**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное автономное образовательное учреждение**

**высшего профессионального образования**

**«Национальный исследовательский университет «МИЭТ»**

**Факультет** Микроприборов и технической кибернетики (МПиТК)

**Кафедра «**Телекоммуникационные системы»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**НА ТЕМУ:**

**Разработка программно-аппаратной системы двухфакторной аутентификации**

**Студент /Р.В. Иванов /**

**(подпись)**

**Руководитель /В. П. Батура/**

**(подпись)**

**Консультант /О. П. Симонова/**

**(подпись)**

**2015 г.**

Техническое задание на дипломный проект

* 1. Назначение и область применения. Программно-аппаратная система двухфакторной аутентификации предназначена для защиты от несанкционированного доступа к информации.
  2. Цель работы:

Обеспечение защиты от несанкционированного доступа автоматизированной системы управления и открытого канала связи, предназначенного для аутентификации, путем применения средств, криптографической защиты информации.

* 1. Задачи проекта:

1. реализовать программную часть системы;
2. разработать криптографический протокол, на основе российских стандартов, предназначенный для проверки подлинности клиента и обеспечения безопасности пользовательских данных, передаваемых по открытому каналу связи;
3. разработать приложение для тестирования;
4. написать документ, описывающий интерфейс доступа библиотеке.
   1. Требования к платформе:
5. программное обеспечение для сервера: ОС Windows Server 2012 R2
6. программное обеспечение для клиента: ОС Linux (встраиваемая система)
   1. Требования к криптографическому протоколу:
7. применять сертифицированные российские стандарты;
8. соответствовать требованию ФСТЭК для информационных систем 3 класса;
9. протокол должен содержать необходимый для данной задачи функционал.
   1. Результат работы:
10. реализована программная часть системы;
11. реализована библиотека, включающая в себе российские криптографические алгоритмы, предназначенные для данного протокола;
12. разработана библиотека, реализующая криптографический протокол;
13. разработана библиотека, предоставляющий интерфейс доступа к считывателю смарт-карты клиента, через USB интерфейс;
14. разработано приложение для тестирования библиотек;
15. представлен руководящий документ, описывающий интерфейс доступа к программной части системы.
    1. Требования к обеспечению надёжного функционирования программ

Надежное (устойчивое) функционирование ПО должно быть обеспечено выполнением совокупности организационно-технических мероприятий. Перечень организационно-технических мероприятий:

1. использование во время разработки ПО систему контроля версий;
2. использованием лицензионных средств для разработки;
3. Microsoft visualstudio 2012;
4. ОС Linux OPENSUSE 13.1;
5. QtCreater.
6. регулярное выполнения ГОСТ 51188-98 защита информации, испытание программных средств на наличие программных вирусов;
7. статический и динамический анализ исходного кода;
   1. Требования к программной документации

Программная документация содержит:

1. техническое задание;
2. программы и методики испытаний;
3. описание интерфейсных функций библиотек:
4. криптографических алгоритмов;
5. криптографического протокола;
6. протокола взаимодействия со смарт-картой по USBинтерфейсу;
7. описание приложения для тестирования библиотек:
8. руководств по применению библиотек.
   1. Стадии разработки

Разработка должна быть проведена в 4 стадии:

1. разработка технического задания;
2. разработка программных обеспечений;
3. тестирование по методикам испытаний:
4. внедрение продуктов.
   1. Этапы разработки

Разработка должна быть проведена в нескольких этапах. На стадии разработки технического задания должно быть поставлены основные требования к системе:

1. функциональные возможности ПО;
2. требования к аппаратным составляющим;
3. требования к защите информации и криптографическому протоколу;
4. сроки разработки.
5. разработка приложений для тестирования криптографического протокола;
6. написание программной документации;
   1. Порядок контроля. Статический анализ исходного кода.

Программный код должен быть исследован с помощью статистического анализа, с целью выявления возможных ошибок, а так же потенциально небезопасного кода на этапе написания программы, должно быть проведено комплексное тестирование.

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных

ИТ – информационные технологии

НСД – несанкционированный доступ

ОС – операционная система

ПО – программное обеспечение

СКЗИ – средства криптографической защиты информации

ЭВМ – электронная вычислительная машина.

СОДЕРЖАНИЕ

[Техническое задание на дипломный проект 2](#_Toc421619841)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 5](#_Toc421619842)

[ВВЕДЕНИЕ 8](#_Toc421619843)

[1 Обзор современных методов аутентификации 10](#_Toc421619844)

[1.1 Основные понятия 10](#_Toc421619845)

[1.2 Однофакторная аутентификация. Основные недостатки и 12](#_Toc421619846)

[методы атак на систему 12](#_Toc421619847)

[1.3 Двухфакторная аутентификация. Преимущества и недостатки 19](#_Toc421619848)

[2 Криптографические протоколы аутентификации 21](#_Toc421619849)

[2.1 Основные понятия 21](#_Toc421619850)

[2.2 Алгоритмы аутентификации клиента 22](#_Toc421619851)

[2.3 Протокол обмена сессионным ключом. 27](#_Toc421619854)

[3 Описание разрабатываемого криптографического протокола 29](#_Toc421619855)

[3.1 Общие сведения. 29](#_Toc421619856)

[3.2 Описание первого этапа протокола 30](#_Toc421619857)

[3.3 Описание второго этапа протокола 31](#_Toc421619858)

[3.4 Описание третьего этапа протокола. 34](#_Toc421619859)

[4 Реализация программной части 36](#_Toc421619860)

[4.1 Общие сведения. Архитектура 36](#_Toc421619861)

[4.2 Алгоритм симметричного шифрования ГОСТ 28147-89 37](#_Toc421619862)

[4.3 Алгоритм вычисления хеш-функции ГОСТ Р11.34-2012 40](#_Toc421619863)

[4.4 Алгоритм обмена сессионным ключом по протоколу Диффи-Хеллмана 41](#_Toc421619864)

[4.5 Формат передаваемых сообщений 42](#_Toc421619865)

[4.6 Реализация первого этапа протокола 43](#_Toc421619866)

[4.5 Реализация второго этапа протокола 44](#_Toc421619867)

[4.6 Реализация третьего этапа протокола 44](#_Toc421619868)

[5 Тестирование. Статический анализ исходного кода. 46](#_Toc421619869)

[ЗАКЛЮЧЕНИЕ 48](#_Toc421619870)

[Список использованных источников 50](#_Toc421619871)

[ПРИЛОЖЕНИЕ A Описание заголовочных файлов (интерфейсные функции и классы) 52](#_Toc421619872)

[ПРИЛОЖЕНИЕ Б Описание кодов ответа и запроса 59](#_Toc421619873)

# ВВЕДЕНИЕ

Защита от несанкционированного доступа, является одним из актуальных и приоритетных задач в обеспечении безопасности информационных и инфокоммуникационных систем. Интерес к данной теме на сегодняшний день велик, так как несанкционированный доступ к информации представляет с собой множество серьезных угроз. Среди них:

* кража конфиденциальной информации;
* изменение критических данных;
* захвата управления над автоматизированными системами и оборудованием;
* несанкционированного использования ресурсов и сервисов;
* дезорганизации работы информационной системы и сопутствующих бизнес-процессов организации;
* беспрепятственного выноса за пределы системы ценных данных.

Наиболее серьёзной угрозой является получение доступа и изменения критически важных данных в системах автоматизированного управления. Такая угроза возможна, вследствие недостаточно надежной системы аутентификации, предназначенного для доступа к автоматизированным системам управления. Для решения данной задачи представлена программно-аппаратная система двухфакторной аутентификации.

Автоматизированные системы управления применяются во всех сферах человеческой деятельности. Начиная от самых простых устройств общего назначения, такие как персональные ЭВМ, заканчивая целыми системами, применяющиеся в производстве, сфере энергетики, транспорта, военной промышленности, космической сфере и т. п. Очевидно, что ущерб, который могут нанести злоумышленники, при получении доступа к таким системам, очень велик. Поэтому **предметом** моего исследования стало: безопасность доступа к системам автоматизации и управления.

Уязвимым местом, в обеспечении информационной безопасности, является *аутентификация*. Многие системы очень часто пренебрегают этот фактор защиты или используют примитивный и не достаточно эффективный, на сегодняшний день, способ. Что приводит к получению доступа в систему злоумышленникам. **Целью** работы является обеспечение защиты от несанкционированного доступа автоматизированной системы управления и открытого канала связи, предназначенного для аутентификации, путем применения средств, криптографической защиты информации.

Для достижения поставленной цели, необходимо решить следующие **задачи**:

1. реализовать программную часть системы;
2. разработать криптографический протокол, на основе российских стандартов, предназначенный для проверки подлинности клиента и обеспечения безопасности пользовательских данных, передаваемых по открытому каналу связи;
3. разработать приложение для тестирования;
4. написать документ, описывающий интерфейс доступа к программной части.

Многие системы автоматизированного управления представляют с собой информационную систему, использующую модель «клиент-сервер». Клиент представляет с собой сложную встраиваемую систему. Большая часть таких систем, работают на специальной версии ОС Linux. Это позволяет построить систему очень гибко, расширяемо и с возможностью выполнять сложные задачи. Именно поэтому платформой для клиента является ОС Linux. Сервер работает на ОС от Microsoft – Windows Server 2012R2, предназначенный специально для серверных систем.

Основной частью работы является разработка программной части системы. Сюда входят, крипто провайдер, включающий в себя реализацию необходимых криптографических алгоритмов, криптографический протокол со стороны клиента и сервера, приложение тестирования, библиотека доступа к считывателю смарт-карты и руководящий документ, где описан интерфейс доступа к программной части. Основную часть программной части составляет криптографический протокол, который предназначен для проверки подлинности клиента и защиты открытого канала связи для аутентификации.

В ходе работы будут рассмотрены современные методы аутентификации, их преимущества и недостатки, основные угрозы и способы защиты от них, проведены сравнения протоколов проверки подлинности клиента, представлены результаты работы и тестирования приложений.

# 1 Обзор современных методов аутентификации

### Основные понятия

*Идентификация* – процедура распознавания субъекта (пользователя, процесса, действующего от имени пользователя, аппаратно-программного компонента) по его уникальному идентификатору, присвоенному субъекту ранее и занесенному в базу данных в момент его регистрации как легального пользователя системы.

*Аутентификация* – процедура проверки подлинности входящего в систему субъекта, предъявившего свой идентификатор.

*Авторизация* – это процедурой предоставления субъекту определённых прав.

В любой системе аутентификации обычно можно выделить следующие элементы:

* субъект, проходящий процедуру аутентификации;
* характеристика субъекта (отличительная черта);
* система, несущая ответственность за аутентификацию и контролирующий её работу (сервер);
* сам механизм аутентификации, то есть принцип работы системы;
* механизм, предоставляющий или лишающий субъекта определенных прав доступа.

Субъект может подтвердить свою подлинность, предъявив, один из следующих аутентификаторов:

1. Аутентификация на основе «субъект знает» –претендент обладает информацией, которой нет y других объектов компьютерной системы (пароль, персональный идентификационный номер, секретный ключ). Эту информацию субъект демонстрирует в протоколах типа «запрос – ответ».
2. Аутентификация на основе «субъект обладает» – претендент имеет некий физический предмет (магнитную карту, интеллектуальную карту, смарт-карту, генератор паролей), который выполняет для него криптографические преобразования информации;
3. Аутентификация на основе «субъект есть» – проверяются некоторые биометрические данные человека (голос, радужная оболочка глаза, отпечатки пальцев и др.)[11].

*Фактор аутентификации* – определенный вид информации, предоставляемый субъектом системе для его аутентификации. Если в процессе аутентификации используется только один способ аутентификации, такая система называется *однофакторной* аутентификацией, если несколько –*многофакторной*. Для идентификации средствами криптографии все эти три метода аутентификации могут быть сведены к одному – к аутентификации на основе владения какой-либо информацией. Действительно, любые биометрические данные или информация, заключенная на физическом носителе, могут быть преобразованы в уникальный ключ (при идентификации при помощи криптографической системы или протокола) или пароль (при аутентификации или идентификации парольными схемами), который будет однозначно определять субъекта[11].

*Протокол аутентификации* – это криптографический протокол, в ходе которого одна сторона убеждается:

1. в идентичности другой стороны, вовлеченной в протокол;
2. в активности другой стороны во время или непосредственно перед моментом приобретения доказательства.

Аутентификация может быть *односторонней* (когда клиент доказывает свою подлинность серверу) и *двусторонней* или *взаимной* (это обоюдная аутентификация между сторонами обмена информацией).

Примером *односторонней* аутентификации является процедура входа в систему Windows NT.

Примером *двусторонней* является использование протокола *kerberos*.

Протоколы аутентификации характеризуются следующими важными характеристиками:

1. *Вычислительная эффективность* количество операций, необходимых для выполнения протокола;
2. *Коммуникационная эффективность* – данное свойство отражает количество сообщений и их длину, необходимую для осуществления аутентификации;
3. *Наличие третьей стороны* – примером третьей стороны может служить доверенный сервер распределения симметричных ключей или сервер, реализующий дерево сертификатов открытых ключей, основа гарантий безопасности – примером могут служить протоколы, обладающие свойством доказательства с нулевым знанием;
4. *Хранения секрета* – имеется в виду способ хранения критичной ключевой информации.

Также, можно классифицировать протоколы аутентификации по уровню обеспечиваемой безопасности или возможности противостоять определенному классу атак.

В соответствии с данным подходом протоколы аутентификации делят на типы:

* простая аутентификация (на основе использования паролей);
* строгая аутентификация (на основе использования криптографических методов и средств);
* протоколы, обладающие свойством доказательства с нулевым знанием.

### Однофакторная аутентификация. Основные недостатки и

### методы атак на систему

Один из способов аутентификации в компьютерной системе состоит во вводе пользовательского идентификатора («логина» от англ. login – регистрационное имя пользователя) и пароля конфиденциальной информации, доступной только пользователю. Достоверная (эталонная) пара логина и значение пароля хранится на сервере базы данных.

Пароли бывают *фиксированные* и *однократные*.Фиксированные или многократные пароли не меняются при каждом обращении, а однократные используются только один раз. Простая аутентификация имеет следующий алгоритм:

1. Субъект запрашивает доступ в систему и вводит личный идентификатор (логин) и пароль. Поскольку пароль должен храниться в тайне, то он должен шифроваться перед посылкой по незащищенному каналу связи.
2. Идентификатор и пароль поступают на сервер аутентификации, где сравниваются с эталонным идентификатором и паролем. В случае их совпадения, пароль считается *достоверным*, а пользователь – *законным*.
3. При совпадении с эталоном, аутентификация признается успешной, при несовпадении – субъект возвращается на первый шаг. При этом пароль и идентификатор могут передаваться по сети следующими способами:

* в открытом виде (например, при доступе к IIS (*InternetInformationServer*) по протоколу *НТТР* в качестве одного из типов поддерживаемой IIS аутентификации);
* идентификатор пользователя, его пароль вначале *шифруются* на согласованном *ключе*, и в зашифрованном виде пересылается системе;
* идентификатор пользователя, его парольвначале подаются на вход *односторонней* функции *(хеш-функции)*, а выход функции пересылается системе;

Схемы организации простой аутентификации отличаются также способами хранения паролей:

1. Хранение паролей в *открытом* виде в системных файлах, с установленными на них атрибутами защиты от чтения и записи (например, при помощи описания соответствующих привилегий в списках контроля доступа к операционной системе). Криптографические механизмы (шифрование, однонаправленные функции) не используются.

Недостаток: если злоумышленник получит в системе привилегии администратора, то ему станут доступны системные файлы и ресурсы. Для усиления защиты этого способа хранения паролей можно записать парольные файлы на внешние носители (touchmemory, смарт-карты и гибкие магнитные диски). Однако такой подход так же не гарантирует полную безопасность.

1. Более надежным методом хранение паролей является хранение в *зашифрованном* или *хешированном* виде. При проверке введенного пароля система вычисляет его хеш-образ и сравнивается результатом хеш-образа пароля, который храниться в базе данных, соответствующий логину пользователя. Такой подход хранения более надежный, чем предыдущий.

Всё-таки, главной угрозой *парольной аутентификации* – это *взлом*, т.е. раскрытие злоумышленником пароля пользователя, дающего право входить в систему.

К основным атакам паролей пользователей относятся:

1. Полный перебор паролей;
2. Атака с помощью словаря;
3. Атака с помощью радужных таблиц;
4. Метод социальной инженерии (основан на предположении, что пользователь использовал в качестве пароля личные сведения – имя или фамилия, дата рождения и т.п.);
5. Установка вредоносных программ для перехвата пароля;
6. Подмена доверенного объекта сети *(IP-spoofing)*;
7. Перехват пакетов *(sniffing)*;

Поэтому, для уменьшения деструктивного влияния человеческого фактора необходимо реализовать ряд требований к выбору и использованию паролей:

1. Задание минимальной длины пароля для затруднения метода полного перебора.
2. Использование в пароле различных групп символов для усложнения подбора пароля.
3. Проверка и отбраковка пароля по словарю для затруднения проведения злоумышленником словарной атаки.
4. Установление максимального срока действия пароля для затруднения метода полного перебора паролей, в том числе и в режиме «offline» при взломе предварительно похищенных учетных записей пользователей.
5. Ограничение числа попыток ввода пароля для предотвращения интерактивного подбора пароля злоумышленником.
6. Использование задержки при вводе неправильного пароля для предотвращения интерактивного подбора пароля злоумышленником (в режиме «onlinе»).
7. Поддержка режима принудительной смены пароля пользователя для эффективности реализации требования, ограничивающего максимальный срок действия пароля. Далее рассмотрим атаки на парольную аутентификацию подробнее.

