

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Институт естественных и точных наук
Кафедра «Математическое и компьютерное моделирование»

РАБОТА ПРОВЕРЕНА
РЕЦЕНЗЕНТ, НАЧАЛЬНИК ФИНАНСО-
ВО-ЭКОНОМИЧЕСКОГО ОТДЕЛА ООО
НЕФТЬ-СЕРВИС _____ ХОРЕВА Е.И.
« ____ » _____ 2017 Г.

ДОПУСТИТЬ К ЗАЩИТЕ
ЗАВЕДУЮЩИЙ КАФЕДРОЙ,
Д.Ф.-М.Н., ДОЦЕНТ
_____ С.А.ЗАГРЕБИНА
« ____ » _____ 2017 Г.

Приложение-идентификатор, основанное на криптографическом методе
доказательства с нулевым решением
ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ– 01.03.02.2017.074.12.000 ВКР

Нормоконтролер,
Доцент каф. МиКМ, к.ф.-м.н.,
_____ Т.А. Макаровских
_____ 2017 г.

Руководитель проекта,
Проф., д.ф.-м.н.,
_____ А.В. Панюков
_____ 2017 г.

Автор работы
Студент группы ЕТ-485
_____ А.Д. Ромасс
_____ 2017 г.

Дипломная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Институт естественных и точных наук
Кафедра «Математическое и компьютерное моделирование»
Направление «Прикладная математика и информатика»

ДОПУСТИТЬ К ЗАЩИТЕ
ЗАВЕДУЮЩИЙ КАФЕДРОЙ,
Д.Ф.-М.Н., ДОЦЕНТ
_____ С.А.ЗАГРЕБИНА
« ____ » _____ 2017 Г.

ЗАДАНИЕ

на выпускную квалификационную работу студента
Ромасса Андрея Дмитриевича
Группа ЕТ-485

1. Тема работы
Приложение-идентификатор, основанное на криптографическом методе доказательства с нулевым решением утверждена приказом по университету от _____ 20_ г. № _____
2. Срок сдачи студентом законченной работы (проекта) _____
3. Исходные данные к работе (проекту)
-Материалы статьи Мэтью Грина Доказательство с нулевым разглашением. Электронная библиотека MSDN.
4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

4.1 Разработка генератора случайных графов
4.2 Разработка приложения основанного на протоколе нулевого разглашения
4.3 Разработка интерфейса приложения-идентификатора
4.4 Создание тестирующей системы для приложения-идентификатора
4.5 Тестирование алгоритмов на функциональность и эффективность
4.6 Анализ полученных результатов
5. Перечень графического материала (с точным указанием обязательных чертежей, плакатов в листах формата А1)

5.1 Приложение-идентификатор, основанное на криптографическом

методе доказательства с нулевым разглашением
5.2 Введение
5.3 Виды алгоритмов шифрования
5.4 Постановка задачи с условием нулевого разглашения
5.5 Задача о магнате
5.6 Алгоритм решения задачи построения протокола нулевого разглашения
5.7 Реализация иерархии классов
5.8 Тестирование алгоритмов на функциональность и эффективность
5.9 Анализ полученных результатов

6. Дата выдачи задания _____

Руководитель _____ А.В.Панюков
(подпись)

Задание принял к исполнению _____ А.Д.Ромасс
(подпись студента)

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов выпускной квалификационной работы (проекта)	Срок выполнения этапов работы (проекта)	Отметка о выполнении руководителя
1. Сбор материала и литературы по теме дипломной работы	30.01.2017 – 25.02.2017	
2. Постановка задач	25.02.2017 – 04.03.2017	
3. Предобработка исходных данных	04.03.2017 – 01.04.2017 –	
4. Программная реализация разработанных алгоритмов	01.04.2017 – 01.05.2017	
5. Подготовка пояснительной записки дипломной работы	01.05.2017 – 14.05.2017	
6. Оформление пояснительной записки	14.05.2017 – 25.05.2017	
7. Получение отзыва руководителя	30.05.2017	
8. Проверка работы руководителем, исправление замечаний	25.05.2017 – 31.05.2017	
9. Подготовка графического материала и доклада	31.05.2017 – 02.06.2017	
10. Нормоконтроль	02.06.2017	
11. Рецензирование, представление зав. кафедрой	10.06.2017	

Заведующий кафедрой _____ /С.А.Загребина/

Руководитель работы _____ /А.В. Панюков/

Студент _____ /А.Д. Ромасс /

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(Национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра «Математическое и компьютерное моделирование»

АННОТАЦИЯ

Ромасс А.Д. Приложение-идентификатор, основанное на криптографическом методе доказательства с нулевым решением/ А.Д. Ромасс – Челябинск: ЮУрГУ, Институт естественных и точных наук, 2017. – 53 с., 13 ил., 1 табл., 2 прил., библиогр. список – 11 названий

В дипломной работе было разработано приложение-идентификатор, основанное на криптографическом методе доказательства с нулевым решением. Были рассмотрены теоретические основы построения криптографически стойкого алгоритма шифрования. Рассмотрены методы построения алгоритма шифрования основанного на протоколе нулевого разглашения. Произведено тестирование и оценка эффективности разработанной программы.

Оглавление

ВВЕДЕНИЕ.....	9
1 ПРИМЕНЕНИЕ КРИПТОГРАФИЧЕСКОГО АНАЛИЗА И ВЫБОРА ТИПА АЛГОРИТМА ШИФРОВАНИЯ.....	11
1.1 Используемые определения и обозначения.....	11
1.2 Алгоритмы шифрования.....	12
1.2.1 Симметричные алгоритмы.....	13
1.2.2 Асимметричные алгоритмы.....	15
1.3 Достоинства алгоритмов на асимметричном ключе.....	18
Выводы по главе один.....	18
2 АНАЛИЗ ПРОТОКОЛА С НУЛЕВЫМ РАЗГЛАШЕНИЕМ.....	19
2.1 Протокол нулевого разглашения.....	19
2.2 Постановка задачи с условием нулевого разглашения.....	20
2.3 Решение задачи о магнате.....	22
2.3.1 Первый этап.....	22
2.3.2 Второй этап.....	22
2.3.3 Третий этап.....	23
Выводы по главе два.....	23
3 ПОСТРОЕНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА НУЛЕВОГО РАЗГЛАШЕНИЯ.....	24
3.1 Построение системы алгоритма.....	24
3.1.1 Постановка задачи приложения.....	25
3.1.2 Иерархия классов.....	27
3.1.3 Клиент.....	27
3.1.4 Проверяющая система.....	27

3.1.5 Модуль нулевого разглашения.....	28
3.2 Генерация графа G	28
3.3 Построение графа G'	30
3.4 Создание тестирующей системы.....	32
3.5 Достижение точности.....	34
3.6 Анализ скорости алгоритма и результатов тестов различных графов G	35
Выводы по главе три.....	37
ЗАКЛЮЧЕНИЕ.....	38
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	40
ПРИЛОЖЕНИЯ	
ПРИЛОЖЕНИЕ А.....	40
ПРИЛОЖЕНИЕ Б.....	48

ВВЕДЕНИЕ

Шифры и коды существуют, с того момента, как человечество научилось записывать свои впечатления об окружающем мире на носителях. Уже обычный алфавит является шифром. Шифр — какая-либо система преобразования текста с секретом (ключом) для обеспечения секретности передаваемой информации. В случае с алфавитом последовательность символов с помощью ключа можно преобразовать в слова понятные для понимания. Сейчас в век цифрового сообщества кодирование информации с помощью шифров развивается в ответ на потребность общества в технологиях защиты личной информации. Так например для большинства пользователей глобальной сети интернет, почта стала неотъемлемым паспортом в сети. И без ее использования становится сложной даже регистрация на любом общественном сайте.

Из этого можно сделать вывод, что реальный паспорт, в сети оказался не нужным и бесполезным. Очевидно, встает проблема реализации интернет идентификатора человека. Например, для части документа оборота или просто регистрации и деанонимизации человека в сети. Актуальность данного направления обусловлена возрастанием числа людей, занятых информационными технологиями, то есть создается глобальное информационное пространство, обеспечивающее:

- эффективное информационное взаимодействие людей;
- их доступ к мировым информационным ресурсам.

Объектом исследования является алгоритм шифрования основанный на протоколе нулевого разглашения.

Предметом исследования является скорость работы приложения-идентификатора основанного на протоколе нулевого разглашения.

Целью работы является создание приложения - идентификатора, основанное на криптографическом методе доказательства с нулевым решением. Для достижения поставленной цели будет использоваться метод Доказательства с нулевым разглашением.

Для достижения поставленной цели были решены следующие **задачи**:

- генерация ключа, где ключ – это гамильтонов путь в графе G , с возможностью создания изоморфного графа по графу G , с последующим выводом нового гамильтонова пути;
- анализ эффективности алгоритма шифрования с использованием протокола нулевого разглашения;
- тестирование алгоритма на крипто устойчивость;
- написание алгоритма шифрование с использованием концепций объектно-ориентированное программирование.

В качестве **информационной базы** используются материалы статьи Мэтью Грина Доказательство с нулевым разглашением, интернет библиотека MSDN.

Работа состоит из введения, трех разделов, заключения, библиографического списка и приложения. Объем работы составляет 48 страниц, объем библиографии – 11 источников.

Первая глава работы описывает сущность и общие принципы криптографии.

Во второй главе описаны алгоритмы и методы шифрования с использованием протокола с нулевым разглашением.

В третьей главе подробно описаны методы решения задачи приложения.

В заключении формулируются выводы по дипломной работе, оценивается степень выполнения поставленных задач.

1 ПРИМЕНЕНИЕ КРИПТОГРАФИЧЕСКОГО АНАЛИЗА И ВЫБОРА ТИПА АЛГОРИТМА ШИФРОВАНИЯ

1.1 Используемые определения и обозначения

Предположим, что отправитель хочет послать сообщение получателю. Более того, этот отправитель хочет послать свое сообщение безопасно: он хочет быть уверен, что перехвативший это сообщение не сможет его прочесть.

Само сообщение называется открытым текстом (иногда используется термин *клер*). Изменение вида сообщения так, чтобы спрятать его суть называется шифрованием. Шифрованное сообщение называется шифротекстом. Процесс преобразования шифротекста в открытый текст называется дешифрованием.

Искусство и наука безопасных сообщений, называемая криптографией, воплощается в жизнь криптографами. Криптоаналитиками называются те, кто постоянно используют криптоанализ, искусство и науку взламывать шифротекст, то есть, раскрывать, что находится под маской. Отрасль математики, охватывающая криптографию и криптоанализ, называется криптологией, а люди, которые ей занимаются, - криптологами. Современным криптологам приходится неплохо знать математику[1].

