МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 кафедры)

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

Системное программирование и компьютерные технологии
(направленность (профиль)/специализация

БАКАЛАВРСКАЯ РАБОТА

на тему Реализация криптографического алгоритма RSA

Студент	Д.С. Дашкевич	
	(И.О. Фамилия)	(личная подпись)
Руководитель	Г.А. Тырыгина	
	(И.О. Фамилия)	(личная подпись)
Консультанты	М.А. Четаева	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Тольятти 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 кафедры)

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная
математика и информатика»
_____
_____
(подпись) (И.О. Фамилия)
«____»_____20___г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Дашкевич Денис Сергеевич

1. Тема «Реализация криптографического алгоритма RSA».

2. Срок сдачи студентом законченной выпускной квалификационной работы май 2018 г.

3. Исходные данные к выпускной квалификационной работе алгоритм RSA, язык программирования C#.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов) титульный лист, аннотация (рус., анг.), оглавление, введение, анализ алгоритма RSA и его реализация, заключение, список литературы, приложения.

5. Ориентировочный перечень графического и иллюстративного материала презентация.

6. Консультанты по разделам

_____

7. Дата выдачи задания «_____»_____20____г.

Руководитель выпускной
квалификационной работы _____ Г.А. Тырыгина
(подпись) (И.О. Фамилия)

Задание принял к исполнению _____ Д.С. Дашкевич

(подпись) (И.О. Фамилия)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 кафедры)

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная математика и информатика»
_____
_____
(подпись) (И.О. Фамилия)
«____»_____20___г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Дашкевич Денис Сергеевич
по теме «Реализация криптографического алгоритма RSA»

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Изучение источников по теме ВКР	Декабрь2017			
Изучение алгоритма RSA	Декабрь2017			
Реализация алгоритма RSA	Январь2018			
Реализация программного кода	Март2018			
Тестирование приложения	Март2018			
Подготовка пояснительной записки	Апрель2018			
Проверка на наличие заимствований (плагиата) в системе «Антиплагиат ВУЗ»	Апрель2018			
Подготовка презентации к защите	Май2018			
Предзащита дипломной работы	Май2018			
Сдача на кафедру комплекта документов для защиты ВКР	Май 2018			
Защита дипломной работы	Июнь 2018			

Руководитель выпускной квалификационной работы

	Г.А. Тырыгина
(подпись)	(И.О. Фамилия)

Задание принял к исполнению

	Д.С. Дашкевич
(подпись)	(И.О. Фамилия)

АННОТАЦИЯ

Дипломная работа посвящена вопросу реализации алгоритма RSA и разработке программы, шифрующей данные при помощи этого алгоритма.

В данную дипломную работу входит пояснительная записка на 52 страницы, введения на 1 страницу, теоретической части, части практической реализации, списка из 20 источников и 1 приложения. Ключевым вопросом в дипломной работе является безопасность передачи данных в сети Интернет между пользователями. Особое внимание уделяется криптографической системе шифрования RSA. Программный код написан с помощью языка программирования C#. Дипломная работа может быть разделена на следующие логические части: описание и суть алгоритма, вспомогательные алгоритмы для шифрования, реализация и сама программа шифрования.

В дипломной работе подробно описывается алгоритм RSA. Сначала мы изучаем его роль в криптографии и безопасности в сети Интернет. Рассматриваем его применимость и систему работы.

В отдельной части дипломной работы подробно рассказывается о принципе работы программы для шифрования алгоритмом RSA, почему была выбрана именно эта платформа и как работает программа.

Примененная методика, позволяет, шифровать и дешифровать данные с использованием криптографической системы шифрования RSA.

ABSTRACT

The title of the graduation work is « Implementation of the cryptographic algorithm RSA ».

The aim of the work is to give some information about algorithm RSA and writing working program for encryption using this algorithm.

The subject of the study is cryptographic algorithm RSA.

The graduation work is devoted to implement algorithm RSA in the programming language C#.

The bachelor's thesis may be divided into several logically connected parts.

The first part of the bachelor's thesis is description and analysis of the RSA algorithm and some algorithms what applying with it.

The second part we consider the qualitative theory of the algorithm and questions related to the simplicity of the number.

The last part describes the software implementation of RSA encryption. It includes the composition of the program, the modules and the composition of the project.

In our world there is a big problem of storing information and even more, keeping it in safety. In order to protect information from unauthorized access and hacking, we need to develop already existing software methods of information encryption, or create new ways. Application of the algorithm RSA in programming allows storing data and keeping it under protection for a long time. For solving this problem, a program was implemented to encrypt by this method in the C# programming language.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С древних времен проблема защиты информации волновала человеческий ум. Вместе с появлением письменности, появилась и необходимость сокрытия смысла написанного. Первые примеры зашифрованного текста можно найти в древних манускриптах Индии и Египта. В течение долгих веков происходило неспешное совершенствование способов шифрования основанных на простейших методах подстановки и перестановки символов. Важным этапом стало изобретение ключа шифрования, который получил широкое применение в эпоху возрождения. Следующим важнейшим шагом в развитии криптографии стало появление механических устройств, таких как роторные машины. Они были изобретены в XIX веке, но широкое применение получили только в XX. И применялись вплоть до появления ЭВМ. В начале XX века криптография окончательно оформилась как самостоятельная наука, использующая множество математических методов.

В середине 1970-х годов случился настоящий прорыв современной криптографии - появились первые асимметричные криптосистемы. Такие системы не требовали наличия скрытого ключа у обеих сторон. Начало этого метода было положено в 1976 году работой «Новые направления в современной криптографии» Уитфилда Диффи и Мартина Хеллмана. В эти исследования впервые в истории криптографии вошли четко сформулированные принципы обмена зашифрованной информации без необходимости обмена секретным ключом. Уже через год, на основании вышеописанного метода, тремя учеными из Массачусетса Роном Ривестом, Ади Шамиром и Леонардом Адлеманом был разработан новый алгоритм шифрования данных, впоследствии получивший название RSA. Этот алгоритм явился первой практической асимметричной криптосистемой, основывающейся на так называемой «проблеме факторизации больших простых чисел». Асимметричная криптография содействовала развитию только появлявшихся на тот момент прикладных областей, таких как

системы электронной цифровой подписи (ЭЦП) и защитные механизмы зарождающихся систем электронных платежей. Так же метод RSA широко используется как дополнительный при шифровании ключа для симметричных систем шифрования.

Широкой применимостью алгоритма RSA и определяется актуальность темы ВКР.

Объект исследования - криптографический алгоритм RSA .

Предмет – реализация алгоритма RSA .

Цель работы – реализовать криптографический алгоритм RSA .

Для реализации цели следует решить задачи:

1. Проанализировать теоретические основания алгоритма RSA.

2. Осуществить программную реализацию алгоритма RSA.

3. Проанализировать различные программные реализации и сравнить их.

ГЛАВА 1 ОПИСАНИЕ И АНАЛИЗ КРИПТОГРАФИЧЕСКОГО АЛГОРИТМА RSA

В 30х годах прошлого века, криптография окончательно сформировалась как наука, основывающаяся на фундаментальных математических дисциплинах, таких как: математическая статистика, теория вероятностей, теория чисел. В результате развития кибернетики и теории алгоритмов появились первые ЭВМ позволяющие оперировать огромными вычислительными ресурсами, что в свою очередь дало ощутимый толчок в развитии криптографических дисциплин.

Теперь невозможно представить современную криптографию без вычислительных машин и электронных средств связи. Они проникали практически во все стези человеческой деятельности. Шифрование и дешифровка определенного массива данных производятся на основе обработки посредством ЭВМ целых чисел. С помощью некоторых функций, определяющихся на множестве целых чисел, можно описать выполнение криптографических методов. Таким образом для решения криптографических задач становится вполне естественным использование математических методов теории чисел. Как следствие этого, сложность теоретико-числовых задач в значительной степени определяет стойкость современных криптографических систем.

При использовании ЭВМ в криптографических построениях математическими методами возникают некоторые ограничения связанные с формализацией данных. Длинные цифровые последовательности необходимо перед шифрованием делить на блоки допустимые возможностями ЭВМ. Так же положим, что в шифровании используются целые, неотрицательные числа, величина которых ограничена неким, определяемым техническими характеристиками числом m. Полученные в ходе шифрования числа будут соответствовать тем же условиям. Такой подход дает возможность считать используемые числа в качестве элементов кольца вычетов Z/mZ . В

результате функцию шифрования можно рассматривать как взаимооднозначное отображение колец вычетов

$$f : Z/mZ \to Z/mZ$$

а число $f(x)$ - сообщение x в зашифрованном виде.

В 1978 году американцы предложили пример функции f, имеющей ряд замечательных преимуществ. На его основе была создана стандартная система шифрования. Название алгоритма состоит из трех имён изобретателей: Рональда Л. Ривеста, Ади Шамира и Леонарда М. Адлемана (R.L.Rivest.A.Shamir.L.Adleman). Создали они его на факультете Массачусетского технологического института.