#### 1.2.1 Полный перебор паролей

Полный *перебор паролей* – прямой перебор всех возможных сочетаний допустимых в пароле символов. Поиск можно несколько ограничить, указав число символов в пароле, их тип (буквы, цифры, спецзнаки) или задав символы, с которых должен начаться поиск.

Поиск можно несколько ограничить, указав число символов в пароле, их тип (буквы, цифры, спецзнаки) или задав символы, с которых должен начаться поиск.

Таблица 1 - Иллюстрация зависимости времени полного перебора паролей от их длины (при скорости перебора паролей 1 000 000 паролей/сек) [1]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алфавит | Мощность алфавита | Время взлома паролей из 3 символов (сек) | Время взлома паролей из 6 символов | Время взлома паролей из 8 символов | Время взлома паролей из 12 символов |
| Маленькие латинские буквы | 26 | 0.02 | 5 мин. | 58 час | 3000 лет |
| Маленькие латинские буквы и цифры | 36 | 0.04 | 36 мин. | 32 дня | 150 000 лет |
| Маленькие и большие латинские буквы, цифры | 62 | 0.02 | 15 час | 7 лет | 100 млн. лет |
| Маленькие и большие латинские буквы, цифры и спецсимволы | 95 | 1 | 8 дней | 193 год | Дольше, чем существует планета Земля |

Сегодня, NIST рекомендует для наилучшей защиты использовать 12-символьный пароль, который должен генерироваться с помощью 95-символьного алфавита (набор ASCII). Примеры ниже иллюстрируют ошибки, приводящие к генерации слабых паролей. Каждый из приведенных паролей основан на простом шаблоне, из чего следует очень низкая энтропия, что позволяет очень быстро их угадывать.

1. Пароль по умолчанию: «password», «default», «admin», «guest» и др. Список паролей по умолчанию широко распространен по интернету;
2. Словарные слова: «chameleon», «RedSox», «sandbags», «bunnyhop!», «IntenseCrabtree» и др., включая слова из не английских словарей;
3. Слова с добавленными числами: «password1», «deer2000», «ivan1234» и др. Могут быть очень быстро проверены;
4. Слова с простой заменой букв: «p@ssw0rd», «l33th4x0r», «g0ldf1sh» и др. Могут быть проверены автоматически с небольшими временными затратами;
5. Удвоенныеслова: «crabcrab», «stopstop», «treetree», «passpass» идр.
6. Последовательности знаков, соответствующие рядом стоящим клавишам на клавиатуре: «qwerty», «12345», «asdfgh», «fred» и др.
7. Числовые последовательности, основанные на хорошо известных наборах: 911, 314159... или 271828..., 112358... и др.
8. Личные данные пользователя: «ivpetrov123», «1/1/1970», номер телефона, "%username%, номер ИНН, адрес и другие.

Атаки с помощью словаря. Атака с помощью словаря– атака на систему защиты, использующая метод полного перебора паролей, при котором перебираются все слова определенного вида и длины из словаря (слова в чистом виде или их зашифрованные образы). Атака основана на предположении, что в пароле используются существующие слова какого-либо языка либо их сочетания.

Различают два вида таких атак: «online» атаки, в которых атакующему для проверки корректности пароля необходимо взаимодействие с сервером. «Offline» атаки, когда злоумышленник может проверить все допустимые пароли, не нуждаясь в обратной связи с сервером. Вероятностная оценка успеха атак по словарю равна отношению количества взломанных паролей при атаке по словарю к общему числу попыток. Для полного перебора или перебора по словарю могут использоваться специальные ПО. Такие как:

* PasswordsPro;
* MD5BFCPF;
* JohntheRipper.

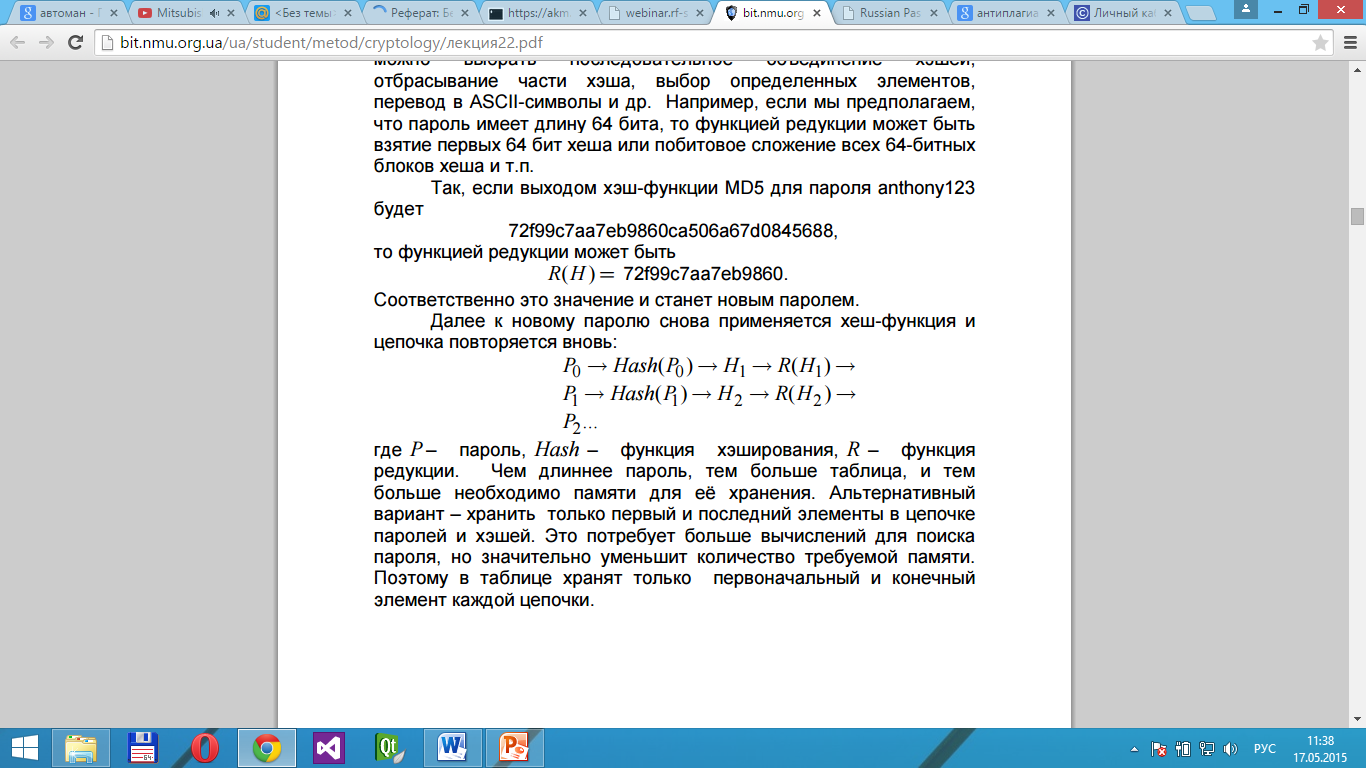
Противодействия online атакам со словарем:

1. Задержка ответа *(delayed response):* для предоставленной пары логин/пароль сервер использует небольшую, специально сгенерированную задержку ответа (не чаще одного ответа в секунду);
2. Блокировка учетной записи *(accountlocking)* после нескольких неудачных попыток ввода пары логин/пароль (например, блокировка на час после пяти неправильных попыток ввода пароля). Первые два способа, в большинстве случаев, мешают проведению словарной атаке и взлому пароля за допустимое время.
3. Предполагается, что ввод верной комбинации логин/пароль производится реальным человеком, а атака по словарю – автоматической программой. Поэтому попытка ввода правильного пароля должна быть «вычислительно простой» для человека, и «вычислительно сложной» для машины.

#### 1.2.2 Атаки с помощью радужных таблиц

Атаки на пароли с помощью радужных таблиц предложены Филиппом Окслином в 2003 году и обусловили существенное увеличение скорости подбора паролей. По сути, технология радужных таблиц представляет собой один из вариантов компромисса «время-память» и позволяет сократить перебор вариантов, например, паролей за счет использования больших предвычисленных таблиц. В таком случае атака состоит из двух этапов — трудоемкого построения таблиц, которое может быть выполнено заранее на мощном оборудовании, и быстрой атаки на хеш-образ пароля, которая может занимать несколько секунд или минут на обычном персональном компьютере. Создание таблицы. Принцип генерации радужной таблицы следующий: каждая цепочка начинается со случайного возможного пароля, который подвергается действию хеш-функции и функции редукции. Функция редукции – это функция, преобразующая выход хеш-функции в некий возможный пароль. В качестве такой функций можно выбрать последовательное объединение хешей, отбрасывание части хэша, выбор определенных элементов, перевод в ASCII-символы и др. Например, если мы предполагаем, что пароль имеет длину 64 бита, то функцией редукции может быть взятие первых 64 бит хеша или побитовое сложение всех 64-битных блоков хеша и т.п.

Так, пусть выходом хэш-функции для пароля будет «72 F9 9С 7A A7 EB 98 60 CA 506A 67 D0 84 56 88», то функцией редукции может быть R(H) = «72 F9 9C 7A A7 EB 98 60». Соответственно это значение и станет новым паролем. Далее к новому паролю снова применяется хеш-функция, и цепочка повторяется вновь:



*P*– пароль, *Hash* – функция хеширования, *R* – функция редукции. Чем длиннее пароль, тембольше таблица, и тем больше необходимо памяти для её хранения. Альтернативный вариант – хранить только первый и последний элементы в цепочке паролей ихеш значений. Это потребует больше вычислений для поиска пароля, но значительно уменьшит количество требуемой памяти. Поэтому в таблице хранят только первоначальный и конечный элемент каждой цепочки.

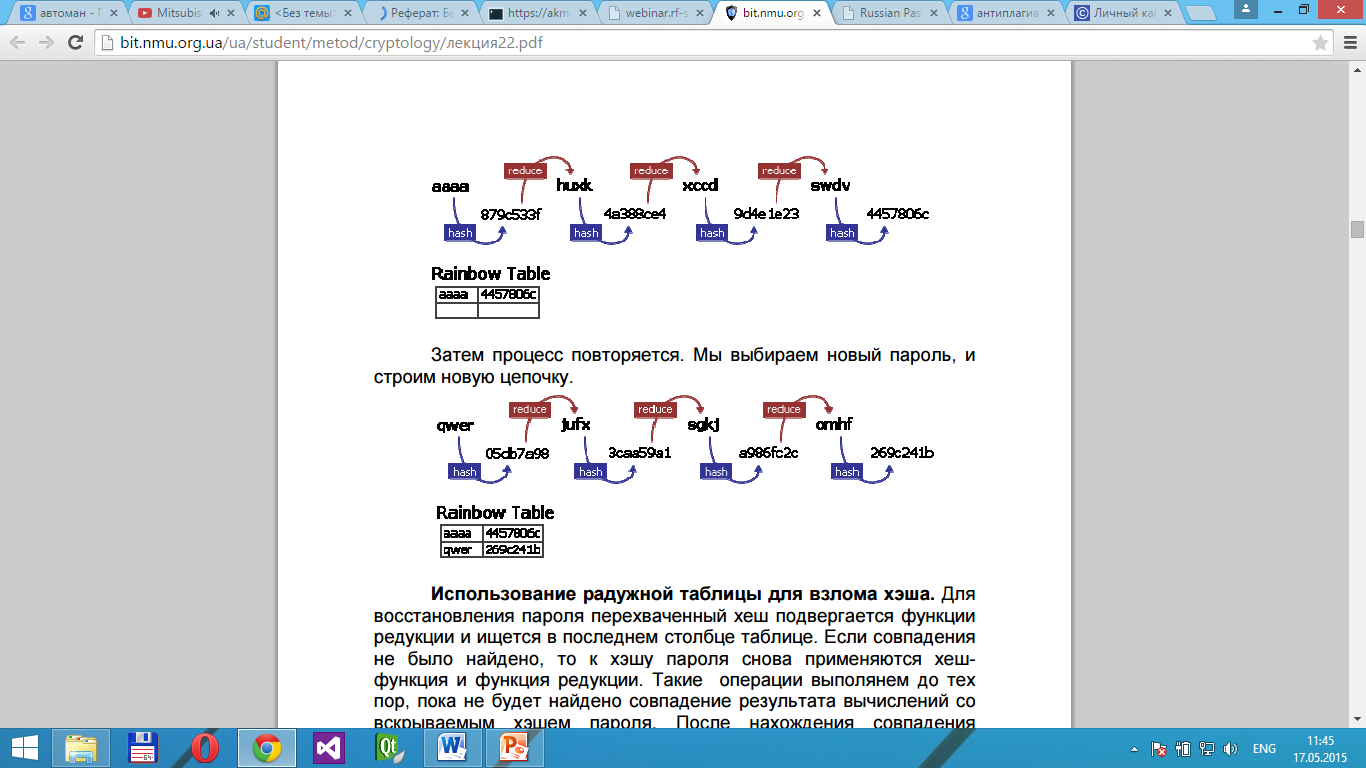


Рисунок 1 – Пример атаки с помощью радужных таблиц на простой пароль

Затем процесс повторяется. Мы выбираем новый пароль, и строим новую цепочку.

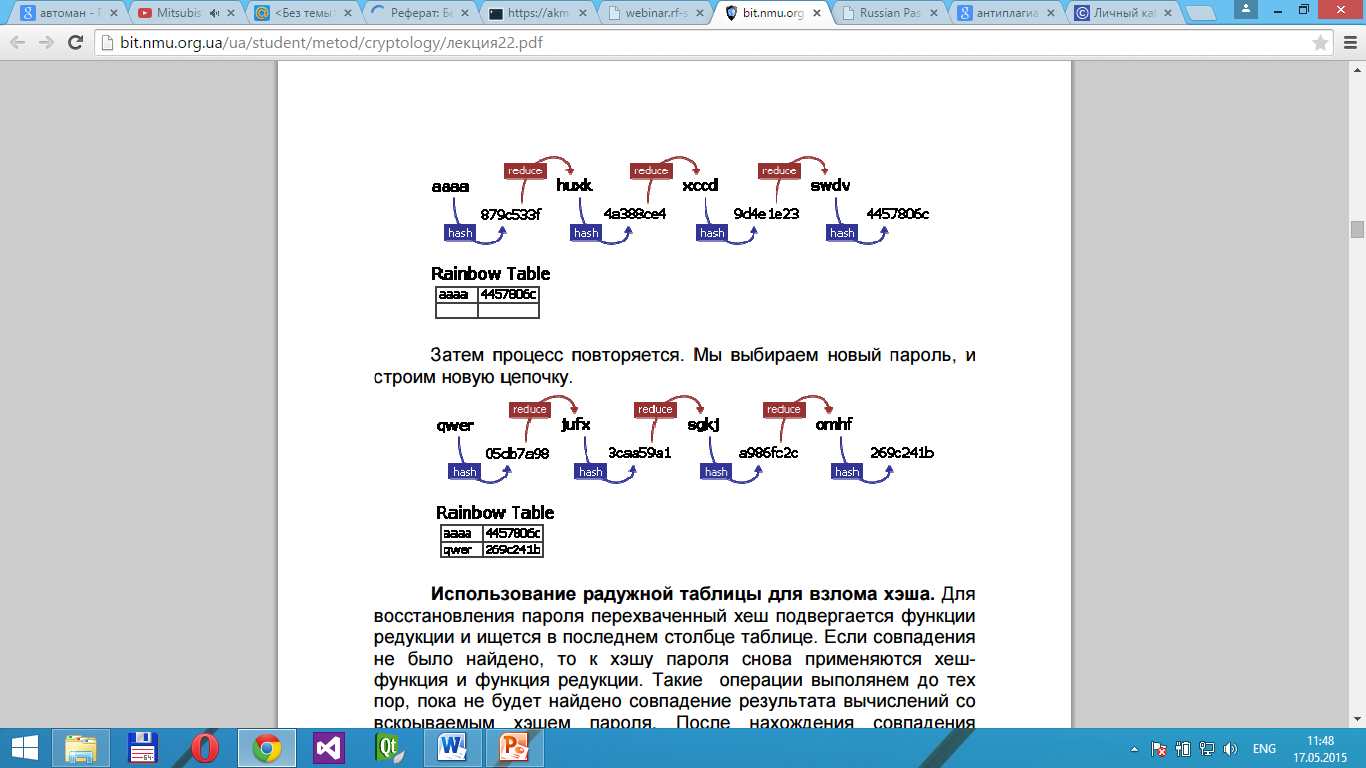


Рисунок 2 – Пример атаки с помощью радужных таблиц на пароль из разных символов

Rainbow Table

|  |  |
| --- | --- |
| *aaaa* | *4457806C* |
| *qwar* | *269C241B* |

Для усложнения подбора пароля и предотвращение атак с помощью радужных таблиц перед хешированием к паролю добавляют соль. Солью (salt) называется (псевдо) случайная битовая строка *salt*, добавляемая к аргументу password (паролю) функции хеширования ℎ для рандомизации хеширования одинаковых сообщений. Соль передается вместе с вычисленным хеш значением. Соль удлиняет пароль, а это значительно осложняет восстановление исходных паролей с помощью предварительно построенных радужных таблиц. Ведь в этом случае атакующемунеобходимо не только сгенерировать таблицы с потенциальными паролями, но сгенерировать их с разным значением соли. Благодаря этому временная сложность атак по словарю или радужным таблицам увеличивается (напомним, что, однако, соль не защищает от полного перебора каждого пароля в отдельности). Например, пусть украден хешированный пароль пользователя, который является одним из 200 000 английских слов из словаря. Система использует 32-х битную соль. Добавление соли делает бессмысленными все посчитанные хеши паролей в радужных таблицах и последние становятся бесполезными. Злоумышленник теперь должен вычислять хеш каждого пароля с каждым из 32 2 4 294 967 296 = 2 32 (4 294 967 296) возможных вариантов соли до тех пор, пока не будет получено совпадение. Общее число возможных комбинаций хеш значений и соли: 32 14 2 200 000 000 858 993 459 10 – порядка 900 триллионов (вместо 200 000).

