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

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему Сравнительный анализ двух реализаций алгоритма шифрования RSA

Студент Д.А. Калашников _____

Руководитель Г.А. Тырыгина _____

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

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

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Калашников Дмитрий Александрович
1. Тема
Сравнительный анализ двух реализаций алгоритма шифрования RSA
2. Срок сдачи студентом законченной бакалаврской работы
24.06.2016
3. Исходные данные к бакалаврской работе
Алгоритм RSA
4. Содержание бакалаврской работы (перечень подлежащих разработке вопросов, разделов)

Введение
Глава 1. Теоретические аспекты алгоритма шифрования
1.1 Основные теоретико-вероятностные понятия
1.1.1 Сравнения
1.1.2 Алгоритм Евклида
1.1.3 Обратимость элементов в Z_i
1.2 Общие сведения об алгоритме
1.3 Криптосистема RSA
1.4 Как работает алгоритм RSA
1.5 Шифровка и дешифровка
1.6 Пример алгоритма RSA
1.7 Китайская теорема об остатках
1.8 Математическая модель алгоритма
1.9 Корректность криптосистемы
1.10 Цифровая подпись
1.11 Выбор простых чисел

1.11.1 Выбор d
1.11.2 Вычисление e из d и ф(n)
1.11.3 Выбор e и вычисление d из e и ф(n)
1.12 Криптоанализ RSA
1.12.1 Факторинг
1.12.2 Перебор делителей
1.12.3 Атака на секретную экспоненту малого размера
1.12.4 Атаки на передачу и связанные сообщения
1.12.5 Атаки на основе реализации
Глава 2. Реализация и тестирование алгоритма
2.1 Разработка алгоритма RSA на языке Java
2.2 Разработка алгоритма RSA на языке C++
2.3 Тестирование алгоритма
Глава 3. Сравнительный анализ реализаций
Заключение
Список используемой литературы

5. Ориентировочный перечень графического и иллюстративного материала Презентация, графики, рисунки

6. Дата выдачи задания «11» января 2016 г.

Руководитель бакалаврской работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ Д.А. Калашников

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

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Калашникова Дмитрия Александровича
по теме Сравнительный анализ двух реализаций алгоритма шифрования RSA

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Изучение теоретических основ	20.01.2016	20.01.2016	Выполнено	
Написание введения бакалаврской работы	01.02.2016	01.02.2016	Выполнено	
Изучение математических основ	25.02.2016	25.02.2016	Выполнено	
Написание 1 главы бакалаврской работы	23.03.2016	23.03.2016	Выполнено	
Написание кода программ	6.04.2016	6.04.2016	Выполнено	
Тестирование реализаций	8.04.2016	8.04.2016	Выполнено	
Написание 2 главы бакалаврской работы	15.04.2016	15.04.2016	Выполнено	
Проведение сравнительного	24.04.2016	24.04.2016	Выполнено	

анализа реализаций				
Написание 3 главы бакалаврской работы	30.04.2016	30.04.2016	Выполнено	
Написание заключения бакалаврской работы	8.05.2016	8.05.2016	Выполнено	
Подведение итогов, редактирование бакалаврской работы	9.05.2016	9.05.2016	Выполнено	
Представление бакалаврской работы	11.05.2016	11.05.2016	Выполнено	
Создание презентационного материала	11.05.2016	11.05.2016	Выполнено	
Предварительная защита	31.05.2016 - 14.06.2016	31.05.2016	Выполнено	
Проверка на наличие заимствований (плагиата) в системе antiplogiat.ru	17.06.2016	17.06.2016	Выполнено	
Сдача на кафедру отзыва научного руководителя и ознакомление с ним	20.06.2016	20.06.2016	Выполнено	
Сдача на кафедру комплекта документов для защиты	24.06.2016	24.06.2016	Выполнено	
Защита бакалаврской работы	27.06.2016 - 30.06.2016	29.06.2016	Выполнено	

Руководитель бакалаврской работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ Д.А. Калашников

Аннотация

Темой данной бакалаврской работы является «Сравнительный анализ двух реализаций алгоритма шифрования RSA».

Работы выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы ПМИб-1201, Калашниковым Дмитрием Александровичем.

Объект работы: алгоритм шифрования RSA.

Предмет исследования: сравнительный анализ двух реализаций алгоритма шифрования RSA на языках программирования C++ и Java.

Цель работы: реализовать алгоритм шифрования RSA на разных языках программирования, выявить его достоинства и недостатки.

Для достижения цели работы необходимо решить следующие задачи:

1) Изучить криптографический алгоритм.
2) Реализовать алгоритм RSA на языках C++ и Java.
3) Проверить работоспособность и корректность работы программ.
4) Выявить достоинства, каждой из реализаций RSA.

Отчет состоит из введения, трех глав и заключения.

В первой главе рассказывается о криптосистеме RSA, рассмотрены условия для ее реализации и основные методы, используемых в атаках против этой криптосистемы.

Во второй главе описана реализация алгоритма на языках Java и C++, и тестирование их работы.

В третьей главе производится сравнительный анализ реализаций.

Бакалаврская работа представлена на 42 страницах, включает 7 иллюстраций, 1 таблицу, 1 приложение, список используемой литературы содержит 21 источник.

Оглавление

Введение

Шифрование играет важную роль в функционировании нашего общества. Например, миллионы людей каждый день делают покупки в Интернете. Каждый раз, когда передается информация о кредитной карте онлайн, существует риск того, что эта информация может быть украдена. Так как может информация передаваться надежно?

Информация о кредитной карте покупателей должна быть зашифрована, прежде чем она будет передаваться по сети Интернет, и поэтому метод шифрования должен быть обнародован. Но метод дешифрования должен быть известен только банку, который обрабатывает платежи.

Шифрование представляет собой преобразование данных в форму, которую практически невозможно прочитать без соответствующих знаний (ключа). Ее цель состоит в том, чтобы обеспечить конфиденциальность, сохранность информации, то есть скрыть от тех, для которых она не предназначена, и от тех, кто не имеет доступ к зашифрованным данным. Дешифрирование является обратной к шифрованию; это преобразование зашифрованных данных обратно в понятную форму.

Шифрование и дешифрование как правило, требуют использования какой-то секретной информации, упомянутой в качестве ключа. В некоторых криптографических алгоритмах, дешифрование просто противоположность шифрованию; эти системы используют симметричный ключ, так как этот ключ используется как для шифрования, так и для дешифрования. Другие ассиметричные алгоритмы, у них два различных ключа: открытый и закрытый ключи. Закрытый (секретный) ключ является ключом дешифрования, а открытый ключ является ключом шифрования.

Криптография делает веб-сайты и электронную почту безопасными. Для этого, передаваемые между компьютерами данные, где информация о том, где они хранятся и кем будут получены, должны быть зашифрованы. Это позволяет людям проводить коммерческие сделки, не опасаясь, что какая-либо

информация находится под угрозой. Криптография очень важна для дальнейшего развития Интернета и электронной коммерции.

Актуальность:

RSA встраивается во многие коммерческие продукты, число которых постоянно увеличивается. Также ее используют операционные системы Microsoft, Apple, Sun и Novell. В аппаратном исполнении RSA алгоритм применяется в защищенных телефонах, на сетевых платах Ethernet, на смарт-картах, широко используется в криптографическом оборудовании. Кроме того, алгоритм входит в состав всех основных протоколов для защищенных коммуникаций Internet, в том числе S/MIME, SSL и S/WAN, а также используется во многих учреждениях, например, в правительственных службах, в большинстве корпораций, в государственных лабораториях и университетах.

Объект работы: алгоритм шифрования RSA.

Предмет исследования: сравнительный анализ двух реализаций алгоритма шифрования RSA на языках программирования C++ и Java.

Цель работы: реализовать алгоритм шифрования RSA на разных языках программирования, выявить его достоинства и недостатки.

Для достижения цели работы необходимо решить следующие задачи:

1) Изучить криптографический алгоритм.
2) Реализовать алгоритм RSA на языках C++ и Java.
3) Проверить работоспособность и корректность работы программ.
4) Выявить достоинства, каждой из реализаций RSA.

Отчет состоит из введения, трех глав и заключения.

В первой главе рассказывается о криптосистеме RSA, рассмотрены условия для ее реализации и основные методы, используемых в атаках против этой криптосистемы.

Во второй главе описана реализация алгоритма на языках Java и C++, и тестирование их работы.

В третьей главе производится сравнительный анализ реализаций.

Глава 1. Теоретические аспекты алгоритма шифрования

1.1 Основные теоретико-вероятностные понятия

1.1.1 Сравнения

Пусть a и b — целые числа. a сравнимо с b по модулю n, если при делении a и b на модуль n, получаются одинаковые остатки. И записывается как:

$$a \equiv b \text{ (по модулю } n\text{)},$$

другими словами, если a-b делится на n.

Отношение сравнения по модулю натурального числа обладает следующими свойствами:

1. Рефлексивность: $a \equiv a$ (по модулю n).

2. Симметричность: $a \equiv b$ (по модулю n) $\rightarrow b \equiv a$ (по модулю n).

3. Транзитивность: $a \equiv b$ (по модулю n) и $b \equiv c$ (по модулю n) $\rightarrow a \equiv c$ (по модулю n).

Все целые числа, которые при делении на n дают один и тот же остаток r, объединяют в один класс Z_m. Очевидно, что при делении на n возможны n остатков:

$$0,1, 2, \ldots, (n\text{-}1).$$

Следовательно, все целые числа можно разбить на n классов:

$$Z_0, Z_1, Z_2, \ldots, Z_{n-1},$$

которые называются классами вычетов по модулю n. Каждый класс Z_m содержит бесконечное количество сравнимых между собой целых чисел, объединенных одним соотношением

$$N = nq + r;\ r < n;\ q = -\infty, \ldots, -1, 0, 1, 2, \ldots, \infty.$$

Над вычетами можно выполнять арифметические операции сложения, вычитания, умножения и возведения в степень, а если число n простое или является некоторой степенью простого числа, то и деление.

Для любых a и b выполняется формула $a + b = a + b$, поэтому операции над вычетами выполняются как над обычными числами, приводя результат к значению, принадлежащему интервалу $[0, n-1]$ путем вычисления остатка от

деления на число n [1, 4]. Так, например, в множестве Z_7 $2 \cdot 5 = 10 = 3$(по модулю 7). Чаще пишут просто: $2 \cdot 5 = 3$(по модулю 7).

