**Содержание**

Введение5

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству6

2. Анализ требований к программному средству и разработка функциональных требований14

3. Проектирование программного средства 21

3.1 Общая схема программного средства21

3.2 Проектирование алгоритмов подачи сюжета22

3.3 Проектирование алгоритмов работы с графикой и звуком27

3.4 Проектирование алгоритмов, реализующих игровую механику 31

3.5 Проектирование алгоритмов работы с настройками35

4. Создание (конструирование) программного средства 38

4.1 Конструирование алгоритма подачи сюжета(загадки) 38

4.2 Работа с графикой 39

**4.2.1 Переходы на другие подуровни39**

**4.2.2 Механизм обзора персонажа40**

4.3 Влияние уровня сложности на урон ловушек, эффекты от применения зелий и магии 40

4.4 Описание классов 41

5. Тестирование и проверка работоспособности программного средства 44

5.1 Таблица тестов44

5.2 Внесённые изменения57

6. Руководство по установке и использованию59

Заключение4

Список использованных источников4

Приложение А4

Приложение Б4

**Введение:**

Широкое развитие информационных технологий и информационных коммуникаций открыло возможность для передачи через сети не только текстовых сообщений, но также файлов различных типов. С развитием локальных сетей появилась потребность в наличии обособленного хранилища данных, которое хранит всю необходимую информацию компании или просто группы лиц.

Однако, развитие технологий хранения и распределения информации неизбежно привело и к развитию методов нелегального её получения и вопрос безопасности файловых серверов встал довольно остро. Организации, чьи локальные сети имеют доступ и в глобальную сеть серьёзно занимаются вопросами защиты своей информации, но владельцы небольших изолированных локальных сетей зачастую уверены, что они в безопасности и банально игнорируют вопрос защиты своих данных, что позволяет злоумышленникам проникнуть в их сеть и без особых проблем получить данных.

Изучив данный вопрос, я поставил себе цель – создать клиент-серверное приложение для малых организаций и локальных сетей, которое бы обеспечивало приемлемый уровень безопасности и при этом было бы удобным в использовании.

**Глава 1.**

На данный момент существует достаточно много решений, дающих пользователям возможность хранить свои данных удалённых серверах.

Основными примерами таковых являются:

1. Google Диск

Google Диск — это файловый хостинг, созданный и поддерживаемый компанией Google. Его функции включают хранение файлов в Интернете, общий доступ к ним и совместное редактирование. В состав Google Диска входят Google Документы, Таблицы и Презентации — набор офисных приложений для совместной работы над текстовыми документами, электронными таблицами, презентациями, чертежами, веб-формами и другими файлами. Общедоступные документы на Диске индексируются поисковыми системами.

Google Диск был представлен 24 апреля 2012 года, а к октябрю 2014 года насчитывал 240 миллионов ежемесячно активных пользователей.

15 мая 2018 года компания Google объявила о переименовании Google Drive (Google Диск) в Google One.

(Google Drive [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.)

Как утверждает компания Google: Файлы, загруженные на Google Диск, хранятся в защищенных центрах обработки данных. Больше ничего о защите данных в данном сервисе неизвестно. Впрочем, обещания компании Google по поводу абсолютной безопасности оказались ложными и в 2015 Google Диск был взломан.

Исследователи выяснили, что все из указанных сервисов предоставляют непрерывный доступ приложений-клиентов к серверам с помощью токенов, которые генерируются при первой аутентификации пользователя. Эти токены хранятся на ПК в специальных файлах, в реестре Windows или в Windows Credential Manager (зависит от приложения).

Аналитики написали приложение под названием Switcher, которое способно проникать на компьютер жертвы в виде вложения в почтовом сообщении или через уязвимость в браузере. Попав на машину, Switcher заменяет токен для доступа к аккаунту в определенном облачном сервисе на свой собственный, связанный с учетной записью злоумышленника.

Затем Switcher перезагружает приложение-клиент облачного сервиса, после чего оно уже использует подменный токен. В результате синхронизация папки на ПК пользователя происходит с облачным аккаунтом злоумышленника. Switcher помещает в эту папку копию подлинного токена, которую он создал заранее. Таким образом, хакер получает в свое распоряжение не только файлы жертвы, но и токен для последующего доступа к аккаунту жертвы уже со своего компьютера.

(Google Drive – безопасность [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://support.google.com/drive/answer/141702?hl=ru&co=GENIE.Platform%3DDesktop>.)

(Google Drive – взлом [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://safe.cnews.ru/news/top/najden_legkij_sposob_vzloma_akkuantov>.)

Исходя из этого, к плюсам данной системы можно отнести:

+ широкие возможности по редактированию информации

+ синхронизация файлов

+ привязка к аккаунту

Основные минусы:

- сомнительная безопасность системы

- наличие только 15 гб бесплатного пространства

- индексация файлов поисковой системой, что может приводит к утечке вашей информации в общий доступ, если вы не успели настроить приватность

1. FTP-сервер

FTP (англ. *File Transfer Protocol* — протокол передачи файлов) — стандартный протокол, предназначенный для передачи файлов по TCP-сетям (например, Интернет). Использует 21-й порт. FTP часто используется для загрузки сетевых страниц и других документов с частного устройства разработки на открытые сервера хостинга.

Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером. Пользователи FTP могут пройти аутентификацию, передавая логин и пароль открытым текстом, или же, если это разрешено на сервере, они могут подключиться анонимно. Можно использовать протокол SSH для безопасной передачи, скрывающей (шифрующей) логин и пароль, а также шифрующей содержимое.

FTP не разрабатывался как защищённый (особенно по нынешним меркам) протокол и имеет многочисленные уязвимости в защите. В мае 1999 авторы RFC 2577 свели уязвимости в следующий список проблем:

* Скрытые атаки (bounce attacks)
* Спуф-атаки (spoof attacks)
* Атаки методом грубой силы (brute force attacks)
* Перехват пакетов, сниффинг (packet capture, sniffing)
* Защита имени пользователя
* Захват портов (port stealing)

FTP не может зашифровать свой трафик, все передачи — открытый текст, поэтому имена пользователей, пароли, команды и данные могут быть прочитаны кем угодно, способным перехватить пакет по сети. Эта проблема характерна для многих спецификаций Интернет-протокола (в их числе SMTP, Telnet, POP, IMAP), разработанных до создания таких механизмов шифрования, как TLS и SSL. Обычное решение этой проблемы — использовать «безопасные», TLS-защищённые версии уязвимых протоколов (FTPS для FTP, TelnetS для Telnet и т. д.) или же другой, более защищённый протокол, вроде SFTP/SCP, предоставляемого с большинством реализаций протокола Secure Shell.

(FTP/FTPS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.)

(SSH [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.)

Основные плюсы:  
+ бесплатность  
+ возможность настройки доступа и аутентификации  
+ расширение безопасности за счёт использования SSL, SSH, FTPS

Минусы:  
- необходимость ручной настройки

Таким образом, первичными требованиями к программному средству являются

1. Обеспечение шифрования данных в процессе передачи и на самом сервере

2. Реализация механизмов регистрации и аутентификации на сервере

3. Разделение рабочих пространств пользователей

4. Обеспечение получения файлов с сервера

5. Обеспечение загрузки файлов на сервер

«Список литературы»

[1] Google Drive [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

[2] Google Drive – безопасность [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://support.google.com/drive/answer/141702?hl=ru&co=GENIE.Platform%3DDesktop>.;

[3] Google Drive – взлом [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://safe.cnews.ru/news/top/najden_legkij_sposob_vzloma_akkuantov>.;

[4] FTP/FTPS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

[5] SSH [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

**2.Анализ требований к программному средству и разработка функциональных требований**

В данной главе будет приведено текстовое описание работы программного средства, на основании которого будут расширены и уточнены первоначальные функциональные требования.

Клиент:

При запуске приложения открывается окно в котором присутствуют поля для ввода логина и пароля, кнопки «Регистрация», «Авторизация» и «Выход». Также должны присутствовать кнопки «Загрузить на сервер» и «Загрузить с сервера». В середине окна располагается поле, в котором пользователь видит список загруженных на сервер файлов.

При нажатии на кнопку регистрация при заполненных полях ввода пароля и логина регистрируется новый пользователь. Серверу отправляется запрос на регистрацию, после чего приходит ответ, который выводится во всплывающем окне.

При нажатии на кнопку авторизация и заполненных полях ввода серверу отправляется запрос на авторизацию вместе с данными для авторизации. После чего на клиент приходит ответ с информации от сервера, а именно статусе авторизации.

При нажатии на кнопку выход серверу отправляется запрос на выход клиента, после чего сервер сохраняет последние изменения и разрывает соединение с клиентом.

При нажатии на кнопку «Загрузить на сервер», в случае, если клиент успешно авторизован происходит открытие диалогового окна, в котором пользователю будет предложено выбрать файл для загрузки, после того, как пользователь выбрал файл и подтвердил выбор, начинается передача данных серверу в зашифрованном виде. В случае, если пользователь не авторизован, то будет выведено соответствующее предупреждение и отправка файла не будет осуществлена.

При нажатии на кнопку «Загрузить с сервера» происходит отправка запроса на сервер, в случае успеха клиенту будет представлено диалоговое окно, в котором можно будет указать место для сохранения файла и имя сохраняемого файла. Если пользователь не авторизован, то будет выведено соответствующее предупреждение.