Исторически первой задачей криптографии была защита передаваемых текстовых сообщений от несанкционированного ознакомления с их содержанием, что нашло отражение в самом названии этой дисциплины, эта защита базируется на использовании "секретного языка", известного только отправителю и получателю, все методы шифрования являются лишь развитием этой философской идеи. С усложнением информационных взаимодействий в человеческом обществе возникли и продолжают возникать новые задачи по их защите, некоторые из них были решены в рамках криптографии, что потребовало развития принципиально новых подходов и методов. С развитием новых методов шифрования появлялись и новые способы обхода шифрования, что наложило свой отпечаток на защиту действительно важной информации.

Средства криптографической защиты государственные тайны требуют специального отношения и людей обеспечивающих безопасность государственных или иных данных. Очень немногие страны мира имеют свои криптографические компании, которые делают средства защиты мирового уровня. Россия одна из немногих стран мира в которой развита криптографическая школа и база позволяющая поддерживать и развивать криптологию. И в коммерческом и в государственном секторе есть компании и организации, которые представляют мировой рынок.

1.2 Алгоритмы шифрования

На сегодняшний день существует масса алгоритмов шифрования, имеющих значительную стойкость перед криптоанализом (криптографическую стойкость).

Криптографическая стойкость (или криптостойкость) — способность криптографического алгоритма противостоять криптоанализу. Стойким считается алгоритм, успешная атака на который требует от атакующего обладания недостижимым на практике объёмом вычислительных ресурсов или перехваченных открытых и зашифрованных сообщений либо настолько значительных затрат времени на раскрытие, что к его моменту защищённая информация утратит свою актуальность.

Поскольку атака методом грубой силы (полным перебором всех ключей) возможна для всех типов криптографических алгоритмов, кроме абсолютно стойких «по Шеннону», для вновь созданного алгоритма она может являться единственной существующей. Способы её оценки основываются на вычислительной сложности, которая затем может быть выражена во времени, деньгах, и требуемой производительности вычислительных ресурсов, например, в MIPS.[2]

Принято деление алгоритмов шифрования на три группы:

- симметричные алгоритмы;
- ассиметричные алгоритмы;
- алгоритмы хэш-функций.

1.2.1 Симметричные алгоритмы

Симметричное шифрование предусматривает использование одного ключа как для шифрования, так и для дешифрования. К симметричным алгоритмам применяются два основных требования, выполнение которых обязательно для алгоритма: полная утрата всех статистических закономерностей в объекте шифрования и отсутствие линейности.

Для обеспечения выполнения первого требования шифр должен иметь сильный эффект воздействия на данные, так в 2-битном изменении входных данных, должно меняться минимум половина выходного шифроблока, то есть 1 бит. Выполнение второго требования включают в себя условие $F(a) \text{ xor } F(b) == F(a \text{ xor } b)$, где a - первый бит числа, b - второй бит числа, в противном случае становится эффективным применение дифференциального крипто анализа к шифру.

Принято разделять симметричные системы на блочные и поточные. В блочных системах все данные разбиваются на блоки с их последующем шифровании. В поточном, как видно из названия, данные обрабатываются по мере получения потока входных данных, последовательно строится выходная гамма шифрования.

Симметричных алгоритмов шифрования:

- DES (значительно устарел) и TripleDES (3DES);
- AES (Rijndael);
- ГОСТ 28147-89;
- Blowfish.

ГОСТ 28147. Это советский и российский стандарт симметричного шифрования, введенный 1 июля 1990 года. Стандарт обязателен для организаций, предприятий и учреждений, применяющих криптографическую защиту данных, хранимых и передаваемых в сетях ЭВМ, в отдельных вычислительных комплексах или ЭВМ. Схема работы алгоритма представлена на рисунке 1.

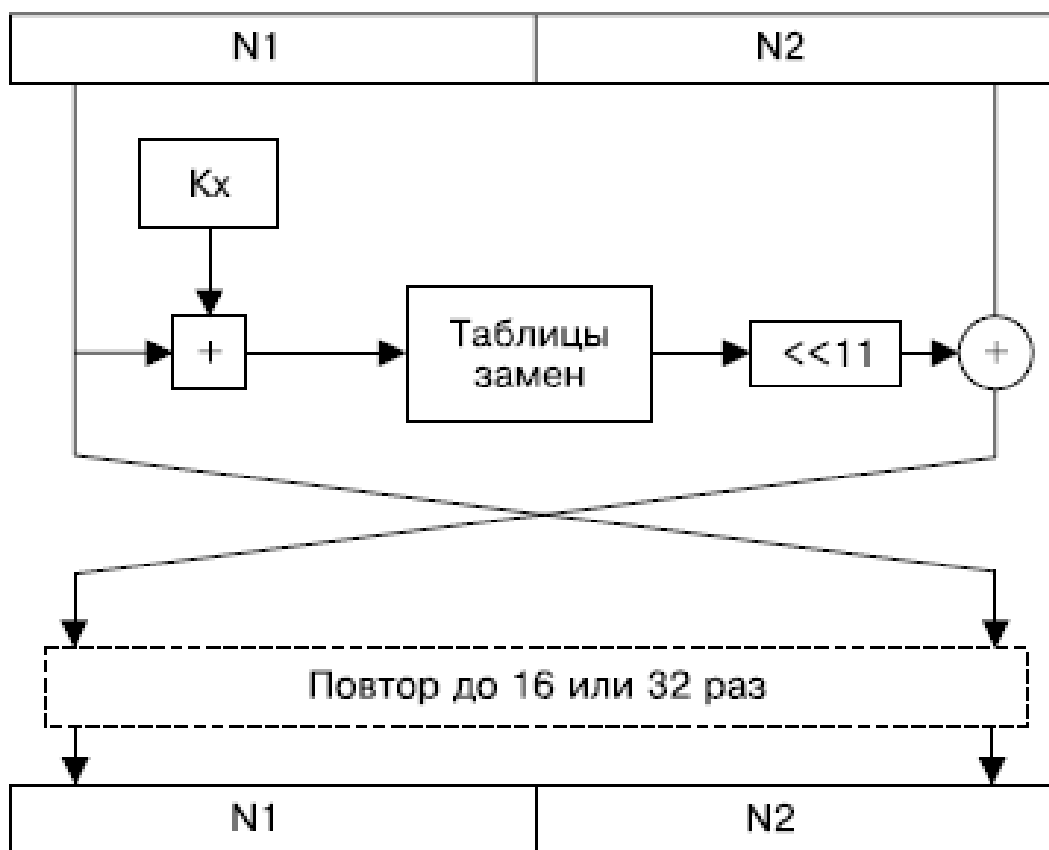


Рисунок 1 – Преобразование выполняется определенное количество раз

Первая операция подразумевает наложение ключа. Содержимое субблока $N1$ складывается по модулю 2 с 32-битной частью ключа Kx . Полный ключ шифрования представляется в виде конкатенации 32-битных подключей: $K0, K1, K2, K3, K4, K5, K6, K7$. В процессе шифрования используется один из этих подключей, в зависимости от номера раунда и режима работы алгоритма.

Вторая операция осуществляет табличную замену. После наложения ключа субблок $N1$ разбивается на восемь частей по четыре бита, значение каждой из которых заменяется в соответствии с таблицей замены для данной части субблока. После этого выполняется побитовый циклический сдвиг субблока влево на 11 бит.

Следовательно при простой реализации алгоритм неустойчивый ко взломам, после раскрытия ключа. Он не подойдет для решения нашей задачи, так как хоть раз раскрыв наш паспорт с использованием этого алгоритма, мы отдадим его получателю, а этого нужно избежать.

1.2.2 Асимметричные алгоритмы

Асимметричный алгоритм шифрования - алгоритм, использующий два математически связанных шифровальных ключа. Один ключ называется секретным (Private Key) и хранится в недоступном месте. Другой ключ называется открытым (Public Key) и свободно предоставляется любым потенциальным пользователям. Связь между Public Key и Private Key выражена неявно, так при условии достаточной длины ключей невозможно вычислить один ключ на основе другого. Общая схема работы представлена на рисунке 2.

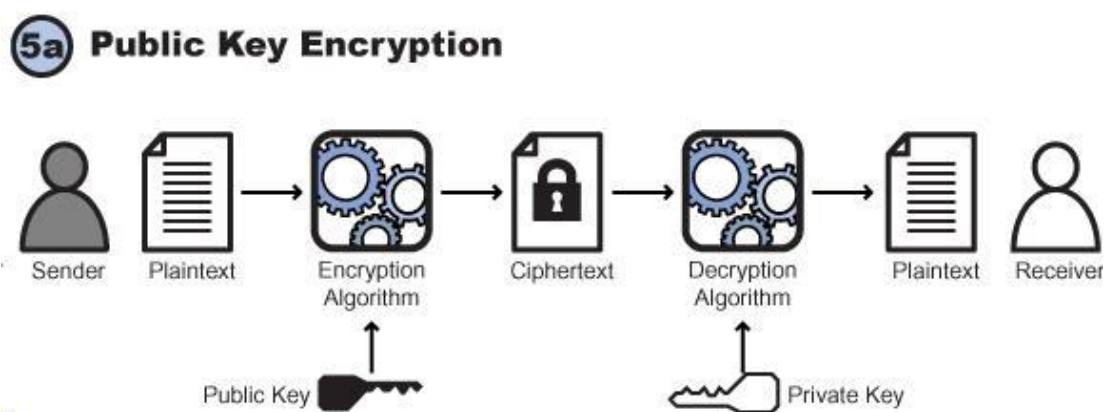


Рисунок 2 – Общая схема работы асимметричного алгоритма

У криптографии с открытыми ключами есть ряд преимуществ перед симметричной криптографией. Наиболее полезное из них касается управления ключами (в частности, их выбором и рассылкой). Но остается сложность в передаче их получателю, без скомпрометированного ключа. В симметричных алгоритмах шифрования ключ зашифрования является также ключом расшифрования, следовательно, первый не может быть раскрыт. Это приводит к тому, что две легальные стороны (отправитель и получатель) договариваются заранее об алгоритме зашифрования и ключах либо при личной встрече, либо при передаче по абсолютно секретному каналу.

При использовании же криптосистем с открытым ключом стороны не обязаны встречаться, знать друг друга и иметь секретные каналы связи. Это преимущество становится еще более актуальным в случае большого количества пользователей системы. Тогда, к примеру, один пользователь может «закрыто» связаться

с другим, взяв некоторую информацию (открытый ключ) из общедоступной базы данных (банка ключей).

Различие ключей (открытого и личного) в криптографии с открытыми ключами позволило создать следующие технологии: электронные цифровые подписи, распределенная проверка подлинности, согласование общего секретного ключа сессии, шифрование больших объемов данных без предварительного обмена общим секретным ключом.

В настоящее время хорошо известен ряд алгоритмов шифрования с открытым ключом. Алгоритмы криптосистемы с открытым ключом можно использовать в трёх назначениях:

- как самостоятельные средства защиты передаваемых и хранимых данных;
- как средства для распределения ключей. Алгоритмы - более трудоёмки, чем традиционные криптосистемы. Поэтому часто на практике рационально с помощью систем с открытым ключом распределять ключи, объем которых как информации незначителен. А потом с помощью обычных алгоритмов осуществлять обмен большими информационными потоками;
- средства аутентификации пользователей (электронная цифровая подпись).