Для каждого пароля должна использоваться уникальная соль. Кроме того введение соли снижает вероятность парольных коллизий. Поэтому соль нерекомендуется повторять. Как пример, рассмотрим организацию парольной защиты в операционной системе Windows NT. Перед тем как получить доступ к ресурсам системы, пользователь должен пройти процедуру входа в систему, при этом подсистема безопасности должна распознать его по имени и проверить подлинность запроса по паролю. Пароль пользователя хранится в базе данных в двух вариантах – в виде, необходимом для проведения аутентификации между компьютерами, работающими под управлением ОС Windows 95. Для формирования NT hash используется алгоритм MD4. Для формирования Lanmanager hash все алфавитные символы пароля приводятся к верхнему регистру, каждая из двух половин 14- байтового символьного пароля обрабатывается независимо от другой. Если длина пароля меньше 14 символов, то вторая половина забивается нулями. На основании двух 7-байтовых половин пароля формируется ключ для шифрования DES некоторого 64-битного числа. В результате получаются две половины 16-байтового хешированного пароля.

Количественная оценка пароля. Пусть A – мощность алфавита паролей (число символов, которые можно использовать для составления пароля), L−длина пароля, тогда

*S =AL* – число всех возможных паролей длины L (пространство атаки);

*v* – скорость пароля подбора злоумышленником;

*T*– максимальный срок действия пароля.

Вероятность подбора пароля равна:

*P = =*

### Двухфакторная аутентификация. Преимущества и недостатки

*Многофакторная* аутентификация имеет преимущества перед системами с*однофакторной* аутентификацией, так как системы с однофакторной аутентификацией не способны обеспечить высокую степень безопасности в современных условиях. При использовании многофакторная аутентификации для того, чтобы пользователю получить доступ к ресурсам, требуется предоставить несколько факторов, по которым можно будет определить, что это именно тот человек, за которого он себя выдает. Частным случаем многофакторной аутентификации служит – *двухфакторная* аутентификация.

Наиболее часто распространённой вариацией использования аутентификации по двум фактором является сочетание смарт-карты и пароля пользователя. Смарт-карта является первым фактором, чем «владеет» пользователь, а пароль является вторым фактором аутентификации, то, что «знает» пользователь. Такой подход имеет ряд преимуществ:

* 1. *повышается уровень* защиты системы, за счёт введения дополнительного фактора;
  2. описанные ранее *методы атаки* на парольную аутентификацию, не могут применяться напрямую;
  3. снижение рисков, связанных с использованием слабых паролей;
  4. относительная низкая стоимость на рынке аппаратных идентификаторов, по сравнению средствами биометрической идентификации.

Однако не существуют система, которая была бы лишена всех недостатков. Двухфакторная аутентификация на основе смарт-карты и пароля, так же имеет недостатки. К ним можно отнести:

1. созданиедополнительных неудобств;
2. возможны потери смарт-карт;
3. усложняется реализация и поддержка такой системы.

Двухфакторная аутентификация может быть реализована с использованием того, что «есть» у пользователя. Сюда можно отнести *биометрические параметры* человека: отпечатки пальцев, голос, сетчатка глаза. Однако использование биометрических средств аутентификации так же имеет ряд недостатков. К ним относиться:

1. изменение биометрических идентификаторов, например, после болезни или несчастного случая;
2. существенно дороже на рынке аппаратные средства определения биометрических параметров человека, например, по сравнению со смарт-картами.
3. ещё более усложняется реализации и поддержка таких систем.

Неоспоримым преимуществом является уникальность биометрических параметров человека, из-за чего повышается безопасность аутентификации.

Таким образом, для решения поставленной задачи, было выбрана двухфакторная аутентификации с помощью смарт-карты и пароля.

# Криптографические протоколы аутентификации

### Основные понятия

*Протокол* – это последовательность шагов, которые предпринимают две или большее количество сторон для решения определённой задачи. Следует обратить внимание на то, что все шаги предпринимаются в порядке строгой очередности и ни один из них не может быть сделан прежде, чем закончится предыдущий.

*Криптографическим протоколом* называется протокол, в основе которого лежит криптографические алгоритмы. Основная цель таких протоколов является сохранение информации втайне от злоумышленников.

Криптографический протокол характеризуется следующими характеристиками:

* вычислительная эффективность – количество операций, необходимых для выполнения протокола;
* коммуникационная эффективность – данное свойство отражает количество сообщений и их длину, необходимую для осуществления аутентификации;
* наличие третьей стороны – примером третьей стороны может служить доверенный сервер распределения симметричных ключей или сервер, реализующий дерево сертификатов открытых ключей;
* основа гарантий безопасности – примером могут служить протоколы, обладающие свойством доказательства с нулевым знанием;
* хранения секрета – имеется в виду способ хранения критичной ключевой информации.

Функции криптографических протоколов:

* аутентификация источника данных;
* аутентификация сторон;
* обеспечение конфиденциальности данных;
* обеспечение целостности данных;
* разграничение доступа к информации.

Классификация криптографических протоколов:

* протоколы шифрования и дешифрования;
* протоколы аутентификации и идентификации;
* протоколы обмена ключами шифрования;
* протоколы электронно-цифровой подписи.

В основе *протоколов шифрования и дешифрования* лежат симметричные и ассиметричные алгоритмы шифрования. Например: российский алгоритм симметричного шифрования *ГОСТ 28147-89*. Алгоритм на вход принимает открытый текст и ключ шифрования, на выходе алгоритм выдает зашифрованный переданным ключом информацию. Процесс дешифрования является обратным процессу шифрования. На вход алгоритма подаётся зашифрованный тест и ключ шифрования, по которому было зашифрована информация. В итоге алгоритм на выходе выдает дешифрованную информацию.

### Алгоритмы аутентификации клиента

Идея протоколов аутентификации, основанных на методе *«запрос-ответ» (challenge-response)*состоит в том, что одна сторона (претендент) доказывает свою идентичность другой стороне (проверяющему), демонстрируя ей знание секрета (в некоторых протоколах секрет известен проверяющему и используется для проверки ответа, в других - вообще нет необходимости, чтобы секрет был известен проверяющему). Претендент должен ответить на запрос, меняющийся во времени, причем ответ должен зависеть и от его секрета и от запроса. Обычно запрос – это число, выбираемое одной стороной в начале протокола. Для противодействия атакам на протокол методами повтора сеанса используют параметры, меняющиеся во времени. Это гарантирует уникальность и актуальность каждого сеанса протокола. В криптографических протоколах широко используется понятие свежести той или иной величины. Оно означает, что значение величины было сгенерировано в начале или входе выполнения текущего сеанса протокола.

В качестве параметров, обеспечивающих эту свежесть, используют три типа параметров:

* Числовые последовательности. Например, последовательность номеров сеансов протокола с увеличением на единицу в каждом последующем сеансе. Недостатком этого способа – необходимость ведения участниками счетчиков последовательностей чисел.
* *Случайные числа*. Легко генерируются всеми сторонами протокола с использованием генератора случайных/псевдослучайных чисел. Обычно случайные числа обозначают символом r с соответствующими индексами.
* *Метки времени* (timestamps). Сторона, генерирующая сообщение, получает метку времени со своих локальных часов (системного таймера) и c помощью криптографии привязывает ее к сообщению. Получатель берет текущее время со своих локальных часов и сравнивает с величиной метки времени, полученной от партнера.

Для использования этого метода необходимо, чтобы:

1. разница между метками времени отправителя и получателя укладывалась в определенный интервал времени – окно принятия;
2. ранее не поступали сообщения с идентичной меткой времени от того же отправителя;
3. часы были синхронизированы. Преимущество третьего метода – не надо поддерживать внутренний счетчик чисел. Системный таймер есть практически на любой вычислительной платформе. Основным международным стандартом по криптографическим протоколам аутентификации является стандарт Международной организации по стандартизации и Международной электротехнической комиссии ISO / IEC 9798, состоящий из пяти частей:
4. ISO / IEC 9798-1 - «General Model»;
5. ISO / EEC 9798-2 - «Mechanisms using symmetric encipherment algorithms»;
6. ISO / EEC 9798-3 - «Entity authentication using a public-key algorithm»;
7. ISO / IEC 9798-4 - «Mechanisms using a cryptographic check function»;
8. ISO / IEC 9798-5 - «Mechanisms using zero knowledge techniques».

В протоколах, на основе «запрос-ответ», основанные на крипто схемах, претендент и проверяющий имеют симметричный секретный ключ либо ключи парно-выборочной связи. Для их получения может использоваться доверенный сервер в режиме реального времени. Стандартом ISO / IEC 9798-2 предусмотрены три способа аутентификации (звездочками в таблицах обозначены необязательные компоненты сообщений):

1. Односторонняя аутентификация, основанная на метке времени.

Если у претендента и проверяющего есть системные часы, нет необходимости посылать запрос.

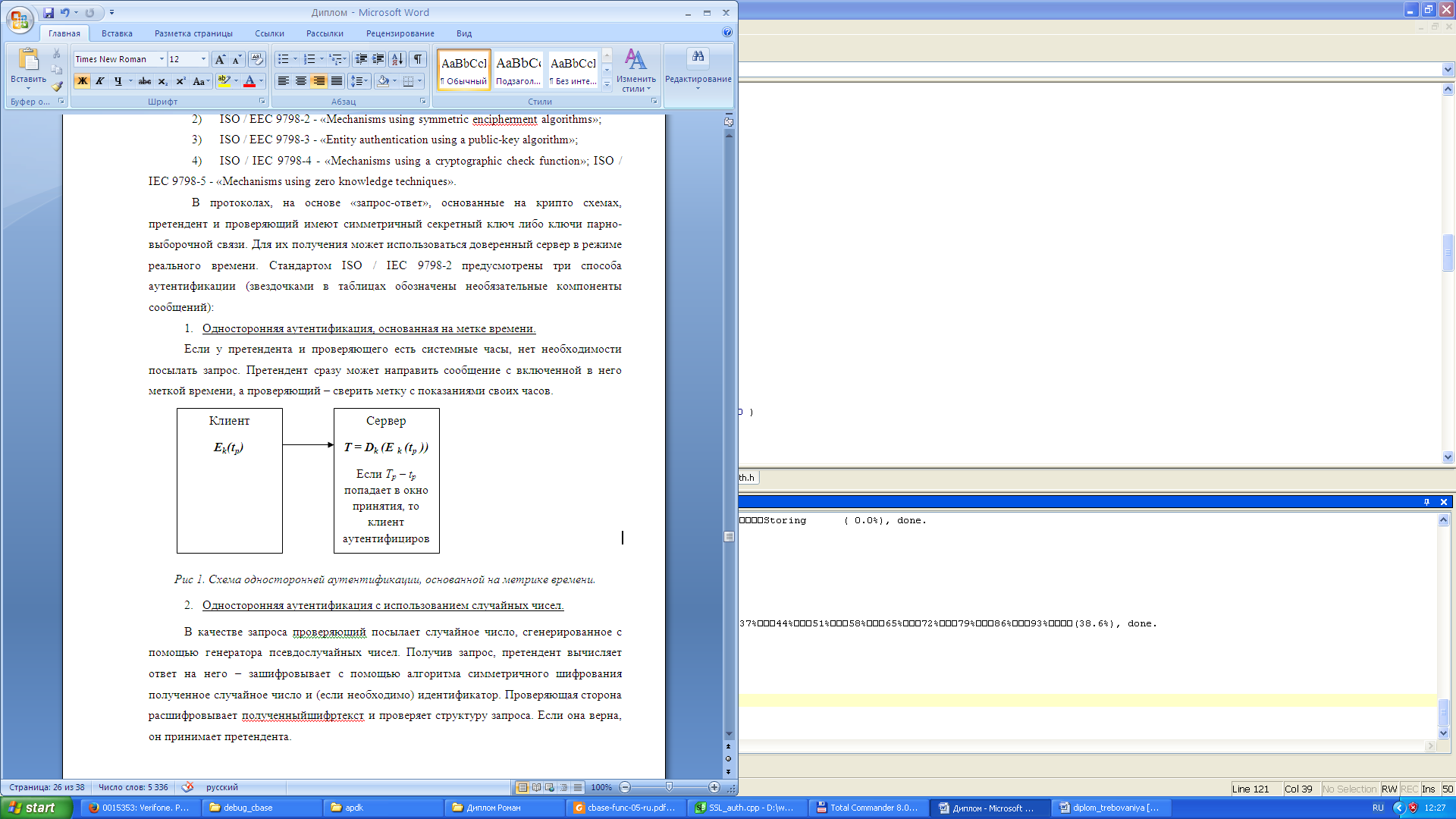
****

Рисунок 3 – Схема односторонней аутентификации, основанной на метрике времени

Претендент сразу может направить сообщение с включенной в него меткой времени, а проверяющий – сверить метку с показаниями своих часов[12].

1. Односторонняя аутентификация с использованием случайных чисел.

В качестве запроса, проверяющий посылает случайное число, сгенерированное с помощью генератора псевдослучайных чисел. Получив запрос, претендент вычисляет ответ на него – зашифровывает с помощью алгоритма симметричного шифрования полученное случайное число и (если необходимо) идентификатор. Проверяющая сторона расшифровывает полученныйшифротекст и проверяет структуру запроса. Если она верна, он принимает претендента[12].

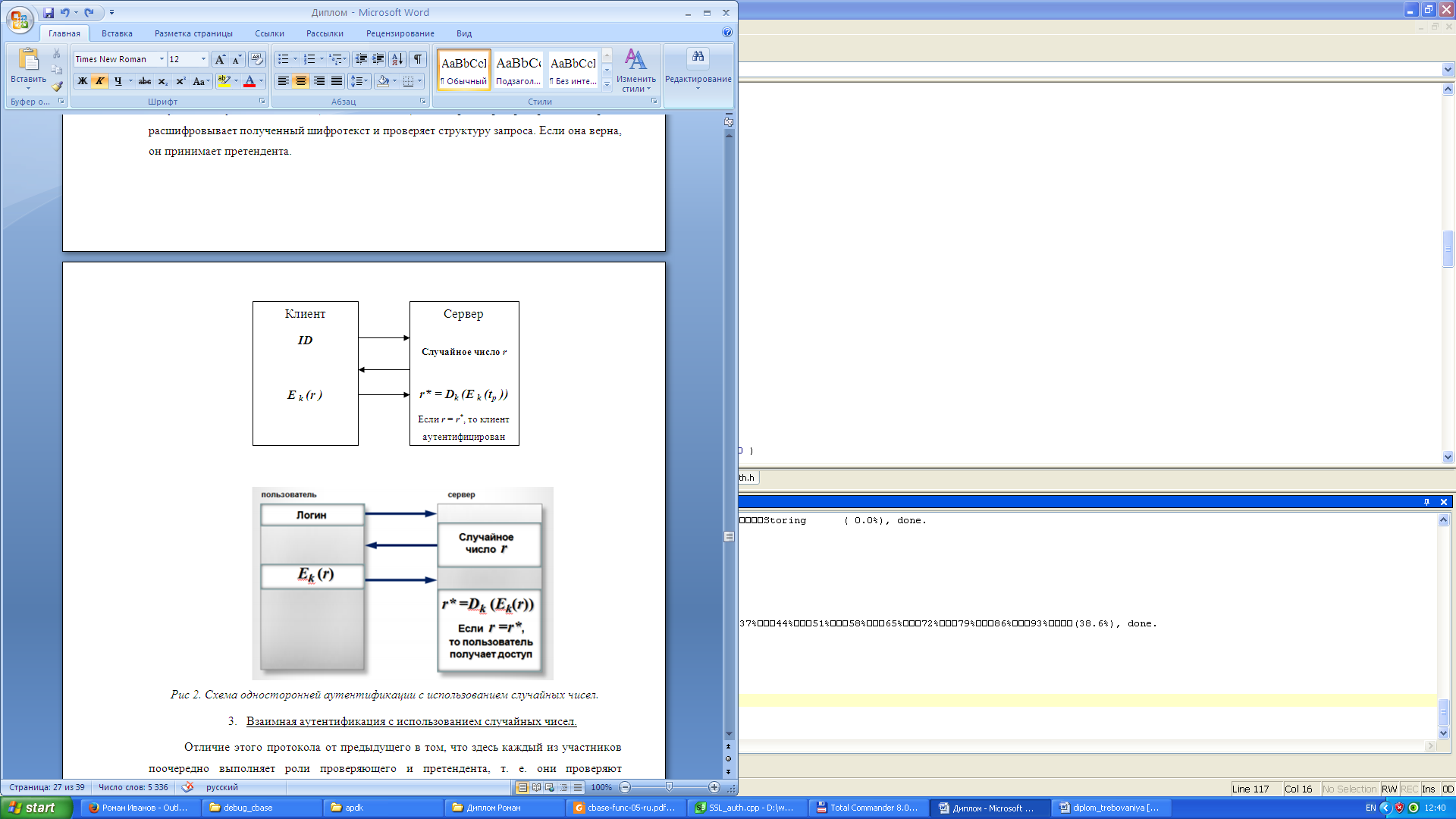


Рисунок 4 – Схема односторонней аутентификации с использованием случайных чисел.