Созданная функция обладает рядом несомненных достоинств, среди которых относительно быстрые алгоритмы вычисления функции $f(x)$ и обратной ей функции $f^{-1}(x)$, а также то, что имеется возможность быстрого вычисления значения $f^{-1}(x)$ при наличии некоторой секретной компоненты функции $f(x)$. Без использования секретной компоненты вычисление $f^{-1}(x)$ становится практически неразрешимой задачей. Такая задача требует для своего решения настолько много времени, что скорее всего поего прошествии станет неактуальной.

В 1977 году широко известный популяризатор математики Мартин Гарднерв журнале «ScientificAmerican» опубликовал статью посвященную алгоритму шифрования RSA. В русском переводе заголовок статьи выглядел довольно интригующе: «Новый вид шифрования, декодирование которого потребует миллионов лет».В основе статьи лежала копия доклада Массачусетского Технологического института. Завершало статью закодированное сообщение, а так же открытый 129-значный ключ к нему.

$$f(x) = 96869613754622061477140922254355882905759991124574319874695120930816298225145708356931476622883989628013391990551829945157815154$$

m=114381625757888867669325779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541

$e = 9007$

За расшифровку сообщения была предложена немалая награда. Статья привлекла внимание весьма обширных кругов как специалистов, так и любителей, что поспособствовало стремительному прогрессу в этой области последующие десятилетия. Сообщение было расшифровано только через 17 лет, для решения задачи были использованы колоссальные для 1994 года вычислительные ресурсы (более 1600 компьютеров, распределенных в Internet), над проектом работало более 600 человек.

Выглядело это решение так:

$p = 34905295108476509491478496199038981334177646384933878439908205 77$

$q = 32769132993266709549961988190834461413177642967992942539798288533$

В современных криптографических алгоритмах RSA рекомендуются к использованию ключи длинной 2048 бит.

Уже в наше время алгоритм шифрования RSA широко используется для шифрования данных электронной почты и других цифровых транзакций через Интернет и он до сих пор обеспечивает высокую сохранность и конфиденциальность информации.

1.1 Метод шифрования RSA

Функция f, реализующая метод RSA, записывается как:

$$f : x \to x^e \pmod{m} \quad (1)$$

Причем m и e здесь натуральные числа

Таким образом, задача расшифровки сообщения $a = f(x)$ сводится к решению сравнения

$$x^e = a \pmod{m} \quad (2)$$

Для получения единственного значения x необходимо соблюсти некоторые условия для m и e.

Для понимания этих условий и средств достижения решения подойдет функция Эйлера. «φ(m) — мультипликативная арифметическая функция, равная количеству натуральных чисел, меньших n и взаимно простых с ним. $\varphi(1)=1$ и $\varphi(p^r)=p^{r-1}(p-1)$ для любого простого числа p и натурального r. А также, $\varphi(ab)=\varphi(a)\varphi(b)$ для любых натуральных взаимно простых a и b». Использование данных свойств упрощает вычисление значения $\varphi(m)$ для разложения на простые сомножители m. Данные свойства позволяют легко определить значения $\varphi(m)$ при известных простых сомножителях числа m.

При условии, что показатель степени e в сравнении (2) взаимно прост с $\varphi(m)$ - сравнение (2) имеет единственное решение. Для нахождения решения, определяем целое число d, удовлетворяющее условиям

$$de \equiv 1 \pmod{\varphi(m)}, \quad 1 \le d < \varphi(m). \tag{3}$$

Это число существует, поскольку $(e, \varphi(m)) = 1$, и, оно единственное. Согласно теореме Эйлера: «для каждого числа x, взаимно простого с m, выполняется сравнение

$$x^{\varphi(m)} \equiv 1 \pmod{m}$$

и, следовательно:

$$a^d \equiv x^{de} \equiv x \pmod{m}. \tag{4}$$

Таким образом, в предположении $(a, m)=1$, единственное решение сравнения (2) возможно найти в виде:

$$x \equiv a^d \pmod{m}. \tag{5}$$

Выполнение этого сравнения возможно и без предположения $(a, m)=1$ если предположить, что число m состоит из различных простых сомножителей. Обозначим $r=(a, m)$ и $s=m/r$. Тогда $\varphi(m)$ делится на $\varphi(s)$, а из (2) следует, что $(x, s)=1$. Как и (4), теперь достаточно легко найти $x \equiv a^d \pmod{s}$. Так же, имеем

$x \equiv 0 \equiv a^d \pmod{r}$.

Полученные сравнения в силу $(r, s) = 1$ дают нам (5).

Функция (1), для алгоритма RSA, может быть рассчитана довольно быстро. Для вычисления обратной к $f(x)$ функция $f^{-1} : x \to x^d \pmod{m}$ действуют те же правила, что и для $f(x)$, заменяются лишь показатели степени e на d. Таким образом, для функции (1) будут полностью выполняться указанные выше свойства 1) и 2).

Исходя из вышеописанного, чтобы вычислить функцию (1) необходимо знать только числа e и m, которые являются открытым ключом шифрования. При этом для получения значения обратной функции необходимо знать секретную компоненту - d. Создается видимость того, что зная число m, не составит труда разложить его на простые сомножители, далее с помощью известных правил получить значение $\varphi(m)$ и при помощи (3) вычислить требуемое значение ключа - d. Однако достаточно просто выполняются все этапы этого вычисления, за исключением первого. Поскольку разложение числа m на простые множители является самой трудоемкой частью вычислений. Несмотря на многолетнюю историю классической теории чисел и открытия в математических методах современной науки, эффективный алгоритм разложения натуральных чисел на множители обнаружен не был. Практически, есть возможность, перебирая все простые числа вплоть до $\sqrt{m}$, и, разделив на них m, найти требуемое разложение. Однако, если учесть, что количество простых чисел в данном интервале асимптотически равно $2\sqrt{m} \cdot \ln m^{-1}$, находится, что при m, записываемом 100 десятичными цифрами, существуют, по крайней мере, не менее $4 \cdot 10^{42}$ простых чисел, на которые понадобится делить m при разложении на множители. Даже по очень приблизительной оценке видно, что для разложения $m > 10^{99}$ на простые сомножители, современный компьютер, выполняющий миллионы делений в секунду, потребует, не менее, чем

10^{28}лет. Разумеется элементарный перебор простых делителей не выглядит достаточно эффективным, существуют и другие методы разложения целых чисел на множители, но их быстродействие оставляет желать лучшего.

В работах авторов алгоритма RSA предложено определять выбор числа mпроизведением простых множителей p и q примерно одинаковых по величине.

Таким образом, если:

$$\varphi(m) = \varphi(pq) = (p-1)(q-1), \tag{6}$$

то единственным условием для выбора показателя степени e в отображении (1) является:

$$(e,\ p-1) = (e,\ q-1) = 1. \tag{7}$$

Таким образом, для того, чтобы организовать, к примеру, зашифрованную переписку с использованием вышеописанных методов RSA необходимо выбрать два достаточно больших простых числа p и q. Произведение этих чисел даст число $m = pq$. Далее необходимо определиться со значениемe, удовлетворяющим условиям (7). В завершение при помощи (6) определяется значение$\varphi(m)$с и с помощью (3) – значениеd. Значения m и e предаются огласке, ключd остается секретным. В результате произведенных действий, у любого участника переписки появляется возможность отправлять кодированные сообщения с помощью функции (1) организатору системы, и, следовательно, организатор, используя (5) может легко расшифровать эти сообщения,

1.2 Сложность теоретико-числовых алгоритмов

В теории чисел сложилась практика определять сложность алгоритмов путем измерения количества арифметических операций выполненных при исполнении абсолютно всех действий описанных алгоритмом. К сожалению, сложность алгоритма не определяет сложность всей процедуры вычисления, поскольку алгоритмом не предписываются

значения чисел участвующих в вычислениях. Понятно, что для перемножения двухсотзначных чисел требуется намного больше вычислительных ресурсов чем для перемножения двух однозначных, хотя для обоих случаев производится одна арифметическая операция. Таким образом, в некоторых случаях учитываются и величины чисел, сложность при этом оценивается количеством битовых операций.

В нашем случае большее значение для определения сложности имеет все-таки количество используемых арифметических операций. При построении эффективных алгоритмов и нахождения оценок сверху по сложности достаточно понятий поля, в котором должен работать алгоритм. Формализация необходима, при отсутствии алгоритма или доказательств нижних границ сложности.

Для описания примеров алгоритмов и оценки их сложности, нет необходимости придерживаться формального описания алгоритмов. Более важное значение имеет смысл выполняемых действий.

Описанный в примере алгоритм вычисляет $a^d \pmod m$ за $O(\ln m)$ арифметических операций. В этом случае, естественно, считается, что величина натуральных чисел a и d не превосходит значение m.

Определим последовательность вычисления $a^d \pmod m$. Для начала необходимо представить d в двоичной системе $d = d_0 2^r + \ldots + d_{r-1} 2 + d_r$, то есть $d_0 = 1$. Далее примем $a_0 = a$ и для $i = 1 \ldots r$ и вычислим $a_i \equiv a_{i-1}^2 \cdot a^{d_i} \pmod m$. В таком случае a_r является искомым вычетом $a^d \pmod m$

Сравнение $a_i \equiv a^{d_0 2^i + \cdots + d_i} \pmod m$, подтверждает правильность вышеописанных действий.