Список функциональных требований

1. Открытие окна программы при её запуске
2. Получение данных о нахождении сервера при запуске программы
3. Установления соединения с сервером при запуске программы
4. Получения ключа для шифрования от сервера
5. Отправка зашифрованного запроса на регистрацию при нажатии на кнопку регистрация и заполненных полях логина и пароля
6. Отправка шифрованного запроса на авторизацию при нажатии на кнопку авторизации и заполненных полях логина и пароля
7. Отправка шифрованного запроса на получение файла
8. Отправка шифрованного запроса на передачу файла
9. Отображение списка текущих файлов пользователя на сервере
10. Шифрование данных и их расшифровка
11. Закрытие соединения при нажатии на кнопку «Выход»
12. Корректная обработка ошибок с выводом пользователю информации о них
13. Возможность попытаться восстановить потерянное соединение

**3. Проектирование программного средства**

**3.1 Общая схема программного средства**

Программное средство Praetorian будет представлять собой оконное приложение, которое будет открывать диалоговые окна для выполнения действий по сохранению или отправке файлов, а также всплывающие окна для уведомления пользователя об ошибках или необходимости совершения предварительных действий. Помимо этого, в состав приложения входит сервер, с которым будут взаимодействовать клиентские приложения.



Рисунок 3.1 – Общая схема программы

Главное окно – стартовое окно программы, в котором пользователю будут доступны возможности по регистрации и авторизации на сервере, возможность загрузки своих файлов на сервер, а также по загрузке своих файлов с сервера. Все данные будут передаваться между клиентом и сервером в зашифрованном виде, а также храниться в зашифрованном виде на самом сервере.

**3.2 Проектирование алгоритмов определения адреса сервера в сети LAN**

Одним из функциональных требований к проекту было возможность динамического определения IP-адреса сервера в локальной сети, для того, чтобы пользователь не был обязан задумываться об этом вопросе, однако возможность ручной настройки IP-адреса сервера будет оставлена и реализовываться через изменение config файла клиента. Для динамического определения IP-адреса можно использовать широковещательный UDP-запрос на локальную сеть, после получением сервером данного запроса в ответ будет выслан IP-адрес сервера. В случае, если ответ от сервера не был получен с первой попытки, то происходит повторная отправка UDP-запроса. Это будет происходить до тех пор, пока не будет получен ответ от сервера, либо не будет превышено число попыток.



Рисунок 3.2. – алгоритм получения адреса сервера в сети

**3.3 Проектирование алгоритмов передачи файлов по сети**

Основной возможностью данного приложения является возможность передавать файлы в шифрованном виде по локальной сети. Для её реализации я буду использовать протокол TCP, поскольку он обеспечивает надёжную передачу пакетов по сети с контролем за потерями, что критически важно при передаче зашифрованных файлов, так как в зависимости от алгоритма шифрования это может привести либо к частичному, либо к полному искажению передаваемой информации. Однако, для осуществления передачи файлов различных размеров необходимо задать определённую структуру передаваемых данных, так как TCP не осуществляет контроль за полным размером передаваемых данных. Для этого, мною была предложена следующая структура для передачи данных серверу и с сервера.

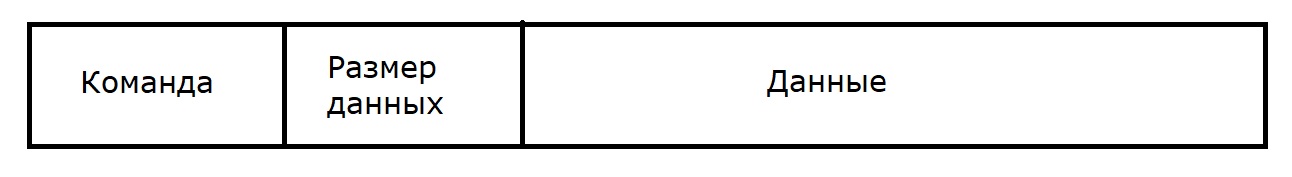


Рисунок 3.3 – Формат запроса

1. Команда – поле команды будет иметь фиксированный размер в 1 байт и содержать в себе численное представление команды, которую будет присылать клиент. Для размера поля команды в 1 байт можно реализовать 255 команд, что, по-моему мнению, на данный момент избыточно, однако позволяет продолжить поддержку проекта и добавление новых команд, без изменения в структуре запроса.
2. Размер данных – поле размер данных будет иметь фиксированный размер в 8 байт и содержать в себе длинное целое число, в котором будет храниться размер поля данных в байтах. Поле размера данных позволит серверу корректно определить размер получаемых данных и выделить на их обработку ровно столько памяти, сколько для этого нужно и позволит снять ограничение на размер передаваемого файла, так как длинное целое число может вместить в себя значение максимальное значение которого равно 18 446 744 073 709 551 615.
3. Данные – поле в котором будут храниться данные, передаваемые серверу, такие как логин/пароль или непосредственно содержимое файла.

Структура ответа от сервера отличается лишь тем, что сервер вместо команды будет присылать свой ответ на команду и в зависимости от его успешности, либо запрошенное содержимое, либо описание ошибки.



Рисунок 3.4. – Алгоритм формирования запроса/ответа

* 1. **Проектирование алгоритмов шифрования**

На данный момент существует большое количество типов и стандартов шифрования. В данной работе я буду использовать готовые библиотеки для реализации шифрования, так как они были написаны специалистами, что минимизирует риск допустить ошибку при реализации или написании собственного алгоритма шифрования, либо самостоятельной реализации уже готового алгоритма.