1. Взаимная аутентификация с использованием случайных чисел.

Отличие этого протокола от предыдущего в том, что здесь каждый из участников поочередно выполняет роли проверяющего и претендента, т. е. они проверяют аутентичность друг друга. Протокол взаимной аутентификации – это по сути два протокола односторонней аутентификации, «упакованные» в три пересылки сообщения. Подобного рода протоколы в силу их симметричности называются протоколами рукопожатия. Этот протокол допускает замену шифра на хеш-функцию с ключом, как указано в стандарте ISO / IEC 9798-4. Для повышения стойкости протокола к передаваемым сообщениям можно добавить метки времени.

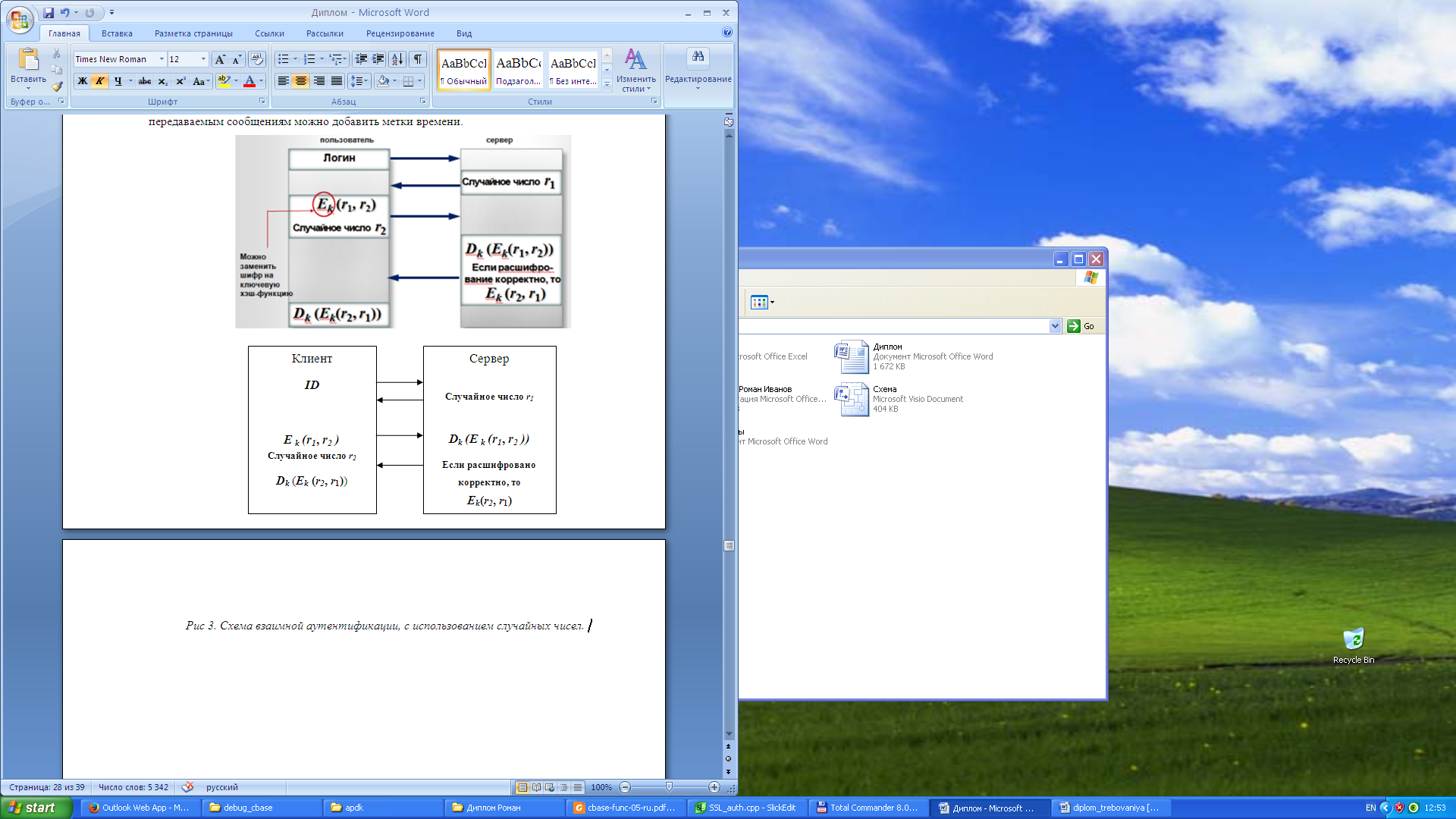
****

Рисунок 5 – Схема взаимной аутентификации, с использованием случайных чисел.

1. Аутентификации с использованиемслучайных чисел и имитовставок.

*Имитовставка* – значение, рассчитанное по входным данным с помощью криптографического алгоритма с использованием секретного элемента (ключа), известного только отправителю и получателю. Имитовставка необходима для аутентификации сообщения, получаемое от клиента.

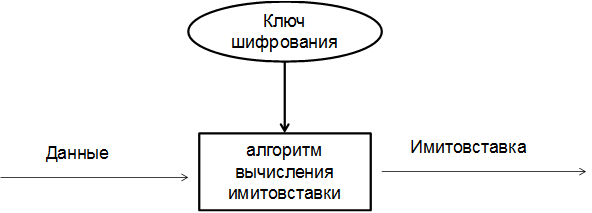
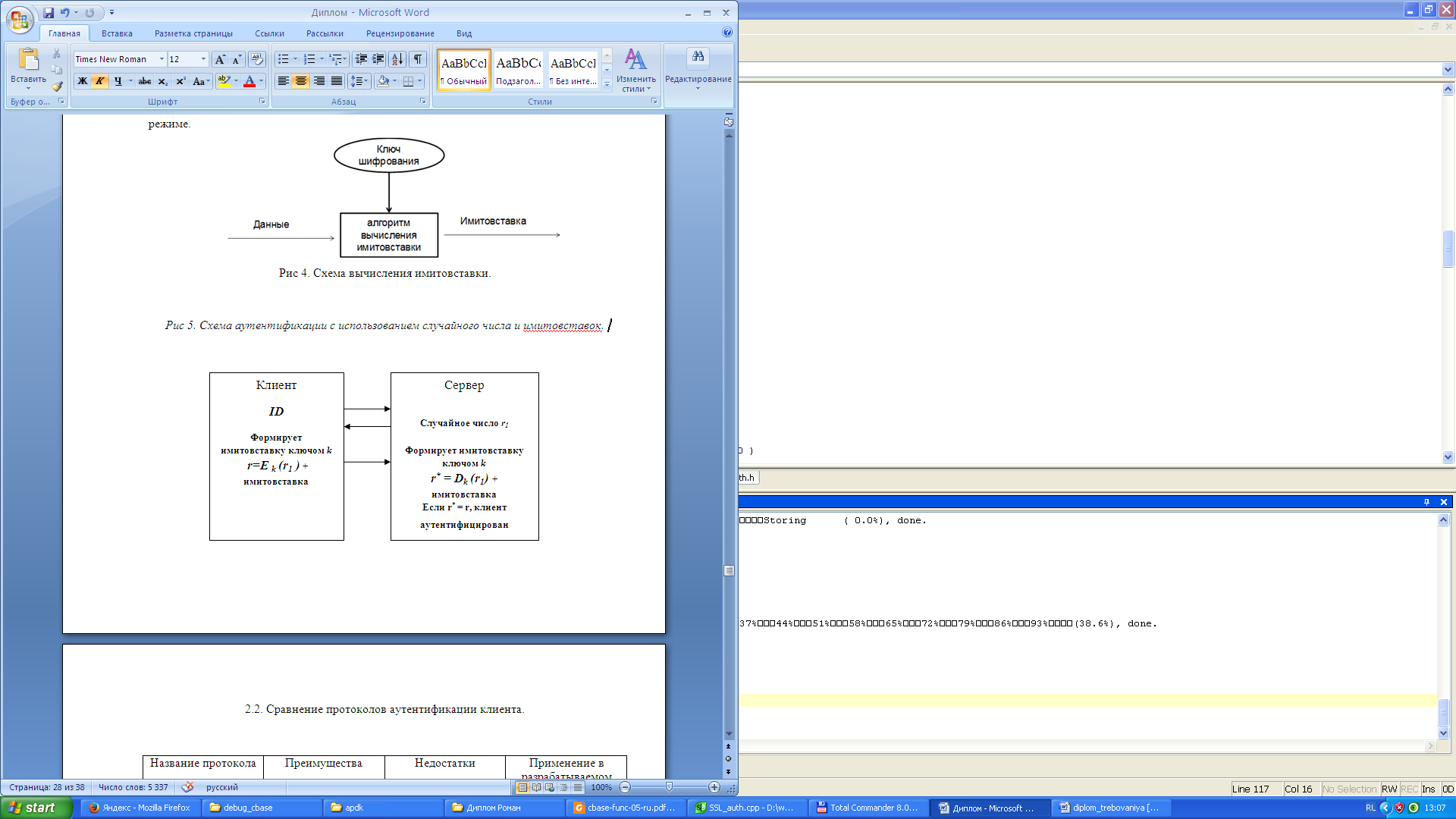


Рисунок 6 – Схема вычисления имитовставки

Клиент, после соединения с сервером, отправляет ID (идентификатор), после чего сервер генерирует случайное число и отправляет клиенту. Клиент на заранее известном ключе вычисляет имитовставку и шифрует полученное от сервера случайное число на этом же ключе, после отправляет серверу. Сервер, зная ID клиента, использует заранее известный ключ и расшифрует сообщение.

****

|  |  |  |  |
| --- | --- | --- | --- |
| Название протокола | Преимущества | Недостатки | Применение в разрабатываемом протоколе |
| Односторонняя аутентификация, основанная на метке времени | простота | 1.Использование системного времени требует дополнительную синхронизацию (в некоторых случаях),  2.Не обеспечивает случайность,  3. Не используется проверка подлинности сообщения | нет |
| Односторонняя аутентификация с использованием случайных чисел | Основан на использовании случайных числах | Не используется проверка подлинности сообщения | нет |
| Взаимная аутентификация с использованием случайных чисел | Основан на использовании случайных числах | 1) предназначен для взаимной аутентификации (не подходит в случае односторонней).  2) не использует проверка подлинности сообщения | нет |
| Аутентификация с использованием случайных чисел и имитовставок | Используется проверка подлинности сообщения | Усложняется реализация | да |

Рисунок 7 – Схема аутентификации с использованием случайного числа и имитовставок

Таблица 2 – Сравнение протоколов аутентификации клиента

После исходное случайное число и добавленное к нему имитовставка соответствуют, тому, что прислал клиент, то аутентификация прошла успешно.

Таким образом, наиболее подходящим алгоритмом для проверки подлинности клиента в данном случае является аутентификация с использованием случайных чисел и имитовставок. Алгоритм симметричного шифрования ГОСТ 28147-89 имеет режим выработки имитовставки, что необходимо для реализации выбранного протокола аутентификации клиента.

### 2.3 Протокол обмена сессионным ключом.

Для передачи секретной информации по открытым каналам связи, для защиты которой используется симметричное шифрование, абонентам необходимо иметь общий секретный ключ. Постоянно использование одного и того же ключа при каждомсеансе связи между абонентами, злоумышленнику позволяет накопить богатый для крипто анализа. Поэтому в целях повышения безопасности обмена секретной информации широко используются сессионные ключи. 

**Сессионный ключ** – это ключ, который используется абонентами в рамках одного сеанса (сессии, раунда) общения. Использование сеансовых ключей позволяет так же ограничить размер ущерба при компрометации ключа.

Возможны следующие разновидности протоколов обмена ключами

* ключ вырабатывается одним из абонентов и высылается второму для последующего обмена информацией;
* совместная выработка ключа абонентами;
* ключ вырабатывается и предоставляется абонентам третьей стороной (доверенным центром).

Наиболее оптимальным и безопасным методом обмена сессионным ключом, является совместная выработка сессионного ключа. Одним из распространённых алгоритмов обмена сессионным ключом является протокол Диффи-Хеллмана.

На первом этапе сервер генерирует клиенту параметры *p*, *g* и *a*, где *p* и *g* общие, не секретные параметры алгоритма, *a*–секретный параметр сервера. Затем вычисляет публичный ключ на основе секретного параметра *a* по формуле *A = ga mod p*. Параметры *p*, *g* и *a* передаются клиенту.

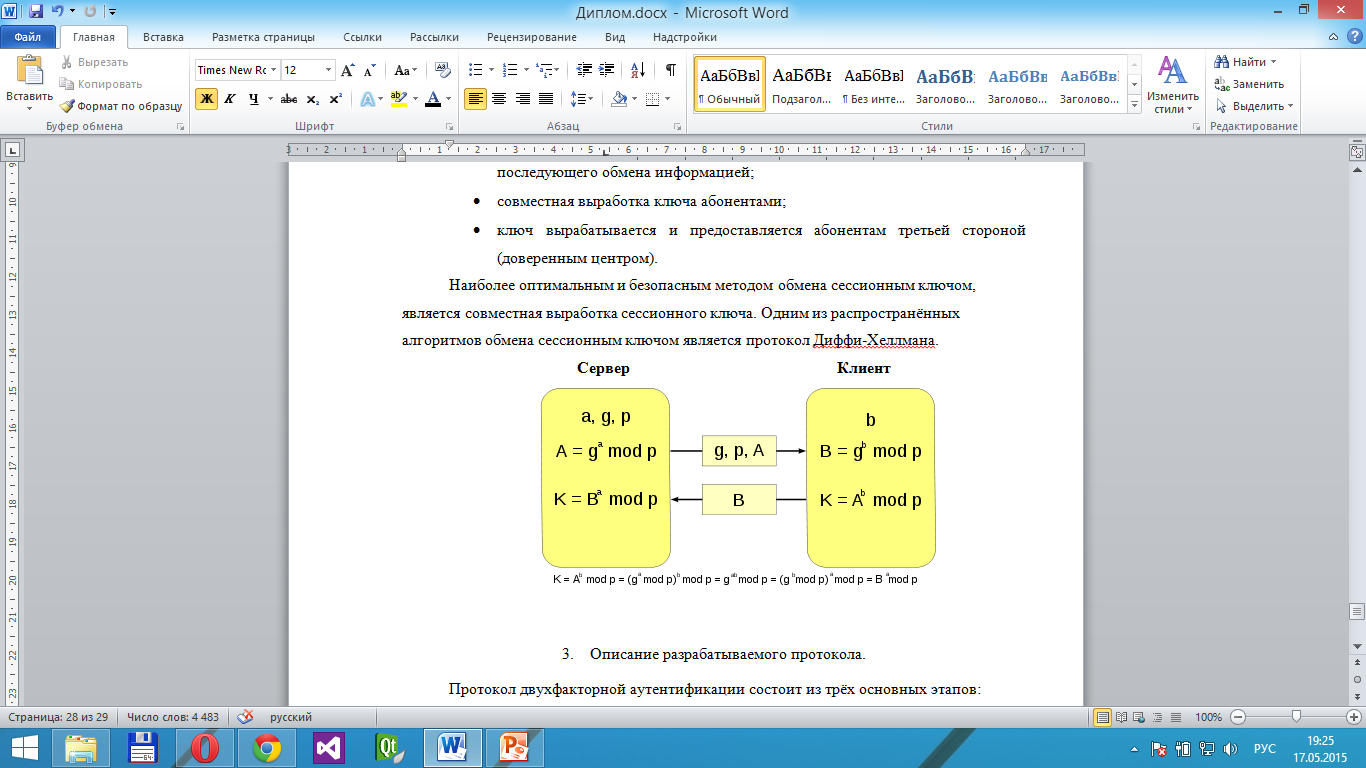


Рисунок 8 – Обмен ключами по протоколу Диффи-Хеллмана

На втором этапе клиент генерирует секретный параметр *b* и, на основе полученный от сервера параметров *p* и *g*,вычисляет открытый ключ клиента по формуле *B = gb mod p* и передает серверу *B*.

