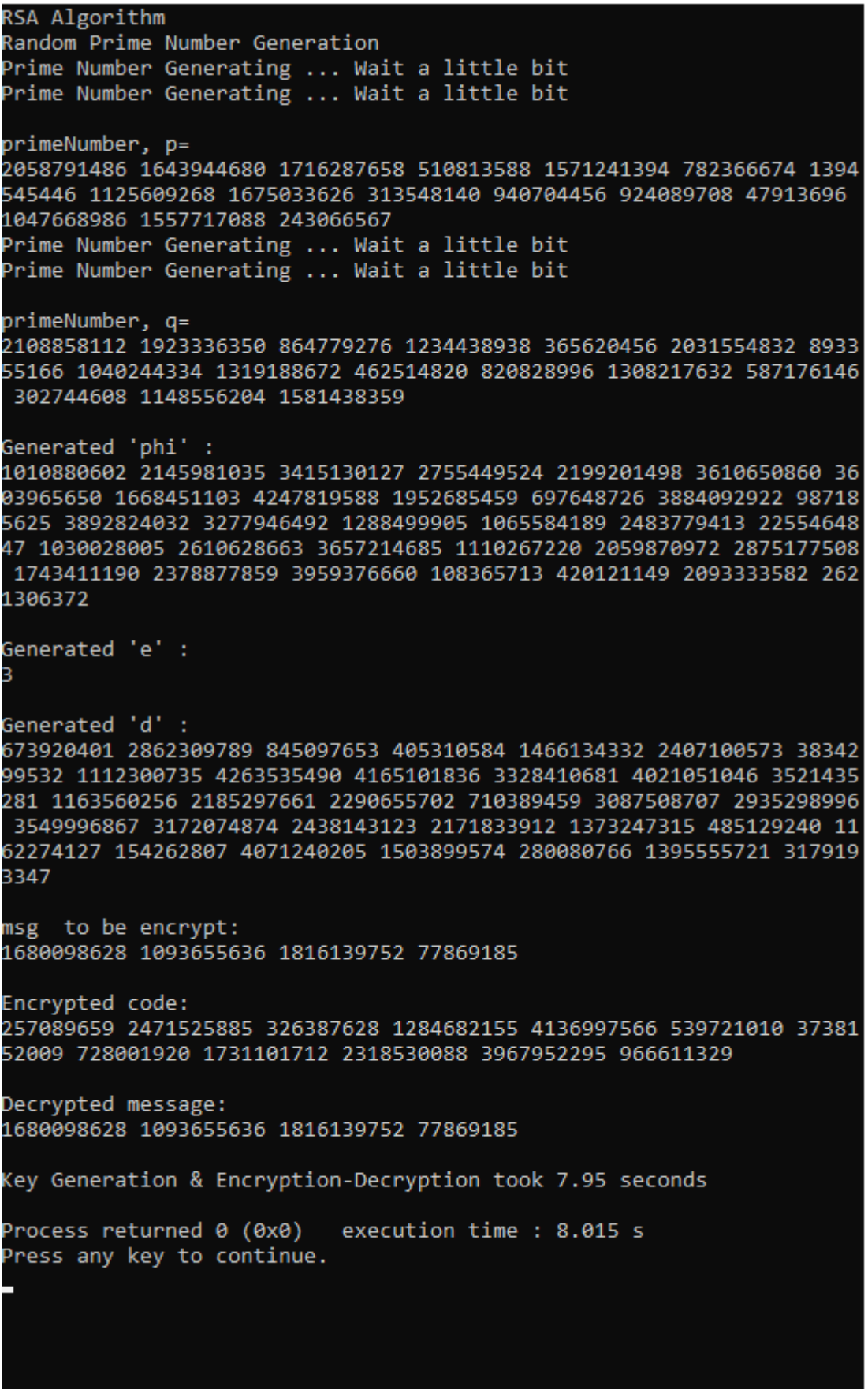


Рисунок 19 пример работы программы

Программа очень удобна для тестирования работы RSA алгоритма, поскольку она позволяет сравнивать время работы RSA алгоритма. Для теста программы возьмём текст размером в 1000, 2000, 3000, 5000 и 10000 случайных знаков и проверим скорость работы RSA.