1.1.2 Алгоритм Евклида

Алгоритм Евклида является простым способом нахождения наибольшего общего делителя двух положительных целых числа a и b (НОД (a, b)), то есть наибольшее целое число, на которое делится оба числа a и b. Этот алгоритм вычисления НОД основан на двух свойствах НОД. Если а наибольшее из двух чисел:

$$\text{НОД}(a, b) = \text{НОД}(a \text{ по модулю } b, b),$$

$$\text{НОД}(a, 0) = a.$$

Второе свойство очевидно из определения. А первое доказывается следующим образом, обозначим c = НОД (a, b), а d = НОД (b, r). Тогда с делится на a и b, поэтому c делится на r. Поэтому, $c \leq d$. Аналогично рассуждая, получим, что: d делится на b и r, поэтому d делится на a. И поэтом, $d \leq c$. И получается, что $c = d$. Если НОД (a, b) = 1, то числа a и b называются взаимно простыми.

Шаги работы алгоритма, для нахождения НОД (a, b):

Шаг 1. Выбрать два натуральных числа a и b.

Шаг 2. Если b = 0, то нужно остановиться, результатом будет значение a, иначе перейти к следующему шагу.

Шаг 3. Заменить значение a на значение b, а значение b остатком от деления a на b и перейти к шагу 2 [1, 3].

1.1.3 Обратимость элементов в Z_i

Определение. Элемент $c \in Z_i$ называется обратимым по умножению, если найдется элемент $d \in Z_i$, такой что

$$c \cdot d \equiv 1 \text{ (по модулю } m).$$

Если для элементов c и d выполняется равенство выше, то d называют обратным элементом по отношению к c и обозначают через c^{-1}.

Итак, для обратимого элемента c выполняется равенство

$$c \cdot c^{-1} \equiv 1 \text{ (по модулю } m).$$

Теорема. Обратимыми по умножению являются те и только те элементы из Z_i, которые взаимно просты с модулем m. Для каждого обратимого элемента $c \in Z_i$ существует только один обратный элемент.

Доказательство. Пусть $c \in Z_i$ и $(c, m) = r > 1$. Тогда ни для какого $d \in Z_i$ сравнение

$$c \cdot d \equiv 1 \text{ (по модулю } m)$$

не может выполняться. Действительно, в случае выполнения этого сравнения m делит $(c \cdot d - 1)$ и r делит как $(c \cdot d - 1)$, так и c, а, следовательно, r делит 1, что невозможно.

Обратно, если $(c, m) = 1$, то, используя алгоритм Евклида, найдем целые числа k и l такие, что (c, m) = $1 = kc + lm$. Отсюда получаем, что m делит $(1 - k \cdot c)$. Следовательно, справедливо соотношение

$$c \cdot k \equiv 1 \text{ (по модулю } m)$$

и, если $k \in Z_i$, то $k = c^{-1}$. Если целое число $k \notin Z_i$, то существует число $\tilde{k} \in Z_i$ такое, что $\tilde{k} \equiv k$ (по модулю m). Следовательно, для некоторого числа $s \in Z$ выполняется равенство

$$k = \tilde{k} + s \cdot m.$$

Поэтому, ввиду

$$c \cdot k \equiv 1 \text{ (по модулю } m),$$

$$c \cdot \tilde{k} \equiv 1 \text{ (по модулю } m) \text{ и } \tilde{k} = c^{-1}.$$

Единственность обратного элемента будем доказывать от противного. Пусть для обратимого элемента $c \in Z_i$ существует два различных обратных элемента d_1 и d_2 из Z_i:

$$c \cdot d_1 \equiv 1 \text{ (по модулю } m),$$

$$c \cdot d_2 \equiv 1 \text{ (по модулю } m).$$

Тогда по свойствам сравнений

$$c \cdot (d_1 - d_2) \equiv 0 \text{ (по модулю } m),$$

или m делит $c \cdot (d_1 - d_2)$. Однако это невозможно [3], поскольку $(c, m) = 1$ и $|d_1 - d_2| < m$.

1.2 Общие сведения об алгоритме

Криптография с открытым ключом состоит из криптосистемы, которая не нуждается в защищенном канале (способе) для обмена любой важной информации, как например передача секретного ключа, которая применяется в криптосистемах с закрытым ключом. Распространенной аналогия для объяснения работы криптографии с открытым ключом является простой почтовый ящик. Например, каждый может положить письмо в почтовый ящик Алисы, но только Алиса, у которой есть ключ для ее почтового ящика, сможет открыть его и прочитать свои письма. То же самое верно для Боба и его почтового ящика. В техническом смысле, это может быть описано следующим образом: Алиса создает пару открытый и закрытый ключи. Затем она показывает ее открытый ключ, что позволяет любому зашифровать сообщения с этим ключом и отправить их к ней. При получении их, она расшифровывает их с помощью своего закрытого ключа. И если Боб тоже хочет получать зашифрованные сообщения, то он должен также создать пару открытый / закрытый ключей и следовать процедуре Алисы. Таким образом, нет необходимости в безопасном канале для передачи ключа, так как нет никаких общих ключей, которые должны быть изменены между несколькими пользователями [8].

Эта процедура требует, чтобы (открытый) ключ для шифрования e был разработан таким образом, чтобы (закрытый) ключ для дешифрования d было крайне сложно определить, при знании e. Она также требуют проверки подлинности самого замка, для защиты от так называемой атаке посредника [12].

1.3 Криптосистема RSA

Наиболее известным шифром с открытым ключом является криптосистема RSA. Она изобретена в 1977 году Роном Ривестом, Ади Шамиром и Леном Адлеманом.

Алгоритм основан на использовании того факта, что задача факторизации является трудной, т.е. легко перемножить два числа, в то время как не

существует полиномиального алгоритма нахождения простых сомножителей большого числа [14].

Он широко используется для обеспечения безопасности связи в сети Интернет, для обеспечения конфиденциальности и подлинности электронной почты, и это стало основой для электронной коммерции. RSA, как правило, присутствует там, где необходима безопасность цифровых данных.

Основная математическая структура функции RSA достаточно проста, и это может быть еще одной причиной его популярности; люди чувствуют себя более комфортно при работе с алгоритмом, который можно легко понять. Он основан на основных алгебраических операциях с большими целыми числами. Перед описанием алгоритма нам нужно установить некоторые условности:

- если a, b и n — целые положительные числа, можно сказать, что a равно b по модулю n (обозначается как $a \equiv b$ по модулю n), если b является остатком от деления на n;
- мы обозначим через Z_n множество целых чисел по модулю n. Этот набор может быть представлен всеми неотрицательными целыми числами, меньших, чем n. Мы можем определить операции сложения и умножения в этом множестве, используя обычные операции с целыми числами, но взяв результат формулы по модулю n, как это определено выше. Они называются модульное сложение и модульное умножение.
- мы обозначим через НОД (a, b) наибольший общий делитель a и b.

Теперь мы можем описать алгоритм RSA:

1) Сгенерировать два различных больших простых чисел p и q, каждый из которых примерно одинакового размера.
2) Вычислить $n = pq$ и $\varphi(n) = (p\text{-}1)\,(q\text{-}1)$.
3) Выбрать целое число e при $1 < e < \varphi(n)$ и НОД (e, $\varphi(n)$) = 1.
4) Вычислить d таким образом, чтобы $de \equiv 1$ по модулю $\varphi(n)$.

Пара целых чисел (e, n) является открытым ключом. Пара (d, n) является закрытым ключом. Целое число n является модулем. Мы будем называть e и d

открытая и закрытая экспоненты, соответственно. Также условно можно называть битовую длину модуля n размером ключа RSA.

Как правило, модуль n длиной в 1024-бит (каждое простое длиной в 512-бит) и открытый показатель степени относительно небольшое целое (3 и 65537 обычно используемые значения). В этом случае закрытая экспонента d будет иметь примерно такой же размер, как n.

Теперь у нас есть:

Шифрование:

- представить сообщение, которое будет зашифровано как целое число $M \in Z_n$;
- зашифровать M, как

$$C \equiv M^e \text{ модулю } n;$$

- в результате шифротекст C может быть расшифрован с помощью вычисления

$$D \equiv C^d \text{ по модулю } n.$$

Это следует из

$$de \equiv 1 \text{ по модулю } \varphi(n),$$

что

$$D = M.$$

RSA также используется для создания цифровых подписей, которые могут обеспечить подлинность и безотказность электронных правовых документов.

Подписание:

• представляет собой сообщение, которое должно быть подписано как целое число $M \in Z_n$;

Это обычно делается сначала путем вычисления хэш сообщения (представив его в виде набора бит).

• M может быть подписан путем вычисления

$$S \equiv M^d \text{ по модулю } n;$$

• подпись может быть проверена путем проверки, если

$$M \equiv S^e \text{ по модулю } n.$$

На практике сообщения кодируются перед шифрованием и подписанием. Использование функции RSA без предварительного кодирования не удовлетворяет основным определениям безопасности и считается небезопасным. Кроме того, тот же самый секретный ключ не должен использоваться для расшифровки и подписи сообщения одновременно. Пользователи должны иметь разные ключи для шифрования и подписи.

Схема шифрования RSA является открытым ключом криптосистемы, а функция лазейка RSA определяется как

$$f(x) = x^e \text{ по модулю } n.$$

Лазейкой является частный показатель d, так как

$$(x^e)^d \equiv x \text{ по модулю } n.$$

Проблема в том, как взломать функцию RSA без знания закрытой экспоненты d называется, как проблема RSA.

Факторизация модуля является разрушительным для системы. Если злоумышленник сможет факторизовать n, он сможет легко вычислить закрытую экспоненту путем решения конгруэнтности

$$ed \equiv 1 \text{ по модулю } \varphi(n),$$

и, таким образом, взломать функцию RSA.

Следует отметить один интересный момент, что нахождение закрытой экспоненты d эквивалентно факторизации модуля n. Это означает, что злоумышленник, который знает, (e, d, n) сможет легко факторизовать n. Это иллюстрирует возможные недостатки криптосистемы RSA. Чтобы избежать возникновения новых простых чисел для каждого пользователя, считалось, что отправитель зашифрованного сообщения может сгенерировать общий модуль и различные пары экспонент (e_i, d_i) для каждого пользователя. Хотя это может показаться на первый взгляд хорошей идей, факт выше свидетельствует о том, что это совершенно небезопасно: любой пользователь, зная свою собственную пару ключей, может факторизовать n, а затем найти все оставшиеся секретные ключи. Это показывает, что модуль RSA никогда не должен использоваться более чем для одного объекта.