На третьем этапе сервер и клиент вычисляют, по имеющим параметрам, общий секретный сессионный ключ по формулам *K = Ba mod p* и *K = Ab mod p* соответственно.

# 3 Описание разрабатываемого криптографического протокола

### Общие сведения.

При разработке протокола решены основные задачи:

1. Выработать механизм проверки подлинности клиента, инициирующее соединение с сервером.
2. Обеспечить безопасность пользовательских данных, при передаче по каналу связи.

Для решения первой задачи, наиболее подходящим алгоритмом, являетсяаутентификация клиента на основе случайных чисел и имитовставок. Для решения второй задачи было выбрано симметричное шифрование на основе российского стандарта ГОСТ 28147-89. Применение симметричного шифрования, требует необходимость выработки сессионного ключа. Эту задачу можно решить алгоритмом Диффи-Хеллмана.

Таким образом, протокол условно делиться на три этапа:

* + 1. Проверка подлинности клиента, инициирующее соединение
    2. Обмен сессионным ключом симметричного шифрования
    3. Безопасная передача пользовательских данных, предназначенных для двухфакторной аутентификация, по открытому каналу связи.

На рисунке 7 представлен алгоритм криптографического протокола. На первом шаге устанавливается соединение клиента с сервером. Когда соединение будет установлено, начинается процесс проверки подлинности клиента. Если клиент аутентифицирован успешно, то переход к обмену сессионным ключом симметричного шифрования по протоколу Диффи-Хеллмана. После обмена сессионным ключом можно перейти к процессу аутентификации пользователей, по безопасному каналу связи.



Рисунок 9 – Алгоритм криптографического протокола

### Описание первого этапа протокола

Из рассмотренных выше протоколов проверки подлинности клиента, наиболее подходящим для данного случая, является аутентификация с использованием случайных чисел и имитовставок. Аутентификация односторонняя, так как только серверу необходимо проверить подлинность клиента. Клиент и сервер заранее имеют секретный общий ключ.Сервер отправляет клиенту случайное число *r*1. Клиентимеющимся у него ключом шифрования, шифрует *r*1 в режиме имитовставки, затем формирует сообщение и отправляет серверу.



Рисунок 10 – Алгоритм проверки подлинности клиента сервером

Сервер в БД ищет ID (идентификатор) клиента, если идентификатор клиента найден, то по соответствующему данному идентификатора ключу, шифрует случайное число *r1* в режиме имитовставки и сравнивает с данными, полученными от клиента. Если результат совпадает, с полученным сообщением, то клиент проходит проверку подлинности на сервере.

* 1. Описание второго этапа протокола

Второй этап протокола представляет с собой алгоритм обмена сессионным ключом для симметричного шифрования.



Рисунок 11 – Алгоритм обмена сессионным ключом со стороны сервера

Сервер генерирует простые числа *p* и *g*, которые являются общими и не секретными параметрами алгоритма Диффи-Хеллмана и передает их клиенту. Для обеспечения контроля целостности передаваемых данных, клиент, после получения сообщения, отправляет серверу, вычисленные хеш значения этих параметров. После успешного обмена параметрами *p* и *g*, сервер генерирует секретное число *a*, вычисляет публичный ключ сервера по формуле *A = g a mod p* и передает его клиенту. Так же, как и с первыми параметрами, в целях обеспечения целостности передаваемых данных, клиент, после получения от сервера публичного ключа, передает обратно его хеш значение. После обмена публичным ключом сервера, следует обменяться публичным ключом клиента.



Рисунок 12 – Блок-схема алгоритма обмена сессионным ключом со стороны клиента

Для этого клиент, так же, генерирует секретное число *b*, вычисляет *B* по формуле

*B=g b mod p* и отправляет результат серверу. Сервер вычисляет хеш значение от *B* и отправит клиенту. После успешного обмен параметрами Диффи-Хеллмана и публичными ключами, клиента и серверимеютвсю информацию для вычисления сессионного ключа. Оба участника протокола, далее вычисляют сессионный ключ симметричного шифрования.

Таким образом, работает второй этап данного протокола, который предназначен для обмена ключами.

* 1. Описание третьего этапа протокола.

Третий этап протокола, представляет с собой безопасный обмен пользовательскими данными, для двухфакторной аутентификации. Сюда относиться идентификатор смарт-карты пользователя и его персональный пароль.



Рисунок 13 – Алгоритм аутентификации пользователей со стороны сервера.

Клиент считывает, с помощью считывателя, идентификаторсмарт-карты пользователя, зашифрует сообщение с сессионным ключом шифрования, который был выработан, на втором этапе проткала, и отправляет серверу. Сервер расшифрует сообщение сессионным ключом и проверяет в базе данных идентификатор клиента. Ели идентификатор был найден, то формирует ответ клиенту о необходимости требования пароля от пользователя, шифрует сообщение и отправляет. Клиент, получив ответ от сервера, предлагает пользователю ввести пароль.



Рисунок 14 –Алгоритм аутентификации пользователей со стороны клиента

После ввод пароля, клиент вычисляет его хеш значение, шифрует сессионным ключом и отправляет серверу. Сервер, получив сообщение, дешифрует его, проверяет в БД на соответствие идентификатора клиента и хеш значения пароля. Если идентификатор соответствует полученному хеш значению от пароля, то сервер формирует ответ об успешной аутентификации пользователя, иначе аутентификация пользователя не было пройдено.

Таким образом, реализуемый в рамках данной задачи протокол состоит из трёх основных этапов. Выполнения всех трёх этапов позволяет решить все поставленные задачи.

4 Реализация программной части

4.1 Общие сведения. Архитектура

Согласно требованиям технического задания, протокол со стороны сервера должен быть реализован на платформе WindowsServer 2012 R2, результатом работ должна быть библиотека, предоставляющий интерфейс доступа к функциям протолкла.

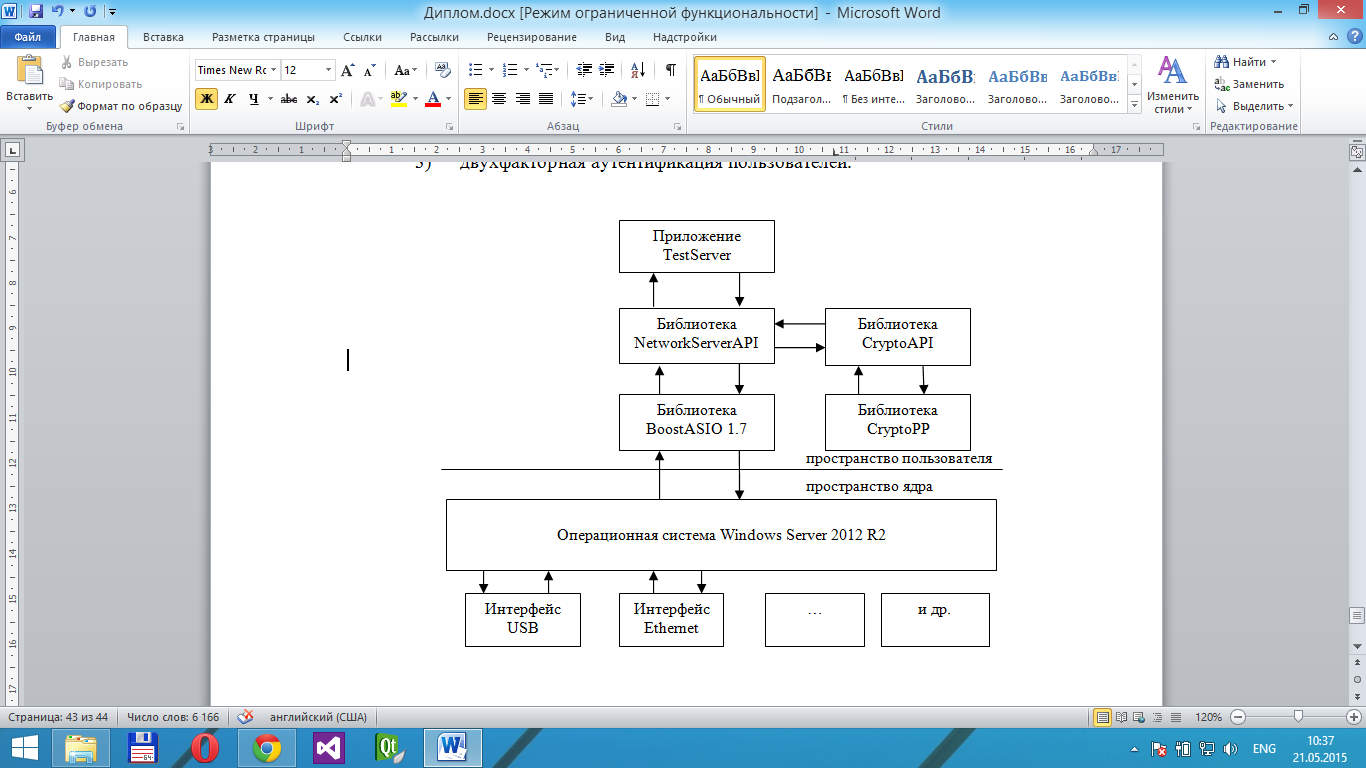


Рисунок 15 – Зависимости модулей протокола со стороны сервера

На рисунке 15 представлена зависимость модулей протокола со стороны сервера. Основной библиотекой, где реализован интерфейс доступа к вышеописанному протоколу, является библиотека «NetworkServerAPI». Эта библиотека связано с «CryptoAPI», которая реализует криптографические алгоритмы, используемые в протоколе. «CryptoAPI» в своей реализации, применяет функции из кроссплатформенной библиотеки «CryptoPP», включающая в себе арифметические и логические операции над длинными числами (в данном случае длина до 256 бит). Сетевая часть зависит от функций, кроссплатформенной библиотеки BoostASIO, которая уже обращается к элементам ядра операционной системы WindowsServer 2012 R2 через интерфейс WinAPI. Приложение «TestServer» предназначен для тестирования протокола со стороны сервера. Такая архитектура является гибким, имеет возможность расширения, благодаря разделению на независимых модулей, которые базируются на кроссплатформенных библиотеках.

Со стороны клиента модули протокола реализованы схожим образом и имеют такие же преимущества, как и у сервера.

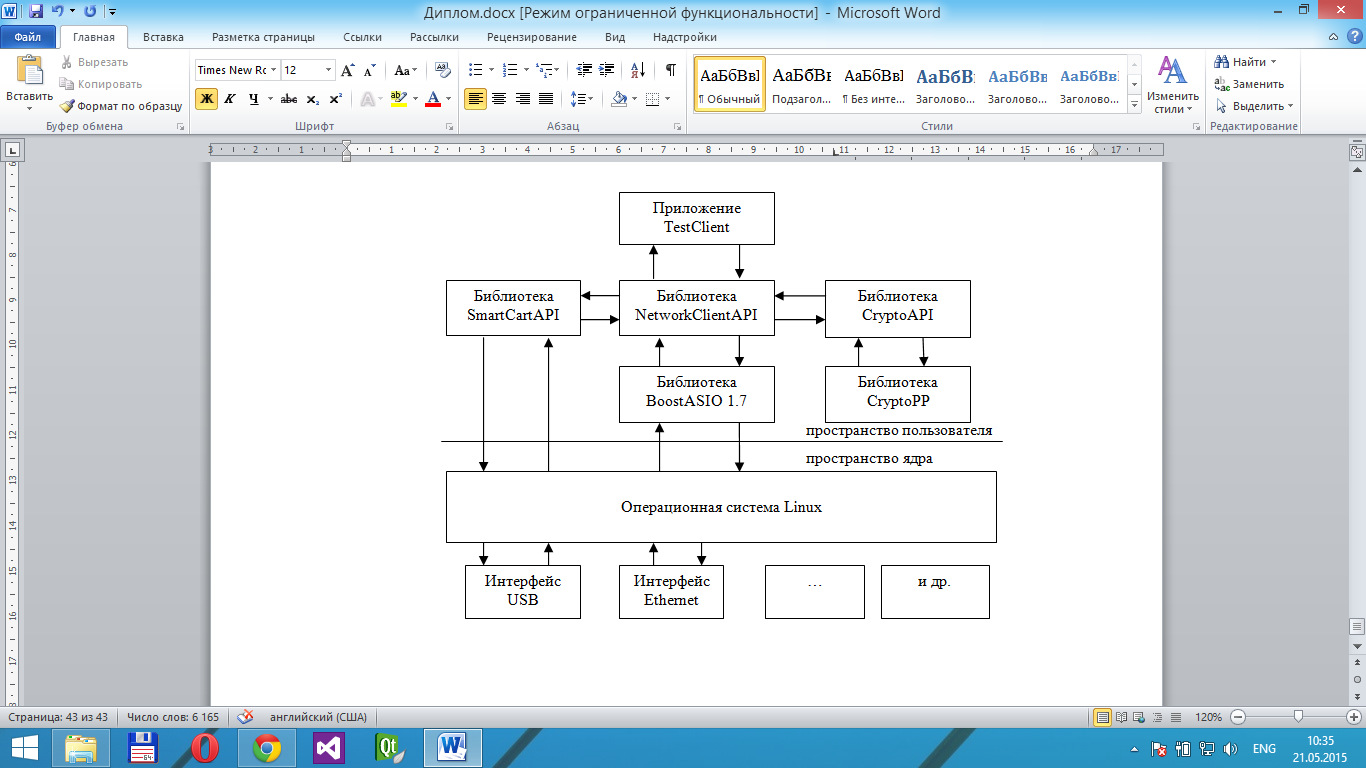


Рисунок 16 – Зависимости модулей протокола со стороны клиента

На рисунке 16, представлены зависимости модулей протокола со стороны клиента. Основной библиотекой здесь является – «NetworkClientAPI», где реализован интерфейс доступа к протоколу со стороны клиента. Криптографические алгоритмы используются из «CryptoAPI», сетевой интерфейс предоставляет «BoostASIO», протокол доступа к считывателю смарт-карты реализовано в библиотеке «SmartCartAPI».

4.2 Алгоритм симметричного шифрования ГОСТ 28147-89

Все криптографические алгоритмы реализованы в библиотеке CryptoAPIна языке С/C++. Эта библиотека предоставляет интерфейс доступа к криптографическим алгоритмам. Алгоритмом симметричного шифрования, который применяется в разработке протокола - *ГОСТ 28147-89*. Это российскийкриптографический стандартсимметричного шифрования. Алгоритм, описанный в данном стандарте, является блочным шифром, длина ключа которого составляет 256 бит, основан на сети Фейстеля. Шифрование происходит в 32 раунда, длина входного блока данных составляет 64 бит информации.Вектор инициализации алгоритма шифрования может иметь различные варианты, поэтому он должен предоставляться в отдельном файле. ГОСТ 28147-89 может работать в одном из четырех режимах[3]:

1. режим простой замены;
2. гаммирование;
3. гаммирование с обратной связью;
4. режим выработки имитовставки.

Для реализации описанного выше протокола понадобиться два режима работы:

* режим простой замены;
* режим выработки имитовставки.

В библиотеке «CryptoAPI»имеется 4 интерфейсных функций, для использования данного алгоритма симметричного шифрования (см. приложение А):

1. функция инициализации (gost\_init);
2. функция шифрования в режиме простой замены(gostcrypt);
3. функция дешифрования в режиме простой замены (gostdecrypt);
4. функция выработка имитовставки (gostimito\_crypt);

Функция *инициализация* принимает в качестве параметра путь к файлу с вектором инициализации в формате \*.bin.Загружает вектор инициализации и выполняет подготовительные этапы алгоритма.

Функция *шифрования в режиме простой замены*, в качестве параметров на вход принимают:

1. массив данных для шифрования размером 64 бит;
2. массив данных, для заполнения результатавыполнения функции (зашифрованные данные);
3. массив размером 256 бит, содержащий ключ шифрования.

Функция *дешифрования в режиме простой замены* имеет параметры:

1. массив зашифрованных данных, размером 64 бит;
2. массив данных, для заполнения результата дешифрования, размером 64 бит;
3. массив размером 256 бит, который содержит ключ, по которому зашифрованы данные.

Функция *выработки имитовставки* принимает:

1. массив данных размером 64 бит;
2. массив для результата, размером 32 бит (вычисленная 32 битная имитовставка).

Рассмотрим результат работы алгоритма.