Выделяют следующие алгоритмы шифрования:

- алгоритм Диффи—Хеллмана;
- алгоритм RSA;
- алгоритм Эль-Гамала;
- алгоритм цифровой подписи (Digital Signature Algorithm).

Рассмотрим работу одного из асимметричных алгоритмов шифрования RSA. RSA (Rivest, Shamir, Adleman) – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Был разработан в 1977 году. Трое учёных из Массачусетского технологического института Рональд Ривест, Ади Шамир и Леонард Адле-

ман занимались разработкой данного алгоритма шифрования, основная идея которого заключалась в том, насколько легко находить большие простые числа и насколько сложно раскладывать на множители произведение двух больших простых чисел. Система была названа по первым буквам фамилий её создателей.

В алгоритме RSA открытый ключ доступен всем, а закрытый хранится только у его хозяина и неизвестен никому другому. С помощью одного ключа можно производить операции только в одну сторону. Если сообщение зашифровано с помощью одного ключа, то расшифровать его можно только с помощью другого. Имея один из ключей вероятность найти другой ключ практически нулевая, если разрядность ключа высока. Алгоритм состоит из трех этапов: генерация ключей, зашифрования, расшифрования. Схема алгоритма представлена на рисунке 3[3].

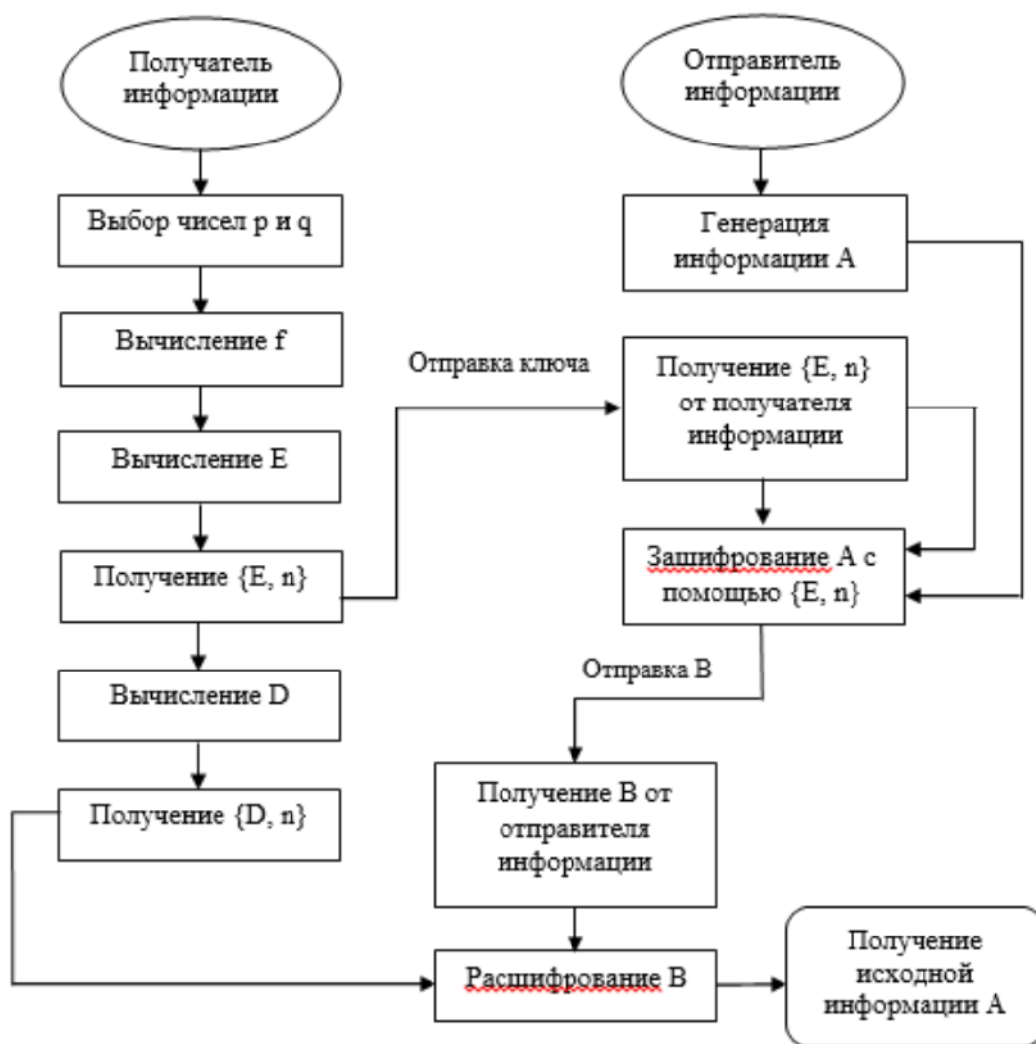


Рисунок 3 – Схема работы RSA.

1.3 Достоинства алгоритмов на асимметричном ключе

Перечислим особенности и преимущества алгоритмов на асимметричном ключе.

Алгоритмы асимметричного ключа делают более легким управление ключами благодаря тому, что не надо искать безопасный канал для передачи копии ключа конечному получателю. Закрытый ключ остается закрытым, а открытый – открытым. Другими словами, большим преимуществом этого механизма над механизмом симметричного ключа является то, что больше нет того секрета, которым требуется делиться. Фактически совершенно не имеет значения, у кого имеется открытый ключ, поскольку он бесполезен без соответствующего закрытого.

Еще одним важнейшим достоинством алгоритмов на асимметричном ключе является то, что они способны предоставить такую цифровую подпись, которую невозможно подделать. Обсуждение этого вопроса будет разобрано в основном разделе работы.

Выводы по главе один

При использовании алгоритма на асимметричном ключе, то это позволит оставить, часть ключа открытым. То есть “При показе паспорта мы его не теряем”. Именно эта особенность и является ключевой при разработке алгоритма.

Следовательно, для решения задачи создания приложения-идентификатора, будет использован алгоритм основанный на ассиметричном ключе. Что повысит устойчивость алгоритма и позволит свободно обмениваться открытым ключом. Хорошим свойством будет возможность использовать алгоритмы криптосистемы с открытым ключом, как самостоятельное средство для защиты хранимой и передаваемой информации. Что позволит расширить функционал системы, не меняя при этом основу системы.

2 АНАЛИЗ ПРОТОКОЛА С НУЛЕВЫМ РАЗГЛАШЕНИЕМ

2.1 Протокол нулевого разглашения

Доказательство с нулевым разглашением (информации) в криптографии — интерактивный криптографический протокол, позволяющий одной из взаимодействующих сторон («The verifier» — Проверяющей) убедиться в достоверности какого-либо утверждения (обычно математического), не имея при этом никакой другой информации от второй стороны («The prover» — Доказывающей)[6].

Понятие протокола нулевого разглашения или Zero Knowledge Proofs было впервые предложен в 1980 специалистами MIT Шафи Голдвассером, Сильвио Микали и Чарльзом Ракофф. (Shafi Goldwasser, Silvio Micali и Charles Rackoff). Эти исследователи работали над проблемой, которая относится к интерактивным системам доказательства - теоретическим системам, в которых первый участник (назовём его "Испытатель" (Prover) обменивается сообщениями со вторым участником "Контролёр" (Verifier), чтобы убедить контролёра, что некоторое математическое утверждение верно[4].

Нужно понимать, что термин нулевого разглашение не равноценен, абсолютной безопасности данных. Но приближен к этому.

Разберем суть протокола с нулевым разглашением. Подтверждение того, что объект действительно обладает «Верным утверждением», требует определенного доказательства от объекта контроллеру. Создается впечатление, что для проверки этого утверждения контроллеру нужна вся информация о Верном утверждении. Но это не так. Создадим систему с клиентом располагающим верным утверждением, который хочет доказать контроллеру, что его утверждение верно. Это доказательство утверждения назовем подтверждением. Большинство систем в реальном мире реализуют это "подтверждение" наихудшим образом из возможных, если смотреть на защищенность этой системы.(можно же просто сказать неудовлетворительно) Клиент просто передаёт оригинальный пароль на сервер, который повторно вычисляет хэш пароля и сравнивает его с сохранённым значением.

Сразу можно выделить проблему: сервер получает пароль клиента в самом притягательном для взлома виде "чистый текст". А пользователь не на 100% уверен в том, что защита сервера не скомпрометирована.

То, что предложили Голдвассер, Микали и Ракофф, стало поводом на появление новых методов подтверждения. В случае полной реализации, доказательства с нулевым разглашением смогут дать подтверждение в описанной выше задаче. При этом не разгласив ни одного бита информации, которая соответствует тому, что "это утверждение верно".

2.2 Постановка задачи с условием нулевого разглашения

Создадим клиента у которого есть его «паспорт» и он хочет убедить контроллера в его достоверности, при этом не отдавая паспорт контролеру. В нашем случае паспорт это абсолютно точное решение NP полной задачи. Такой, как нахождение гамильтонова пути в графе[5].

Предположим, что есть магнат транспортной компании и ему нужно развернуть новую транспортную сеть.

При этом у него есть полный список городов и дорог между ними – граф G (рисунок 4), где каждая вершина графа представляет собой город, и соединяющие линии (границы) показывают дороги связывающие их. И ему нужно построить путь, проходящий через всю торговую сеть, при этом посетить каждый город только один раз, при этом путь должен учитывать, что проход по каждой дороге можно проехать только 1 раз и вернуться в изначальную точку. То есть найти гамильтонов путь в графе G . Обозначим путь буквой P .

Пример графа представлен на рисунке 4.

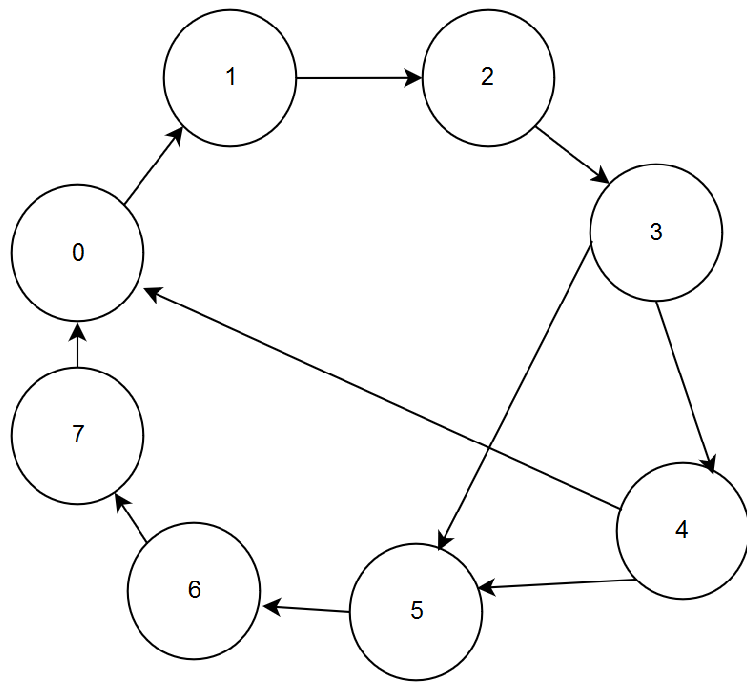


Рисунок 4 – Граф G

Для такого не большого графа решение можно получить путем простого перебора представлен на рисунке 5. Простое решение имеет вид: 0,1,2,3,4,5,6,7.