Сложность алгоритма оценивается значением $O(\ln m)$. Исходя из того, что каждое из вычислений на втором шаге требует не более трех умножений по абсолютной величине $r \le \log_2 m$ раз.

Заметим, что второй алгоритм – это классический евклидовый алгоритм для нахождения наибольшего общего делителя целых чисел для заданных натуральных чисел a и b .

Для нахождения общего делителя этим методом, необходимо вычислить остаток от деления числа a на b, $a = bq + r$, $0 \le r < b$. Обозначим его r. Если $r = 0$, тогда b есть наше искомое число. Если $r \neq 0$, то заменяем исходные числа $\langle a, b \rangle$ парой $\langle b, r \rangle$ и повторяем операцию вычисления остатка от деления.

$$
\begin{aligned}
a &= b \cdot q_1 + r_1, \quad 0 < r_1 < b \\
b &= r_1 \cdot q_2 + r_2, \quad 0 < r_2 < r_1 \\
r_1 &= r_2 \cdot q_3 + r_3, \quad 0 < r_3 < r_2 \\
r_2 &= r_3 \cdot q_4 + r_4, \quad 0 < r_4 < r_3 \\
&\vdots \\
r_{k-2} &= r_{k-1} \cdot q_k + r_k, \quad 0 < r_k < r_{k-1} \\
r_{k-1} &= r_k \cdot q_{k+1}
\end{aligned}
$$

После незначительно правки алгоритма Евклида, появляется возможность решения сравнения $ax \equiv 1 \pmod{b}$ где $(a,b) = 1$. Эта задача равносильна поиску целых решений уравнения $ax + by = 1$.

Для решения уравнения $ax + by = 1$ необходимо определить матрицу $E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, далее вычисляем остаток r от деления числа a на b, $a = bq + r$, $0 \le r < b$. В случае $r = 0$, второй столбец матрицы E даёт вектор $\begin{pmatrix} x \\ y \end{pmatrix}$ возможных решений уравнения. Если $r \neq 0$, заменяем исходную матрицу, матрицей $E \cdot \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$. Повторяем вышеописанные действия до получения $r = 0$.

Все три описанных выше алгоритма являются так называемыми полиномиальными алгоритмами. Алгоритмы, сложность которых оценивается сверху степенным образом, зависимо от длины записи входящих чисел и носят название полиномиальных. Сложно алгоритмов подобного типа определяется величиной $O(\ln^c m)$, где c – является некой константой и равна единице ($c = 1$), когда наибольшее из чисел, которые обрабатываются алгоритмом, не превосходит m.

Однако, такое явление в теории чисел как полиномиальные алгоритмы встречается довольно редко. Собственно как и оценки сложности алгоритмов обычно основаны на каких-либо не доказанных, но правдивых гипотезах, которые зачастую относятся к аналитической теории чисел.

Хоть и в части задач действенные алгоритмы совсем не известны, порой в таких случаях все же предоставляется возможным предложить определенную последовательность действий, быстро приводящую к нужному результату, «если повезет». Также есть алгоритмы, которые выдают правильный результат – вероятностные. Но им присуща вероятностная оценка времени работы. Работа этих алгоритмов, в обычных случаях, зависит от одного или ряда параметров и при наихудшем исходе такие алгоритмы работают слишком долго. При лучшем раскладе, выбранный параметр обеспечивает мало затратное по времени завершение работы. Эти алгоритмы, при достаточном количестве «хороших» значений, при реализации работают достаточно эффективно, но при этом хороших оценок сложности они не имеют.

Для примера, можно рассмотреть вероятностный алгоритм, который позволяет решать по простому модулю полиномиальные сравнения. Допустим p — простое число, и оно мнимо большое, а $f(x) \in Z[x]$ - многочлен, степень которого предположительно ограничена. Задачей является нахождение решений сравнения.

$$f(x) \equiv 0 \pmod{p}. \qquad (8)$$

Если степень многочлена $f(x)$ равна 2, возможно решение квадратичных сравнений. Иначе говоря, необходимо найти в поле $F_p = Z/pZ$ все элементы, которые будут удовлетворять уравнению $f(x) = 0$.

По малой теореме Ферма, все элементы поля F_p – однократные корни многочлена $x^p - x$. Следовательно, при вычисление наибольшего общего делителя $d(x) = (x^p - x, f(x))$, мы определяем многочлен $d(x)$, где, в поле F_p, множество корней совпадет с множеством корней многочлена $f(x)$, при том все эти корни однократны. В случае, когда многочлен $d(x)$ имеет нулевую степень (лежит в поле F_p) сравнение (8) не имеет решений.

Чтобы вычислить многочлен $d(x)$ сначала лучше вычислить, пользуясь алгоритмом возведения в степень, подобно приведенному выше, многочлен $c(x) \equiv x^p \pmod{f(x)}$. После, при помощи аналога алгоритма Евклида вычислить $d(x) = (c(x) - x, f(x))$. Выполняется все это за полиномиальное количество вычислительных операций.

Если продолжить искать возможные решения сравнения $f(x) \equiv 0 \pmod{p}$, то можно предположить, что в кольце многочленов $F_p[x]$ будет справедливо равенство

$$f(x) = (x - a_1) \cdot \ldots \cdot (x - a_n),\ a_i \in F_p,\ a_i \neq a_j$$

Таким образом, продолжаем решать задачу путем нахождения делителей многочлена $f(x)$ в кольце $F_p[x]$. Для этого выбрав элемент $\delta \in F_p$. Находим наибольший общий делитель.

$$g(x) = \left(f(x), (x + \delta)^{\frac{p-1}{2}} - 1 \right)$$

В случае если многочлен $g(x)$ окажется собственным делителем $f(x)$, многочлен $f(x)$ распадётся на два множителя, причем с каждым из них независимо необходимо проделать все операции, предусмотренные данным

методом. В случаях $g(x)=1$ или $g(x)=f(x)$, необходимо просто выбрать новое значение δ и произвести вычисления сначала.

В данном случае, количество операций при нахождении общего делителя оценивается величиной $O(\ln p)$.Остается выяснить самый важный вопрос, а именно, насколько долго придётся вести перебор числа δ, для нахождения собственного делителя $f(x)$.

При всём том, что количество всех решений уравнения $(x+a_1)^{\frac{p-1}{2}}=(x+a_2)^{\frac{p-1}{2}}$ в поле F_p не превосходит $\frac{p-3}{2}$. Становится вполне определенным, что удовлетворяющее условиям

$$(\delta+a_1)^{\frac{p-1}{2}}\neq(\delta+a_2)^{\frac{p-1}{2}},\ \delta\neq a_1,\delta\neq a_2$$

подмножество $D\subset F_p$ элементов δ ,состоит более, чем из $\frac{p-1}{2}$ элементов.

В итоге приходим к выводу, что значений δ, при которых многочлен $f(x)$ распадётся на два собственных множителя не менее чем $\frac{p-1}{2}$.

Подытожим вышесказанное. При произвольном выборе $\delta\in F_p$, вероятность не разложении на множители данного многочлена после k повторений не превосходит 2^{-k} . Сростом k вероятность стремительно убывает. На практике подтверждается достаточно эффективная работа этого метода.

Необходимо отметить, что при оценке вероятности мы использовали всего два корня многочлена $f(x)$. При усложнении задачи и увеличении $n>2$ эта вероятность значительно уменьшается. При проведении более тонкого анализа становится ясно, что вероятность для многочлена $f(x)$ не распасться на множители при однократном проходе алгоритма не превосходит $2^{-n}+O(p^{-1/2})$.

Если в сравнении $f(x) \equiv 0 \pmod{p}$ заменить простой модуль p составным модулем m, то практическая задача нахождения решений сравнения становится очень сложной.

ГЛАВА 2 ЗАДАЧИ ПРАКТИЧЕСКОЙ РЕАЛИЗАЦИИ АЛГОРИТМА

В практической реализации качественного алгоритма RSA,для получения продукта с высокой криптоустойчивостью существует несколько важных проблем, которые связаны с большими числами и операциями над ними. Вспомним вкратце механизм алгоритма:

1. Берем простые числа p и q.
2. Находим n = p * q.
3. Вычисляем m = (p - 1) * (q - 1).
4. Подбираем d взаимно простое с m.
5. Находим ключ e, такой чтобы e * d = 1 (mod m).

Таким образом данным алгоритмом реализуется создания как открытого, так и закрытого ключа.

На основании этих данных, с помощью открытого ключа производится шифрование,адля дешифровки данных необходим так же и закрытый ключ. Эти действия можно отобразить формулами:

$b = a^e \pmod n$ – шифрование;

$a = b^d \pmod n$ – дешифровка.