Шифрованием называется обратимый процесс преобразования информации с целью их сокрытия от непривилегированных лиц. Важной особенностью любого метода шифрования считается использование ключа, утверждающего выбор определённого преобразования из всех возможных для данного метода.

Для того, чтобы информация, прошедшая шифрование, превратилась в бессмысленный для постороннего пользователя набор символов, были разработаны специальные алгоритмы шифрования. В целом, все алгоритмы можно разбить на 2 группы: симметричные и асимметричные.

Симметричные алгоритмы (*AES, CAST, ГОСТ, Blowfish, DES*) используют для шифровки и дешифровки один ключ данных. Главный недостаток этих алгоритмов – в случае кражи ключа шифрования, злоумышленник может похитить и расшифрованные данные. Кроме того, существуют технологии криптоатак, позволяющие дешифровать данные без использования ключа шифрования.

Асимметричные алгоритмы (El-Gamal, RSA) используют для шифрования разные ключи – открытый и закрытый. Открытый ключ передаётся по незащищённому каналу и предназначен для проверки ЭЦП и шифровки сообщения. Для генерации ЭЦП и дешифровки применяется секретный ключ. Асимметричные алгоритмы шифрования частично решают проблему перехвата ключей, так как для успешного расшифрования текста ему придётся решить проблему дискретного логарифма.

Так как шифрование данных будет происходить при передаче с клиента на сервер и на самом сервере, то необходимо выбрать алгоритмы для обоих случаев.

1. Передача между клиентом и сервером

А) Симметричные алгоритмы  
Возможность применения в данном случае симметричных алгоритмов представляется возможным в двух случаях:

- клиент и сервер имею одинаковый ключ по умолчания

- клиент и сервер реализуют механизм обмена ключом, который позволяет двум сторонам получить одинаковый ключ, при этом не открывая его злоумышленнику. В качестве такого механизма может выступать алгоритм Диффи-Хелмана

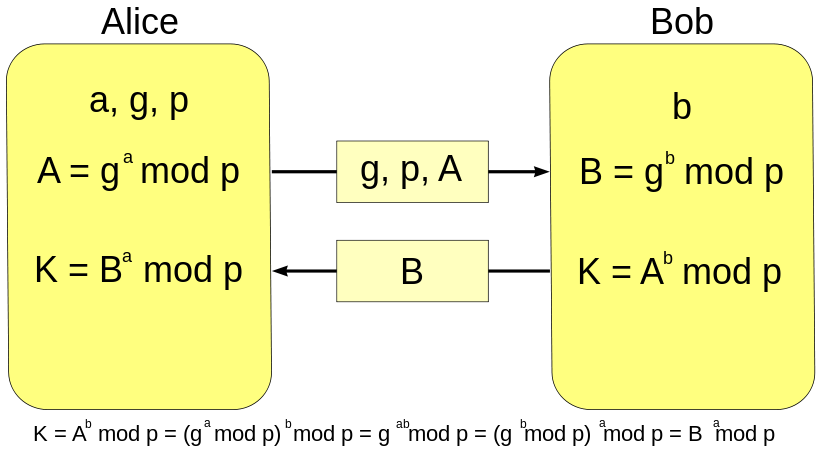
****

Рисунок 3.5. – протокол Диффи-Хелмана

При работе алгоритма каждая сторона:

1. генерирует случайное натуральное число *a* — закрытый ключ
2. совместно с удалённой стороной устанавливает открытые параметры *p* и *g* (обычно значения *p* и *g* генерируются на одной стороне и передаются другой), где

*p* является случайным простым числом

*(p-1)/2* также должно быть случайным простым числом (для повышения безопасности)

*g* является первообразным корнем по модулю *p (*также является простым числом*)*

1. вычисляет открытый ключ *A*, используя преобразование над закрытым ключом

*A = ga* mod *p*

1. обменивается открытыми ключами с удалённой стороной
2. вычисляет общий секретный ключ *K*, используя открытый ключ удаленной стороны *B* и свой закрытый ключ *a*

*K = Ba* mod *p*

*К* получается равным с обеих сторон, потому что:

*Ba* mod *p = (gb* mod *p)a* mod *p = gab* mod *p = (ga* mod *p)b* mod *p = Ab* mod *p*

В практических реализациях для *a* и *b* используются числа порядка 10100 и *p* порядка 10300. Число *g* не обязано быть большим и обычно имеет значение в пределах первого десятка.

При этом симметричные алгоритмы в общем случае дают меньшую вычислительную нагрузку на процессор, что положительно скажется на производительности сервера.