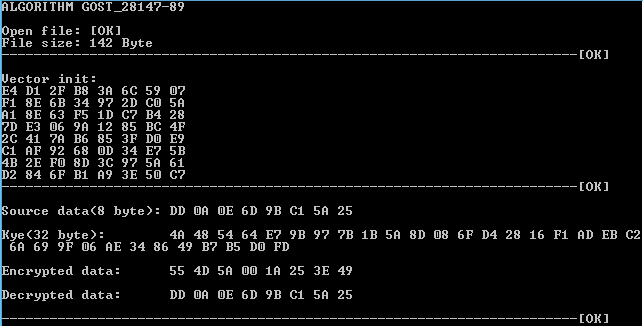


Рисунок 17 – Результат работы алгоритма ГОСТ 28147-89

На рисунке 17 показан результат работы алгоритма ГОСТ 28147-89. Вначале, открывается файл размером 142 байта, содержащий вектор инициализации, считывается содержимое файла и производится подготовительный этап.Затем, продемонстрирована работа функции шифровании в режиме простой замены. 64 бита данных шифруются 256 битным ключом. В результате зашифрованные указанным ключом данные. Для проверки правильности работы функции выполняется обратный процесс шифрованию – дешифрование. В итоге дешифрованные данные совпадают исходными данными. Отсюда можно сделать вывод о правильности работы алгоритма.

* 1. Алгоритм вычисления хеш-функции ГОСТ Р11.34-2012

ГОСТ Р 11.34-2012 – российский криптографический стандарт, который определяет алгоритм вычисления хеш-функции. Алгоритм, описанный в этом стандарте, имеет в два режима[4]:

* + 1. Режим вычисления хеш-функции, длина которой 256 бит;
    2. Режим вычисления хеш-функции, длина которой 512 бит.

Интерфейсные функции, реализующие данный криптографический стандарт, следующие:

1. функция, вычисляющая хеш-функцию, значение, которого 256 бит (hash\_256);
2. функция, вычисляющая хеш-функцию, значением 512 бит (hash\_512).

Полное описание функций (см. приложение А).

Функция hash\_256, принимает в качестве параметра массив данных, от которых необходимо вычислить хеш значение и массив размером 256 бит для получения результата.

Функция hash\_512, так же принимает в качестве параметра массив данных, от которых необходимо вычислить хеш значение и массив размером 512 бит, куда будет записан результат.

Рассмотрим результат работы алгоритма вычисления хеш-функций.

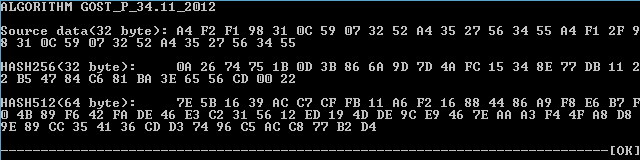


Рисунок 18 – Результат работы алгоритма вычисления хеш-функций

На рисунке 18 представлен результат работы алгоритма вычисления хеш-функций. Из исходных данных размером 32 байта, вычислено значение хеш-функции 256 бит и 512 бит.

* 1. Алгоритм обмена сессионным ключом по протоколу Диффи-Хеллмана

Протокол обмена сессионным ключом Диффи-Хеллмана состоит из нескольких этапов (см. п. 2.3), в каждом из которых производятся арифметические и логические операции с очень длинными числами, так как длина ключа шифрования ГОСТ 28147-89 составляет 256 бит. Поэтому для реализации данного алгоритма использовано дополнительная библиотека с открытым исходным кодом, написанное на языке С++ - CryptoPP.

Библиотека поддерживает платформы:

* ОС Windows;
* ОС Linux.

Интерфейс доступа к реализации протокола обмена сессионным ключом реализует класс (*diffy\_helman*), содержащий следующие открытые методы (полное описание см. приложение А):

1. метод для генерации параметра *p* (*get\_p*);
2. метод для генерации параметра *g* (*get\_g*);
3. метод для расчёта публичного ключа (*generate\_A*);
4. метод для расчёта сессионного ключа*(generate\_K*).

Метод для *генерации параметраp*, в качестве параметра получает массив, в куда будет заполнен результат, после выполнения функции.

Метод *для генерации параметраg*, так же в качестве параметра получает массив, для получения результата функции.

Метод *для расчёта публичного ключа*, в качестве параметра получает публичный ключ клиента, если функцию использует сервер или публичный ключ сервера, если использующий – клиент.

Метод для расчёта сессионного ключа, принимает два массива:

1. массив, который содержит публичный ключ;
2. массив для получения результата выполнения функции.

Таким образом, реализован интерфейс доступа к протоколу обмена сессионным ключом Диффи-Хеллмана.

Теперь рассмотрим пример, демонстрирующий работу алгоритма. На рисунке 19 продемонстрировано реализация алгоритма Диффи-Хеллмана.

Сгенерирован параметр *p* в 256 бит, параметр *g* в 256 бит. На основе этих параметров вычислен публичный ключ для сервера и публичный ключ для клиента. Затем, на основе публичных ключей клиента и сервера вычислен сессионный ключ.

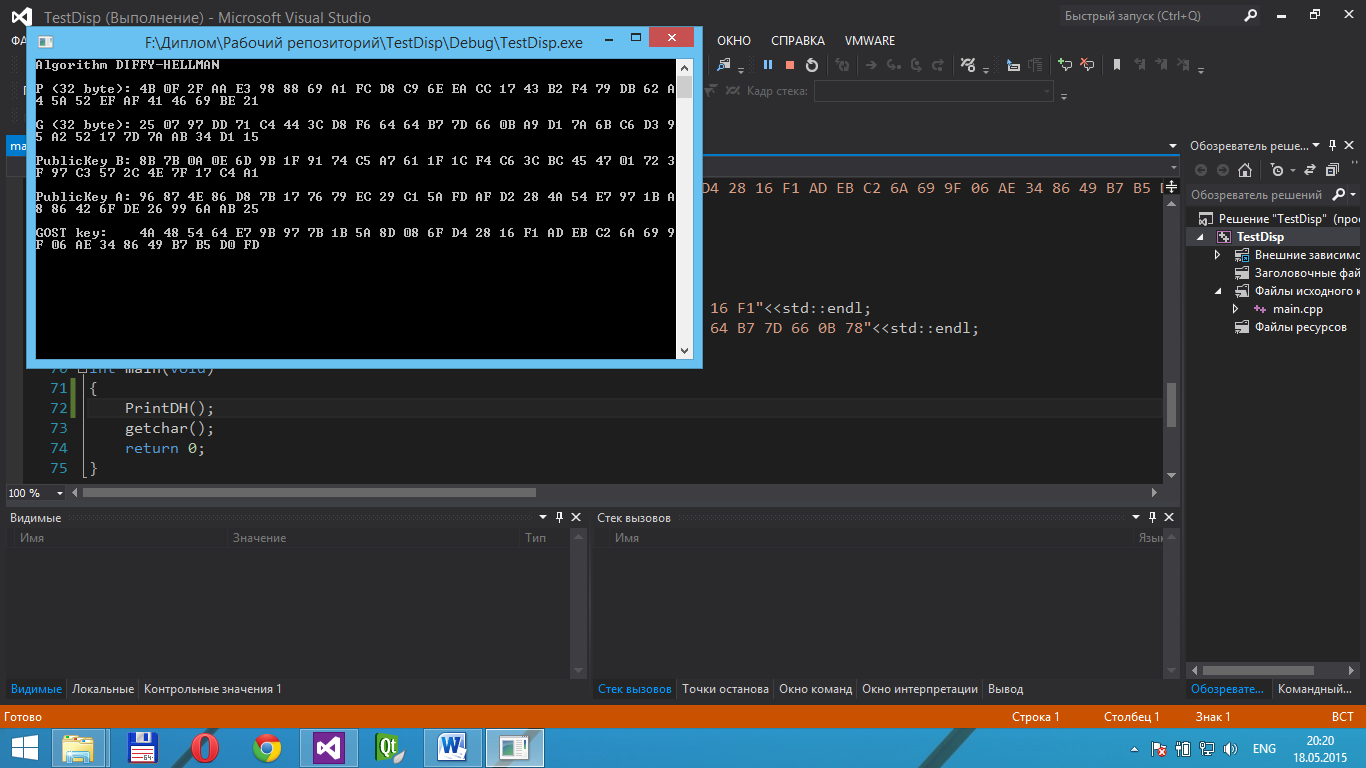


Рисунок 19 – Результат работы алгоритм обмена сессионным ключом Диффи-Хеллмана

Таким образов, рисунке19наглядно демонстрирует обмен сессионным ключом шифрования.

* 1. Формат передаваемых сообщений

При реализации различных протоколов передачи данных, важной частью является формирование пакетов данных. В разных протоколах это делается по-разному. Формат передачи сообщений формирует на самой ранней стадии разработки любого протокола, так как он общи для клиента и для сервера, они оба должны уметь разбирать сообщения и понимать их структуру.

Рассмотрим формат передаваемых сообщений в данном протоколе.

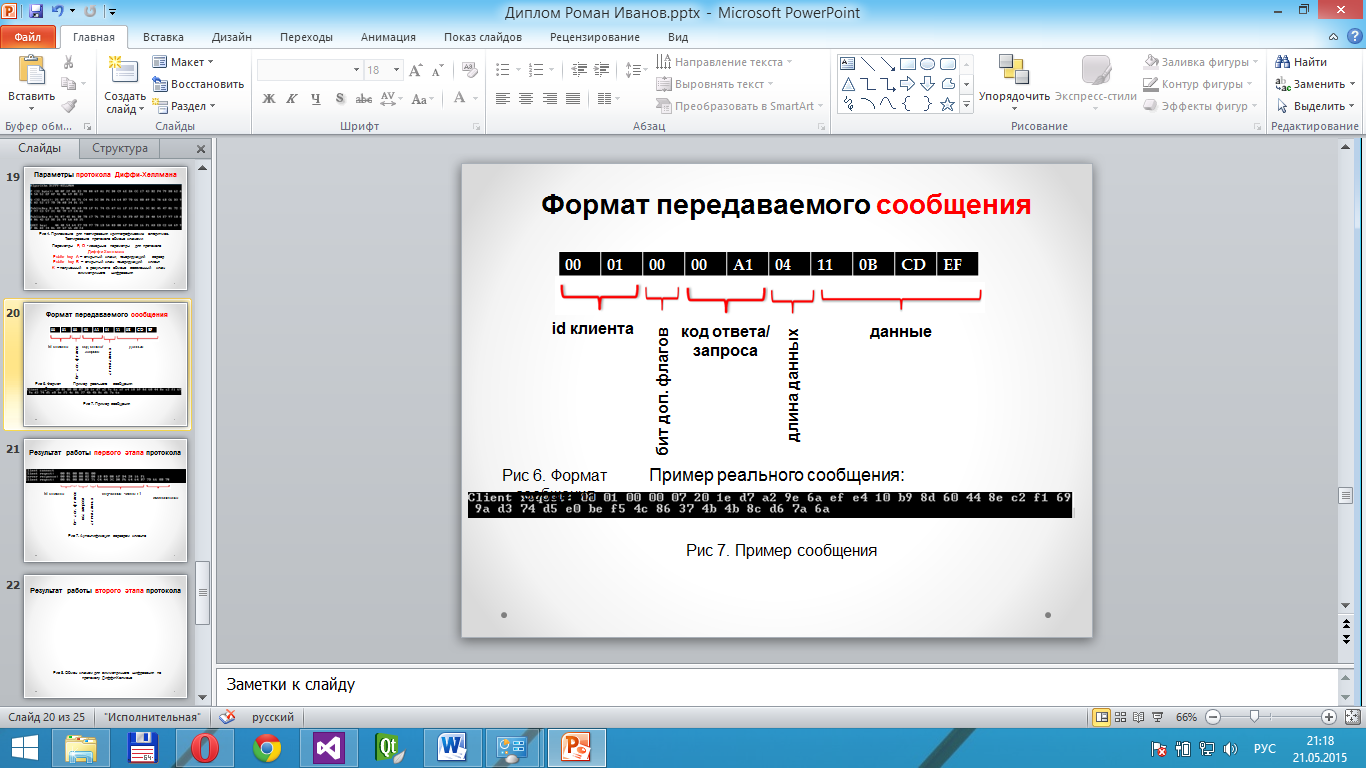


Рисунок 20 – Формат передаваемого сообщения

На рисунке 20 представлен формат передаваемого сообщения. Первые два байта данных отведены для ID (идентификатора) клиента. На примере клиент с *ID = 0x01*. Следующий байт содержит *специальные флаги* и режимы. В данном случае все флаги не активны. Далее 2 байта отводятся под код ответа или запроса. Исходя из этого кода, клиент или сервер узнают, что содержит сообщение. Следующий байт показывает длину передаваемых данных. В этом примере 4 байта. После размера данных идут сами данные.



Рисунок 21 – Пример реального сообщения

На рисунке 21, представлен пример передачи реального сообщения. Из примера следует, что ID = 0x01, флаги отсутствуют, код сообщения 0x07, длина данных 20 байт.

Полное описание кодов ответа и запроса приведено в приложении Б.

* 1. Реализация первого этапа протокола

Как уже было отмечено, криптографический протокол аутентификации пользователей состоит из 3 основных этапов:

1. Проверка подлинности клиента
2. Обмен сессионным ключом шифрования
3. Безопасная передача пользовательских данных, предназначенных для двухфакторной аутентификация, по открытому каналу связи.

Первый этап проткала, реализует аутентификацию клиента. В основе протокола проверки подлинности клиента лежит аутентификация с использованием случайных чисел и имитовставок.

Алгоритм симметричного шифрования ГОСТ 28147-89, как уже отмечалось, имеет режим имитовставки, которое применяется для проверки подлинности сообщения.