На современном этапе развития вычислительной техники, для алгоритма RSA рекомендуется использовать ключи длиной не менее 768 бит, а для особо секретных данных, не менее 2048 бит.Причем в ближайшей перспективе, в связи с ростом электронных технологий и вычислительных мощностей, требования к криптостойкости, а соответственно для RSA и к длине ключа будут только расти. Исходя из условий алгоритма, при такой длине ключа числа pиq всего в 2 раза короче. Для выполнения действий со столь большими числами необходимо использование специальных алгоритмов и структур данных. Для реализации алгоритма RSA необходимо решение ряда задач: генерация простых и взаимно простых чисел, сложение, деление, умножение, возведение в большую степень по модулю целого числа, определение остатка от деления, хранение чисел.

2.1 Проверка большого числа на простоту

Первая же проблема возникает при выборе простых чисел pиq, для достижения достаточной криптоустойчивости. Самый простой и самый точный способ – разделить предполагаемое простое число на все числа меньше его, но он труднореализуем уже при разрядности 32 бит. Существует множество способов определения простоты большого числа с вероятностью стремящейся к 1. Рассмотрим некоторые способы, относительно просто реализуемые программными средствами.

Оптимизированный перебор делителей. Этот метод основан на утверждении, что наименьший делитель составного числа n не превосходит √n. Работа этого метода основана на доказательстве этого утверждения от противного. Примем число k являющееся наименьшим делителем n, причём $k > \sqrt{n}$. Тогда $n = k$ l, где $l \in N$, причём $l \leqslant \sqrt{n}$, таким образом l также является делителем числа n, к тому же, меньшим, чем k, что противоречит нашему предположению. При переборе потенциальных делителей, можно оборвать перебор, как только k достигнет $\sqrt{n}$: в случае если делителей до окончания перебора не найдено, то их не существует. Чем больше значение проверяемого числа, тем больше выигрыш в производительности по сравнению с простым перебором.

Еще один метод - Решето Эратосфена, его приписывают древнегреческому учёному Эратосфену Киренскому, но не смотря на такой древний возраст алгоритма, он прекрасно реализуется программными средствами. Для использования этого метода необходимо взять числа от 2 до n. В оригинальном варианте внести их в таблицу. Зачеркнув чётные числа, следующие после двойки, отмечаем двойку. Далее находим следующее за двойкой не зачёркнутое число - тройку, обводим его, и зачеркиваем каждое третье, начиная с шести, число после тройки. Далее снова находим первое после тройки незачёркнутое число, после чего зачеркнём каждое пятое. Подобные действия необходимо повторять, пока все числа в таблице не

будут отмечены тем или иным образом. Все не зачёркнутые числа будут в точности простыми.

Эти методы являются достаточно простыми. Если рассматривать более сложные варианты, то несомненный интерес вызывает так называемый тест AKS. Название теста представляет собой аббревиатуру, по первым буквам фамилий ученых, его создавших – Агравал, Кайал, Саксен. Этот алгоритм получил широкое распространение и является единственны полиномиальный детерминированный алгоритм проверки числа на простоту. Сложность этого алгоритма составляет $O(log^{19} N)$

Описание этого теста выглядит так: «Если существует $r \in Z$ такое что$O_r\ n\ > log^2 n$ и для любого a от 1 до $\overline{\varphi(r)}\log(n)$ выполняется сравнение $(x+a)^n \equiv\ x+a\ ^n\ mod x^r, n$, то n – либо простое число, либо степень простого числа».

2.2 Нахождение взаимно простых чисел и алгоритм Евклида

В ходе реализации алгоритма RSA возникает необходимость нахождения числа d взаимно простого с m, то есть у этих чисел не должно быть общих делителей. При этом число d должно быть меньше m, поскольку разрядность числа d определяется суммой бит чисел p и q . Что бы реализовать поиск взаимно простых чисел необходим алгоритм находящий наибольший общий делитель этих чисел. Это алгоритм Евклида. На практике алгоритм работает очень быстро.

Для определения наибольшего общего делителя по алгоритму Евклида, необходимо вычислить остаток - r от деления числа a на b , $a = bq + r$, $0 \le r < b$.

В случае получения нулевого остатка делаем вывод о том,что b есть искомое число. Если же $r \neq 0$, то необходимо заменить исходную пару чисел $\langle a, b \rangle$ парой $\langle b, r \rangle$ и вновь приступим к вычислению остатка.

Существует теорема для алгоритма Евклида: «Алгоритм Евклида требует не более $5p$ операций деления с остатком, где p есть количество цифр в десятичной записи меньшего из чисел a и b».

Для доказательства возьмем.

$r_0 = a > b$, $r_1, r_2, \ldots, r_n$ - некую последовательность делителей, появляющихся в процессе вычисления остатка от деления a на b, следуя алгоритму Евклида. Тогда

$$r_1 = b, \ldots, 0 \le r_{i+1} < r_i, \quad i = 0,1, \ldots m-1.$$

Далее задействуем последовательность Фибоначчи

$u_0 = 1,\ u_1 = 1,\ u_{k+1} = u_k + u_{k-1},\ k \ge 1.$

Неравенство $r_{i+1} \ge u_{n-i}$ доказывается индукцией по i от $i = n-1$ до $i = 0$ $r_{i+1} \ge u_{n-i}$. Поскольку $u_n \ge 10^{n-1/5}$, то неравенства $10^p > b = r_1 \ge u_n \ge 10^{n-1/5}$ и $n < 5p+1$.

2.3 Определение ключа алгоритма RSA

Для определения ключа, необходимо найти число e, полностью удовлетворяющее условию:

$e * d = 1 \ mod \ m$.

Для взаимнопростых чисел d и m используют модифицированный алгоритм Евклида. Для решения этой задачи следует решить уравнение $\mathrm{m} * \mathrm{x} + \mathrm{d} * \mathrm{e} = 1$ в натуральных числах. Количество производимых операций и скорость работы алгоритма приблизительно соответствуют параметрам алгоритма Евклида.

Следуя этому методу определения ключа, необходимо сначала определить матрицу:

$$\mathrm{E} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Далее необходимо вычислить остаток от деления a на b.

$a = b * q + r.$

x
y

В случае r = 0, решение определяется по второму столбцу матрицы:

$$E = E * \begin{bmatrix} 0 & 1 \\ 1 & -q \end{bmatrix}$$

Если вычисленный остаток не равен нолю, необходимо произвести замену пары чисел <a, b>, парой <b, r> и повторить операцию вычисления остатка.

Следуя данному методу, необходимо производить все вычисления по модулю большего из чисел a и b. Отрицательное число $-q$ заменяется положительным, для получения которого необходимоиз числа, взятого в качестве модуля вычесть число q.

К примеру, в случае если из чисел a и b большим будет число a, то выполнять все вычисления можно по модулю a, при этом $-q$ будет представлено как $a - q$.

2.4 Практическая математика для длинных чисел

При практической реализации алгоритма RSA появляется необходимость использования методов «длинной математики» при проведении вычислений. Это обусловлено прямой зависимостью криптоустойчивости данных, шифрованных по методу RSA, от длины ключей шифрования.

Хранить числа удобнее в массиве состоящем из 2-х байтовых переменных. В программах на большинстве языков программирования высокого уровня удобнее производить умножение двухбайтовых переменных, поэтому четырёхбайтовый результат лучше разделить на две части для дальнейшей обработки.

Существуют сложные и эффективные алгоритмы для умножения длинных чисел, такие как: умножение Карацубы, умножение посредством преобразования Фурье. Но в нашем случае проще воспользоваться классическим методом "в столбик", при котором промежуточные результаты

можно вычислять в разном порядке. Например, сначала умножается весь первый множитель на первую цифру второго, далее на вторую, третью…результаты при этом записывая строками. В итоге получится матрица, после построчного сложения которой, получим результат. Однако для некоторых случаев эффективен подход вычисления матрицы по столбцам, а не по строкам. Для получения итогового результата значения столбцов складываются между собой.

Большое значение для данной реализации имеет алгоритм возведения в степень по модулю натурального числа. Для этого случая вполне подойдет довольно простой алгоритм, который сводит возведение в степень к серии умножений. Таким образом, алгоритм возведения в степень по модулю, состоит из собственно «алгоритма возведения в степень» и получения остатка от деления результата на n, причем в результате необходим только остаток, частное в этом случае не имеет значения. В получившемся едином алгоритме производятся две операций одновременно, таким образом метод RSA реализовывается в полном объеме.

2.5 Криптоустойчивость RSA

Насколько бы не была сложной криптосистема, всегда найдутся желающие ее взломать. Известны несколько способов взлома систем RSA. Самый эффективный вариант - нахождение секретной компоненты, которая соответствовала бы используемому открытому ключу. Злоумышленник в этом случае получит полный доступ к данным зашифрованным открытым ключом и получит возможность подделывать подписи. Для успеха такой атаки, необходимо найти главные сомножители p и q. Закрытый ключ d легко вычисляется при наличии у взломщика компонент p, q и e. Таким образом безопасность RSA зависит от разложения на сомножители n,что при достаточной длине ключа труднодостижимо. В современном криптоанализе не существует эффективных способов решения такой задачи.

Существует еще один способ полного взлома схем RSA. Он основан на вычислении корня степени e из $mod\ n$. Исходя из того, что $C = M ** e * (mod\ n)$, корнем степени e из $(mod\ n)$ является само сообщение M. Таким образом, для такого взлома не нужен закрытый ключ, достаточно вычислить корень. Такой метод взлома пока существует только теоретически, практические методы на данный момент не известны.