Б) Ассиметричные алгоритмы шифрования

Основным плюсом данных алгоритмов является наличие двух ключей – открытого и закрытого, при этом сообщение зашифрованное открытым/закрытым ключом можно расшифровать только имея закрытый/открытый ключ шифрования. Данный механизм позволяет клиенту и серверу обменяться открытыми ключами, после чего вести защищённую передачу данных, так как злоумышленник, не имея закрытого ключа не сможет расшифровать данные, а в случае, если он захочет самостоятельно вычислить ключ, то ем придётся столкнуться с проблемой дискретного логарифма. Однако, несмотря на высокую криптостойкость ассиметричные алгоритмы имеют два существенных недостатка:  
- Высокая вычислительная нагрузка на компьютер(клиент, сервер)  
- Полная потеря криптоскойкости в случае разрешения проблемы дискретного логарифма

Вывод:

На основании этого, в качестве алгоритма для шифрования данных, передаваемых между клиентом и сервером будет применяться симметричный алгоритм шифрования с использованием алгоритма Диффи-Хелмана. В качестве алгоритма будет выбран AES. Advanced Encryption Standard, также известный как Rijndael— симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES. Надёжность данного алгоритма признана настолько высокой, что Агентство национальной безопасности США постановило, что при помощи AES можно шифровать сведения, составляющие государственную тайну США.

2) Шифрование на сервере

Для шифрования на сервере может использоваться любой тип шифрования так как передачи ключа не происходит. Поэтому данный анализ будет производиться в контексте сложности операции шифрования/дешифрирования, поскольку серверу необходимо будет обрабатывать большие объёмы данных и работать с большим числом пользователей.

А) Симметричные алгоритмы

DES — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 бита. В основе алгоритма лежит сеть Фейстеля с 16 циклами и ключом, имеющим длину 56 бит.

Входными данными для DES служат:

* блок размером n бит;
* ключ размером k бит.

На выходе (после применения шифрующих преобразований) получается зашифрованный блок размером n бит, причём незначительные различия входных данных, как правило, приводят к существенному изменению результата.

Блочные шифры реализуются путём многократного применения к блокам исходного текста некоторых базовых преобразований.

Базовые преобразования:

* сложное преобразование на одной локальной части блока;
* простое преобразование между частями блока.

Так как преобразования производятся поблочно, требуется разделение исходных данных на блоки необходимого размера. При этом формат исходных данных не имеет значения (будь то текстовые документы, изображения или другие файлы). Данные должны интерпретироваться в двоичном виде (как последовательность нулей и единиц) и только после этого должны разбиваться на блоки. Все вышеперечисленное может осуществляться как программными, так и аппаратными средствами.

Таким образом основная сложность алгоритма DES заключается в выполнениях операции перестановки и хранению таблиц шифрования для DES (так называемых S-box), а также в необходимости написания алгоритма генерации ключа, так как на основании алгоритма DES выделяются множества слабых и частично слабых ключей.