Обратим внимание, что факт выше не доказывает эквивалентность между задачей RSA и целочисленной задачей факторизации. Проблемой RSA является, учитывая (e, n) и C, нахождение M такого, что

$$C \equiv M^e \text{ по модулю } n$$

(т.е. вычислить e-ые корни по модулю n). нахождение секретного ключа d является гораздо более амбициозной целью, и выше, мы видим, что это так же сложно, как факторизация целого числа n. В настоящее время не известно, что проблемы RSA и целочисленная факторизация целые решаются одинаково. На самом деле, есть доказательства того, что, по крайней мере для малых открытых экспонент, взлом RSA может быть проще, чем факторинг. Это не дает никаких признаков уязвимости в системе RSA, хотя; это только показывает, что проблемы не могут быть эквивалентными (проблема RSA по-прежнему, вероятно, будет трудной задачей) [7, 9].

1.4 Как работает алгоритм RSA

Сначала получатель формирует n, как произведение двух больших чисел p и q, выбранных случайным образом, но так, чтобы p и q не могли быть легко найдены из n. В этом случае получатель выбирает случайное целое число e между

$$1 \text{ и } \varphi(n) = (p-1)\,(q-1),$$

таким образом, чтобы оно было взаимно простым с $\varphi(n)$ и, используя алгоритм деления Евклида, вычислить

$$d = e^{-1} \text{ в } Z\varphi(n).$$

Числа n и e делаются общедоступными, а d, p, q держатся в секрете.

Любой, кто захочет послать сообщение m, где

$$0 \leq M < n,$$

получателю шифрует сообщение с помощью функции кодирования

$$C = E_e\,(M) = M^e \text{ (по модулю } n)$$

и передает c. Поскольку получатель имеет сведения о d, получатель может расшифровать с, используя функцию декодирования

$$M = D_e\,(C) = C^d \text{ (по модулю } n) \text{ [18].}$$

Общая схема выглядит следующим образом:

1. Каждый пользователь генерирует пару ключей: один для шифрования и один для дешифрования.

2. Каждый пользователь публикует свой ключ шифрования, размещает его в открытом для всех доступе. Второй ключ, соответствующий открытому, сохраняется в секрете.

3. Если пользователь A собирается послать сообщение пользователю B, он шифрует сообщение открытым ключом пользователя B.

4. 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 [6].

1.5 Шифровка и дешифровка

Для шифрования сообщения M, с использованием открытого ключа шифрования (e, n), нужно делать следующее. (e и n пара положительных целых чисел.)

Во-первых, представить сообщение как целое число от 0 до n-1. (Поделив сообщение на блоки, и представить каждый блок в качестве целого числа.) С помощью любого стандартного представления. Цель здесь не в шифровании сообщения, а только, в получении его цифровой формы, необходимой для шифрования.

Затем, зашифровать сообщение, возведя его в e-ую степень по модулю n. То есть, результат (шифротекст C) остаток от деления M^e на n.

Для расшифровки шифротекста, необходимо возвести его в степень d, опять же по модулю n. Алгоритмы шифрования и дешифрования E и D, выглядят следующим образом:

$$C \equiv E(M) \equiv M^e \text{ (по модулю } n\text{), для сообщения } M.$$

$$D(C) \equiv C^d \text{ (по модулю } n\text{), для шифротекста } C.$$

Обратим внимание, что шифрование не приводит к увеличению размера сообщения; и сообщение, и зашифрованный текст являются целыми числами в диапазоне от 0 до n-1.

Ключ шифрования — это пара натуральных чисел (e, n). Аналогичным образом, выглядит ключ дешифрования — пара натуральных чисел (d, n). Каждый пользователь делает свой ключ шифрования открытым (общедоступным), и сохраняет соответствующий ключ дешифрования в секрете. (Эти числа должны быть должным образом проиндексированы, например, как n_A, e_A, и d_A, так как каждый пользователь имеет свой собственный набор. Тем не менее, мы будем рассматривать только частный случай, и будем опускать нижние индексы.)

Для выбора ключей шифрования и дешифрования нужно использовать следующее.

Во-первых, вычислить n как произведение двух простых чисел p и q:

$$n = p \cdot q.$$

Эти простые числа очень большие, "случайные" простые числа. Несмотря на то, что n будет общедоступным, факторы p и q будут скрыты от всех из-за огромной сложности факторизации n. Это также скрывает нахождение d из e.

Затем выбирается целое число d, чтобы оно было большим, случайным целым числом, взаимно простым с

$$(p\text{-}1) \cdot (q\text{-}1).$$

То есть, проверить, что d удовлетворяет:

$$\text{НОД}\ (d, (p\text{-}1) \cdot (q\text{-}1)) = 1.$$

Целое число e наконец вычисляется из p, q, и d оно должно быть "мультипликативным обратным" d, по модулю

$$(p\text{-}1) \cdot (q\text{-}1).$$

Таким образом, мы имеем [2, 5]

$$e \cdot d \equiv 1 \text{ (по модулю } (p\text{-}1) \cdot (q\text{-}1)).$$

1.6 Пример алгоритма RSA

Предположим, что секретное сообщение подлежащие кодированию - *HI*. Сначала преобразуем его в десятичное число:

“HI" = 08, 09

Где 'H' и 'I' являются 8ой и 9ой буквами алфавита, если A = 1, B = 2, и

Z = 26.

1. Это сообщение будет кодироваться как $m = 0809$.

2. Затем, предполагаемый получатель сообщения создает свой открытый ключ. Он или она делают это, выбирая два простых числа и вычисляя их произведение. Для этого, например, мы выбираем два различных двузначных простых числа, (23 и 43) – при использовании 200-значных простых чисел в фактической реализации было бы трудно показать на бумаге:

$$23 \times 43 = 989$$

В общих обозначениях, $p = 23$, $q = 43$, and $n = 989$.

3. Далее, получатель выбирает значение: e; такое что,

$$\text{НОД}\ (e, (p - 1)\ (q - 1)) = 1:$$

$$\text{НОД}\ (e, 924) = 1, \text{например}, e = 5.$$

4. Вычислить d, так, чтобы

$$de \equiv 1 \bmod ((p - 1)\ (q - 1)):$$

Эта обратная e может показать, что число 185, как известно, существует благодаря тому, как e была выбрана, поэтому:

$$185 \times 5 \equiv 1 \text{ (по модулю 924)} \quad d = 185$$

Сейчас существуют следующие значения, указанные в таблице 1.1:

Таблица 1.1 — Значения необходимые для работы алгоритма RSA

p	q	n	e	d
23	43	989	5	185

5. Получатель сообщения теперь публикует значения n и e в качестве упорядоченной пары (n, e), сохраняя при этом p, q, d в секрете. Эта идея представляет собой схожесть с открытым ключом: эти значения опубликованного открытого ключа указывают на то, что будущий отправитель сообщения сможет получить доступ к зашифрованным данным для отправки, в данном случае, пара (n, e) делается владельцем открытого ключа.

6. Для шифрования отправитель сообщения вычисляет:

$$c \equiv m^e \text{ по модулю n}$$

где c зашифрованное сообщение:

$$127 \equiv 0809^5 \text{ (по модулю 989)}$$

Зашифрованный текст в данном примере - 127. Теперь это отправляется получателю сообщения.

7. Сообщение расшифровано получателем при помощи формулы:

$$m \equiv c^d \text{ по модулю } n$$

$$m \equiv 127^{185} \text{ (по модулю 989)} \equiv 809.$$

Так как с 80 выходит из символьного пространства, получается, что сообщение не может быть`80', `9', поэтому оно должно быть дополнено ведущим 0, чтобы стать: 08,09. Итак, расшифрованный текст – ‘HI’ [15].

1.7 Китайская теорема об остатках

Пусть m_1 и m_2 - два положительных целых числа. Принимая во внимание любые два целых числа a и b, существует целое число x такое что

$$x \equiv a \text{ (по модулю } m_1),$$

$$x \equiv b \text{ (по модулю } m_2).$$

Кроме того, любые два решения этих уравнений совпадают друг с другом по модулю m_1m_2.

Доказательство

Рассмотрим числа

$$a, a + m_1, a + 2m_1, \ldots, a + (m_2 - 1)\, m_1.$$

Каждое из этих чисел совпадает с a по модулю m1.

Предположим, что два этих числа равны друг другу по модулю m_2. Покажем, что это приводит к противоречию. Пусть два равных числа

$$a + im_1 \text{ и } a + jm_1,$$

при

$$0 \le i < j \le m_2 - 1,$$

так что

$$a + jm_1 \equiv a + im_1 \text{ (по модулю } m_2).$$

Из этого следует, что

$$jm_1 \equiv im_1 \text{ (по модулю } m_2).$$

Так как m_1 и m_2 взаимно просты, то по теореме аннулирования

$$j \equiv i \text{ (по модулю } m_2).$$

Так m_2 делится на j - i. Но это невозможно, так как

$$0 \le i < j \le m_2 - 1.$$

Мы показали, что никакие из двух чисел из списка

$$a, a + m_1, a + 2m_1, ..., a + (m_2 - 1) m_1.$$

не совпадают друг с другом по модулю m_2. То есть, эти числа дают различные остатки при делении на m_2. Есть m_2 чисел в списке, и есть m_2 возможных остатков. Таким образом, мы должны получить все возможные остатки. В частности, существует целое число k при

$$0 \le k \le m_2 - 1,$$

такое что

$$a + km_1 \equiv b \text{ (по модулю } m_2).$$

Теперь мы можем полагать, что

$$x \equiv a + km_1,$$

и легко увидеть, что x удовлетворяет двум обязательным уравнениям.

Для того чтобы доказать окончательное утверждение теоремы, предположим, что оба x_1 и x_2 являются решениями уравнений. Пусть

$$z = x_1 - x_2.$$

Тогда

$$z \equiv 0 \text{ (по модулю } m_1)$$

и

$$z \equiv 0 \text{ (по модулю } m_2).$$

Поэтому, z делится на оба m_1 и m_2. Так как m_1 и m_2 взаимно просты, то отсюда следует, что z делится на m_1m_2. Следовательно [11],

$$x_1 \equiv x_2 \text{ (по модулю } m_1m_2).$$

1.8 Математическая модель алгоритма

Давайте представим *M* целым числом между 0 и *n* - 1. Если сообщение слишком длинное, то разделим его на блоки и зашифруем их по-отдельности. Пусть *e, d, n* положительные целые числа, с (*e*, *n*) в качестве ключа шифрования, и (*d, n*) ключа дешифрования,

$$n = pq.$$

Теперь мы зашифруем сообщение, возведя его в e-ую степень по модулю n, чтобы получить зашифрованное сообщение C. Затем мы расшифруем C возведя его в d-ую степень по модулю n, тем самым получим исходное сообщение M:

$$C \equiv E(M) \equiv M^e \text{ (по модулю } n)$$
$$M \equiv D(C) \equiv C^d \text{ (по модулю } n).$$

Заметим, что сохраняется тот же размер информации. Также простота заключается в том, что ключи шифрования и дешифрования, находятся в паре чисел (e, n) и (d, n), соответственно. И каждый пользователь имеет свою уникальную пару.