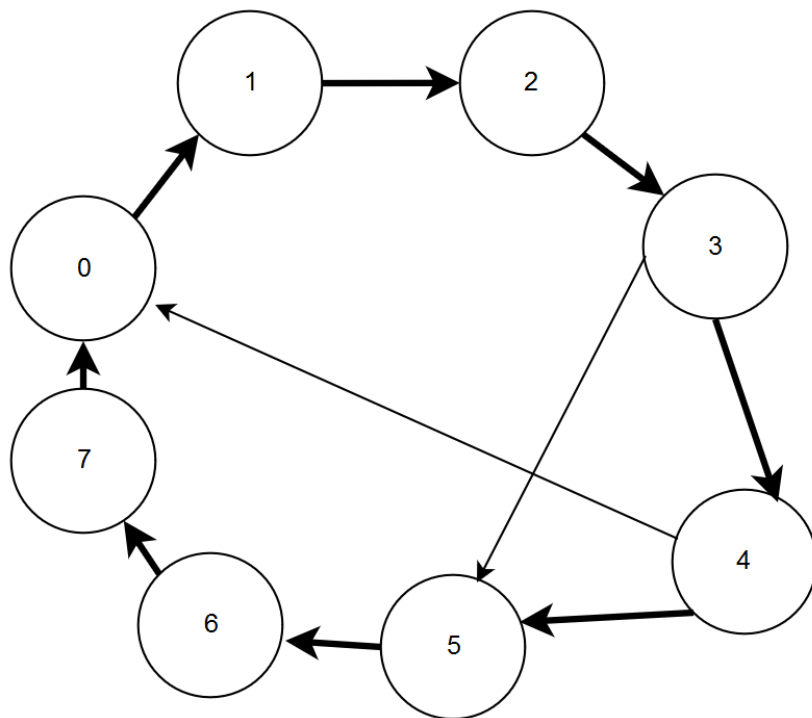


Рисунок 5 – Гамильтонов путь P

То, что описано выше — только частный случай знаменитой теоретической проблемы. Она носит название "поиск гамильтонова пути в графе". Для некоторых графов очень трудно найти решение или даже определить, что такое решение

существует. Задача поиска гамильтонова пути в графе (и поиск ответа на вопрос, существует ли такое решение для данного случая), как известно, относятся к разряду NP полных задач[5].

Предположим, что размер графа G настолько большой, что стандартных вычислительных мощностей не хватает, что бы найти в нем путь P . В этом случае, нужно поручить задачу организации, у которой хватит вычислительной мощности. Пусть такая организация существует и имеет название BigItCompany. Но как убедиться в наличии решения компанией BigItCompany, не подсмотрев сам путь P ?

Получаем задачу о магнате:

Магнат располагает информацией о сети городов G . Но не знает гамильтонова пути в ней P . Компания BigItCompany знает путь P для графа G . Как убедить магната в том, что компания действительно располагает решением задачи поиска гамильтонова пути в графе. При этом, не показывая сам путь P .

2.3 Решение задачи о магнате

Выделим понятие информационной защищенности обеих сторон:

- 1) магнат должен быть уверен в наличии достоверного решения компанией BigItCompany поставленной задачи на 99%;
- 2) компания BigItCompany не должна раскрыть свое решение до того как первый пункт не выполнится.

Будем решать задачу в несколько этапов.

2.3.1 Первый этап

Пусть компания BigItCompany может генерировать граф изоморфный исходному за линейное время. И изменять ответ P на P' в новом графе G' .

Назовем это генерированием открытого ключа.

2.3.2 Второй этап

Компания BigItCompany показывает магнату граф G' и предлагает выбрать случайным образом из 2 вариантов:

- 1) показать магнату путь P' – это докажет, что он действительно есть;
- 2) показать как был получен граф G' из графа.

Если магнат получит обе информации, то нарушится пункт 2 информационной защищенности стороны BigItCompanу. И он сможет решить задачу. Но он получит только условную половину решения, после своего выбора, так как при выборе пункта 1. Ему предстоит решить еще одну NP полную задачу изоморфизма графов. А при выборе пункта 2. Он не получит путь P .

Но при это после одной такой итерации можно убедиться, что вероятность обмана компанией BigItCompanу магната равна 50%. Если они достоверно ответят на пункт который он выберет.

Назовем это одной итерацией алгоритма. В которой каждый раз будет генерироваться новый граф G' .

2.3.3 Третий этап

Таким образом, что после 10 итераций вероятность достоверного решения у компании BigItCompanу будет равна 99.9%. При этом не нарушится не один из пунктов информационной защищенности.

Выводы по главе два

Задача о магнате аналогична задаче создания интернет паспорта, основанного на протоколе нулевого разглашения.

Следовательно, основываясь на задаче о магнате, можно сделать вывод, что алгоритм решения задачи построения протокола нулевого разглашения сводится к четырем основным пунктам:

- 1) построение графа G ;
- 2) построение графа G' ;
- 3) выбор проверяющей системы одного из варианта или достоверности G' или отображения пути P' ;
- 4) повторения итерации 3, до завершения границы необходимой точности.

3 ПОСТРОЕНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА НУЛЕВОГО РАЗГЛАШЕНИЯ

3.1 Построение системы алгоритма

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Где класс — универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей).

Объект — это сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Так же важными принципами ООП являются Инкапсуляция, Наследование и Полиморфизм.

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, C++, C#, Java или Ruby) отождествляют инкапсуляцию с сокрытием, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид по-

лимоморфизма — параметрический — в ООП называют обобщённым программированием[7].

Методология построения программы основанное на этих принципах, становится более устойчивой к различным ошибкам. Увеличивается масштабируемость программы и ее тестируемость.

Следовательно, мы будем руководствоваться именно этими принципами при построении алгоритма программы.

3.1.1 Постановка задачи приложения

Сформулируем описание программы на основе задачи о Магнате.

Приложение-идентификатор, основанное на криптографическом методе доказательства с нулевым решением или нулевым протоколе.

Клиент располагает паспортом, который представлен в виде открытого графа G , и закрытого ключа – гамильтонова пути в графе G , назовем его P . Далее на каждом из n этапов генерируется открытый граф G' . Проверяющая система запрашивает у клиента вопросы двух типов. При этом случайно выбирая только один вопрос из 2 возможных:

- 1) показать наличие пути P' в графе G' ;
- 2) доказать изоморфность графов G и G' .

Покажем одну итерацию алгоритма в виде блок схемы, которая представлена на рисунке б.

Повторяя n итераций алгоритма, увеличивается точность предположения гипотезы о наличии у клиента паспорта. Сервер проверяет достоверность информации об ответе клиента на запрос и дает задание клиенту на следующую итерацию в виде случайной выбранного типа запроса.

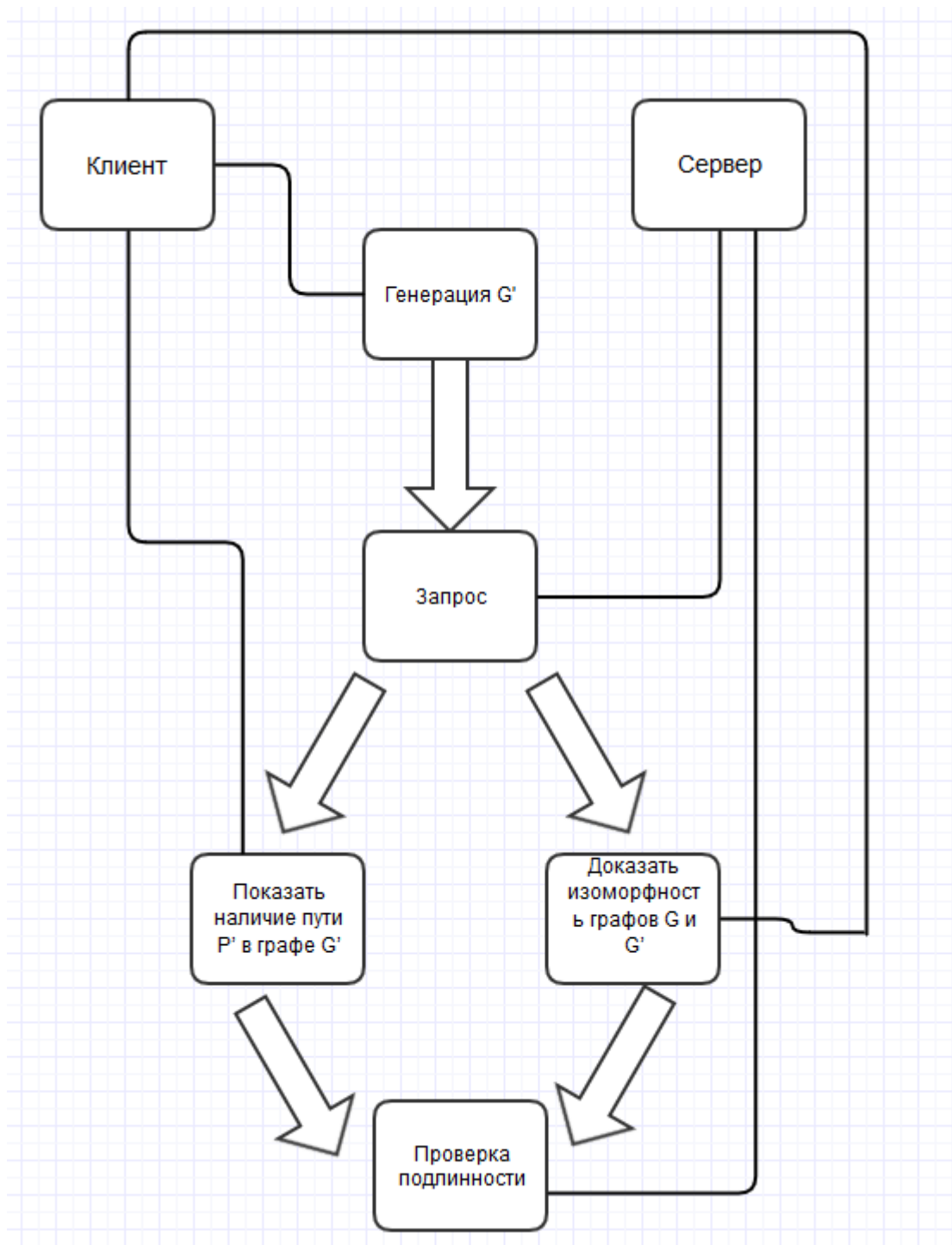


Рисунок 6 – Схема одной итерации

Повторяем до достижения точности достоверности алгоритма *Prec*. Точность порядка 99% достигается за 5 шагов.