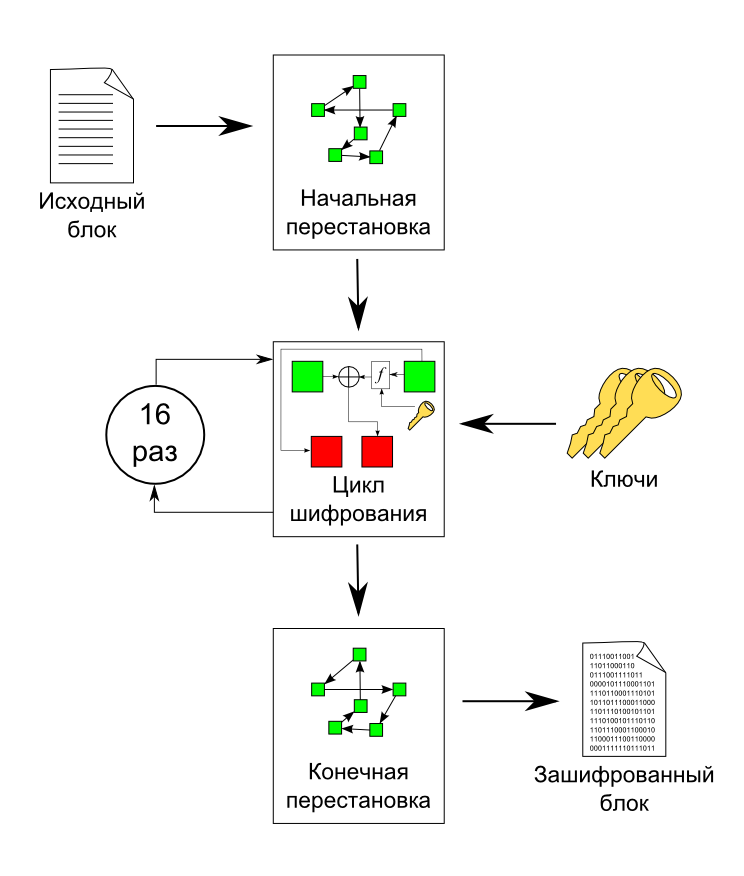


Рисунок 3.6 – Схема шифрования в алгоритме DES

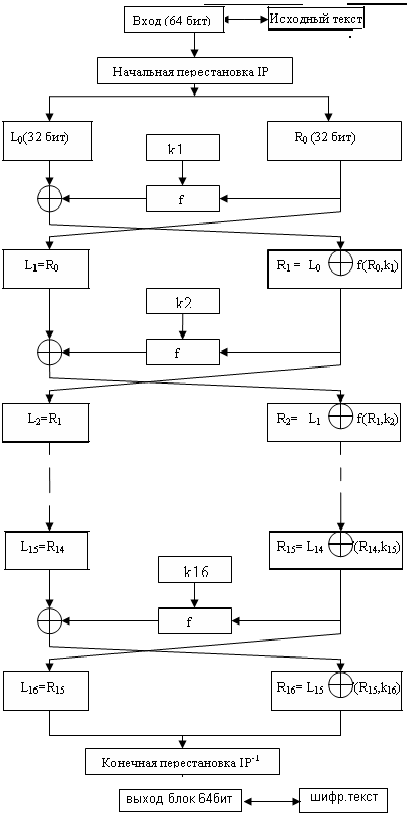


Рисунок 3.7 – подробная схема шифрования алгоритма DES

Blowfish — криптографический алгоритм, реализующий блочное симметричное шифрование с переменной длиной ключа. Разработан Брюсом Шнайером в 1993 году. Представляет собой сеть Фейстеля. Выполнен на простых и быстрых операциях: XOR, подстановка, сложение. Является незапатентованным и свободно распространяемым.

Алгоритм состоит из двух частей: расширение ключа и шифрование данных. На этапе расширения ключа исходный ключ (длиной до 448 бит) преобразуется в 18 32-битовых подключей и в 4 32-битных S-блока, содержащих 256 элементов.

Blowfish зарекомендовал себя как надёжный алгоритм, поэтому реализован во многих программах, где не требуется частая смена ключа и необходима высокая скорость шифрования/расшифровывания.

* хэширование паролей
* защита электронной почты и файлов
  + GnuPG (безопасное хранение и передача)
* в линиях связи: связка ElGamal (не запатентован) или RSA (действие патента закончилось в 2000 году) и Blowfish вместо IDEA
  + в маршрутизаторе Intel Express 8100 с ключом длиной 144 бита
* обеспечение безопасности в протоколах сетевого и транспортного уровня
  + SSH (транспортный уровень)
  + OpenVPN (создание зашифрованных каналов)

В целом алгоритм Blowfish подходит для реализации шифрования данных на сервере, так как базируется на простых операциях, что положительно сказывается на быстродействии сервера. Однако, в рамках разработки планируется ввести систему, которая осуществляет перешифрование данных через определённый промежуток времени, что может привести к появлению слабых S-блоков в шифраторе.

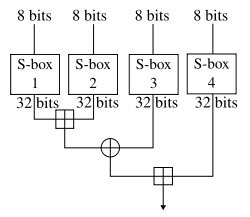


Рисунок 3.8 – функция преобразования Blowfish

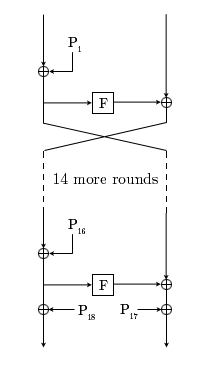


Рисунок 3.9 – алгоритм шифрования Blowfish

Потоковое шифрование – основная идея потоковых криптосистем заключается в шифровании исходного текста М с помощью криптографического ключа К, длина которого равна длине текста. Каждый бит шифротекста С является исключающим или соответствующих битов исходного текста и ключевого потока. Для генерации ключа используют последовательные сдвиговые регистры с обратной связью, криптостойкость алгоритма зависит от генерирующего многочлена.

Основным плюсом данного метода является его простота реализации и высокая производительность, так как операция xor является одной из самой малозатратных, при этом можно обеспечить высокую степень криптоскойкости за счёт изменения генерирующего многочлена.

Б) Ассиметричные алгоритмы

Несмотря на то, что на данный момент существует большое количество криптосистем с открытым и закрытым ключом, их всех объединяет один недостаток – это достаточно сложные вычислительные операции, которые нужно проводить для шифрования, дешифрования данных, такие как нахождение большого целого простого числа и возведение в степень по модулю. Поэтому, я считаю, что применение ассиметричных алгоритмов для шифрования данных на сервере не рациональным, поскольку представленные выше алгоритмы шифрования в данном случае будут иметь почти аналогичную криптостойкость, при этом экономя ресурсы сервера.

Таким образом, на основании анализа, мною был предпочтён алгоритм потокового шифрования, так как он наилучше сочетает в себе качества, соответствующие требованиям проекта.

**4. Создание (конструирование) программного средства**

На этапе конструирования основной моей задачей стала реализация функциональных требований на основе разработанных алгоритмов. В данной главе я планирую рассмотреть наиболее интересные решения, внесённые мной в программное средство.