Возможны и атаки позволяющие узнать содержимое только одного сообщения. Например «атака по предполагаемому открытому тексту». Для такой атаки необходимо кроме зашифрованного текста иметь и часть этого же сообщения открытым текстом. Далее текст шифруется открытым ключом и сравнивается с шифрованным оригиналом.

ГЛАВА 3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА

3.1 Описание состава программных средств

Криптографический алгоритм RSA был реализован с помощью интегрированного пакета C#. Этот язык – объектно-ориентирован. Основан на формах, которые интегрированы с программированием для Windows и использующих компонентную технологию. Среда визуального программирования C# способствует компонентному подходу к созданию приложений, и позволяет, без особых затрат времени, ёмко и качественно "собрать" интерфейс программы, для того, чтобы большую часть времени тратить именно для реализации составленного алгоритма .

Компилятор в машинный код обеспечивает высокую производительность, которая является необходимой для построения приложений. Этот компилятор в настоящее время является самым быстрым в мире. Он позволяет легко разбирать и проводить быстрые проверки готового программного блока, а также обеспечивать качество кода. Кроме того, C# обеспечивает эффективную и не затратную по времени разработку без необходимости писать вставки на C# или заниматься написанием кода вручную (хотя это возможно).

3.2 Описание модулей программы

Программа включает в себя два программных модуля.

Модуль содержит главную кнопочную форму, представлено на рисунке 1.

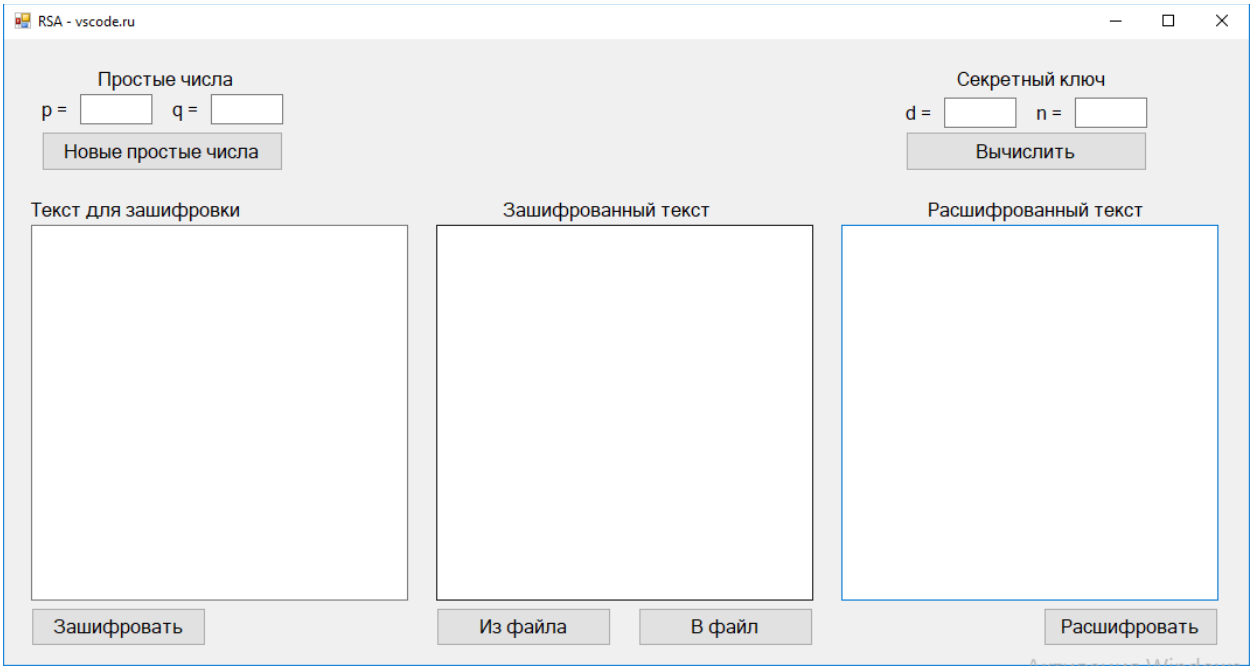

Рисунок 1 - Главная кнопочная форма

Заголовок главной формы изменен на «RSA». Компонент bi_Maximize свойства BorderIcons установлен в значение False, а свойство BorderStyle изменено на bsSingle.

Форма Form1 содержит следующие компоненты: buttonEncrypt, buttonDecipher, label1, textbox_p, label2, textbox_q, label3, label4, label5, label6, textbox_d, textbox_n, textbox_ToEncrypt, textbox_Encrypted, textbox_Decrypted, bt_NewPrimeNumbers, label7, label8, label9, bt_FromDisk, bt_ToDisk, bt_CalculateKey[1].

Первоначальные свойства компонентов изменены следующим образом:

buttonEncrypt:

Caption – «Зашифровать»;

buttonDecipher:

Caption – «Расшифровать»;

bt_FromDisk:

Caption – «Из файла»;

[1] Начальные индексы обозначают наименования компонентов в библиотеке C#: bt – Button.

bt_ToDisk:

Caption – «В файл»;

Bt_NewPrimeNumbers:

Caption – «Новые простые числа»;

bt_CalculateKey:

Caption – «Вычислить»;

Enabled – False;

Свойство Enabled всех компонентов Edit, кроме textbox_p и textbox_q, изменены на False, а свойство Text – на пустую строку. На этапе шифрования они служат лишь для вывода информации о ключах. Компоненты textbox_p и textbox_q служат для ввода двух случайных чисел, на основе которых находятся два ближайших простых числа и генерируются открытый и секретный ключи.

Компоненты Label служат для пояснения назначения каждого из расположенных на форме компонентов. Их свойств Caption изменено следующим образом: label1- «Простые числа», label2 – « p = », label3 – « q = » label4 – «Секретный ключ», label5 – « d = », label6 – « n = », label7 – «Текст для зашифровки», label8 – «Зашифрованный текст», label9 – «Расшифрованный текст»

Свойство Enabled компонентов textbox_ToEncrypt и textbox_Decrypted было изменено на False.

Свойство Filter компонентов OpenDialog1 и OpenDialog2 было изменено следующим образом: «Текстовые файлы (*.txt)|*.txt|Все файлы (*.*)|*.*», а свойство FilterIndex обоих компонентов было установлено в значение 1, что в соответствии со свойством Filter обозначает открытие текстовых файлов по умолчанию.

Программный код модуля приведен в приложении А.

3.3 Состав проекта

Таблица 2 Состав проекта

Наименование	Обозначение	Примечание
Crypto_v2.exe	Исполнимый файл	-
Settings.settings	Файл параметров проекта	Содержит текущие установки проекта.
protect_inf_LR1.csproj	Файл проекта	Связывает все файлы, составляющие приложение
Assemblyinfo.cs	Файл сборки	Управление общими сведениями о сборке
App.config	Файл конфигурации проекта	Содержит установки конфигурации проекта
Resources.resx	Файл ресурсов	Содержит пиктограммы, графические изображения и другие ресурсы, используемые в проекте
Form1.cs	Файл программного модуля для форм	Определяет функциональность формы
Crypto_v2.pdb	Файл отладки	Содержит отладочные данные и сведения о состоянии проекта
Protect_inf_LR1.csproj.FileListAbsolute	Состав проекта	Описывает состав файлов проекта

3.4 Описание программы

Программная реализация криптографического алгоритма RSA разработана под операционную систему Windows XP/7/8/10.

Программа написана на объектно-ориентированном языке программирования высокого уровня C#.

Программа представляет собой приложение, способное зашифровывать загружаемые пользователем файлы, генерируя открытый и секретный ключи на основе введенных случайных чисел, а также расшифровывать ранее зашифрованные файлы.

Программа состоит из одного модуля. Модуль Form1.cs содержит главную форму. На форме расположены основные компоненты, позволяющие производить все необходимые операции по шифрованию и расшифровке файлов.

Модуль содержит следующие процедуры.

Кнопка bt_FromDisk активирует процедуру «procedure TForm1.button_CryptFileOpenClick (Sender: TObject);», которая запускает диалоговое окно открытия файла для шифрования. В случае открытия текстового файла его содержимое будет выведено в текстовом поле TextBox_ToEncrypt. Кроме открытия файла, данная процедура делает доступными кнопки кодирования (button_Encrypt) и генерации ключей (button_CalculateKey), очищает содержимое полей textbox_Enctypted и textbox_Decrypted, если в них уже находился какой-либо текст, и очищает поля для ввода чисел и вывода сгенерированных ключей, вызывая процедуру «procedure Clear (Sender: TObject);».

Кнопка button_Decipher вызывает процедуру «procedure TForm1.btn_DecryptClick (Sender: TObject);». Данная процедура запускает процесс расшифровки файла, который должен быть предварительно открыт. Расшифровка требует ввода пары чисел, составляющих секретный ключ. Для этого предназначены поля ввода textBox_d и textBox_n, которые становятся доступными после открытия файла для расшифровки. Если не введено хотя бы одно число из этой пары, программа выдает сообщение об ошибке.