Выбирая два случайных больших числа p и q, мы перемножаем их и получаем $n = pq$. Несмотря на то, что n является публичным, он не будет раскрывать значения p и q, так как это практически невозможно, учитывая форму n, тем самым делает практически невозможным получить d из e.

Теперь мы хотим получить соответствующие e и d. Мы выбираем d, так чтобы оно была случайным большим числом и взаимно простым с

$$(p- 1) \cdot (q - 1),$$

это означает, что следующее уравнение должно выполняться:

$$\text{НОД } (d, (p - 1) \cdot (q - 1)) = 1.$$

Мы хотим вычислить e из d, p, и q, где e мультипликативная обратная d. Это означает, что нам нужно, чтобы выполнялось следующие выражение

$$e \cdot d = 1 \text{ (по модулю } \varphi(n)).$$

Здесь мы выводим функцию Эйлера $\varphi(n)$, которое является положительным целым числом, меньшим чем n, и которое взаимно простое с n. Для простых чисел p, это, например,

$$\varphi(p) = p - 1.$$

Для n, получаем, в силу элементарных свойств функции Эйлера, что

$$\varphi(n) = \varphi(p) \cdot \varphi(q)$$
$$= (p - 1) \cdot (q - 1)$$

$$= n - (p + q) + 1.$$

Из этого уравнения мы получаем $\varphi(n)$, которое подставляя в

$$e \cdot d = 1 \text{ (по модулю } \varphi(n))$$

получим

$$e \cdot d \equiv 1 \text{ (по модулю } \varphi(n))$$

что эквивалентно

$$e \cdot d = k \cdot \varphi(n) + 1$$

для некоторого целого числа k.

По закону модулярной арифметике, мультипликативная обратная a по модулю m существует тогда и только тогда, когда a и m взаимно просты. Действительно, так как d и $\varphi(n)$ взаимно просты, d имеет мультипликативную обратную e в кольце целых чисел по модулю $\varphi(n)$.

Теперь мы можем смело уверять в следующем:

$$D(E(M)) \equiv (E(M))^d \equiv (M^e)^d \text{ (по модулю } n) = M^{e \cdot d} \text{ (по модулю } n)$$

$$E(D(M)) \equiv (D(M))^e \equiv (M^d)^e \text{ (по модулю } n) = M^{e \cdot d} \text{ (по модулю } n)$$

Кроме того, так как

$$e \cdot d = k \cdot \varphi(n) + 1,$$

мы можем подставить это в приведенные выше уравнения и получить

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \text{ (по модулю } n).$$

Понятно, что мы хотим, чтобы это равнялось M. Чтобы доказать это, нужно будет использовать тождество из Эйлера и Ферма: для любого целого M взаимно простым с n, мы получаем

$$M^{\varphi(n)} \equiv 1 \text{ (по модулю } n).$$

Поскольку ране мы указали, что $0 \leq M < n$, мы знаем, что M не будет взаимно простым с n тогда и только тогда, когда M будет одним из p и q, из целых чисел в этом интервале. Таким образом, шансы на то, что M станет p или q равны $2/n$. Это означает, что M почти определенно взаимно простое с n, поэтому имеет равенство

$$M^{\varphi(n)} \equiv 1 \text{ (по модулю } n)$$

и, используя его, мы получаем:

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n) + 1} \equiv (M^{\varphi(n)})^k M \equiv 1^k M \text{ (по модулю } n) = M.$$

Оказывается, что это работает для всех M, и действительно формулы

$$D(E(M)) = M$$

и

$$E(D(M)) = M$$

Справедливы для всех M, при $0 \leq M < n$. Поэтому E и D являются обратными перестановками [20].

1.9 Корректность криптосистемы

Малая теорема Ферма.

Если p любое простое число, то имеют место следующие два эквивалентных свойства.

(1) Для каждого целого числа, $a \in Z$, если a не делится на p, мы имеем:

$$a^{p-1} \equiv 1 \text{ (по модулю } p).$$

(2) Для каждого целого числа, $a \in Z$, мы имеем:

$$a^p \equiv \alpha \text{ (по модулю } p).$$

Доказательство.

1) Рассмотрим целые числа

$$a, 2\,a, 3\,a, \ldots, (p-1)\,\alpha$$

и пусть

$$r_1, r_2, r_3 \ldots, r_{p-1}$$

будет последовательностью остатков от деления чисел в первой последовательности по p.

Поскольку

$$\text{НОД } (\alpha, p) = 1,$$

то ни одно из чисел не делится на p, поэтому

$$1 \leq r_i \leq p - 1,$$

для

$$i = 1, \ldots, p - 1.$$

Покажем, что эти остатки различны. Если нет, тогда говорят, что,

$$r_i = r_j,$$

при

$$1 \leq i < j \leq p - 1.$$

Но тогда, из-за того, что,

$$ai \equiv r_i \text{ (по модулю } p)$$

и

$$aj \equiv r_j \text{ (по модулю p)},$$

мы получаем, что

$$aj - ai \equiv r_j - r_i \text{ (по модулю } p),$$

и потому, что

$$r_i = r_j,$$

мы получаем,

$$a\,(j - i) \equiv 0 \text{ (по модулю } p).$$

Это означает, что p делится на

$$a\,(j - i),$$

но

$$\text{НОД}\,(a, p) = 1$$

поэтому, согласно предложению Евклида, p должна делится на j - i. Тем не менее,

$$1 \leq j - i < p - 1,$$

таким образом, мы получаем противоречие, и все остатки являются различными.

Есть p - 1 различных остатков и все они отличны от нуля, поэтому мы должны иметь, что

$$\{r_1, r_2, \ldots, r_{p-1}\} = \{1, 2, \ldots, p - 1\}.$$

Для любого натурального m, и для всех

$$a_1; a_2; b_1; b_2 \in Z,$$

выполняются следующие свойства. Если

$$a_1 \equiv b_1 \text{ (по модулю } m) \text{ и } a_2 \equiv b_2 \text{ (по модулю } m),$$

тогда

(1) $a_1 + a_2 \equiv b_1 + b_2$ (по модулю m).

(2) $a_1 - a_2 \equiv b_1 - b_2$ (по модулю m).

(3) $a_1a_2 \equiv b_1b_2$ (по модулю m).

Используя свойство (3) из равенств выше, мы получаем, что

$$a \cdot 2a \cdot 3a \cdot \cdot \cdot (p - 1)\, a \equiv 1 \cdot 2 \cdot 3 \cdot \cdot \cdot (p - 1) \text{ (по модулю } p),$$

то есть,

$$(a^{p-1} - 1) \cdot (p - 1)! \equiv 0 \text{ (по модулю } p).$$

Снова p делится на

$$(a^{p-1} - 1) \cdot (p - 1)!,$$

а потому что p взаимно простое с $(p - 1)!$, оно должно делиться на

$$a^{p-1} - 1,$$

как и утверждали.

2) Если

$$\text{НОД } (a, p) = 1,$$

мы доказали в (1), что

$$a^{p-1} \equiv 1 \text{ (по модулю } p),$$

из которого мы получаем

$$a^p \equiv \alpha \text{ (по модулю } p),$$

так что

$$a \equiv a \text{ (по модулю } p).$$

Если a делится на p, тогда

$$a \equiv 0 \text{ (по модулю } p),$$

что подразумевает

$$a^p \equiv 0 \text{ (по модулю } p),$$

и поэтому,

$$a^p \equiv \alpha \text{ (по модулю } p).$$

Таким образом, (2) имеет место для всех $a \in Z$ и мы только что доказали, что из (1) следует (2). И наконец, если (2) имеет место и

$$\text{НОД } (a, p) = 1,$$

а p делится на

$$a^p - a = a(a^{p-1} - 1),$$

и оно должно делится на

$$a^{p-1} - 1,$$

которое показывает, что (1) имеет место и так, (2) следует из (1).

Теперь легко доказать правильность RSA.

Для любых двух простых чисел p и q, если e и d — любые два положительных числа, такие что

1. $1 < e, d < (p - 1)(q - 1)$.

2. $ed \equiv 1$ (по модулю $(p - 1)(q - 1)$),

то любого $x \in Z$ мы имеем, что

$$x^{ed} \equiv x \text{ (по модулю } pq).$$

Доказательство.

Поскольку p и q - два различных простых числа, по предложению Евклида достаточно доказать, чтобы p и q делились на

$$x^{ed} - x.$$

Покажем, что

$$x^{ed} - x$$

делится на p, доказательство делимости q похоже.

По условию (2), мы имеем, что

$$ed = 1 + (p - 1)(q - 1)k,$$

при $k \geq 1$, так что

$$1 < e, d < (p - 1)(q - 1).$$

Поэтому, если мы напишем, что

$$h = (q - 1)k,$$

мы получим $h \geq 1$ и

$$\begin{aligned} x^{ed} - x &\equiv x^{1+(p-1)h} - x \text{ (по модулю } p) \\ &\equiv x((x^{p-1})^h - 1) \text{ (по модулю } p) \\ &\equiv x(x^{p-1} - 1)((x^{p-1})^{h-1} + (x^{p-1})^{h-2} + \cdots + 1) \\ &\text{(по модулю } p) \\ &\equiv (x^p - x)((x^{p-1})^{h-1} + (x^{p-1})^{h-2} + \cdots + 1) \text{ (по модулю } p) \\ &\equiv 0 \text{ (по модулю } p); \end{aligned}$$

потому что,

$$x^p - x \equiv 0 \text{ (по модулю } p),$$

по малой теореме Ферма [13].

1.10 Цифровая подпись

Если электронная почта система заменит существующую бумажную почтовую систему для бизнес-операций, то должно быть возможным ставить "подпись". Получатель подписанного сообщения имеет доказательства того, что сообщение было прислано от отправителя. Этот метод надежнее, чем просто аутентификация (когда получатель может проверить, что сообщение пришло от отправителя); получатель может убедить "судью", что подписывающий послал сообщение. Для этого он должен убедить судью, что он не подделывал сам, помеченное сообщение! В задачи аутентификации получатель не беспокоиться о таком, так как он только хочет удостовериться, что сообщение пришло от отправителя.

Электронная подпись должна зависеть от сообщения, а также от того, кто подписывает это сообщение. В противном случае получатель может изменить сообщение перед показом пары сообщение-подпись судье. Или он мог бы добавить подпись к любому бы то ни было сообщению, так как невозможно обнаружить электронного "вырезания и вставки текста."