* 1. **Конструирование алгоритма авторизации**

Основная задача авторизации – убедится в том, что данный пользователь имеет право пользоваться сервером и дать ему соответствующее право на это. Авторизация на сервере будет происходить путём передачи по сети хеширования логина и пароля. При этом, поскольку перед началом пользования клиент не имеет ключа шифрования, то передача их в открытом виде недопустима. Для того, чтобы злоумышленник не смог получить их, логин и пароль перед отправкой будут хешироваться алгоритмом SHA256. После приёма хешей, сервер проверяет, существует ли такой пользователь в реестре пользователей и в случае нахождения отправляет клиенту данные для вычисления ключа шифрования. После этого начинается второй этап проверки, клиент шифрует полученным ключом свои логин и пароль и передаёт серверу, после чего сервер расшифровывает их сравнивает с первоначально полученными хешами. В случае совпадения, авторизация проходит успешно и пользователю отправляется подтверждение авторизации.

Данный алгоритм необходим для того, чтобы злоумышленник, не смог перехватить пакеты и авторизоваться вместо пользователя. Случае атаки злоумышленник может получить только хеш-пароля и логина, данные для вычисления ключа, который он вычислить не сможет и зашифрованные логин и пароль, что недостаточно для того, чтобы авторизоваться вместо другого пользователя

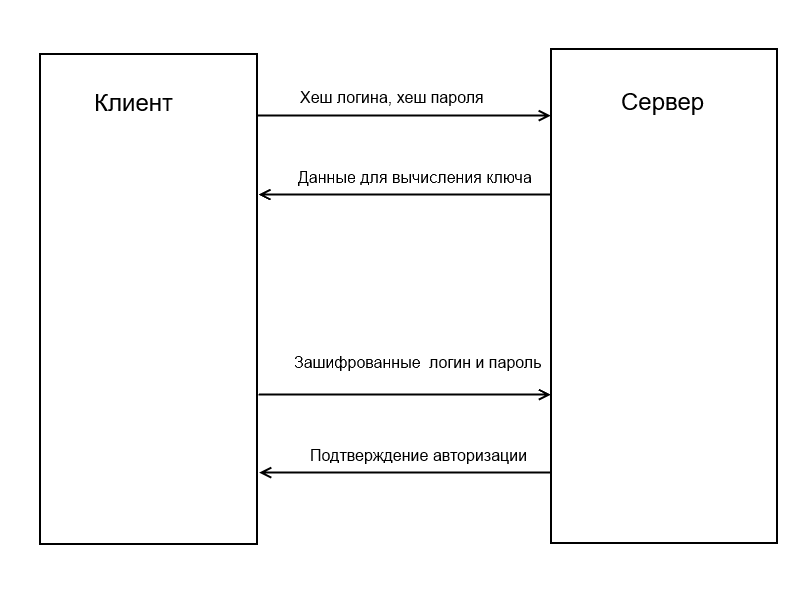


Рисунок 4.1 – Схема авторизации

* 1. **Конструирование алгоритма определения нахождения сервера**

Поскольку адрес сервера в локальной сети может быть непостоянным и для того, чтобы избавить пользователей от необходимости ручной настройки его положения через config файл, был сконструирован алгоритм нахождения сервера. В его основе лежит широковещательная рассылка UDP пакетов по сети.

Платформа .NET предоставляет стандартный класс для реализации обмена дейтаграммами через UDP, а именно public class UdpClient : IDisposable, который содержит метод [JoinMulticastGroup(Int32, IPAddress)](https://msdn.microsoft.com/ru-ru/library/xt8ahkww(v=vs.110).aspx), который позволяет присоединиться к группе широковещательной рассылки. После чего используются стандартные методы Send() и Receive().

В случае получения сервером дейтаграммы, содержащей запрос на получение адреса, сервер получает из дейтаграммы адрес клиента и отправляет ему ответный UDP пакет, со своим адресом. После чего клиент получает адрес и инициирует TCP-подключение для передачи данных.

Для реализации процедуры нахождения сервера используется класс UDPLocator

**4.3 Описание классов**

**Класс Client**

. Предназначен для хранения параметров текущего пользователя

Описание класса:

* Поля
* Login – поле, хранящее логин пользователя
* Password – поле, хранящее пароль пользователя
* isAuthorised – поле, хранящее текущее состояние пользователя в контексте авторизации на сервере
* Методы
* Client() – конструктор класса

**Класс Connection**

Предназначен для реализации соединения и передачи данных между клиентом и сервером.