Кнопка bt_FromDisk запускает процедуру открытия файла для расшифровки «procedure TForm1.btn_DecryptFileOpenClick (Sender: TObject);». Данная процедура выводит диалоговое окно открытия файла, подобное представленному на рисунке 3. Кроме того, она выполняет очистку текстовых полей textbox_ToEncrypt и textbox_Decrypted, делает недоступными кнопки кодирования (button_Encrypt) и генерации ключей (btn_CalculateKey), свойство кнопки дешифрования (button_Decipher) устанавливает в значение True и очищает поля для ввода чисел и ключей процедурой «procedure Clear (Sender: TObject);». Свойство PasswordChar компонентов textbox_n и textbox_d, служащих для ввода чисел, составляющих секретный ключ, устанавливается в «#». Это делается затем, чтобы невозможно было подсмотреть секретный ключ во время его ввода пользователем. Кнопка button_Encrypt активирует процедуру «procedure

TForm1.btn_EncryptClick (Sender: TObject);». Эта процедура запускает процесс шифрования файла сгенерированными ключами. Генерация ключей производится на основе двух введенных пользователем простых чисел. Если введенные числа не позволяют рассчитать секретный ключ, выводится сообщение об ошибке.

Кнопка btn_CalculateKey запускает процедуру «procedure TForm1.btn_KeyGenerateClick (Sender: TObject);», которая генерирует секретный и открытый ключи на основе введенных пользователем в поля ввода textbox_p и textbox_q случайных чисел. Если числа не введены, программа выводит сообщение об ошибке. Сгенерированные ключи отображаются в соответствующих компонентах Edit. Открытый ключ d отображается компонентом textbox_d, закрытый – textbox_n. Остальные поля предназначены для вывода дополнительной информации.

Кнопка btn_ToDisk запускает процедуру «procedure TForm1.btn_SaveToFileClick (Sender: TObject);». Данная процедура открывает диалоговое окно сохранения файла. Кнопка btn_ToDisk становится доступной только в том случае, если текстовое поле textbox_ToEncrypt содержит какую-либо информацию, то есть если в этом поле содержится зашифрованная информация, которую можно сохранить в файл.

Помимо вышеперечисленных процедур, модуль включает также следующие процедуры и функции:

«procedure Clear (Sender: TObject);» - производит очистку полей ввода чисел и ключей, а также дополнительной информации о ходе их генерации. Обнуляет ключи.

«function MemodN (m, e, n: int64): longword;» - реализует процесс кодирования числовых блоков по формуле $C(i) = M(i)^e \bmod n$ и раскодирование по формуле $M(i) = C(i)^d \bmod n$.

«function CodeAlgoritm (CodeText: String; se, sn: LongWord): string;» - разбивает сообщения на блоки, преобразует их в числовое представление и передает работу функции MemodN.

Тексты вышеперечисленных процедур и функций находятся в приложении А.

Для полноценного функционирования программы необходимо: персональный компьютер с процессором тактовой частоты в 1.6 ГГц или более , имеющий не менее 1 Гб оперативной памяти(1.5 Гб при выполнении в виртуальной машине); жесткий диск 5400 об/мин; видеоадаптер с поддержкой DirectX 9 (разрешение 1024x768 и выше). При увеличении чисел, на основе которых генерируются ключи, может потребоваться процессор, обладающий большей производительностью, поскольку время генерации ключей обратно пропорционально производительности процессора. Мелким и средним предприятиям и частным лицам рекомендуется использовать ключи длиной от 56 до 96 бит, так как им не требуется обеспечение секретности на несколько лет вперед.

Для запуска программы необходимо вставить носитель информации в соответствующее устройство чтения (накопитель на гибких магнитных дисках), скопировать исполняемый файл «Crypto_v2.exe» в отдельную папку на жестком диске и произвести двойной щелчок левой кнопкой мыши по исполняемому файлу. После запуска приложения на экран выводится главное окно, представленное на рисунке 2. Для генерации ключей необходимо ввести два случайных числа в поля, либо сгенерировать их кнопкой генерации новых простых чисел, расположенные возле пояснительной надписи «Простые числа», и нажать кнопку «Вычислить». В том случае, если хотя бы одно число не будет введено, программа выдаст сообщение об ошибке и ключи сгенерированы не будут.

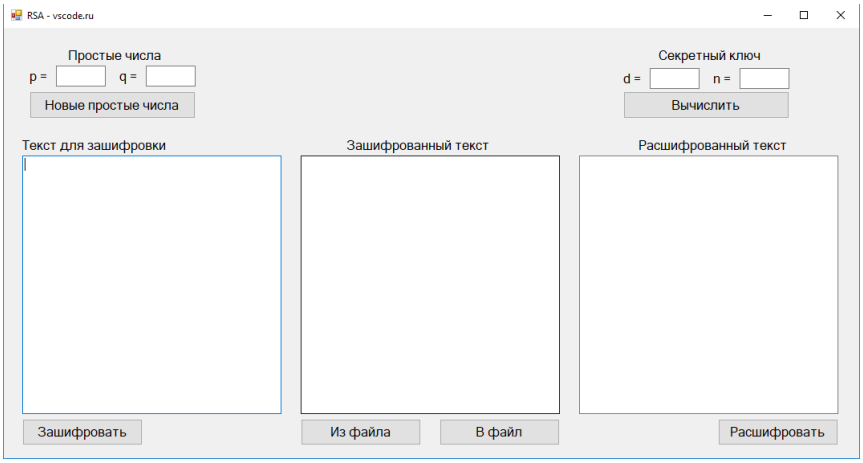

Рисунок 2 - Главное окно приложения

После успешной генерации ключей они будут выведены в соответствующие поля. При этом пара чисел e и n будет являться открытым ключом, а пара d и n – секретным. Пользователь может распространить открытый ключ, разослав его другим пользователям, от которых он желает получать зашифрованные сообщения и файлы. Секретный ключ d необходимо сохранить в тайне – он будет использоваться исключительно для расшифровки.

Для того, чтобы зашифровать файл, необходимо нажать на кнопку «Из файла» и в появившемся диалоговом окне «Открыть» выбрать необходимый файл, представлено на рисунке 3. В случае необходимости шифрования текстовых сообщений можно воспользоваться текстовым полем «Текст для зашифровки», набрав в нем нужный текст.

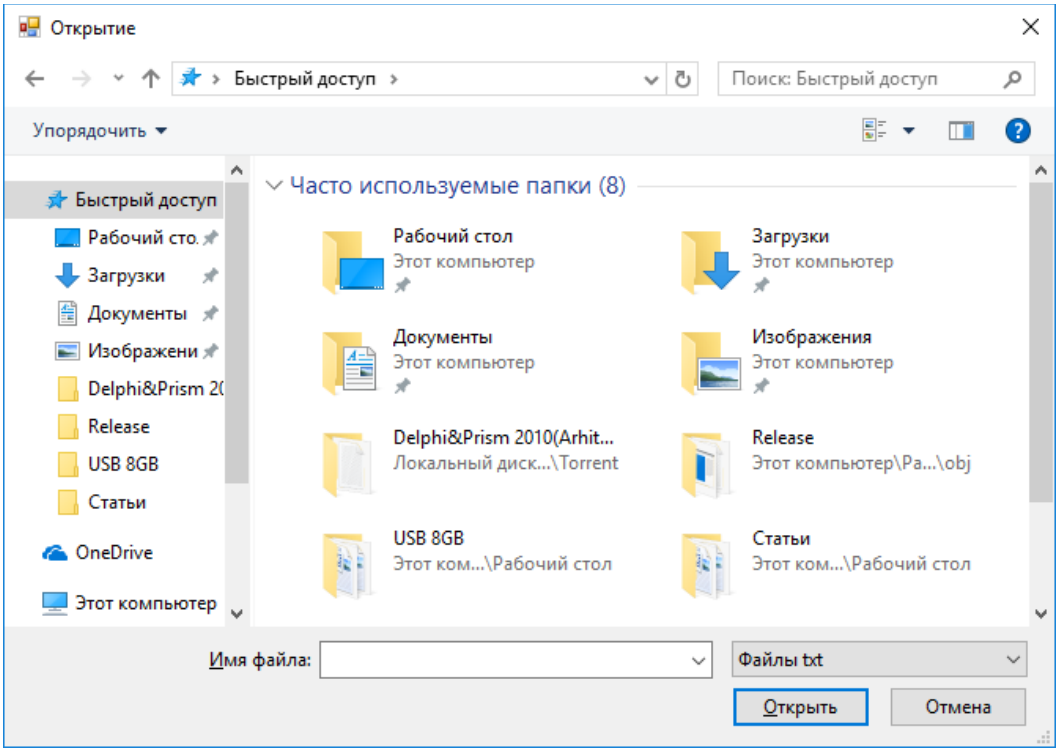

Рис. 3 Диалоговое окно открытия файла

Для шифрования введенного текста или открытого файла требуется нажать кнопку «Зашифровать». Результат работы алгоритма шифрования будет выведен в текстовом поле «Зашифрованный текст». Если процесс генерации ключей не был завершен успешно, то при попытке шифрования будет выведено сообщение об ошибке и процесс кодирования прервется.