Таблица 1 – время шифрования и расшифрования сообщения

| Количество знаков | Время, затраченное на зашифровку | Время, затраченное на расшифровку |
|---|---|---|
| 1000 | 0.01 секунд | 0.04 секунды |
| 2000 | 0.01 секунд | 0.07 секунд |
| 3000 | 0.01 секунд | 0.11 секунд |
| 5000 | 0.01 секунд | 0.17 секунд |
| 10000 | 0.01 секунд | 0.39 секунд |

Из полученных результатов можно сделать заключение, что чем больше знаков в тексте, то тем дольше времени требуется алгоритма на их зашифровку и расшифровку. К тому же можно заметить, что расшифрование занимает значительно больше времени, чем шифрование. Так же можно заметить, что время расшифровки увеличивается пропорционально количеству символов в линейной зависимости. Это очень хорошо можно просмотреть на графике на рисунке 20.

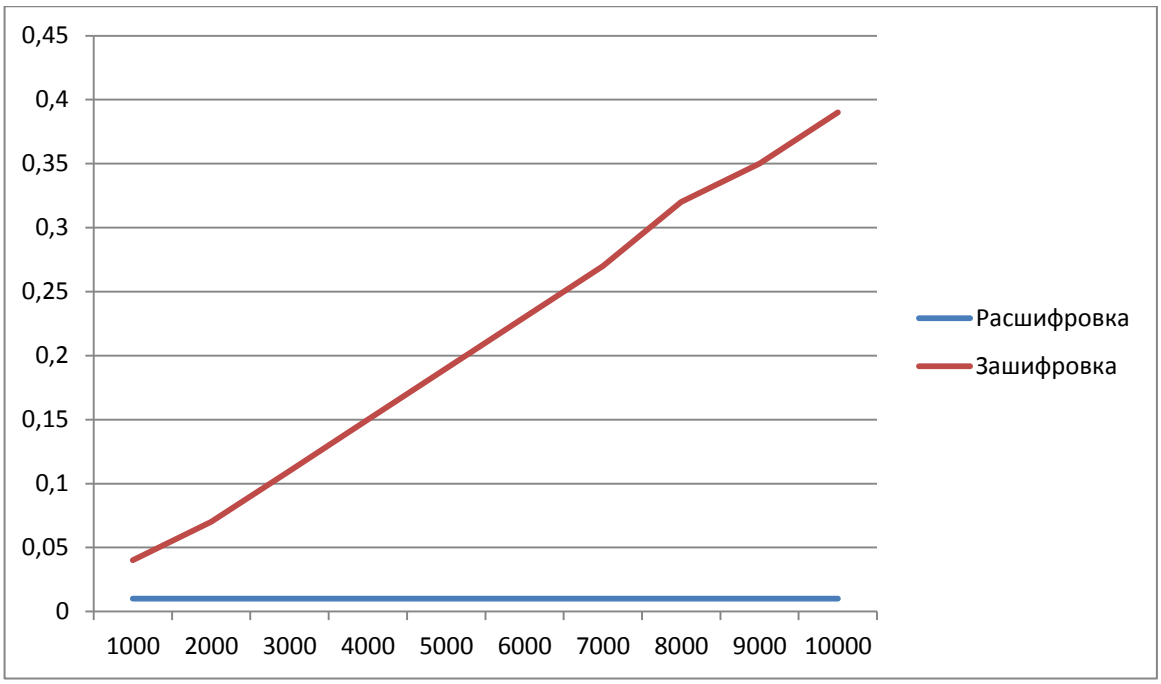


Рисунок 20 график зависимости времени от количества символов

Выводы по третьей главе:

В третьей главе выпускной квалификационной работы было проведено тестирование реализованного программного кода. Показана работа алгоритма и были проведены разные вычисления.