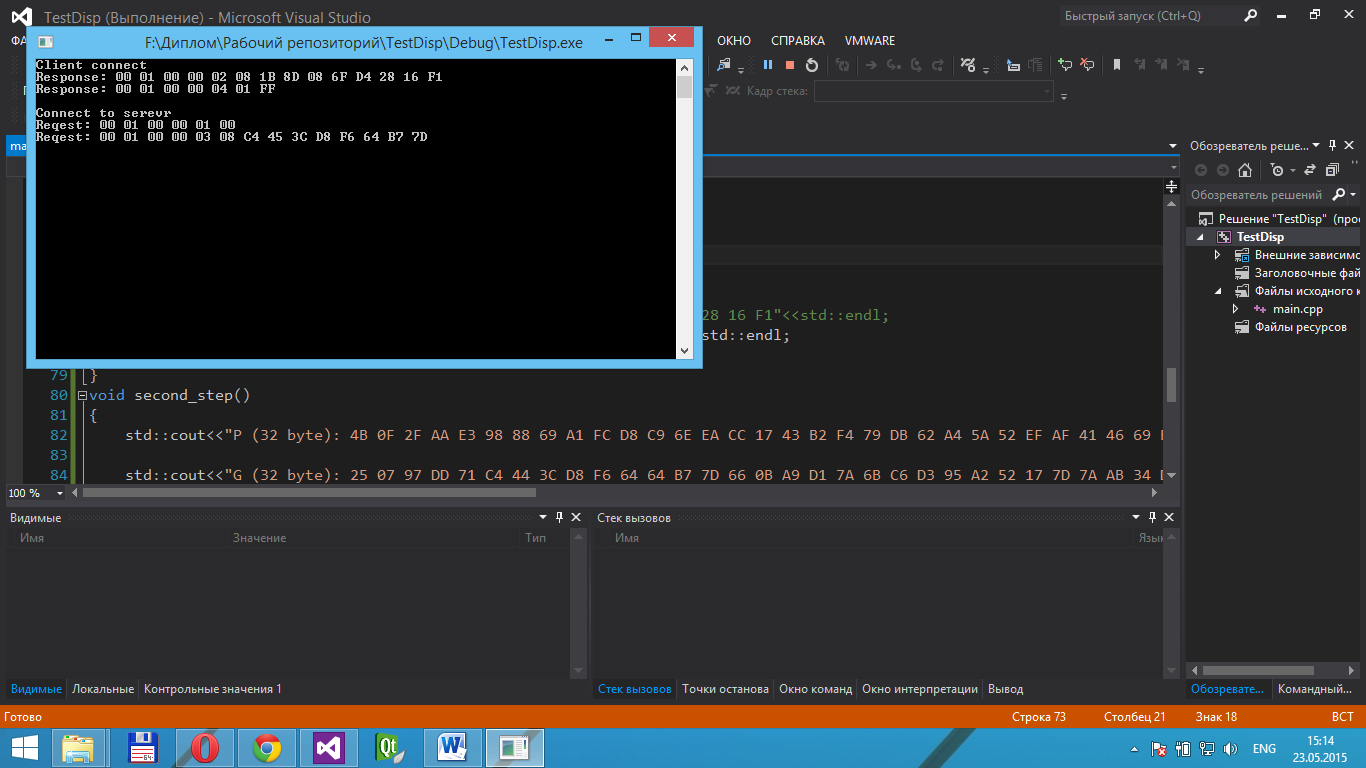


Рисунок 22 – Первый этап протокола со стороны сервера

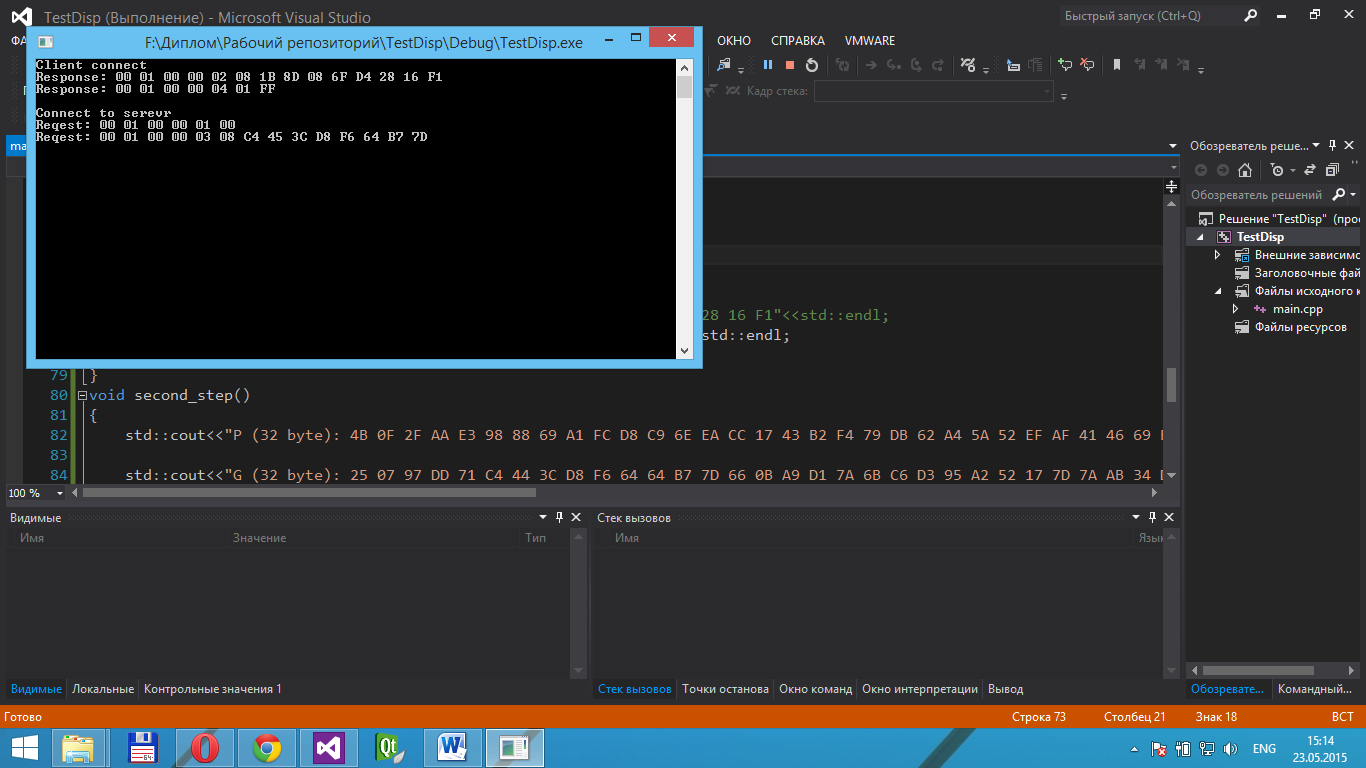


Рисунок 23 – Первый этап протокола со стороны клиента

На рисунке 22 *и* рисунке 23 продемонстрирован первый этап протокола. Клиент, *ID* которого **0x01**, соединяется с сервером и отправляет запрос:«**00 01 00 00 01 00**». Сервер сгенерировал случайное число:«**1B 8D 08 6FD4 28 16 F1**» и ответит клиенту. Клиент, на заранее известном ключе, шифрует алгоритмом шифрования ГОСТ 28147-89 в режиме *имитовставки* и отправляет результат серверу. Сервер, по тому же ключу шифрует переданное клиенту случайное число и сравнивает с поученным ответом от клиента. Если результат совпадает, то клиент проходит проверку подлинности, и сервер отправляет соответствующий ответ клиенту.

* 1. Реализация второго этапа протокола

Второй этап протокола предназначено для обмена сессионным ключом шифрования. Обмен ключом производиться с помощью алгоритма Диффи-Хеллмана.

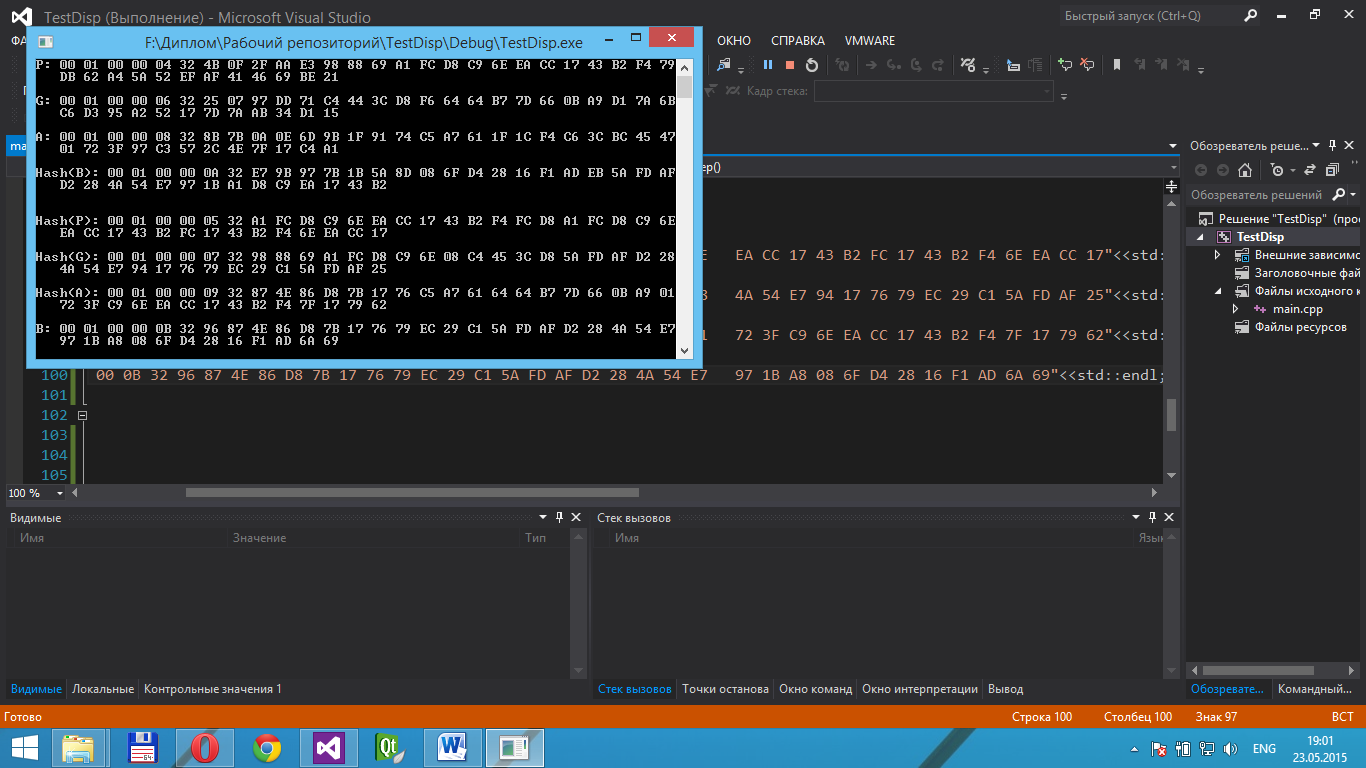


Рисунок 24 – Второй этап протокола со стороны сервера

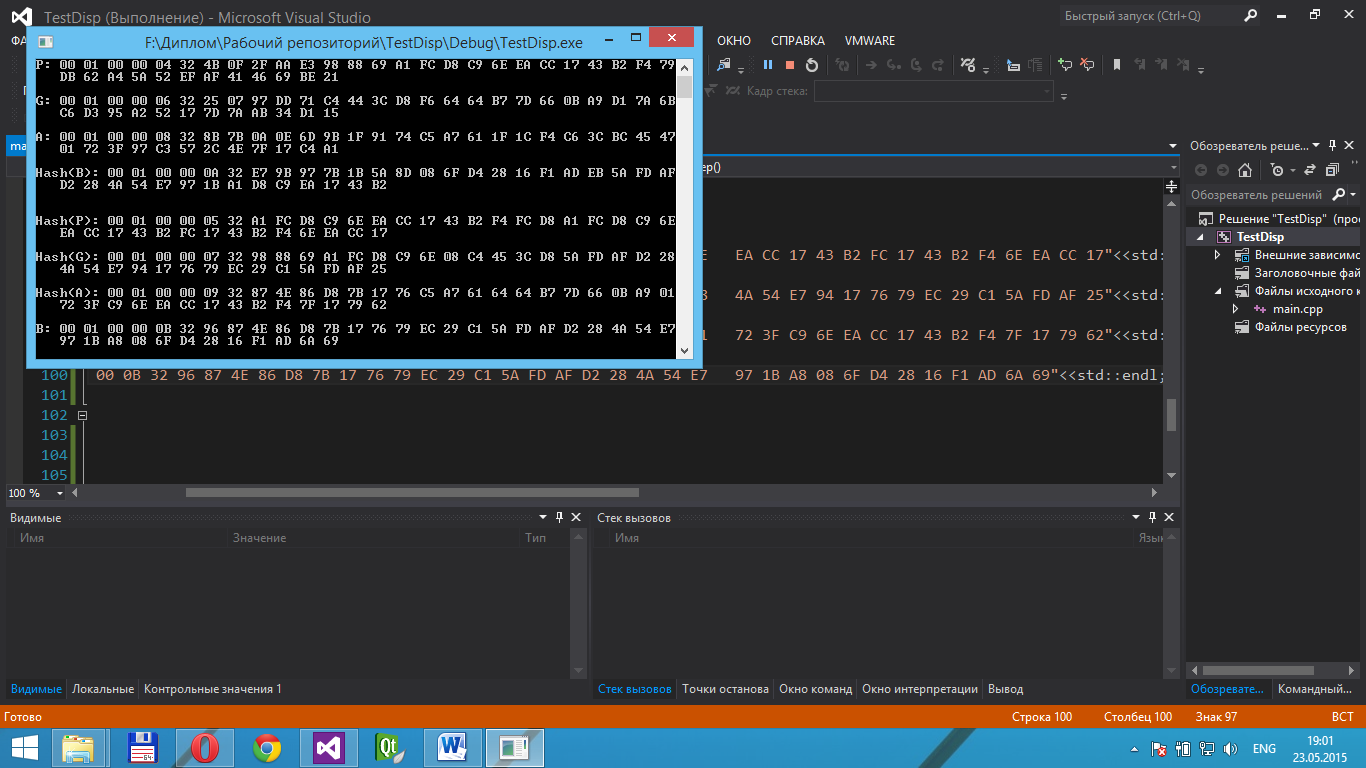
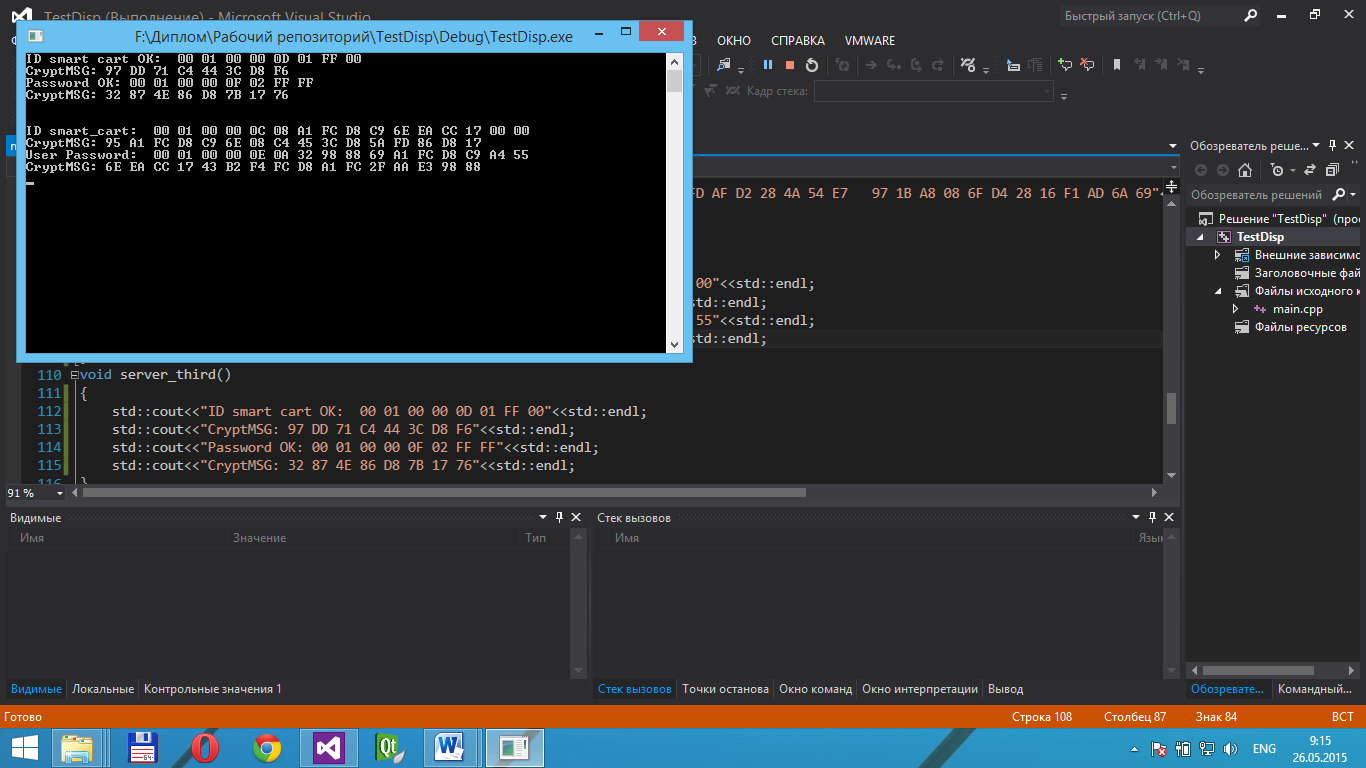


Рисунок 25 – Второй этап протокола со стороны клиента

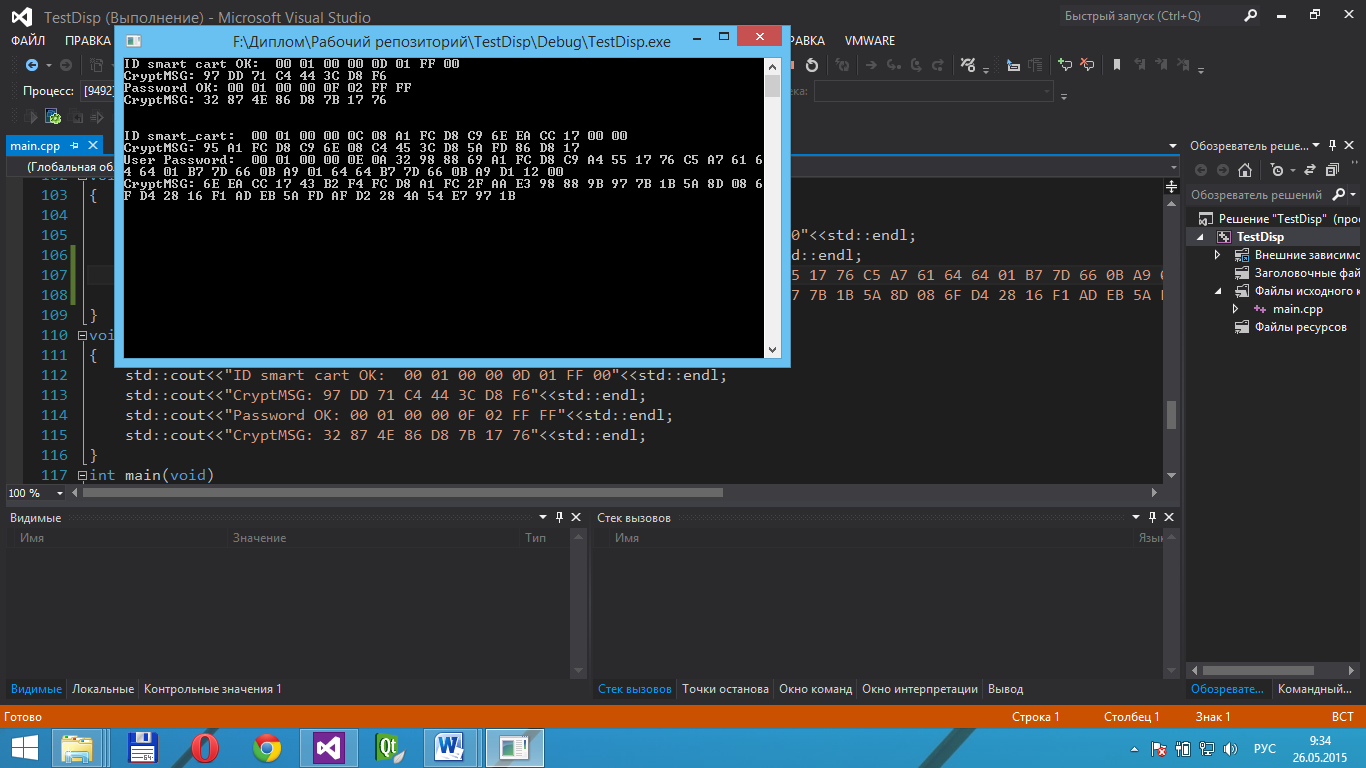
Рассмотрим второй этап протокола на примере, которой продемонстрирован на рисунке 24 и 25. Сервер генерирует параметр *p*, формирует сообщение с кодом ответа 0x04 и отправляет клиенту. Клиент, получив сообщение, хеш значение от данных, формирует сообщение с кодом ответа 0x05, отправляет серверу. Сервер сравнивает хеш значение полученных данных с хеш значением переданных данных. Если результат совпадает, то генерирует второй параметр *g*и таким же образом отправляется клиенту. Далее продемонстрирован обмен публичными ключами клиента и сервера. После обмена всеми параметрами алгоритма Диффи-Хеллмана, вычисляется сессионный ключ и переход к третьему этапу.