После достижения нужной точности *Prec* принимается верность утверждения, что клиент располагает своим паспортом. Так как на каждом этапе сервер получает только одну половинку паспорта клиента, которой не достаточно для воссоздания реального приватного ключа, а так же получения частичной инфор-

мации о нем, то применив протокол нулевого разглашения, мы добились выполнения условия о защите информации клиента.

3.1.2 Иерархия классов

Если представить иерархию классов программы в простом виде, то она будет иметь вид, представленный на рисунке 7.

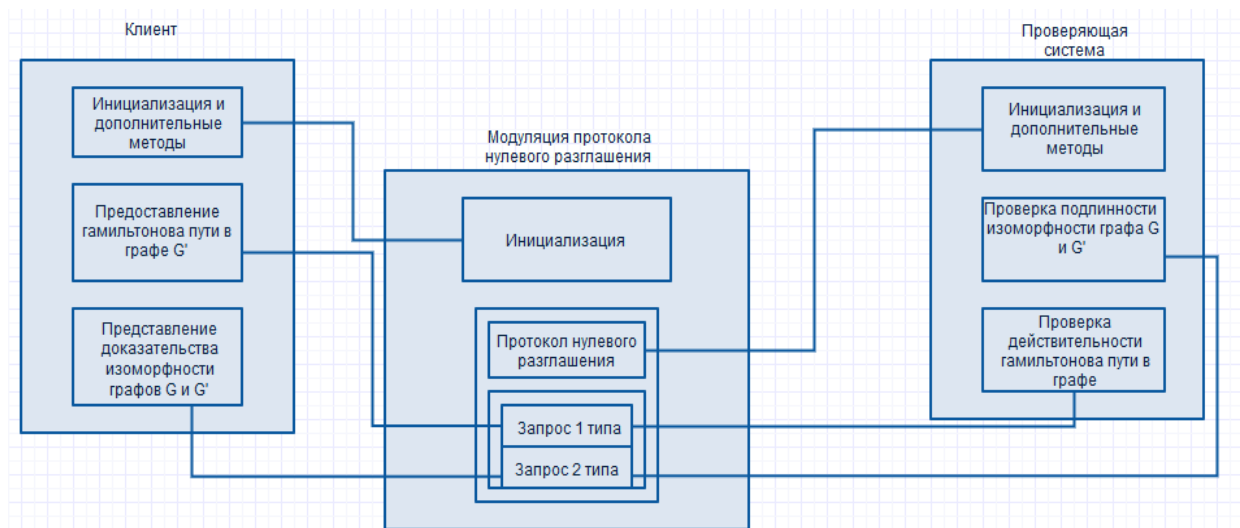


Рисунок 7 – Иерархия классов

Можно выделить три основных класса: клиент, модуль нулевого разглашения, проверяющая система.

Опишем каждый из них подробнее.

3.1.3 Клиент

Модуль клиента, представляет из себя класс хранящий информацию обо всей доступной информацией, такой как G, G', P, P' и др. Так же в нем описаны методы генерации графа, открытых и закрытых ключей.

Для тестирования и отображения результата в нем описаны методы вывода и отображения графа, как открытого, так и изоморфного ему. От клиента требуется быстро выдавать ответы на запросы 1 и 2 типа. А так же генерировать граф G' на каждый итерации алгоритма.

3.1.4 Проверяющая система

Модуль проверяющей системы или Checker должен проверять достоверность представляемой информации. А именно ответы на запросы двух типов. То есть

наличие гамильтонова пути в графе по ключу (1 тип) и доказательство изоморфности графов G и G' по представляемому ключу (2 тип).

3.1.5 Модуль нулевого разглашения

Модуль нулевого разглашения должен модулировать работу протокола нулевого разглашения. Подробнее описан в 3.4 .

3.2 Генерация графа G .

Для генерации графа был выбран алгоритм основанный на балансированном среднем состоянии вершины.

Алгоритм “GenerateGraph”:

Шаг 1. Добавить в граф созданный гамильтонов путь path.

Шаг 2. Для каждой вершины V генерировать число count смежных вершин;

Шаг 3. Добавить count вершин в hash set вершины V .

Конец Алгоритма

Гамильтонов путь (или гамильтонова цепь) — путь (цепь), содержащий каждую вершину графа ровно один раз. Гамильтонов путь, начальная и конечная вершины которого совпадают, называется гамильтоновым циклом;

Структура данных hash-set — это такая редкая структура, что присутствует не во всех стандартных библиотеках. В .NET Framework, например, она появилась только с версии 3.5. Однако в некоторых случаях hash-set может быть весьма полезен.

Set - это коллекция, элементы которой неупорядоченные и уникальны (т.е. присутствуют в коллекции не более одного раза). Set поддерживает базовые операции:

- 1) добавить элемент;
- 2) удалить элемент;
- 3) проверить, существует ли элемент;
- 4) перебрать все элементы.

Реализации set различаются только структурой данных, используемой для поиска — это может быть массив, список, дерево, hash-таблица или что угодно

еще. На практике, чаще всего используется hash-таблица, поскольку дает максимальную производительность с приемлемыми накладными расходами.

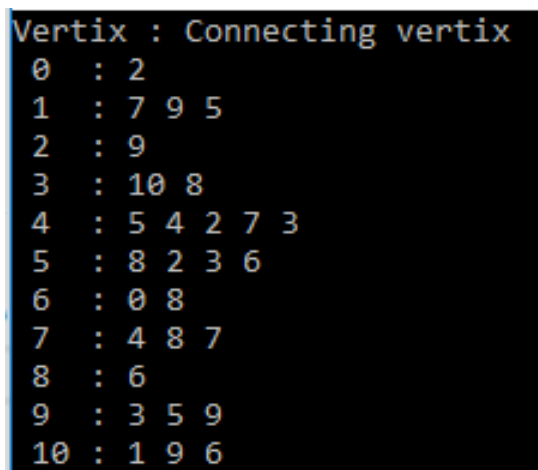
В данном случае для каждой вершины V хранится hash table смежных вершин. Алгоритм метода GenerateGraph представлен на листинге 1.

```
public void GenerateGraph()
{
    for (int i = 0; i < _size - 1; ++i)
    {
        _graph[_path[i]].Add(_path[i + 1]);
    }
    _graph[_path[_size - 1]].Add(_path[0]);

    Random rnd = new Random();
    for (int i = 0; i < _size; ++i)
    {
        int count = rnd.Next((_size - 1)/2);
        for (int j = 0; j < count; ++j)
        {
            _graph[i].Add(rnd.Next(_size - 1));
        }
    }
}
```

Листинг 1 – Метод GenerateGraph

С помощью метода ShowOpenKey. Выведем на консоль граф с малым количеством ребер и вершин. Результат вывода программы представлен на рисунке 8.



```
Vertex : Connecting vertex
0 : 2
1 : 7 9 5
2 : 9
3 : 10 8
4 : 5 4 2 7 3
5 : 8 2 3 6
6 : 0 8
7 : 4 8 7
8 : 6
9 : 3 5 9
10 : 1 9 6
```

Рисунок 8 – Генерация малого графа

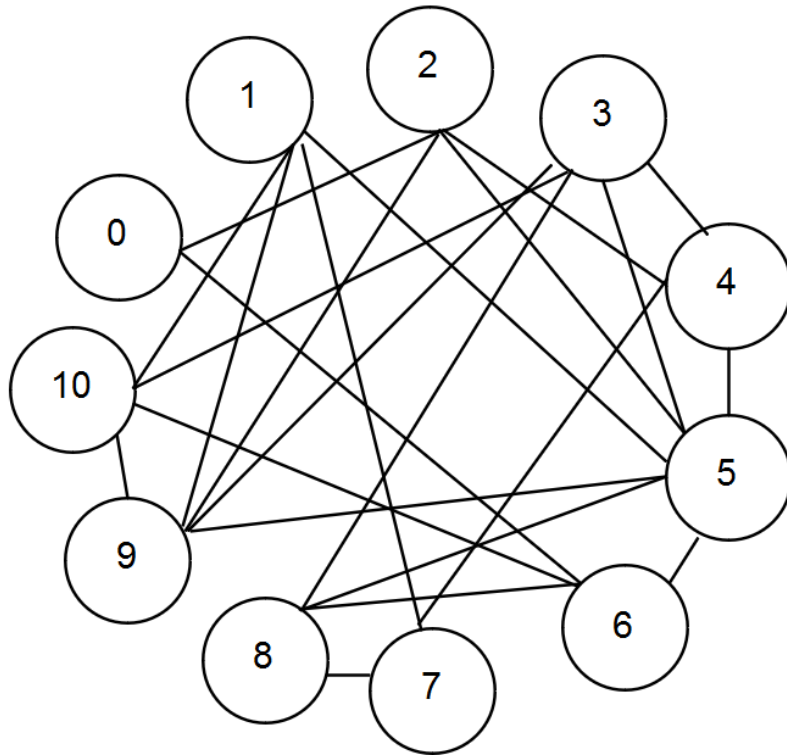


Рисунок 9 – Сгенерированный граф G в графическом виде

3.3 Построение графа G' .

В теории графов изоморфизмом графов $G = (V_G, E_G)$ и $H = (V_H, E_H)$ называется биекция между множествами вершин графов $f: V_G \rightarrow V_H$ такая, что любые две вершины u и v графа G смежны тогда и только тогда, когда вершины $f(u)$ и $f(v)$ смежны в графе H .

Таким образом нужно построить изоморфный граф, графу G . С сохранением нового открытого ключа. Для этого воспользуемся методом “замешивания” вершин в графе. Алгоритм “IsomorphicTransformation”:

Шаг 1. Создать скрытый ключ изоморфности каждой вершины с ее соответствием;

Шаг 2. Инициализировать OpenKey или открытый граф;

Шаг 3. Добавить все вершины в новый граф в соответствии с ключом `_key`.

Конец алгоритма.

Код метода IsomorphicTransformation представлен на листинге 2.

```

public void IsomorphicTransformation()
{
    _key = Shafle(_size);
    OpenKey = new List<HashSet<int>>();
    for (int i = 0; i < _size; ++i)
    {
        OpenKey.Add(new HashSet<int>());
    }
    for (int i = 0; i < _size; ++i)
    {
        foreach (var vertex in _graph[i])
        {
            OpenKey[_key[i]].Add(_key[vertex]);
        }
    }
}

```

Листинг 2 – Метод IsomorphicTransformation

С помощью метода ShowIsoporphicKey. Выведем в консоль изоморфный граф с малым количеством ребер и вершин. Результат представлен в виде:

Номер вершины: список смежных вершин перечисленных через пробел.