После того, как информация будет зашифрована, можно будет сохранить файл кнопкой «В файл», которая позволяет сохранить зашифрованную информацию в отдельный файл. При нажатии этой кнопки открывается диалоговое окно «Сохранить как», представлено на рисунке 4. От пользователя требуется ввести имя файла и путь, по которому он будет сохранен на носителе информации. В том случае, если такой файл уже существует, будет выдано соответствующее сообщение и предложение заменить уже существующий файл новым. Для расшифровки файла, который был ранее зашифрован, необходимо нажать на кнопку «Открыть файл для расшифровки» и в диалоговом окне открытия файла, представлено на рисунке 3, выбрать зашифрованный файл. Содержимое текстовых файлов в

данном случае будет выведено в верхней части поля «Зашифрованный текст».

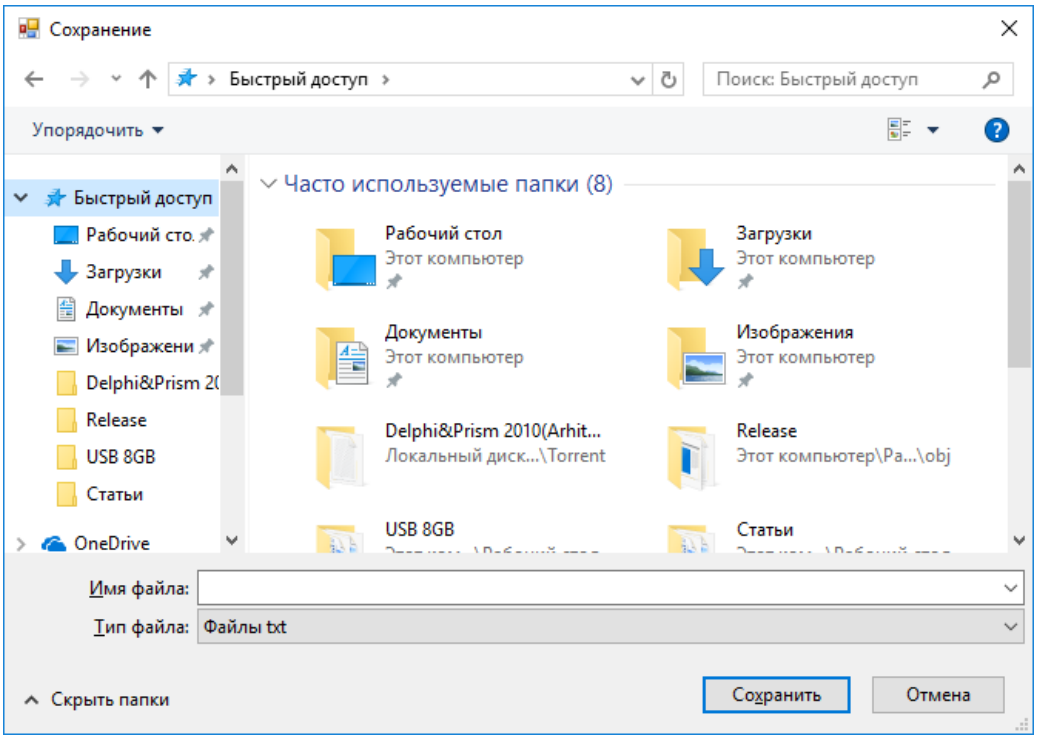

Рисунок 4 - Диалоговое окно сохранения файла

Процедура расшифровки открытого ранее файла требует ввода числа n и секретного ключа d, которые должны были быть сохранены после генерации ключей. Каждый вводимый при этом символ будет отображаться на экране в виде символа «#», поскольку в этом случае сводится к минимуму угроза подглядывания секретного ключа посторонним лицом при вводе.

Если не было введено хотя бы одно число из пары, составляющей секретный ключ, программа выдаст сообщение об ошибке, и процесс расшифровки прервется.

Для завершения работы приложения необходимо либо нажать кнопку «Выход», либо щелкнуть мышью по кнопке закрытия главного окна в его заголовке.

Во время работы программы возможно появление следующих сообщений.

Если пользователем не введено хотя бы одно из случайных чисел, необходимых для генерации ключей, будет выведено сообщение об ошибке, представлено на рисунке 5.

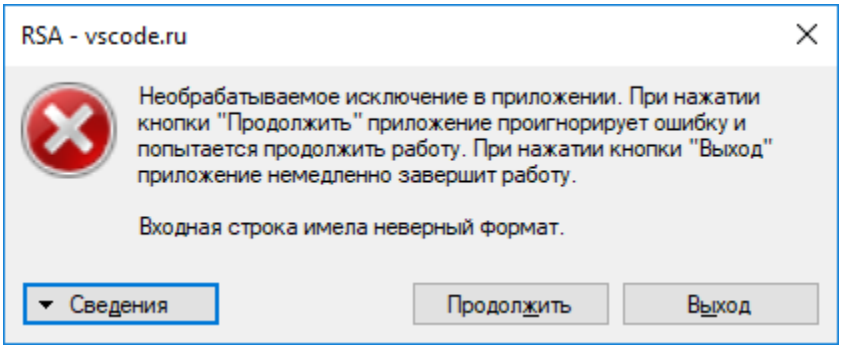

Рисунок 5 - Сообщение об ошибке в процессе генерации ключей

Если пользователем введены не простые числа, по которым нельзя рассчитать секретный ключ d, будет выведено сообщение, представлено на рисунке 6.

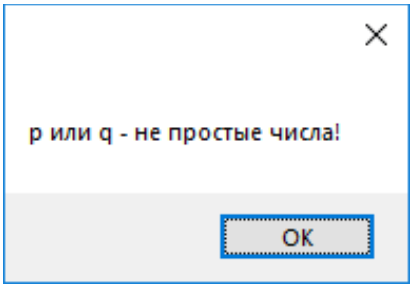

Рисунок 6 - Сообщение об ошибке при расчете секретного ключа

Если пользователь не ввел хотя бы одно число из пары, составляющей закрытый ключ, при расшифровке файла, будет выведено сообщение об ошибке, представленное на рисунке 7.

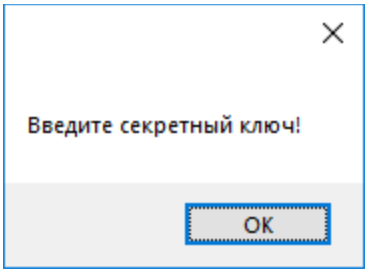

Рисунок 7 - Сообщение об ошибке в процессе дешифрования

В данном разделе было описано проектирование и использование программного комплекса, предназначенного для шифрования файлов как для передачи по сети, так и для безопасного хранения их на носителях информации.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы программно реализован криптографический алгоритм RSA и разработана программа на языке C#, демонстрирующая работу данного алгоритма. Программа способна зашифровывать загружаемые пользователем файлы, генерируя открытый и секретный ключи на основе введенных случайных чисел, а также может расшифровывать ранее зашифрованные файлы.

Входные данные для шифрования файлов – два случайных числа, а также файл, который предварительно должен быть открыт (или текст, введенный в предназначенном для этого текстовом поле). На основе введенных пользователем чисел определяются два ближайших к ним простых числа. Число n – результат их перемножения, – будет входить в пары, являющиеся открытым и закрытым ключами. При этом пара чисел p и q является открытым ключом, а d и n – секретным.

На основе анализа современных методов и средств защиты информации в сетях и перспектив развития информационных технологий, можно выделить основные факторы, затрудняющие решение проблем информационной безопасности в ЭВМ и их сетях:

- массовое и повсеместное применение;

- стабильно возрастающая сложность функционирования;

-множественные варианты программного обеспечения ПК, архитектурных решений, и легкая приспособляемость для выполнения различных задач пользователей.

Криптографические средства являются одним из множества средств информационной защиты в ЭВМ и их сетях. Их задача состоит в том, чтобы защищать информацию при передаче по линиям связи, хранении на носителях, а так же защищает от ввода ложной информации.

Практическая реализация криптографических средств защиты может быть программной, то есть шифрование и дешифрование информации реализуется специальной программой, и технической, с помощью

определенных технических средств, которые могут реализовать алгоритм шифрования.

Данный программный модуль может применяться любыми организациями, сохранение конфиденциальности информации которых имеет важное стратегическое значение. Кроме того, его можно использовать в составе других программных комплексов, к примеру, для шифрования паролей в целях разграничения доступа к ресурсам.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Алферов А.П. Основы криптографии. Учебное пособие. 2-е изд., испр. и доп./Алферов А.П., Зубов А.Ю., Кузьмин А.С., - М.: Гелиос АРВ, 2002. – 480 с., ил.

2. Бабаш А.В. Криптография. Под редакцией А.В. Бабаш, Г.П. Шанкин. – М.: СОЛОН-Р, 2007. – 512с. – (Серия книг «Аспекты защиты»). ISBN 5-93455-135-3

3. Баричев С.Г., Серов Р.Е. Основы современной криптографии. Учебное пособие. — М.: Горячая Линия — Телеком, 2006. — 152 с.