* 1. Реализация третьего этапа протокола

Третий этап протокола предназначен для обеспечения защиты пользовательских данных, передаваемых по открытому каналу связи, с помощью симметричного шифрования. Данные пользователя предназначены для аутентификации на сервере. Рассмотрим пример, который демонстрирует процесс передачи данных на третьем этапе протокола.



*Рисунок 23 – Третий этап протокола со стороны сервера*



*Рисунок 24 – Третий этап протокола со стороны клиента*

На *рисунках 23* и *24* продемонстрирован пример третьего этапа протокола со стороны сервера и клиента. Клиент посылает сообщение, содержащее идентификатор смарт-карты в шифрованном виде, сервер проверяет существующий идентификатор в базе данных и отправляет запрос ввода пароля пользователя, в случае нахождения в базе данных идентификатора клиента. Клиент, получив пароль от пользователя, отправляет в его хеш в зашифрованном виде серверу, и сервер определяет, аутентифицирован пользователь или нет.

# 5 Тестирование. Статический анализ исходного кода.

Качество программного обеспечения можно определить как совокупную характеристику исследуемого ПО с учётом следующих составляющих, описанных в ISO 9126:

* надёжность
* сопровождаемость (улучшение, поддержка и расширяемость)
* практичность
* эффективность
* функциональность

К наиболее эффективным методам тестирования можно отнести комбинацию стратегий черного ящика и белого ящика. Стратегия «черного ящика» предполагает изучение и тестирование внешнего поведения функционирующей программы; стратегия «белого ящика» предполагает изучение и тестирование внутренней архитектуры и логики программы на основе ее исходных кодов.

При использовании метода чёрного ящика (black-box testing) тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Например, тестирующий модуль может виртуально нажимать клавиши или кнопки мыши в тестируемой программе с помощью механизма взаимодействия процессов. Эти события вызывают тот же отклик, что и реальные нажатия клавиш и кнопок мыши. Как правило, тестирование чёрного ящика ведётся с использованием спецификаций или иных документов, описывающих требования к системе.

Программа тестируется так, как с ней будет работать конечный пользователь, и он ничего не знает о внутренних механизмах и алгоритмах, по которым работает программа. Другими словами, он запускает приложение на выполнение и тестирует его функциональность, используя пользовательский интерфейс для ввода входных и получения выходных данных. Но как при этом обрабатываются входные данные, он не знает. Цель данного метода – проверить работу всех функций приложения на соответствие функциональным требованиям.

Для достижения данных результатов, необходимо проводить комплексное тестирование, статический и динамический анализ исходного кода. Для тестирования программного кода без его непосредственного запуска применяется метод белого ящика. При тестировании с использованием метода белого ящика тестировщик имеет доступ к исходному коду ПС и может писать код, который связан с библиотеками тестируемого ПС. Это типично для компонентного тестирования (юнит-тестирования, unit-testing), при котором тестируются только отдельные части системы. Такие тесты основаны на знании кода приложения и его внутренних механизмов.

Метод белого ящика часто используется на стадии, когда приложение ещё не собрано воедино, но необходимо проверить каждый из его компонентов, модулей, процедур и подпрограмм. Также отметим, что компонентным тестированием (unit testing), чаще всего занимается программист, хорошо понимающий код.

В ходе работы было применено тестирование методом «белого ящика» и «черного ящика». Каждая библиотека отдельно было тестировано. Разработано специальное приложение, предназначенное для тестирования. Достигнуто большое количество покрытия кода, путем воздействия множеством входных данных. Результат удовлетворяет требованиям. При использовании метода «черного ящика», применены внешние воздействия такие, как:

* обрыв лини связи
* передача клиентом некорректных данных
* передача, используя неправильный формат данных
* передача, искаженных данных

Полученный результат соответствует ожидаемому.

Для выявления небезопасного кода и согласно требованиям руководящего документа ФСТЭК, проведен статический анализ кода с помощью статического анализатор PSV Studio. Все выявленные предупреждения были исправлены.

# ЗАКЛЮЧЕНИЕ

Таким образом, исследуя выбранную тему, я пришел к выводу о том, что защита от несанкционированного доступа в систему, является одним из актуальных и приоритетных задачи в обеспечении безопасности информационных и инфокоммуникационных систем. Одним из возможных уязвимостей, в комплексной безопасности, является система аутентификации.

Неспроста, для достижения основной цели, было выбрано именно двухфакторная аутентификация. Сложная многофакторная аутентификация позволяет предотвратить намерения злоумышленника получить секретные и конфиденциальные данные. После анализа существующих методов, наиболее оптимальным способом оказалась именно двухфакторная аутентификация. Такой способ обладает значительными достоинствами:

1. введения дополнительного, второго фактора аутентификации, повышает уровень защиты системы;
2. методы атаки на парольную аутентификацию, не могут применяться напрямую;
3. относительная низкая стоимость на рынке аппаратных идентификаторов, по сравнению средствами биометрической идентификации.

Однако, для получения выше названных преимуществ двухфакторной аутентификации, необходимо решить следующие задачи:

1. проверить подлинность клиента;
2. обеспечить безопасность, передаваемых по открытому каналу связи пользовательских данных для аутентификации.

Для решения этих задач, был разработан криптографический протокол на основе российских стандартов. Протокол использует симметричное шифрование ГОСТ 28147-89, алгоритм вычисления хеш-функций ГОСТ Р 34.11-2012 и обмен сессионным ключом по алгоритму Диффи-Хеллмана. Удовлетворяет всем необходимым требования:

1. эффективная вычислительная сложность;
2. коммуникационная эффективность;
3. наличие третей стоны;
4. безопасное хранение секретов (ключей).

Так же требования к платформе, которые предъявлены в техническом задании, полностью соответствуют.

В ходе работы получены следующие результаты:

1. реализована библиотека, включающая в себе российские криптографические алгоритмы, предназначенные для данного протокола;
2. разработана библиотека, реализующая криптографический протокол;
3. разработана библиотека, предоставляющий интерфейс доступа к считывателю смарт-карты клиента, через USB интерфейс;
4. разработано приложение для тестирования библиотек;
5. представлено описание интерфейсных функций библиотек.

Выполнены требования и по безопасности написания исходного кода ПО. Проверены все исходные коды на наличие уязвимостей, с помощью статического анализатора кода (PSV Studio).

Таким образом, все поставленные цели и задачи выполнены успешно. Результат работы соответствует всем требования, предъявленным в техническом задании.

# Список использованных источников

1. Батура В. П., Панасенко С. П., «Основы криптографии для экономистов», М.: Издательство «Финансы и статистика», 2005 – 176 с.
2. Брайан В., Ритчи Деннис М. Язык программирования Си. Издание 3-е, М.: 2009г,
3. ГОСТ 28147-89. «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования», 1990 – 28 с.
4. ГОСТ Р 34.11-2012. «Информационная технология. Криптографическая защита информации. Функция хеширования». 2013 г – 24 с.
5. М. Вельшенбах Криптография на Си и С++ в действии. Учебное пособие. – М.: Издательство Триумф, 2004 — 464 с:
6. Панасенко С.П., [«Алгоритмы шифрования, специальный справочник», Санкт-Петербург, Издательство «БХВ-Петербург», 2009 - 576 с.](http://www.panasenko.ru/Book2/page_book2.html" \o "ссылка на сайт Сергея Панасенко)
7. Шнайер Б., «Прикладная криптография», - М.: Издательство ТРИУМФ, 2002г.
8. ФСТЭК, «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия не декларированных возможностей», 1999г.
9. ФСТЭК, «Средства вычислительной техники. Межсетевые экраны. Защита от несанкционированного доступа. Показатели защищенности от несанкционированного доступа к информации», 1992 г.
10. ФСТЭК, «Автоматизированные системы. Защита от несанкционированного доступа к информации. Классификация автоматизированных систем и требования по защите информации», 1992 г.
11. ISO/IEC 9798-1:2010. «Информационные технологии. Методы защиты. Аутентификация объектов. Часть 1. Общие положения», 2010г.
12. ISO/IEC 9798-2:2008. «Информационные технологии. Методы защиты. Аутентификация объектов. Часть 2. Механизмы с применением алгоритмов симметричного шифрования», 2008г.
13. ISO/IEC 9797-1:2011, «Информационные технологии. Методы защиты. Коды аутентификации сообщений (MAC). Часть 1. Механизмы, использующие блочный шифр», 2011г.
14. ISO/IEC 9798-4:1999. «Информационные технологии. Методы защиты. Аутентификация объектов. Часть 4. Механизмы с применением криптографической функции проверки».

# ПРИЛОЖЕНИЕ A Описание заголовочных файлов (интерфейсные функции и классы)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Файл: crypt\_gost\_28147-89\_API.h

\* Описание: содержит объявление функций

\* криптографического алгоритма ГОСТ 28147-89

\* Автор: Иванов Роман Витальевич (c)

\* Год: 2014 - 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef CRYPT\_GOST\_28147\_89\_API\_H\_

#define CRYPT\_GOST\_28147\_89\_API\_H\_

#define CRYPT\_GOST\_API \_\_declspec(dllexport)

#define CRYPT\_GOST\_28147\_89\_API\_VERSION 0x01

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* crypt\_gost\_28147-89 version (версия dll)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if \_DEBUG

#define DEBUG\_INFO\_PRINT 1

#else

#define DEBUG\_INFO\_PRINT 0

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* вывод на консоль отладочной информации

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "crypt\_gost\_types.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* блок подстановки K состоит из 8 узлов замены k1, k2, k3, k4, k5,

\* k6, k7, k8 по 64 бит каждый \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static byte k87[key\_size256];

static byte k65[key\_size256];

static byte k43[key\_size256];

static byte k21[key\_size256];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* API functions (интерфейсные функции)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция key\_box\_init() инициализирует ключ шифрования

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CRYPT\_GOST\_API void key\_box\_init(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция gostcrypt() шифрует 64 бита информации

\* в режиме простой замены

\* Параметры функции:

\* 1. {word32 const \*in\_data} массив из 2 элементов по 32 бит,

\* входные данные для шифрования

\* 2. {word32 \*out\_data} массив из 2 элементов по 32 бит

\* для зашифрованных данных

\*

\* 3. {word32 const \*gost\_key} - массив из 8 элементов типа word32

\* (ключ шифрования)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CRYPT\_GOST\_API void gostcrypt( word32 const \*in\_data, word32 \*out\_data, word32 const \*gost\_key);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция gostdecrypt() расшифрует 64 бита информации

\* в режиме простой замены

\* Параметры функции:

\* 1. {word32 const \*in\_data} массив из 2 элементов по 32 бит,

\* входные данные для расшифрования

\*

\* 2. {word32 \*out\_data} массив из 2 элементов по 32 бит

\* для расшифрованных данных

\*

\* 3. {word32 const \*gost\_key} - массив из 8 элементов типа word32

\* (ключ расшифрования)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CRYPT\_GOST\_API void gostdecrypt(word32 const \*in\_data, word32 \*out\_data, word32 const \*gost\_key);

CRYPT\_GOST\_API word32 read\_vector\_init(const char \*path, const unsigned int \*length);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* private functions (вспомогательные функции)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static word32 f(word32 x);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Файл: diffy\_helman.h \* Описание: содержит объявление функций протокола обмена ключами

\* Диффи-Хеллмана

\* Автор: Иванов Роман Витальевич (c)

\* Год: 2014 – 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_DIFFY\_HELMAN\_H\_

#define \_DIFFY\_HELMAN\_H\_

#define DIFFY\_HELMAN\_API \_\_declspec(dllexport)

#if \_DEBUG

#define DEBUG\_INFO\_PRINT 1

#else

#define DEBUG\_INFO\_PRINT 0

#endif

#include "inc\_crypt\_pp.h"

const std::size\_t gost\_key\_size = 256;

class diffy\_helm

{

public:

DIFFY\_HELMAN\_API explicit diffy\_helm(void);

DIFFY\_HELMAN\_API explicit diffy\_helm(byte \*p, const std::size\_t &p\_length, byte \*q, const std::size\_t &q\_length, byte &g);

DIFFY\_HELMAN\_API ~diffy\_helm();

DIFFY\_HELMAN\_API void get\_p(byte \*p, const std::size\_t &p\_size)const;

DIFFY\_HELMAN\_API void get\_q(byte \*q, const std::size\_t &q\_size)const;

DIFFY\_HELMAN\_API void get\_g(byte &g);

DIFFY\_HELMAN\_API void generate\_A(byte \*A, const std::size\_t &A\_size);

DIFFY\_HELMAN\_API void generate\_K(byte \*B, const std::size\_t &B\_size, byte \*gost\_K, const std::size\_t &gost\_K\_size);

private:

diffy\_helm(diffy\_helm &);

diffy\_helm& operator=(diffy\_helm &);

private:

CryptoPP::AutoSeededRandomPool rnd\_;

CryptoPP::AutoSeededRandomPool rnd\_A\_;

CryptoPP::DH dh\_;

CryptoPP::DH dhA\_;

CryptoPP::Integer p\_;

CryptoPP::Integer g\_;

CryptoPP::Integer q\_;

CryptoPP::SecByteBlock \*priv\_A\_;

CryptoPP::SecByteBlock \*pub\_A\_;

CryptoPP::Integer key\_;

bool is\_generateA\_;

};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Файл: server\_network\_dll.h

\* Описание:содержит обяъявление функций для работы сервреа с сетью

\* Автор: Иванов Роман Витальевич (c)

\* Год: 2014

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef \_NETWORK\_SERVER\_DLL\_H\_

#define \_NETWORK\_SERVER\_DLL\_H\_

#define NETWORK\_SERVER\_API \_\_declspec(dllexport)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* define ServerNetworkDll version

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define NETWORK\_SERVER\_DLL\_VERSION 0x01

#include "inc\_boost\_heders.h"

#include "crypt\_gost\_types.h"

#include "server\_types.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Класс, инкапсулирующий сетевую часть сервера

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace server

{

class data\_parser;

class server\_network : private boost::noncopyable

{

private:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Конструктор класса server

\* Параметры конструктора:

\* 1. Логический порт сервера

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

explicit server\_network(const std::uint32\_t port);

public:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Деструктор класса server

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

~server\_network();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция для начала работы сервреа

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция для передачи данных клиенту

\* Параметры функции:

\* 1. массив данных

\* 2. размер массива

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void send\_bytes(byte \*data, const std::size\_t &size);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция для приёма данных от клиента

\* Параметры функции:

\* 1. массив данных

\* 2. размер массива

\* Возвращаемое значение: число принятых байтов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

boost::int32\_t read\_bytes(byte \*data, const std::size\_t &size);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция возвращает текущий логический порт

\* сервера

\* Возвращаемое значение: текущий логический порт

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::uint32\_t current\_port()const;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функция возвращает дата время соединения

\* клиента

\* Возвращаемое значение: строка дата и время

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::string client\_connect\_data\_time()const;

server\_network \*get\_instanse(const std::uint32\_t &port);

private:

void disable\_connect();

private:

SERVER\_INFO server\_info\_;

server\_time time\_;

std::string client\_connect\_time\_;

std::uint32\_t port\_;

boost::asio::io\_service io\_service\_;

boost::asio::ip::tcp::acceptor acceptor\_;

boost::asio::ip::tcp::socket socket\_;

data\_parser \*data\_parser\_;

PSERVER\_NET\_BUF net\_data\_;

PSERVER\_DATA\_BUFF pars\_data\_;

};

}

#endif

# ПРИЛОЖЕНИЕ Б Описание кодов ответа и запроса