Как пользователь Боб может отправить Алисе "подписанное" сообщение M в криптосистеме с открытым ключом? Он первый вычисляет свою "подпись" S для сообщения M с использованием D_B:

$$S = D_B(M).$$

Затем он шифрует S с помощью E_A (для конфиденциальности), и посылает результат $E_A(S)$ Алисе. Ему не нужно отправлять M, а также; она может быть вычислена из S.

Алиса первая расшифровывает шифротекст D_A, чтобы получить S. Она знает, кто является предполагаемым отправителем подписи (в данном случае, Боб). Затем она извлекает сообщение с процедурой шифрования отправителя, в этом случае E_B:

$$M = E_B(S).$$

Теперь она обладает парой сообщение-подпись пары (M, S) со свойствами, аналогичными подписанного бумажного документа.

Боб не может позже отрицать, отправив Алисе это сообщение, так как никто не мог создать

$$S = D_B(M).$$

Алиса может убедить "судью", что

$$E_B(S) = M,$$

так что у нее есть доказательства того, что Боб подписал этот документ.

Ясно, что Алиса не может изменить M на другую версию M', так как тогда она должна будет создать соответствующую подпись

$$S' = D_B(M').$$

Поэтому Алиса получившая сообщение, которое "подписано" Бобом, может "доказать", что он послал это сообщение, но она не может изменить его. (При этом она не может подделать подпись для любого другого сообщения.) [10, 17]

1.11 Выбор простых чисел

Каждый пользователь должен выбрать два больших случайных числа p и q, чтобы создать свои собственные ключи шифрования и дешифрования. Эти цифры должны быть большими, так чтобы не представлялось возможным вычисление для кого-либо фактора

$$n = p \cdot q.$$

(Помните, что n находится в открытом доступе, но не p и q.) Рекомендуется использовать 100-значные (десятичные) простые числа p и q, так что получается n состоит из 200 цифр.

Для простоты вычисления простого случайного числа, будем использовать генерацию простых чисел, с последующей проверкой их на простоту.

Кроме этого p и q не должны быть одинаковыми или близкими, так как можно воспользоваться методом Ферма для факторизации n и решить уравнение [16]

$$(\frac{p+q}{2})^2 - n = (\frac{p-q}{2})^2.$$

1.11.1 Выбор *d*

Легко выбрать число *d*, взаимно простое с $\phi(n)$. Например, любое простое число, которое будет больше, чем максимум от (*p*, *q*). Важно, чтобы *d* было выбрано из больших чисел, чтобы его не нашли прямым поиском [16].

1.11.2 Вычисление *e* из *d* и $\phi(n)$

Для вычисления *e*, используется следующий вариант алгоритма Евклида для вычисления наибольшего общего делителя $\phi(n)$ и *d*. Вычислить

$$\text{НОД}\ (\phi(n), d)$$

путем вычисления ряда

$$x_0, x_1, x_2, \ldots,$$

где

$$x_0 \equiv \phi(n),\ x_1 = d \text{ и } x_i + 1 \equiv x_i - 1 \text{ (по модулю } x_i),$$

до достижения x_k равным 0. Тогда

$$\text{НОД}\ (x_0, x_1) = x_{k-1}.$$

Далее нужно рассчитать для каждого x_i чисел a_i и b_i такие, что

$$x_i = a_i \cdot x_0 + b_i \cdot x_1.$$

Если

$$x_{k-1} = 1,$$

то b_{k-1} является мультипликативной инверсией

$$x_1 \text{ (по модулю } x_0).$$

Так как *k* будет меньше, чем $2\log_2(n)$, это вычисление будет происходит очень быстро.

Если *e* окажется меньше, чем $\log_2(n)$, необходимо начать вычисление простой заново, путем выбора другого значения *d* [16].

1.11.3 Выбор *e* и вычисление *d* из *e* и $\phi(n)$

Выбрать случайное простое число *e*, которое будет удовлетворять двум требованиям:

1) $1 < e < \phi(n)$.
2) Быть взаимно простым со значением функции $\phi(n)$.

Число d вычисляется, как мультипликативное обратное к числу e по модулю $\phi(n)$, то есть число, удовлетворяющее:

$$d \cdot e \equiv 1 \text{ (по модулю } \phi(n)).$$

1.12 Криптоанализ RSA

В то время как криптография наука о создании шифров, криптоанализ связан с исследованием взлома шифров. Криптография и криптоанализ науки взаимодополняющие: развитие одной обычно сопровождается дальнейшим развитием другой. В частности, криптоанализ является важным инструментом для оценки уязвимости криптосистем.

В схеме шифрования, основной целью злоумышленника является восстановление открытого текста M из соответствующего шифротекста. Если оно окажется успешным, мы говорим, что он взломал систему. В случае цифровых подписей, цель злоумышленника заключается в формировании подписи. Более амбициозная атака является восстановление секретного ключа d. Если это получится, злоумышленник тогда может расшифровать все шифротексты и подделывать подписи по собственному желанию. В этом случае единственным решением является аннулирования (изъятие из оборота) ключа.

Ниже приводится краткое описание основных методов, используемых для атаки криптосистемы RSA. Таких как основные методы факторинга, нападения на основную математическую функцию и атаки, которые используют недостатки в реализациях.

Для этого будем использовать три сущности, участвующих в системе как Алиса, Боб и Кэролайн. Алиса и Боб желают безопасно обмениваться данными друг с другом, в то время как Кэролайн злоумышленник, которой пытается помешать коммуникации [19].

1.12.1 Факторинг

Задача нахождения нетривиальных факторов заданного положительного составного целого числа n называется целочисленной задачи факторизации. Проблема факторизации считается трудной задачей, то есть, нет полиномиального алгоритма, который решает эту проблему для значительной

части возможных случаев. Факторинг не всегда трудно провести, но легко создать пример, при котором возникают такие проблемы, путем простого умножения двух больших простых чисел. То есть именно то, что мы делаем при работе с RSA.

Как мы уже видели, безопасность криптосистемы RSA тесно связана с целочисленной задачи факторизации. Если злоумышленник сможет провести успешную факторизацию модуля n, он сможет вычислить секретный ключ. В этом случае означает, что он полностью нарушил схему шифрования: он не только сможет восстановить конкретный открытый текст M, но и расшифровать все зашифрованные тексты, зашифрованные с помощью соответствующего открытого ключа.

Поэтому, хотя методы факторинга не используются на практике в нападениях на RSA, они играют важную роль при выводе нижних границ для размеров ключей и свойств параметров безопасности. Принимая во внимание значение данных и модели угроз, параметры должны быть выбраны таким образом, что факторинг модуля будет является невозможным для вычисления.

Методы факторизации можно разделить на методы специального назначения и общего назначения. Методы специального назначения зависят от особых свойств числа, которые будут учитываться, таких как размер наименьшего множителя p в n, факторизация для p - 1, и т.д. методы общего назначения зависят только от размера n [17, 19].

1.12.2 Перебор делителей

Перебор делителей можно рассматривать как исчерпывающий поиск секретного ключа RSA. Метод заключается в попытке разделить n на каждое число из последовательности простых чисел, пока один из делителей не будет найден. Поскольку составное число n должно иметь простой делитель

$$\leq n^{1/2},$$

поэтому надо только проверить для всех простых чисел до квадратного корня из n. Тогда из теоремы о простых числах следует, что количество попыток ограничено

$$(2n^{1/2}) / (\log(n)).$$

Хотя этот метод является очень эффективным при попытке факторизации случайно выбранного составного числа или относительно малого числа, оно бесполезно против вида чисел, используемых в настоящее время с RSA [16].

1.12.3 Атака на секретную экспоненту малого размера

Дешифрования RSA и подписание его требует большой вычислительной мощности, которые занимают много времени, в зависимости от длины секретного показателя *d*. Таким образом, для некоторых маломощных устройств могут использовать небольшую *d* вместо случайной, в целях повышения производительности. Однако атака благодаря Винеру показывает, что выбор небольшой *d* может привести к полному разрушению системы. Более конкретно, он показал, что, если пара модуль n и d являются секретным ключом, при

$$d < 1/3\ (n)^{1/4},$$

а также открытым ключом является пара (*e, n*), то злоумышленник может эффективно восстановить *d*.

В практическом плане это означает, что для типичного 1024-битного RSA-модуля, частный показатель *d* должен быть не менее 300 бит в длину. Если открытый показатель степени *e* выбирается 65537 (наиболее часто используется значение), и вычислить *d*, как

$$de \equiv 1 \text{ по модулю } (n),$$

то можно гарантировать, что нахождение *d* почти так же сложно, как и нахождение *n*, то эта атака не должна представлять угрозы [21].

1.12.4 Атаки на передачу и связанные сообщения

Если Боб транслирует шифрование одного и того же сообщения *M* достаточно большому количеству получателей (которые имеют различные открытые ключи (e_i, n_i)), Hastad описал нападение, в котором злоумышленник может эффективно восстановить *M*, если все общедоступные экспоненты малы. Эта атака может иметь смысл в случаи, если вместо отправки

$$(M)^{ei} \text{ по модулю } n_i$$

каждому получателю, Боб посылает

$$(f_i(M))^{ei} \text{ по модулю } n_i,$$

где f_i известные полиномы. Это делается путем решения системы одномерных уравнений по модулю относительно простых составных чисел, которые могут быть найдены, если имеется достаточно много уравнений.

Существует также атака, когда Боб посылает Алисе связанные зашифрованные сообщения, используя один и тот же модуль. Если M_1 и M_2 — 2 сообщения, такие, что

$$M_1 = f(M_2),$$

где f является известной полиномиальной функцией, и Боб посылает

$$C_1 = (M_1)^e \text{ по модулю } n$$

и

$$C_2 = (M_2)^e \text{ по модулю } n$$

то злоумышленник также может восстановить оба сообщения, если e мало. Примером такого сценария является, где Боб посылает первое зашифрованное сообщение Алисе, которое перехватывает Каролина. Но из-за того, что Алиса ответила, Боб посылает сообщение снова, используя различные дополнения (функции f). Тогда Кэролайн может восстановить M с помощью атаки выше.

Обе эти атаки являются более эффективными, если открытый показатель степени $e = 3$, хотя они могут быть предотвращены, если мы удалим связь между сообщениями, как правило, путем добавления какого-то случайного дополнения [21].