Также во время проведения тестирования было замечено, что количество времени, затраченного на шифрование и расшифрование, линейно зависит от количества символов, которые требуется зашифровать или расшифровать.

# Заключение

В ходе выполнения бакалаврской работы был рассмотрен криптографический алгоритм RSA, а также реализован программный код, который показывает работу RSA. Было разработано консольное приложение, с помощью которого можно отследить время работы алгоритма, время шифровки и расшифровки. Также приложение полностью автономное и отлично подходит для рассмотрения и оценки работы RSA. Были также рассмотрены все теоретические аспекты данного алгоритма, его сильные и слабые стороны, а также некоторые возможные атаки на алгоритм.

Также в работе была рассмотрена криптостойкость данного алгоритма. Самая большая опасность для алгоритма – атака путём факторизации, но при грамотном пользовании алгоритмом для реализации атаки потребуется много много ресурсов и времени. Остальные же атаки направлены на неправильную и слабую реализацию алгоритма.

Основная положительная сторона алгоритма – его ассиметричность. Алгоритм позволяет передавать данные с помощью открытых и закрытых ключей, и никто не сможет прочитать передаваемое сообщение, кроме владельца закрытого ключа. Ещё одна положительная сторона алгоритма – его криптостойкость. При соблюдении правил генерации ключей, алгоритм очень сложен для взлома.

Недостатком же данного алгоритма является его долгое время работы. При сравнении с другими алгоритмами, алгоритм RSA значительно уступает симметричным алгоритмам по времени зашифровки сообщений и не всегда подходит для шифрования очень больших сообщений. Тем не менее, с помощью RSA можно передать ключ для расшифровки сообщения.

Таким образом, в данной работе были изучены основные аспекты работы алгоритма RSA, выявлены его основные достоинства и недостатки и реализован программный код, который соответствует механизмам алгоритма RSA.

## Список используемой литературы

1. Алферов А. П.. Основы криптографии / А.П. Алферов, А.Ю. Зубов, А.С. Кузьмин, А.В. Черемушкин. – Москва, Гелиос АРВ, 2005. – 480с.

2. Бабаш А. В. Криптографические методы защиты информации. Криптографические методы защиты информации: Учебно-методическое пособие. Москва : ИЦ РИОР, НИЦ Инфра-М, 2013.

3. Баричев С. Г., Гончаров В. В., Серов Р. Е. Основы современной криптографии. Учебное пособие. Москва : Горячая Линия – Телеком, 2006 – 42с.

4. Виноградов И. М. Основы Теории Чисел. Москва : Гостехиздат, 1949. – 180с.

5. Жданов О. Н., Лубкин И. А. Алгоритм RSA. Методические указания к выполнению лабораторных работ. Красноярск : СибГАУ, 2007. – 38с.

6. Жданов О.Н., Золотарёв В.В. Методы и средства криптографической защиты. Красноярск : СибГАУ, 2007. – 217с.

7. Ишмухаметов Ш. Т., Рубцов Р. Г. Математические основы защиты информации: учеб. пособие. Казань : Казанский федер. Унт, 2012. -138с.

8. Кнут Д. Э. Искусство программирования. Том 1. Москва : Вильямс, 2017. – 720с.

9. Коутинхо С. А., Ландо С. К. Введение в теорию чисел. Алгоритм RSA. Перевод с англ. Москва : ПОСТМАРКЕТ, 2001.

10. Макаров А. С.. Теория и практика хакерских атак. Москва : МИК, 2015.

11. Мао В.. Современная криптография. Теория и практика. Москва : Вильямс, 2005. – 768с.

12. Ожиганов А. А. Криптографические системы с секретным ключом. Санкт-Петербург : МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ УНИВЕРСИТЕТ ИТМО, 2015 – 67с.