Результат выполнения программы представлен на рисунке 9.

```

Vertex : Connecting vertex
0 : 1 2 6 4
1 : 4
2 : 8
3 : 2
4 : 3 1
5 : 10 1 5
6 : 7 1
7 : 9 8 4
8 : 6 0 8
9 : 5 8 0
10 : 0 10 2 5 6

```

Рисунок 10 – Изоморфный граф G'

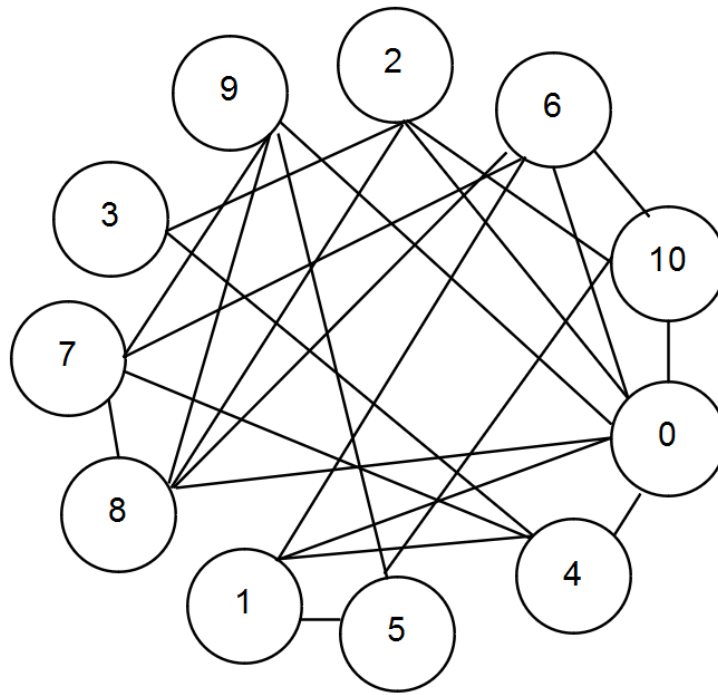


Рисунок 11 – Сгенерированный граф G' в графическом виде

3.4 Создание тестирующей системы.

Для построения тестирующей системы или проверяющего узла. Создадим класс `modelOfZeroKnowledge`, который будем моделировать внутри себя поведение системы основанной на протоколе нулевого соглашения.

Алгоритм `Precision`.

Шаг 1. Создать ключ `PrivateKey`;

Шаг 2. Проверить гипотезу о достоверности наличия скрытого ключа;

Шаг 3. Сгенерировать изоморфный граф G' ;

Шаг 4. Смодулировать случайный выбор из двух возможных: предоставления ключа изоморфности графа и его проверка или предоставления гамильтонова пути в графе G' и его проверка;

Шаг 5. Скорректировать достоверность и вывод ошибки в случае не правильного исхода;

Шаг 6. Возврат к шагу 2, если точность предположения не достигла заданной *Prec*.

Конец алгоритма.

Код метода `precision` класса `modelOfZeroKnowledge` приведен в листинге 3.


```

public void precision(double P)
{
    double reliability = 0;
    double rate = 0.5;
    int i = 0;
    var rnd = new Random();
    while (i < 100 && reliability < P)
    {
        OurKey.IsomorphicTransformation();
        singleCheker = new Checker(OurKey.OpenKey);
        int l = rnd.Next(2);
        bool f = true;
        if (l == 1)
        {
            f= singleCheker.CheckPath(OurKey.GetPrivateKeyPath());
        }
        else
        {
            f = singleCheker.CheckIsomorphic(OurKey.graph,OurKey.GetIsomorphicKey());
        }
        if (f)
        {
            reliability += rate;
            rate /= 2;
            Console.WriteLine("success on " + reliability);
        }
        else
        {
            Console.WriteLine("EROOR ");
            break;
        }
    }
}
}

```

Листинг 3 – Метод Precision

3.5 Достижение точности

Опишем задачу и результат каждой итерации алгоритма нулевого разглашения в рамках теории вероятности.

Под вероятностью случайного события в математике понимают меру возможности осуществления данного события в конкретных условиях эксперимента (испытания) [8].

Определение: Вероятностью $P(A)$ случайного события A называется отношение количества t элементарных событий, благо-препятствующих событию A , к общему количеству элементарных событий: $P(A) = t/n$.

Теперь выделим событие в задаче протокола нулевого разглашения. **Предположим**, клиент хочет обмануть сервер и у него нет паспорта. Тогда верно следующие, он не знает ответа на один из двух типов запросов. Так как если ему известны ответа на 2 вида запроса на хотя бы одной итерации, то он знает паспорт. Ему нужно лишь преобразовать граф G' обратно к G и путь P' к P . Так же очевидно что он может подделать заранее только один из видов запроса. Это следует из того, что клиент уже показал граф G' .

И так получаем вероятность этого события равна, $\frac{1}{2}$ так как видов запроса 2, а клиент может подделать только 1 из них и он точно не знает какой именно вопрос ему будет задан.

То есть вероятность, того что клиент обманывает сервер равна $\frac{1}{2}$ на каждой итерации. Так как вероятности каждого события независима.

Теорема умножения вероятностей: вероятность совместно-го появления независимых событий равна произведению их вероятностей. Для двух событий [8]. Получаем вероятность обмана клиентом сервера после n итераций равна $\frac{1^n}{2}$. Значит вероятность события обратного, а именно того что клиент действительно обладает паспортом равна $1 - \frac{1^n}{2}$ [8].

Получаем точность утверждения “Клиент обладает реальным паспортом” равна $1 - \frac{1^n}{2}$. То есть после 10 итераций вероятность события будет равна $p = 0,9990234375$.

3.6 Анализ скорости алгоритма и результатов тестов различных графов G

Проанализируем скорость выполнения алгоритма. Произведем замеры времени выполнения проверки достоверности гипотезы при точности 0.99 и 0.999. На основе графов размеров 20000-150000 с шагом 10000. Для каждого измерения были взяты данные усредненные по трем измерениям для достижения точного вычисления результата.

Проанализировав алгоритм, можно сделать вывод, что основная нагрузка вычислений ложится на клиента. Проверка занимает линейное время выполнения. Вычислительная сложность — понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных[9]. Тогда как сложность генерации графов G' зависит от коэффициента hash-table. Что накладывает дополнительную константу на вычисления со стороны клиента. Результат работы программы представлен в таблице 1.

Таблица 1 – Скорость проверки достоверности гипотезы от размера графа

Количество вершин графа G	0.990	0.999
20000	609	886
30000	1030	1345
40000	1369	1832
50000	1682	2366
60000	1725	2688
70000	2328	3044
80000	2754	3734
90000	3296	3944
100000	3566	4891
110000	4104	5095
120000	3995	5949
130000	4622	6434
140000	4599	7322
150000	5255	7086

Построим графики зависимостей времени исполнения модуляции от размера графа – рисунок 12 и 13.

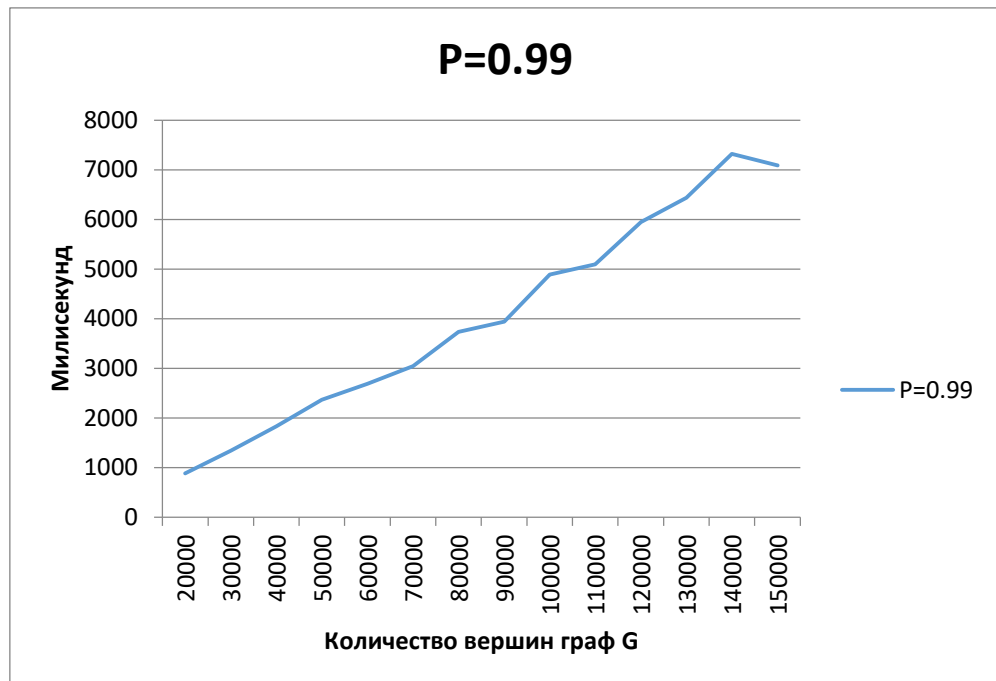


Рисунок 12 – Точность гипотезы $Prec = 0.99$

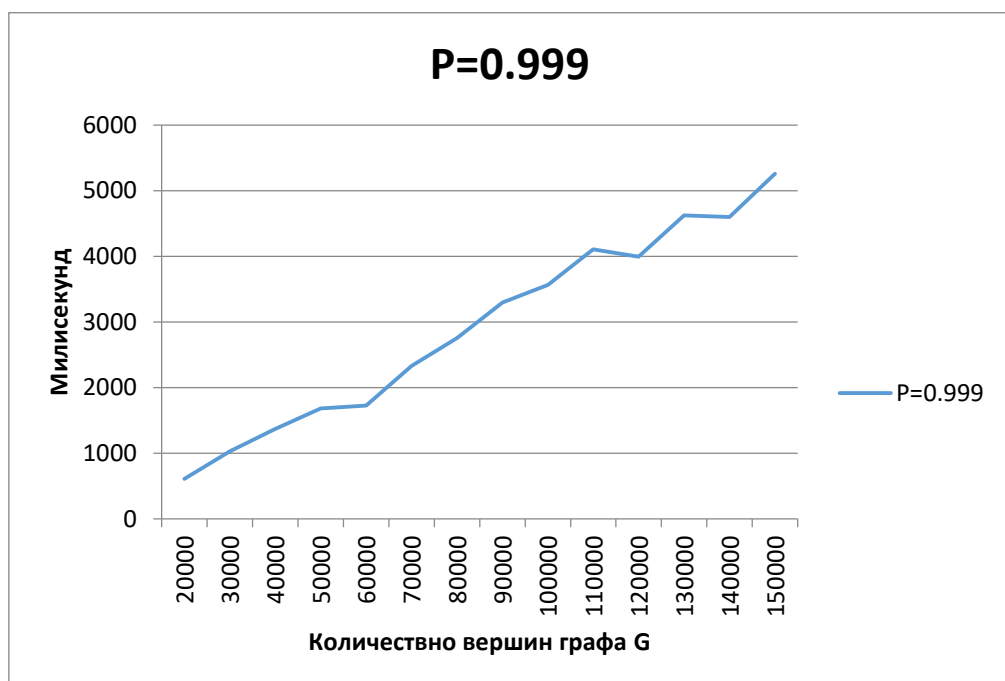


Рисунок 13 – Точность гипотезы $Prec = 0.999$

Из графика можно сделать вывод, что зависимость носит линейный характер и выбор hash-table при генерации графов был оправдан, так как быстрее чел

линейную сложность этот алгоритм иметь не может. Это следует из того, что инициализация графа занимает линейное время, а сложность алгоритма можно охарактеризовать его самым медленным участком [10].