1.12.5 Атаки на детали реализации

Все атаки против RSA, что описаны выше, применяются к основным криптографическим примитивам и параметрам. С другой стороны, атаки на основе реализации (называемые также атаки по сторонним каналам) ориентированы на конкретные детали реализации. В этом случае злоумышленник обычно использует некоторую дополнительную информацию, вытекающую из реализации функции RSA или использование ошибок в реализации. Такие атаки, как правило, применяются на смарт-картах и маркеров безопасности, и являются более эффективными, когда злоумышленник имеет

криптографический модуль. Трудно защититься от атак по сторонним каналам; обычно пытаются уменьшить количество утекающей информации или сделать ее не нужной для злоумышленника [21].

Глава 2. Реализация и тестирование алгоритма

2.1 Разработка алгоритма RSA на языке Java

Исходя из требований к программному продукту, алгоритм нахождения чисел, являющимися ключами, реализован в виде функции. На рисунке 2.1 представлен код для нахождения, открытого и секретного ключей:

```
/** Создаются ключи для шифровки и дешифровки. */
public RSA() {
  SecureRandom r = new SecureRandom();
  BigInteger p, q;
   p = BigInteger.probablePrime(bitlen / 2, r);
   //проверка что p и q не равны
   do{
       q = BigInteger.probablePrime(bitlen / 2, r);
   }while( q.compareTo( p ) == 0 );

  n = p.multiply(q);

  BigInteger phi = (p.subtract(BigInteger.ONE)).
          multiply(q.subtract(BigInteger.ONE));

  // проверка e на удолетворение условию  1 < e < phi
  do
  {
      e = BigInteger.probablePrime(bitlen / 2, r);

  }while (phi.gcd(e).compareTo(BigInteger.ONE) > 0
          && e.compareTo(phi) < 0 && e.compareTo(BigInteger.ONE) > 0);

  d = e.modInverse(phi);
}
```

Рисунок 2.1 — Функция создания ключей на Java

Встроенные методы: probablePrime() — для нахождения простого числа, gcd() — для проверки чисел на взаимную простоту. А также метод modInverse() — нахождение обратной по модулю, позволяют уменьшить объём кода, что влияет на быстродействие программ.

Блок-схема работы алгоритма RSA, изображена на рисунке 2.2.

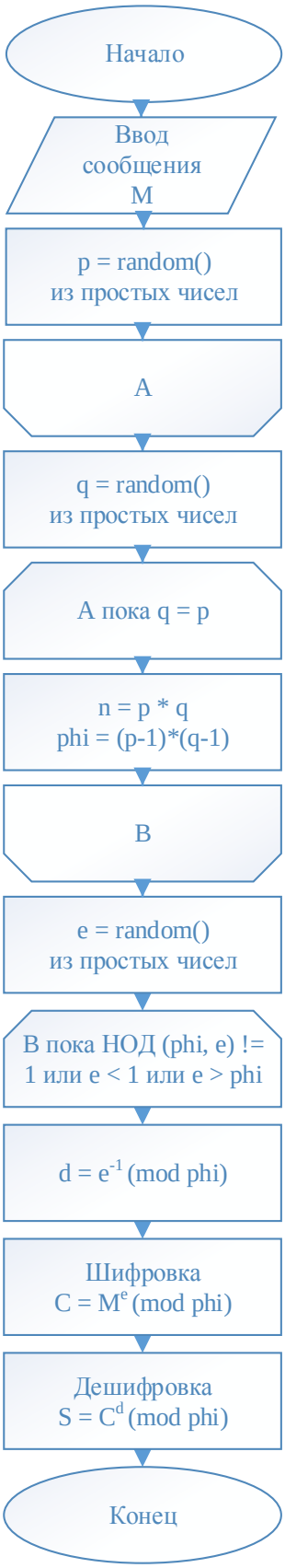

Рисунок 2.2 — Блок-схема работы алгоритма RSA в представленных реализациях

2.2 Разработка алгоритма RSA на языке C++

Для простоты и более точного анализа, алгоритм нахождения ключей, также реализован в виде функции. На рисунке 2.3 представлен код для нахождения, открытого и секретного ключей:

```
//нахождение открытого и секретного ключей
void RSA() {
    // Генерация двух простых чисел p и q
    long long int p = rand() % 10000;
    long long int p_simple = sundaram(p);
    long long int q = 0, q_simple = 0;
    do {
        q = rand() % 10000;
        q_simple = sundaram(q);
    } while (q_simple == p_simple);
    //Находим число n.
    n = p_simple*q_simple;
    long long int phi = (p_simple - 1)*(q_simple - 1);

    //Генерация числа e, для которого является истинным
    //условие 1 < e < phi
    do {
        e = rand() % 10000;
        e = sundaram(e);
    } while (gcd(e, phi) != 1 && e >= phi && e <= 1);
    d = modInverse(e, phi);
}
```

Рисунок 2.3 — Функция создания ключей на C++

Из-за неимения необходимых встроенных функций в языке C++, мы пишем их сами. Функция sundaram() — для нахождения простого числа по алгоритму "решето Сундарама", ближайшего к заданному. А функции gcd() и modInverse() похожи на встроенным методы в Java с аналогичными названиями.

2.3 Тестирование алгоритма

В программе используются простые числа с количеством бит 1024, генерируемые с помощью алгоритма решето Сундарама, которые проверяются на взаимную простоту по алгоритму Евклида. Для возведения в степень при шифровании используется последовательное возведение в квадрат; при дешифровании - Китайская теорема об остатках.

Для тестирования работы алгоритма шифрования, будем использовать текст, состоящий из 5 и 169 слов. В ходе выполнения программы должны

самостоятельно создать исходя из выбранного количества бит, ключи для шифрования и дешифрования текста. Пример работы программы на языке Java, изображен на рисунке 2.4.

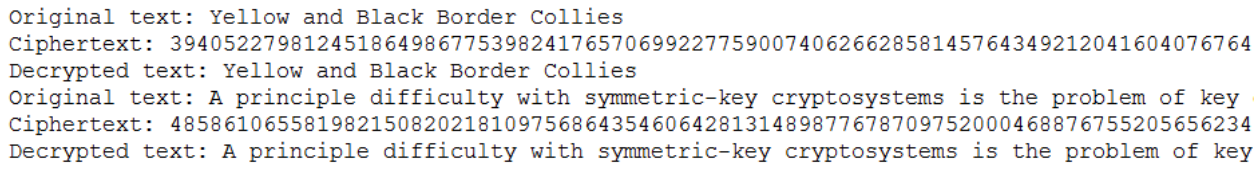

Рисунок 2.4 — Результат работы программы на языке программирования Java

В ходе тестирования алгоритма мы убедились, что программы на языке Java шифрует и расшифровывает текст. Так же независимо от длины сообщения.

Теперь проверим поведение алгоритма, при тестировании на языке C++, пример работы которого изображен на рисунке 2.5.

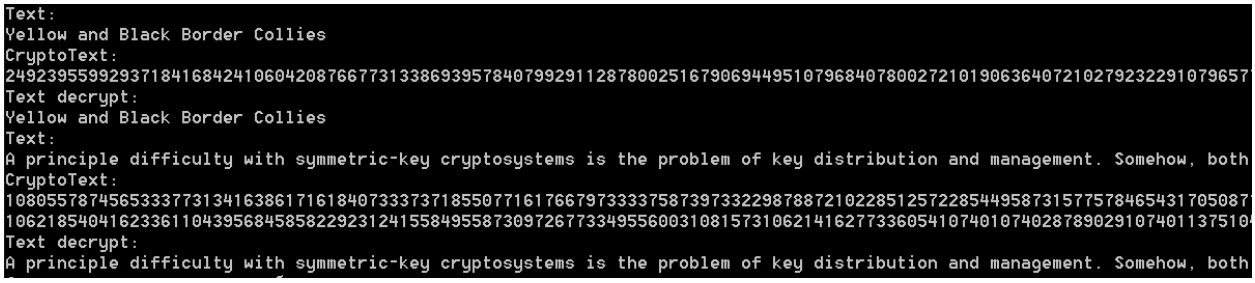

Рисунок 2.5 — Результат работы программы на языке программирования C++

Теперь вы с уверенностью может сказать, что обе реализации алгоритма шифрования работают правильно и соответствуют требованиям к программному продукту.

Глава 3. Сравнительный анализ реализаций

При разработке алгоритма шифрования была предусмотрена возможность измерения времени работы программ в миллисекундах.

Бралось системное время перед началом шифрования и после завершения расшифровки, полученные данные, записывались в файл. Для большей достоверности вычислим среднее время шифрования. Среднее время будем рассчитывать за 1000 проходов. Это позволит максимально снизить уровень влияния аппаратуры, на время работы алгоритма RSA.

Как мы видим, время шифрования на языке C++ в разы быстрее, чем на Java. В нашем случае время работы алгоритма зависит от языка программирования, длинны наших строк и количества шифруемых данных. Теперь переведем все эти цифры в графики. Результат работы двух реализаций на разных языках программирования при малом количестве текста, представлен на рисунке 3.1.

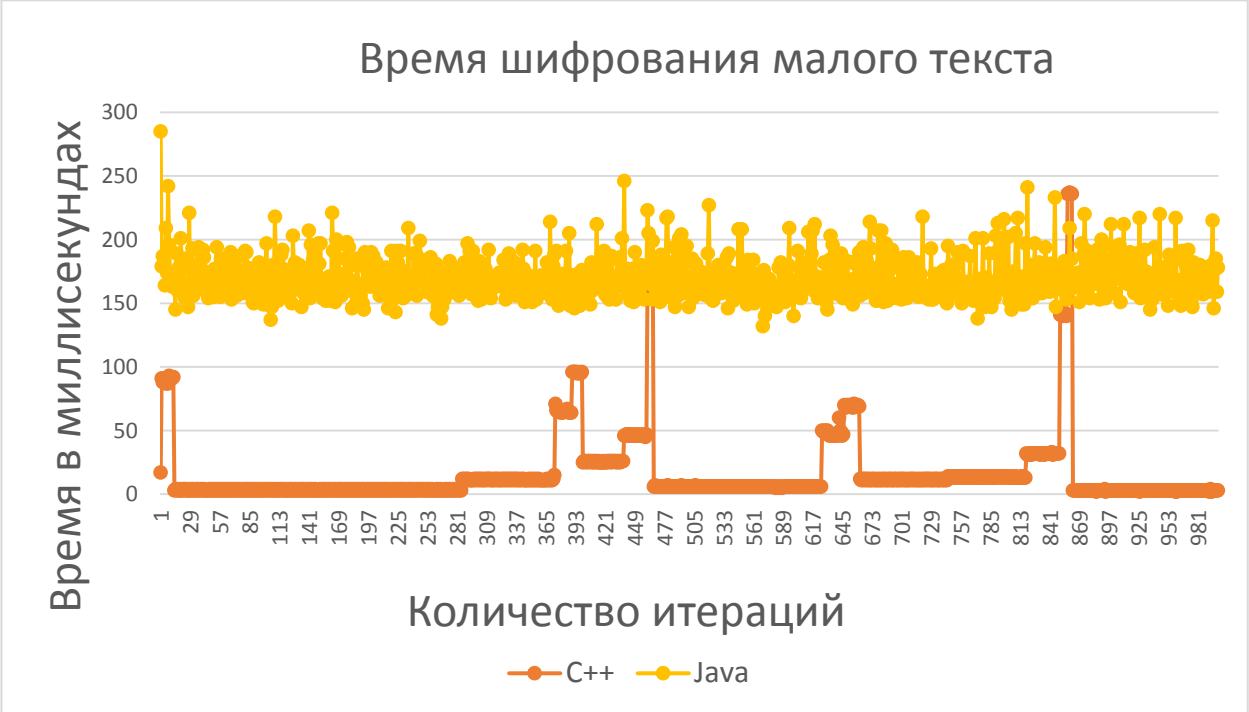

Рисунок 3.1 — Результат работы программ при малом количестве данных

На оси абсцисс у нас расположено количество итераций, а на оси ординат время (в миллисекундах) шифрования и дешифрования данных.