13. Орлов В. В., Алексеев А. П. Стенографические и криптографические методы защиты информации. Самара : 2010 – 288с.

14. Смарт Н. Криптография. Москва : Техносфера, 2005. - 528 с.

15. Шилдт Г. C++ Руководство для начинающих. Москва : Вильямс, 2005. – 672с.

16. Шнайер Б.. Прикладная криптография. Москва : Вильямс, 2015. – 1024с.

17. Ященко В. В., Черемушкин А.В. Введение в криптографию. 4 издание. Москва : Издательство МЦНМО, 1999 – 272с.

18. Katz J. Introduction to Modern Cryptography: Principles and Protocols (Chapman & Hall/CRC Cryptography and Network Security Series) 1st Edition, Kindle Edition. London : Chapman and Hall/CRC, 2007 – 552 с.

19. Rivest R. L., Shamir A., Adleman L.. RSA (Ron Rivest, Adi Shamir and Leonard Adleman) algorithm digital signature method//US4405829A, 1983. U.S. Patent 4,405,829.

20. Skobic V, Dokic B., Ivanovic Z.. Hardware Modules of the RSA// Serbian Journal of Electrical Engineering. 2014. Vol. 11, – 121с.

21. Tenzer T., Super Secreto - The Third Epoch of Cryptography: Multiple, exponential, quantum-secure and above all, simple and practical Encryption for Everyone, Kindle Edition. London : Chapman and Hall/CRC, 2022 – 481с.

22. Welsh D., Codes and Cryptography. London : Oxford University Press, 1988 – 272с.

Приложение А

**Листинг программы с использованием библиотеки**

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <exception>
#include <string>
#include <memory>
#include "BigInteger.h"
#include "RSABigInteger.h"
#include <ctime>

using std::cout;
using std::cerr;
using std::string;
using std::auto_ptr;

using namespace std;

#define SIZEN 96        //     word size bit = N*32
#define HALFSIZE 48      //

#define WORD_COUNT 390
#define SIZE 1024                  //N

int main(int argc,  char** argv)
{
```

```
    srand ( time(NULL) );

       int nSize;
    if(argc  < 2)
       nSize = SIZE;
    else
       nSize = atoi(argv[1]);

    nSize = nSize/32;

    cout<<"RSA Algorithm"<<endl;
    RSABigInteger rsa(WORD_COUNT);
    cout<<"Random Prime Number Generation"<<endl;

    struct timespec tstart={0,0}, tend={0,0};
    clock_gettime(CLOCK_MONOTONIC, &tstart);

    BigInteger                                    primeNumberP(WORD_COUNT),
primeNumberQ(WORD_COUNT);
    rsa.primeNumberGeneration(primeNumberP, nSize/2);
    cout<<"\nprimeNumber, p=\n";
    primeNumberP.showDigits();

    rsa.primeNumberGeneration(primeNumberQ, nSize/2);
    cout<<"\nprimeNumber, q=\n";
    primeNumberQ.showDigits();

    rsa.init(primeNumberP,primeNumberQ);
```

```
    BigInteger rslt(WORD_COUNT);

    BigInteger msg(WORD_COUNT);
    RSABigInteger operationCheck(WORD_COUNT);
    operationCheck.randomNGeneration(msg, 4);
    cout<<"\nmsg  to be encrypt: "<<endl;
    msg.showDigits();

    rsa.encryption(msg,rslt);
    cout<<"\nEncrypted code:\n";
    rslt.showDigits();

    rsa.decryption(rslt,msg);
    cout<<"\nDecrypted message:\n";
    msg.showDigits();

    clock_gettime(CLOCK_MONOTONIC, &tend);
    printf("\nKey Generation & Encryption-Decryption took %.2f seconds\n",
         ((double)tend.tv_sec + 1.0e-9*tend.tv_nsec) -
         ((double)tstart.tv_sec + 1.0e-9*tstart.tv_nsec));

    return 0;
}
```