Выводы по главе три

После измерения скорости модуляции взаимодействия по протоколу нулевого разглашения, составлена таблица и построен график. По графику видно, что скорость работы программы носит линейный характер. Это подтверждает аналитическую скорость работы hash-table.

После сравнения результатов полученных с построения двух графиков точности, можно сделать вывод: скорость работы модуляции для достижения точности $Prec = 0.999$, не имеет существенных отличий от точности гипотезы $Prec = 0.99$. Это означает хорошую устойчивость алгоритма шифрования.

ЗАКЛЮЧЕНИЕ

Протокол нулевого разглашения – это еще не полностью изученная область криптографии, основанная на математической теории о хранении информации. На данный момент нет реализации применения теории на практике.

В ходе работы был разработан алгоритм, модулирующий действия клиента и сервера при взаимодействии по протоколу нулевого разглашения. На каждом этапе алгоритма было соблюдено условие информационной защищенности сторон. Следовательно, протокол неразглашения был выполнен.

При разработке алгоритма учитывалась скорость его работы. Скорость работы алгоритма можно обозначить самым медленным его участком. Так, весь алгоритм модуляции выполняется за сложность $O(N * t)$, где N - количество вершин графа, t – средняя степень вершины графа. Следующая сложность достигнута путем использования hash-table, при проверки наличия ребра между вершинами. Так как в hash-table запрос наличия вершины выполняется за $O(1)$, или за константное время.

Модуляция построена на трех независимых классах, что позволяет выполнять условие нулевого разглашения, сохранения информационной защищенности сторон. Каждый класс имеет доступный интерфейс для масштабируемости и возможного встраивания в иные системы. Методы и переменные имеют название строго соответствующее задаче, которую они выполняют, что позволило создать “читаемый код”, в рамках концепции ООП.

Разработана система тестирования, позволяющая тестировать, как отдельные участки модуляции на скорость. Например, одну итерацию модуляции, так и полноценное модулирование системы. Каждый из участков в случае недопустимого поведения алгоритма, сообщает о месте и наличии ошибки. Это позволило провести ряд тестов и предоставить достоверные данные о скорости работы алгоритма.

Языком разработки алгоритма был выбран C#. Из-за того что, язык программирования C# претендует на подлинную объектную ориентированность, ориентирован на безопасность кода, содержит унифицированную систему типизации (в сравнении с C и C++), имеет расширенную поддержку событийно-ориентированного программирования. Именно эти факторы позволили создать надежную и точную модуляцию работы протокола нулевого разглашения.

Для анализа результатов был выбран подход графического отображения табличной зависимости, из которой можно сделать вывод, что скорость алгоритма носит линейный характер. Следовательно, достигнута предельно возможная сложность алгоритма взаимодействия. Благодаря распределению нагрузки между сервером и клиентом уменьшается среднее время работы алгоритма.

В заключении, актуальность поиска нового вида шифрования и сохранности электронных подписей клиентов, важная задача. Так как, чем дольше известен алгоритм шифрования, тем больше находится методов его взлома. Протокол нулевого разглашения лишен такого изъяна. Это следует из математической базы алгоритма. Именно этот ключевой фактор является основной особенностью протокола нулевого разглашения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК:

1. Фергюсон, Н. Практическая криптография. / Н. Фергюсон, Б.Шнаер.- Пер. с англ. — М. : Издательский дом “Вильямс”, 2004. — 432 с.
2. Черемушкин, А.В. Криптографические протоколы. Основные свойства и уязвимости / А.В. Черемушкин - М. : Академия, 2009. — 272 с.
3. Коваленко Е.А. Алгоритм шифрования данных RSA /Е.Ф.Коваленко, О.С. Ключко// Электронный журнал: наука, техника и образование. 2016. №3 (7). С.24-34.
URL:<http://ntojournal.ru/uploads/articles/ead0c9f23f54e72352b27416ea9ef7fb.pdf> (дата обращения 31.05.2017);
4. Matthew Green, Zero Knowledge Proofs// Math \cap Programming
URL: <https://jeremykun.com/2016/07/05/zero-knowledge-proofs-a-primer>
5. Окольнишникова Е. А. , “О числе гамильтоновых циклов в гамильтоновых плотных графах”/ Е. А. Окольнишникова, Матем. тр., 8:2 (2005), 199–206; Siberian Adv. Math., 16:4 (2006), 79–85
6. Goldreich O. A Short Tutorial of Zero-Knowledge // Secure Multi-Party Computation — Amsterdam, Berlin, Tokyo, Washington, D.C.: IOS Press, 2013. — P. 28–60. — 285 p. — ISBN 978-1-61499-168-7
7. Павловская, Т. А. С / С++. Программирование на языке высокого уровня / Т.А. Павловская – СПб.: Питер, 2003. — 461 с.
8. Гмурман, В. Е. Теория вероятностей и математическая статистика / В. Е. Гмурман. – М.: Высшая школа, 1977. — 401 с.
9. Гирш Э. А. «Сложность вычислений и основы криптографии». Курс лекций описывающий основы сложности вычислений и криптографии;
URL: <http://logic.pdmi.ras.ru/csclub/courses/complexityandcryptography>
10. Ахо А., Построение и анализ вычислительных алгоритмов/ Ахо А., Хопкрофт Дж., Ульман Дж. М.: Мир, 1979. 535 с;
11. Басакер, Р. Конечные графы и сети / Р. Басакер, Т. Саати. – М.: Наука, 1974. — 367 с.

ПРИЛОЖЕНИЕ А
ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Институт естественных и точных наук
Кафедра «Математическое и компьютерное моделирование»

Приложение-идентификатор, основанное на криптографическом методе
доказательства с нулевым решением
ТЕХНИЧЕСКОЕ ЗАДАНИЕ
НА РАЗРАБОТКУ ПРОГРАММНОГО КОМПЛЕКСА «ИДЕНТИФИКАТОР»
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ– 01.03.02.2017.074.12.000 ВКР

Нормоконтролер,
Доцент каф. МиКМ, к.ф.-м.н.,
_____ Т.А. Макаровских
_____ 2017 г.

Руководитель проекта,
Проф., д.ф.-м.н.,
_____ А.В. Панюков
_____ 2017 г.

Автор работы
Студент группы ЕТ-485
_____ А.Д. Ромасс
_____ 2017 г.

Челябинск, 2017

1. ВВЕДЕНИЕ

1.1 Наименование программного изделия

Полное наименование программы - «Приложение - идентификатор, основанное на криптографическом методе доказательства с нулевым решением». Краткое наименование – программа-идентификатор.

1.2 Область применения

Программа предназначена для регистрации и повторной проверки личности пользователя при входе в систему.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1 Документ, на основании которого ведется разработка

Приложение - идентификатор разрабатывается в выпускной работе в соответствии с учебным планом кафедры МиКМ («Математическое и компьютерное моделирование») ЮУрГУ.

2.2 Организация, утвердившая этот документ и дата его утверждения

Задание утверждено на д.ф.-м.н, проф. Кафедры МиКМ, А. В. Панюковым

2.3 Наименование темы разработки

Наименование темы разработки - ZeroPasport

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Разработка является выпускной квалификационной работой.

4. ТРЕБОВАНИЯ К ПРОГРАММЕ

4.1 Требования к функциональным характеристикам

4.1.1 Состав выполняемых функций

4.1.1.1 Программа идентификатор должна осуществлять проверку подлинности по ключу.

4.1.1.2 При запуске программы с помощью файла ZeroPasport.exe программа должна обеспечить загрузку рабочего окна программы. Загрузка и сохранение файла осуществляется через главное меню программы.

4.1.1.5 В любой момент работы программы при нажатии клавиши F1 либо при выборе пункта «Помощь» главного меню должны выводиться тексты файлов помощи со списком всех возможных команд программы на данный момент.

4.1.1.6 Программа должна обеспечить вывод на принтер содержимое текстового файла стандартными символами принтера с размером текста, заданным пользователем.

4.1.2 Организация входных и выходных данных

В процессе работы программы входной информацией являются коды клавиш, нажимаемых пользователем на клавиатуре, согласно режимам, определяемым выходной информацией.

4.1.3 Временные характеристики и размер занимаемой памяти

Время реакции программы на нажатие любой из клавиш не должно превышать 0.25 с, за исключением реакций на чтение и запись входных и выходных файлов.

4.2 Требования к надежности

4.2.1 Требования к надежному функционированию

Программа должна нормально функционировать при бесперебойной работе ЭВМ. При возникновении сбоя в работе аппаратуры восстановление нормальной работы программы должно производиться после:

- 1) перезагрузки операционной системы;
- 2) запуска исполняемого файла программы, повторного выполнения действий, потерянных до последнего сохранения информации в файл на диске.

Уровень надежности программы должен соответствовать технологии программирования, предусматривающей:

- 1) инспекцию исходных текстов программы;
- 2) автономное тестирование модулей программы;
- 3) комплексное тестирование программы.

4.2.2 Контроль входной и выходной информации

Программа должна контролировать выбор пользователем пункта меню «Выход» и предупреждать о возможной потере несохраненных изменений.

4.2.3 Время восстановления после отказа

Время восстановления после отказа должно состоять из:

- 1) времени запуска пользователем исполняемого файла программы;

2) времени повторной загрузки текстового файла.

4.3 Условия эксплуатации

Программа должна храниться в виде двух маркированных копий: эталонной и рабочей. Периодическая перезапись информации должна осуществляться согласно нанесенной маркировке. Условия хранения дисков с программой должны соответствовать нанесенной на них маркировке.

4.4 Требования к составу и параметрам технических средств

Программа должна корректно работать на следующем или совместимом с ним оборудовании:

- 1) персональный компьютер с процессором Pentium и выше;
- 2) принтер.

4.5 Требования к информационной и программной совместимости

4.5.1 Требования к информационным структурам на входе и выходе

Требования к информационным структурам на входе и выходе определены в п.4.1.2.

4.5.2 Требования к методам решения

Требования к методам решения определены в пп.4.1.1.2. Выбор остальных методов решения осуществляется разработчиком без согласования с заказчиком.

4.5.3 Требования к языкам программирования

Язык программирования выбирается разработчиком без согласования с заказчиком.

4.5.4 Требования к программным средствам, используемым программой

Для работы программы необходима операционная система Windows XP и выше.