А теперь посмотрим время работы программ с большим количеством дынных, представленных на рисунке 3.2.

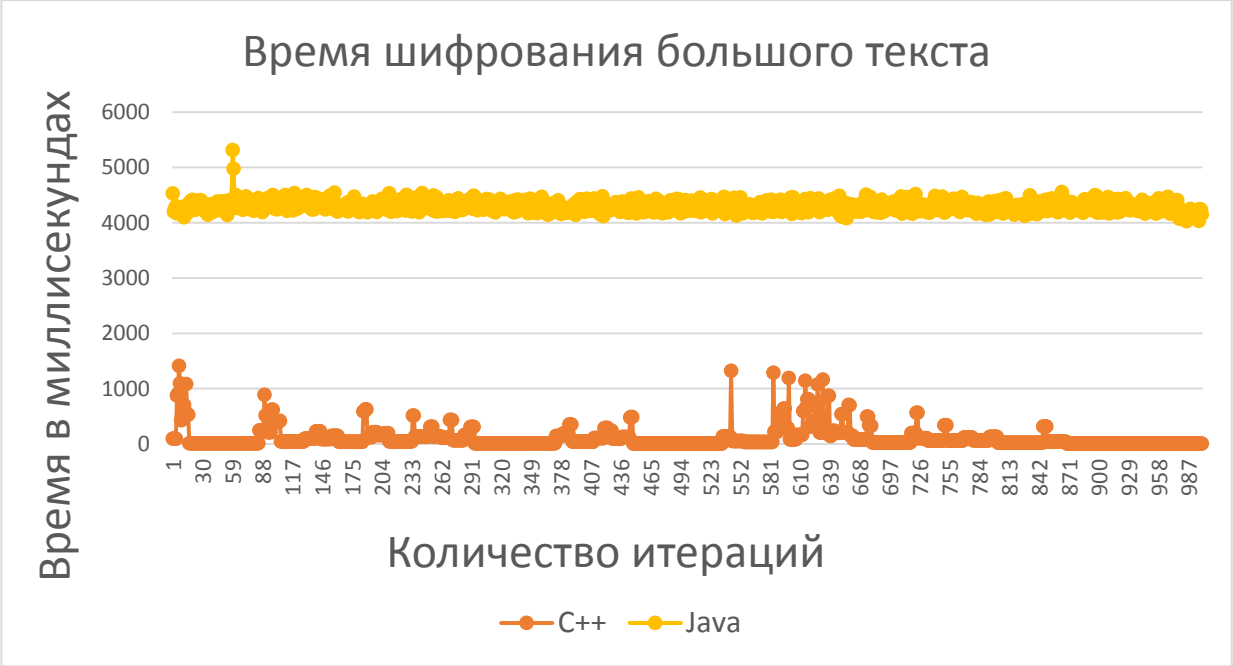

Рисунок 3.2 — Результат работы программ при большом количестве данных

Как видно на графиках, в результате работы, время шифровки и расшифровки данных, на языке C++, все же в несколько раз меньше. Погрешность в графиках обуславливается аппаратными характеристиками компьютера, на котором проходило тестирование.

Заключение

В ходе выполнения бакалаврской работы рассмотрены алгоритм шифрования RSA, его теоретические аспекты и математические основы, достоинства и недостатки алгоритма, а также его реализации на двух языках программирования, Java и C++.

Криптосистема RSA является надежным алгоритмом для шифрования с открытым ключом и электронной подписи во всем мире. Она используется в наиболее популярных продуктах, где необходима безопасность, и протоколах, используемых сегодня, и может рассматриваться в качестве одной из основ для безопасного общения в сети Интернет.

Основная функция и свойства алгоритма широко изучались математиками и специалистами по безопасности на протяжении более четверти века. Хотя и было разработано множество атак в течение этого периода, используя специальные свойства функции RSA, а также подробную информацию в конкретной реализации, он не был взломан на протяжении многих лет, и его безопасность никогда не подставлялось под сомнение. Нет надежного и эффективного способа взлома данного алгоритма, и большинство проблем, связанных с ним, являются результатом неправильного использования системы, плохого выбора параметров или недостатков в реализации. На самом деле, годы исследований, повысило доверие к безопасности RSA, и есть все основания полагать, что он будет оставаться наиболее часто используемым алгоритмом с открытым ключом еще долгие годы.

Был изучен алгоритм RSA и пошагово описан ход выполнения алгоритма шифрования RSA. Рассмотрено доказательство его корректности и пример его работы.

Стойкость RSA зависит от сложности задачи факторизации, поэтому атаки, основанные на решении этой задачи, представляют наибольшую угрозу. А остальные атаки направлены на недостатки и особенности реализации криптосистемы.

Была рассмотрена возможность использования алгоритма RSA более чем двумя участниками. Это позволяет большому числу пользователей передавать шифрованную информацию, и какие могут возникнуть при этом проблемы.

Также был реализован алгоритм шифрования RSA на двух языках программирования Java и C++. Реализации были написаны максимально приближенно друг к другу для более точного сравнительного анализа.

Для сравнительного анализа реализаций алгоритма RSA были добавлены методы подсчета времени, позволяющие выявить их быстродействие. Для получения более точных результатов, были проведены не менее 1000 итераций шифрования и дешифрования данных с малым и большим размером текста.

Практическое сравнение реализаций показало, что быстродействие реализации, написанной на языке программирования C++, превосходит похожую реализацию на языке программирования Java. Зато для простоты написания лучше использовать язык Java.

Список используемой литературы

1. Герман, О.Н. Теоретико-числовые методы в криптографии: учебник / О.Н. Герман, Ю.В. Нестеренко. — М.: Академия, 2012. — 272 с.

2. Ишмухаметов, Ш.Т. Математические основы защиты информации: учеб. пособие / Ш.Т. Ишмухаметов, Р.Г. Рубцов — Казань: Казанский федер. Ун-т, 2012. — 138 с.

3. Нестеренко, А.Ю. Теоретико-числовые методы в криптографии: учеб. пособие / А.Ю. Нестеренко — М.: Моск. гос. ин-т. электроники и математики, 2012 — 224 с.

4. Орлов, В.А. Теория чисел в криптографии: учеб. пособие / В.А. Орлов, Н.В. Медведев, Н.А. Шимко, А.Б. Домрачева — М.: Изд-во МГТУ им. Н.Э. Баумана, 2011 — 223 с.

5. Романьков, В.А. Введение в криптографию. Курс лекций / В.А. Романьков. — М.: ФОРУМ, 2012. — 240 с.

6. Салий, В.Н. Криптографические методы и средства защиты информации: учеб. пособие / В.Н. Салий; Саратов: Саратовский гос. ун-т имени Н.Г. Чернышевского, 2012. — 41 с.

7. Ященко, В.В. Введение в криптографию. / В.В. Ященко — 4-е изд., [доп.]. — М.: МЦНМО, 2012. — 348 с.

8. Boucinha, F. A Survey of Cryptanalytic Attacks on RSA. 2011. [Электронный ресурс]. Режим доступа: https://fenix.tecnico.ulisboa.pt/downloadFile/395143450047/dissertacao.pdf

9. Bowman, J.C. Coding theory & cryptography. 2015. [Электронный ресурс]. Режим доступа: https://www.math.ualberta.ca/~bowman/m422/m422.pdf

10. Cozzens, M.J. The Mathematics of Encryption: An Elementary Introduction. 2013. [Электронный ресурс]. Режим доступа: https://books.google.ru/books?id=GbKyAAAAQBAJ&printsec=frontcover&dq=The+Mathematics+of+Encryption:+An+Elementary+Introduction&hl=ru&sa=X&ved=0ahUKEwif8-7t-

sLMAhWFkSwKHShUB3kQ6AEIJTAA#v=onepage&q=The%20Mathematics%20of%20Encryption%3A%20An%20Elementary%20Introduction&f=false

11. Evans, M. RSA Encryption. 2011. [Электронный ресурс]. Режим доступа: http://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Encryption5.pdf

12. Gagneja, K. A Survey and Analysis of Security Issues on RSA Algorithm. / K. Gagneja, K.J. Singh. 2015. [Электронный ресурс]. Режим доступа: http://www.maxwellsci.com/print/RJASET/11-847-853.pdf

13. Gallier, J. Notes on public key cryptography and primality testing part 1: randomized algorithms Miller-Rabin and Solovay-Strassen tests. 2016. [Электронный ресурс]. Режим доступа: http://www.cis.upenn.edu/~jean/RSA-primality-testing.pdf

14. Hawryluk, C. RSA Encoding. 2013. [Электронный ресурс]. Режим доступа: http://www.math.tamu.edu/~david.larson/hawryluk13.pdf

15. Jenings, T.A. The mathematics of cryptography & data compression. 2012. [Электронный ресурс]. Режим доступа: https://www.carroll.edu/library/thesisArchive/Anderson%20J_2012final.pdf

16. Kak, A. Public-Key Cryptography and the RSA Algorithm. 2016. [Электронный ресурс]. Режим доступа: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf

17. Lupafya, C. Cryptanalysis Attacks on RSA. 2012. [Электронный ресурс]. Режим доступа: https://vlebb.leeds.ac.uk/bbcswebdav/orgs/SCH_Computing/MSCProj/reports/1213/Lupafya.pdf

18. McNeely, P. Variations and Attacks on the RSA Algorithm. / P. McNeely, B. Walker. 2013. [Электронный ресурс]. Режим доступа: https://petemcneely.files.wordpress.com/2013/03/variations-and-attacks-on-the-rsa-algorithm-paper.pdf

19. Ruohonen, K. Mathematical cryptology. 2014. [Электронный ресурс]. Режим доступа: http://math.tut.fi/~ruohonen/MC.pdf

20. Skobic, V. Hardware Modules of the RSA Algorithm. / V. Skobic, B. Dokic, Z. Ivanovic. 2014. [Электронный ресурс]. Режим доступа: http://www.doiserbia.nb.rs/img/doi/1451-4869/2014/1451-48691400011S.pdf

21. Wang, J. Attacks against RSA Cryptosystems in Thirty Years. 2011. [Электронный ресурс]. Режим доступа: http://cis.sjtu.edu.cn/download/d/df/RSA_Attacks.