4. Болотов А.А. Элементарное введение в эллиптическую криптографию: Алгебраические и алгоритмические основы./Болотов А.А., Гашков С.Б., Фролов А.Б. – М.: КомКнига, 2012. – 306 с.

5. Брассар Ж. Современная криптология. Руководство. Москва, Издательско-полиграфическая фирма ПОЛИМЕД 1999. – 142 с.

6. Брюс Шнайер Прикладная криптография. Издательство: Триумф, 2012. – 815. ISBN 978-5-89392-527-2

7. Коблиц Н. Курс теории чисел и криптографии. Москва: Научное издательство “ТВП”, 2001. - 260 с.

8. Коутинхо С. Введение в теорию чисел. Алгоритм RSA. Перевод с англ. С.А. Кулешова под редакцией С.К. Ландо. М.: ПОСТМАРКЕТ, Москва, 2001. – 328 с.

9. Нильс Фергюсон, Брюс Шнайер Практическая криптография.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2005. – 424 с.: ил. – Парал. тит. англ. ISBN 5-8459-0733-0 (рус.)

10. Панасенко С.П. Алгоритмы шифрования. Специальный справочник. – Издательство “БХВ-Петербург”, 2009. – 576 с.: ил. ISBN 978-5-9775-0319-8

11. Ожиганов А.А. Криптография: учебное пособие. – СПб: Университет ИТМО, 2016. – 140 с.

12. Орлов В. А. Шимко Н.А., Медведев Н.В. Домрачева А.Б. Теория чисел в криптографии : учеб. пособие – М.: Издательство МГТУ им. Н.Э Баумана, 2011. – 223, [1] с. ISBN 978-5-7038-3520-3

13. Осипян В.О., Осипян К.В. криптография в задачах и упражнениях. – М.: Гелиос АРВ, 2004. – 144с., ил. ISBN 5-85438-009-9

14. Попов А. М. Алгоритм RSA. Методические указания к выполнению лабораторных работ. сост. : О. Н. Жданов, И. А. Лубкин ; Сиб. гос. аэрокосмич. ун-т. – Красноярск, 2007. – 38 с.

15. Ростовцев А.Г. Маховенко Е.Б. Теоретическая криптография. НЛО «Профессионал» Санкт-Петербург, 2004. – 480 с.

16. Сидельников В.М. Криптография и теория кодирования // Материалы конференции "Московский университет и развитие криптографии в России" (МГУ, 17-18 октября 2002 г.) – 21 с.

17. Смарт Н. Криптография Москва: Техносфера, 2005. - 528 с. ISBN 5-94836-043-1

18. Черемушкин А.В. Лекции по арифметическим алгоритмам в криптографии. Научный редактор А.Б. Пикчур. Технический редактор В. Шувалов. Издательство Московского Центра непрерывного математического образования. Лицензия ИД №01334 от 2000г.

Электронные ресурсы

19. Алгоритм шифрования RSA на пальцах. Электронный ресурс: http://teh-box.ru/informationsecurity/algoritm-shifrovaniya-rsa-na-palcax.html

20. Галина Басалова. Основы криптографии: Информация. Электронный ресурс https://www.intuit.ru/studies/courses/691/547/info

ПРИЛОЖЕНИЕ А

Программный код реализации алгоритма шифрования RSA

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Numerics;

namespace protect_inf_LR1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      public char separator = Convert.ToChar("-");

      //зашифровать
      private void buttonEncrypt_Click(object sender, EventArgs e)
      {
         if ((textBox_p.Text.Length == 0) && (textBox_q.Text.Length == 0))
         {
            MessageBox.Show("Введите p и q!");
            return;
         }
         long p = Convert.ToInt64(textBox_p.Text);
         long q = Convert.ToInt64(textBox_q.Text);

         if(p < 31 || q < 31)
         {
            MessageBox.Show("p и q должны быть больше 31");
            return;
         }
```

```
    if (!IsTheNumberSimple(p) || !IsTheNumberSimple(q))
    {
        MessageBox.Show("p или q - не простые числа!");
        return;
    }

    long n = p * q;
    long m = (p - 1) * (q - 1);
    long d = Calculate_d(m);
    long e_ = Calculate_e(d, m);

    textBox_Encrypted.Text = RSA_Endoce(textBox_ToEncrypt.Text, e_, n);

    textBox_d.Text = d.ToString();
    textBox_n.Text = n.ToString();

}

//расшифровать
private void buttonDecipher_Click(object sender, EventArgs e)
{
    if ((textBox_d.Text.Length > 0) && (textBox_n.Text.Length > 0))
    {
        long d = Convert.ToInt64(textBox_d.Text);
        long n = Convert.ToInt64(textBox_n.Text);

        textBox_Decrypted.Text = RSA_Dedoce(textBox_Encrypted.Text, d, n);

    }
    else
        MessageBox.Show("Введите секретный ключ!");
}

//проверка: простое ли число?
private bool IsTheNumberSimple(long n)
{
    if (n < 2)
        return false;

    if (n == 2)
        return true;

    for (long i = 2; i < n; i++)
        if (n % i == 0)
```

```
            return false;

        return true;
    }

    //зашифровать
    private string RSA_Endoce(string s, long e, long n)
    {
        string result = "";

        BigInteger bi;

        for (int i = 0; i < s.Length; i++)
        {
            //int index = Array.IndexOf(characters, s[i]);

            int index = (int)s[i];

            bi = new BigInteger(index);
            bi = BigInteger.Pow(bi, (int)e);

            BigInteger n_ = new BigInteger((int)n);

            bi = bi % n_;

            result += (bi.ToString()) + separator;
        }

        return result;
    }

    //расшифровать
    private string RSA_Dedoce(string input, long d, long n)
    {
        string result = "";

        BigInteger bi;

        string[] strings = input.Split(separator);

        foreach (string item in strings)
        {
            if(item == "") { continue; }
            bi = new BigInteger(Convert.ToDouble(item));
            bi = BigInteger.Pow(bi, (int)d);
```

```
            BigInteger n_ = new BigInteger((int)n);

            bi = bi % n_;

            int index = Convert.ToInt32(bi.ToString());

            result += (char)index;
        }

        return result;
    }

    //вычисление параметра d. d должно быть взаимно простым с m
    private long Calculate_d(long m)
    {
        long d = m - 1;

        for (long i = 2; i <= m; i++)
            if ((m % i == 0) && (d % i == 0)) //если имеют общие делители
            {
                d--;
                i = 1;
            }

        return d;
    }

    //вычисление параметра e
    private long Calculate_e(long d, long m)
    {
        long e = 10;

        while (true)
        {
            if ((e * d) % m == 1)
                break;
            else
                e++;
        }

        return e;
    }

    private void bt_NewPrimeNumbers_Click(object sender, EventArgs e)
```

```
{
    int p = 0, q = 0;

    Random rnd = new Random();

    while (!IsTheNumberSimple(p))
    {
        p = rnd.Next(31, 211);
    }

    while (!IsTheNumberSimple(q) || q == p)
    {
        q = rnd.Next(31, 211);
    }

    textBox_p.Text = Convert.ToString(p);
    textBox_q.Text = Convert.ToString(q);

}

private void bt_FromDisk_Click(object sender, EventArgs e)
{
    OpenFileDialog OPF = new OpenFileDialog();
    OPF.Filter = "Файлы txt|*.txt";
    if (OPF.ShowDialog() == DialogResult.OK)
    {
        textBox_Encrypted.Text = "";

        StreamReader sr = new StreamReader(OPF.FileName);

        while (!sr.EndOfStream)
        {
            textBox_Encrypted.Text += sr.ReadLine();
        }

        sr.Close();
    }
}

private void bt_ToDisk_Click(object sender, EventArgs e)
{
    SaveFileDialog SVF = new SaveFileDialog();
    SVF.Filter = "Файлы txt|*.txt";
    if (SVF.ShowDialog() == DialogResult.OK)
    {
```

```
            StreamWriter sw = new StreamWriter(SVF.FileName);
            sw.WriteLine(textBox_Encrypted.Text);
            sw.Close();

        }

    }

    private void bt_CalculateKey_Click(object sender, EventArgs e)
    {

        if ((textBox_p.Text.Length == 0) && (textBox_q.Text.Length == 0))
        {
            MessageBox.Show("Введите p и q!");
            return;
        }

        long p = Convert.ToInt64(textBox_p.Text);
        long q = Convert.ToInt64(textBox_q.Text);

        if (p < 31 || q < 31)
        {
            MessageBox.Show("p и q должны быть больше 31");
            return;
        }

        if (!IsTheNumberSimple(p) || !IsTheNumberSimple(q))
        {
            MessageBox.Show("p или q - не простые числа!");
            return;
        }

        long n = p * q;
        long m = (p - 1) * (q - 1);
        long d = Calculate_d(m);
        long e_ = Calculate_e(d, m);

        textBox_d.Text = d.ToString();
        textBox_n.Text = n.ToString();

    }
  }
}
```