4.6 Требования к маркировке и упаковке

Диски с эталонным и рабочими экземплярами программы должны иметь маркировку, состоящую из надписи «ZeroPassport», надписи «эталон» или «рабочая», даты последней перезаписи программы. Упаковка должна соответствовать условиям хранения диска. На упаковке должны быть указаны условия транспортирования и хранения диска.

4.7 Требования к транспортированию и хранению

Условия транспортирования и хранения диска должны соответствовать п.4.6.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Состав программной документации должен включать следующие документы:

- 1) технический проект программы по ГОСТ 19.404-79 в машинописном исполнении;
- 2) описание программы по ГОСТ 19.402-78 на компакт-диске;
- 3) текст программы по ГОСТ 19.401-78 на компакт-диске;
- 4) руководство программиста по ГОСТ 19.504-79 на компакт-диске в виде файла README.txt.

Пояснительная записка «Технический проект программы» должна содержать следующие разделы.

1. Раздел «Входные данные» (характер, организация и предварительная подготовка входных данных).

2. Раздел «Выходные данные» (характер и организация и выходных данных).

3. Раздел «Описание логической структуры» при технологии структурного программирования должен включать следующие материалы:

- описание связей программы с другими программами;
- описание внутренних массивов и переменных, которые используются в межмодульном обмене данными;

4. Раздел «Используемые технические средства» (типы ПК, на которых возможно выполнение программы; устройства, используемые при выполнении программы).

5. Раздел «Вызов и загрузка» (виды носителей программы, их используемый объем; способы вызова программы с соответствующих носителей данных; входные точки в программу – запуск программы).

6. Раздел «План мероприятий по разработке и внедрению программы» (план мероприятий разрабатывается для реализации программы коллективом программистов; планом должны быть предусмотрены контрольные временные точки реализации, в течение которых происходит интеграция разработанных модулей и те-

стирование уже разработанной части программы; приводится состав тестов и принципы их подготовки для тестирования уже созданного фрагмента программы для каждой из контрольных точек).

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Технико-экономические показатели должны определяться заказчиком без участия исполнителя.

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Разработка программы должна выполняться по следующим этапам:

- 1) разработка, согласование и утверждение технического проекта программы с пояснительной запиской – 5 недель.
- 2) разработка рабочего проекта программы с комплексным тестированием – 6 недель;
- 3) приемка-сдача с исправлением обнаруженных недостатков в программе и программной документации – 2 недели.
- 4) внедрение.

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

8.1 Виды испытаний

Испытания программы и верификация документации должны проводиться в организации заказчика с привлечением сторонних экспертов. Проверочные тесты должны готовиться заказчиком.

8.2 Общие требования к приемке

Приемка программы должна осуществляться заказчиком. Программа должна считаться годной, если она удовлетворяет всем пунктам данного технического задания.

ПРИЛОЖЕНИЕ Б
КОД ПРИЛОЖЕНИЯ

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Институт естественных и точных наук
Кафедра «Математическое и компьютерное моделирование»

Приложение-идентификатор, основанное на криптографическом методе
доказательства с нулевым решением
ТЕКСТ ПРОГРАММЫ «ИДЕНТИФИКАТОР»
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ– 01.03.02.2017.074.12.000 ВКР

Нормоконтролер,
Доцент каф. МиКМ, к.ф.-м.н.,
_____ Т.А. Макаровских
_____ 2017 г.

Руководитель проекта,
Проф., д.ф.-м.н.,
_____ А.В. Панюков
_____ 2017 г.

Автор работы
Студент группы ЕТ-485
_____ А.Д. Ромасс
_____ 2017 г.

ПРИЛОЖЕНИЕ Б

Файл program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Diagnostics;
using System.Threading;
using System.Runtime.Remoting.Metadata.W3cXsd2001;
using System.Text;
using System.Threading.Tasks;

namespace example
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 20000; i < 160000; i += 10000)
            {
                var model = new modelOfZeroKnowledge(new PrivateKey(i, 30));
                Console.WriteLine(model.ShowTimerOfAll(0.999));
            }
        }
    }

    class modelOfZeroKnowledge
    {
        public Checker singleCheker;
        public PrivateKey OurKey;

        public modelOfZeroKnowledge(PrivateKey Key)
        {
            OurKey = Key;
        }
        public int ShowTimerOfAll(double P)
        {
            Stopwatch stopWatch = new Stopwatch();
            stopWatch.Start();

            precision(P);

            stopWatch.Stop();
            // Get the elapsed time as a TimeSpan value.
            TimeSpan ts = stopWatch.Elapsed;
            string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}",
            ts.Minutes, ts.Seconds, ts.Milliseconds);
            //Console.WriteLine("RunTime " + elapsedTime);
            return ts.Seconds * 1000 + ts.Milliseconds;
        }
        public void ShowTimerOfoneTick()
        {
            Stopwatch stopWatch = new Stopwatch();
            stopWatch.Start();

            precision(0.5);

            stopWatch.Stop();
            // Get the elapsed time as a TimeSpan value.
```



```

    TimeSpan ts = stopwatch.Elapsed;
    string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}",
    ts.Minutes, ts.Seconds, ts.Milliseconds);
    Console.WriteLine("RunTime " + elapsedTime);
}
public void precision(double P)
{
    double reliability = 0;
    double rate = 0.5;
    int i = 0;
    var rnd = new Random();
    while (i < 100 && reliability < P)
    {
        OurKey.IsomorphicTransformation();
        singleCheker = new Checker(OurKey.OpenKey);
        int l = rnd.Next(2);
        bool f = true;
        if (l == 1)
        {
            f= singleCheker.CheckPath(OurKey.GetPrivateKeyPath());
        }
        else
        {
            f = singleCheker.CheckIsomorphic(OurKey.graph,OurKey.GetIsomorphicKey());
        }
        if (f)
        {
            reliability += rate;
            rate /= 2;
            //Console.WriteLine("success on " + reliability);
        }
        else
        {
            //Console.WriteLine("EROOR ");
            break;
        }
    }
}
}
class Checker
{
    public List<List<int>> Key;

    public Checker(List<HashSet<int>> OpenKey)
    {
        Key = new List<List<int>>();
        for (int i = 0; i < OpenKey.Count; ++i)
        {
            Key.Add(new List<int>());
            foreach (var vertex in OpenKey[i])
            {
                Key[i].Add(vertex);
            }
        }
    }

    public Boolean CheckPath(List<int> path)//проверяю есть ли из каждой вершины действи-
    тельно путь
    {
        Boolean answer = true;
        for (int i = 0; i < path.Count-1; ++i)
        {
            if (!Key[path[i]].Contains(path[i + 1])) answer = false;
        }
    }
}

```

```

    }
    if (!Key[path[path.Count-1]].Contains(path[0])) answer = false;
    return answer;
}

public Boolean CheckIsomorphic( List<HashSet<int>> graph, List<int> IsomorphicKey)
{
    Boolean answer = true;

    for (int i = 0; i < graph.Count; ++i)
    {
        if (graph[i].Count() != Key[IsomorphicKey[i]].Count()) answer = false;
        var IsomorphicV = new HashSet<int>();
        foreach(var v in graph[i])
        {
            IsomorphicV.Add(IsomorphicKey[v]);
        }

        foreach(var v in Key[IsomorphicKey[i]])
        {
            if (!IsomorphicV.Contains(v)) answer = false;
        }
    }

    return answer;
}
}
class PrivateKey
{
    public List<HashSet<int>> graph;//внос исправления с хэш сетов не проверял граф вроде
работает

    private int _size;
    public List<int> _path;
    //создать изоморфный ключ
    public List<HashSet<int>> OpenKey;
    private List<int> _IsomorphicKey;
    public int degree;

    public void InitKey(int size)
    {
        _size = size;
        _path = new List<int>();
        graph = new List<HashSet<int>> >();
        for (int i = 0; i < _size; ++i)
        {
            graph.Add(new HashSet<int>());
        }
    }

    }
    public PrivateKey(int size,int d)
    {
        InitKey(size);
        degree = d;
        GeneratePath();
        GenerateGraph();
    }
    public PrivateKey(int size, List<int> path,List<HashSet<int>> > graph)//check it
    {
        InitKey(size);
        foreach (var vertex in path)
        {

```

```

        _path.Add(vertex);
    }
    foreach (var vertex in graph)
    {
        this.graph.Add(vertex);
    }
}
private List<int> Shafle(int size)
{
    var answer=new List<int>();
    for (int i = 0; i < _size; ++i)
    {
        answer.Add(i);
    }
    Random rnd = new Random();
    for (int i = 0; i < _size; ++i)
    {
        var a = rnd.Next(_size);
        var b = rnd.Next(_size);//переписать нормально
        int copy = answer[a];
        answer[a] = answer[b];
        answer[b] = copy;
    }
    return answer;
}
public void GeneratePath()
{
    _path = Shafle(_size);
}
public void GenerateGraph()
{
    for (int i = 0; i < _size - 1; ++i)
    {
        graph[_path[i]].Add(_path[i + 1]);
    }
    graph[_path[_size - 1]].Add(_path[0]);

    Random rnd = new Random();
    for (int i = 0; i < _size; ++i)
    {
        int count = rnd.Next(degree);
        for (int j = 0; j < count; ++j)
        {
            graph[i].Add(rnd.Next(_size - 1));
        }
    }
}

public List<int> GetPrivateKeyPath()
{
    var privateKey=new List<int>();
    for (int i = 0; i < _path.Count; ++i)
    {
        privateKey.Add(_IsomorphicKey[_path[i]]);
    }
    return privateKey;
}
public List<int> GetIsomorphicKey()
{
    return _IsomorphicKey;
}
public void IsomorphicTransformation()
{

```

```

        _IsomorphicKey = Shafle(_size);

        OpenKey = new List<HashSet<int>>();
        for (int i = 0; i < _size; ++i)
        {
            OpenKey.Add(new HashSet<int>());
        }
        for (int i = 0; i < _size; ++i)
        {
            foreach (var vertex in graph[i])
            {
                OpenKey[_IsomorphicKey[i]].Add(_IsomorphicKey[vertex]);
            }
        }
    }

    public void ShowOpenKey()
    {
        Console.WriteLine("Vertex : Connecting vertex");
        for (int i = 0; i < graph.Count; ++i)
        {
            string numberOfvertex = "";
            numberOfvertex += i;
            if (numberOfvertex.Count() < 2) numberOfvertex += " ";
            Console.Write(" "+numberOfvertex + " : ");
            foreach(var v in graph[i])
            {
                Console.Write(v+" ");
            }
            Console.WriteLine("\n");
        }
    }

    public void ShowIsoporphicKey()
    {
        Console.WriteLine("Vertex : Connecting vertex");
        for (int i = 0; i < OpenKey.Count; ++i)
        {
            string numberOfvertex = "";
            numberOfvertex += i;
            if (numberOfvertex.Count() < 2) numberOfvertex += " ";
            Console.Write(" " + numberOfvertex + " : ");
            foreach (var v in OpenKey[i])
            {
                Console.Write(v + " ");
            }
            Console.WriteLine("\n");
        }
    }
}
}
}
}
}
}

```