Приложение А

Листинг кода, реализаций алгоритма RSA

Реализация на языке Java:

```
//неизменяемый класс, для работы с целыми числами
import java.math.BigInteger;
//класс, беспечивающий криптостойкую генерацию случайных чисел
import java.security.SecureRandom;
//для работы с файлом
import java.io.*;
import java.awt.Desktop;
/**
* Реализация алгоритма шифрования RSA.
*/
public class RSA {

        public static double runningTime;
        private static BigInteger n, e, d;
        private int bitlen = 1024;

        /** Создаются ключи для шифровки и дешифровки. */
        public RSA() {
          SecureRandom r = new SecureRandom();
          BigInteger p, q;
          p = BigInteger.probablePrime(bitlen / 2, r);
          //проверка что p и q не равны
          do{
                q = BigInteger.probablePrime(bitlen / 2, r);
          } while( q.compareTo( p ) == 0 );
          n = p.multiply(q);
          BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
          // проверка e на удолетворение условию  1 < e < phi
          do{
                  e = BigInteger.probablePrime(bitlen / 2, r);
         }while (phi.gcd(e).compareTo(BigInteger.ONE) > 0
                  && e.compareTo(phi) < 0 && e.compareTo(BigInteger.ONE) > 0);
```

```
  d = e.modInverse(phi);
}

/** Проверка работы программы. */
public static void main(String[] args) {
        //для открытия файла после завершения работы программы
        Desktop desktop = null;
        if (Desktop.isDesktopSupported()) {
            desktop = Desktop.getDesktop();
        }
        int k = 0, z = 1;
        //для записи времени в файл
        try (FileWriter writer = new FileWriter("D:\\X-Rey\\javarez.txt", false)){
                do{
                        runningTime = System.currentTimeMillis();
                        //Создаем экземляр класса для генерации RSA ключей
                        RSA rsa = new RSA();
                        //сообщения для шифровки
                         String text = "";
                          if(z==0){
                                text = "Yellow and Black Border Collies";
                          }
                           if(z==1){
                                text = "A principle difficulty with symmetric-key
```
cryptosystems is the problem of key distribution and management. Somehow, both parties, which may be quite distant from each other, have to securely exchange their secret keys before they can begin communication. One technique for avoiding the problem of key exchange makes use of two secure envelopes, or locks, which each party alternately applies and later removes from the sensitive data, as the data makes a total of three transits between sender and receiver. The required three transmissions makes this method awkward to use in practice. In public - key cryptosystems, key exchange is avoided altogether by making copies of the receiver's encrypting key (lock) available to anyone who wants to communicate with him. Both the secure envelope technique and the public - key technique require that the encrypting key e is designed so that the decrypting key d is extremely difficult to determine from knowledge of e. They also require authentication of the lock itself, to guard against so - called man - in - the - middle attacks.";

```
}
 byte[] temp = new byte[1] ;
 byte[] digits = text.getBytes();
 BigInteger [] plaintext = new BigInteger[digits.length] ;
 BigInteger [] ciphertext = new BigInteger[digits.length] ;
 BigInteger [] decrypttext = new BigInteger[digits.length] ;

 //ШИФРОВКА
 for(int i = 0; i < plaintext.length; i++){
        temp[0] = digits[i];
        plaintext[i] = new BigInteger(temp);
        ciphertext[i] = plaintext[i].modPow(e, n);
 }

 //ДЕШИФРОВКА
 for(int i = 0; i < ciphertext.length; i++){
        decrypttext[i] = ciphertext[i].modPow(d, n);
 }
 byte[] byteArray = new byte[decrypttext.length] ;
 String rs = "";
 try {
        for(int i = 0 ; i < decrypttext.length ; i++ ) {
                byteArray[i] = decrypttext[i].byteValue();
        }
        rs=new String( byteArray );
 }
 catch (Exception exc) {
        System.out.println(exc);
 }

 System.out.println("Original text: " + text);
 String str = "";
 try {
        for ( int i = 0 ; i < ciphertext.length ; i++ ){
                str = ciphertext[i].toString();
```

```
                    }
                }
                catch (Exception exc) {
                    System.out.println(exc);
                }
                System.out.println("Ciphertext: " + str);
                System.out.println("Decrypted text: " + rs);

                //Подсчет времени
                String txt = Integer.toString((int)((runningTime =
                    System.currentTimeMillis() - runningTime)));
                writer.write(txt);
                writer.write(System.getProperty("line.separator"));
                k++;
            } while(k<1000);
        }
        catch(IOException ex){
            System.out.println(ex.getMessage());
        }

        //Открытие файла с полученным временем
        try {
            desktop.open(new File("D:\\X-Rey\\javarez.txt"));
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

Реализация на языке C++:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <fstream>
#include <string>

using namespace std;

long long int n, e, d;
```

```
//Алгоритм "решето Сундарама". Выбирает все простые числа
//до заданного (случайно сгенерированного).
long long int sundaram(long long int n)
{
	long long int *a = new long long int[n], i, j, k;
	memset(a, 0, sizeof(long long int) * n);
	for (i = 1; 3 * i + 1 < n; i++)
	{
		for (j = 1; (k = i + j + 2 * i*j) < n && j <= i; j++)
			a[k] = 1;
	}
	//Выбирает из списка простых чисел ближайшее к заданному.
	for (i = n - 1; i >= 1; i--)
		if (a[i] == 0)
		{
			return (2 * i + 1);
			break;
		}
	delete[] a;
}

//Алгоритм Евклида. Алгоритм для нахождения наибольшего
//общего делителя двух целых чисел. Используется для проверки
//чисел на взаимопростоту.
long long int gcd(long long int a, long long int b)
{
	long long int c;
	while (b)
	{
		c = a % b;
		a = b;
		b = c;
	}
	return abs(a);
}


// Нахождение числа обратного по mod числу a
long long int modInverse(long long int a, long long int m)
{
	a = a%m;
	for (long long int x = 1; x<m; x++)
		if ((a*x) % m == 1)
			return x;
}

//нахождение открытого и секретного ключей
void RSA() {
	// Генерация двух простых чисел p и q
	long long int p = rand() % 10000;
	long long int p_simple = sundaram(p);
```

```
	long long int q = 0, q_simple = 0;
	do {
		q = rand() % 10000;
		q_simple = sundaram(q);
	} while (q_simple == p_simple);
	//Находим число n.
	n = p_simple*q_simple;
	long long int phi = (p_simple - 1)*(q_simple - 1);

	//Генерация числа e, для которого является истинным
	//условие 1 < e < phi
	do {
		e = rand() % 10000;
		e = sundaram(e);
	} while (gcd(e, phi) != 1 && e >= phi && e <= 1);
	d = modInverse(e, phi);
}

// Находит (a ^ b) mod n
long long int modpow(long long int a, long long int b, long long int mod)
{
	long long int t;
	if (b == 1)
		return a;
	t = modpow(a, b / 2, mod);
	if (b % 2 == 0)
		return (t*t) % mod;
	else
		return (((t*t) % mod)*a) % mod;
}

void main()
{
	int k = 0, z = 1;
	//для записи времени в файл
	ofstream out("D:/X-Rey/crez.txt");
	do {
		// Переменные для вычисления времени работы программы

		unsigned int start = clock();

		srand((unsigned)time(NULL));

		//создание ключей
		RSA();

		string txt = "";
		if (z == 0) {
			txt = "Yellow and Black Border Collies";
		}
		if (z == 1) {
```

```
		txt = "A principle difficulty with symmetric-key cryptosystems is the problem of key distribution and management. Somehow, both parties, which may be quite distant fromeach other, have to securely exchange their secret keys before they can begin communication. One technique for avoiding the problem of key exchange makes use of two secure envelopes, or locks, which each party alternately applies and later removes from the sensitive data, as the data makes a total of three transits between sender and receiver. The required three transmissions makes this method awkward to use in practice. In public - key cryptosystems, key exchange is avoided altogether by making copies of the receiver's encrypting key (lock) available to anyone who wants to communicate with him.Both the secure envelope technique and the public - key technique require that the encrypting key e is designed so that the decrypting key d is extremely difficult to determine from knowledge of e.They also require authentication of the lock itself, to guard against so - called man - in - the - middle attacks.";
	}

	//Ввод шифруемых данных.
	const int MAX = txt.length();
	long long int *Text = new long long int[MAX];

	//перевод данных в кода ASCII
	for (int i = 0; i < (int) txt.length(); i++) {
		Text[i] = static_cast<long long int>(txt[i]);
	}

	//Массив для хранения шифротекста.
	long long int *CryptoText = new long long int[MAX];
	long long int *Tdecrypt = new long long int[MAX];

	//ШИФРОВАНИЕ данных по формуле c = (m^e) mod n.
	for (int j = 0; j < MAX; j++) {
		CryptoText[j] = modpow(Text[j], e, n);
	}

	//ДЕШИФРОВАНИЕ данных по формуле m = (c^d) mod n
	for (int j = 0; j < MAX; j++) {
		Tdecrypt[j] = modpow(CryptoText[j], d, n);
	}
	string dectxt = "";
	for (int j = 0; j < MAX; j++)
	{
		dectxt += static_cast<char>(Tdecrypt[j]);
	}

	//вывод
	cout << "Original text: " << endl;
	cout << txt;
	cout << endl << endl;

	cout << "Ciphertext: " << endl;
	for (int j = 0; j < MAX; j++)
	{
		cout << CryptoText[j];
	}
```

```
            cout << endl << endl;

            cout << "Decrypted text: " << endl;
            cout << dectxt;
            cout << endl;

            delete[] Text;
            delete[] CryptoText;
            delete[] Tdecrypt;

            unsigned int end = clock(); // конечное время
            unsigned int time = end - start;
            out << time << endl;
            k++;
        } while (k<1000);
        out.close();

        //открытие файла с полученным временем
        system("notepad.exe D:/X-Rey/crez.txt");
}
```