Описание класса:

* + Поля
* Client – поле, хранящее текущий TcpClient
* IP – поле, хранящее IP-адрес клиента
* Socket – поле, хранящее номер сокета, на который будут посылаться TCP-пакеты
* Stream – поле, хранящее текущий сетевой поток
* AES – поле, хранящее текущий экземпляр объекта шифрования/дешифрирования
  + Методы
* Connection() – конструктор класса
* SetConnection() – метод, устанавливающий соединение между клиентом и сервером
* Reg() – метод, реализующий механизм регистрации на сервере
* Auth() – метод, реализующий механизм авторизации на сервере
* Get() – метод, предназначенный для получения данных с сервера
* Send() – метод, предназначенный для отправки данных на сервер
* Reconnect() – метод, реализующий механизм восстановления подключения

**Класс AppCommands**

Предназначен для хранения команд клиента и сервера. Данный класс является абстрактным

Описание класса:

* + Поля
* ClientCommands – поле, хранящее коды команд клиента
* ServerAnswers – поле, хранящее коды ответов сервера

**Класс AES\_DiffieHelman**

Предназначен для реализации объекта, отвечающего за шифрование и дешифрирование данных

Описание класса:

* + Поля
* PublicKey – поле, хранящее материал ключа, передаваемый по сети
* Key – поле, хранящее ключ, используемый для шифрования данных.
* ChangeMechanism – поле, хранящее стандартный объект, реализующий протокол Диффи-Хелмана
* Aes – поле, хранящее стандартный объект, реализующий алгоритм AES
* Client\_IV – поле, хранящее данные для получения ключа на основе эллиптических кривых
  + Методы
* AES\_DiffieHelman() – конструктор класса
* GetSecretKey() – метод, предназначенный для получения секретного ключа
* Encrypt() – метод, предназначенный для шифрования данных
* Decrypt() – метод, предназначенный для дешифрирования данных

**5. Тестирование и проверка работоспособности программного средства**

Процессу тестирования подвергались версии проекта различной степени завершённости. Если какая-то из проблем, решённая в предыдущих версиях, не описана в следующих, то она была решена и больше не возникала. В случае, если программное средство не прошло какой-либо из тестов, то описание проблемы и её решения можно найти ниже таблицы тестов.

**5.1 Таблица тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер тестируемого функционального требования | Описание цели тестирования | Ожидаемый результат | Полученный результат |
| 1 | Проверка отображения на экране окна приложения | Окно приложения меню отобразилось после запуска программы | Соответствует ожидаемому |
| 2 | Проверка получения данных о нахождении сервера при запуске программы | При открытии приложения сервер присылает свой IP адрес в локальной сети | Соответствует ожидаемому |
| 3 | Проверка установления соединения с сервером при запуске программы | При запуске программы устанавливается TCP-соединение с сервером | Соответствует ожидаемому |
| 4 | Проверка получения ключа для шифрования от сервера | После установления соединения с сервером клиенту присылается ключ для шифрования данных | Соответствует ожидаемому |
| 5 | Проверка отправки зашифрованного запроса на регистрацию при нажатии на кнопку регистрация и заполненных полях логина и пароля | Клиент отправляет серверу зашифрованный запрос на регистрацию | Соответствует ожидаемому |
| 6 | Проверка отправки шифрованного запроса на авторизацию при нажатии на кнопку авторизации и заполненных полях логина и пароля | Клиент отправляет серверу зашифрованный запрос на авторизацию | Соответствует ожидаемому |
| 7 | Проверка отправки шифрованного запроса на получение файла | Клиент отправляет серверу зашифрованный запрос на получение файла | Соответствует ожидаемому |
| 8 | Проверка отправки шифрованного запроса на передачу файла | Клиент отправляет серверу зашифрованный запрос на передачу файла и передаёт файл в зашифрованном виде | Соответствует ожидаемому |
| 9 | Проверка отображения списка текущих файлов пользователя на сервере | Пользователю виден список всех его файлов, которые загружены на сервер | Соответствует ожидаемому |
| 10 | Проверка шифрования данных и их расшифровки | Клиент корректно расшифровывает и зашифровывает передаваемые данных и запросы | Соответствует ожидаемому |
| 11 | Проверка закрытия соединения при нажатии на кнопку «Выход» | При нажатии кнопки «Выход» осуществляется разрыв соединения и деавторизация пользователя | Соответствует ожидаемому |
| 12 | Проверка корректной обработки ошибок с выводом пользователю информации о них | Программа ведёт себя корректно при возникновении ошибок и уведомляет пользователя текстовыми сообщениями с содержанием ошибки | Соответствует ожидаемому |
| 13 | Проврка возможности попытаться восстановить потерянное соединение | В случае потери соединения приложения производит попытки переподключения | Соответствует ожидаемому |

**6. Руководство по установке и использованию**

Программное средство Praetorian будет поставляться в комплекте, состоящим из двух программ, а именно клиента и сервера, после чего администратор локальной сети должен выделить компьютер-сервер и установить на него серверную часть Praetorian, а на остальные компьютеры установить программу-клиент.

Для запуска программного приложения, необходимо открыть файл Praetorian.exe, для быстрого доступа рекомендуется создать ярлык на рабочем столе. Вид окна может отличаться, в зависимости от используемой операционной системы и цветовой схемы.

При открытии приложения на экране должно появится окно следующего вида:

**Окно приложения**

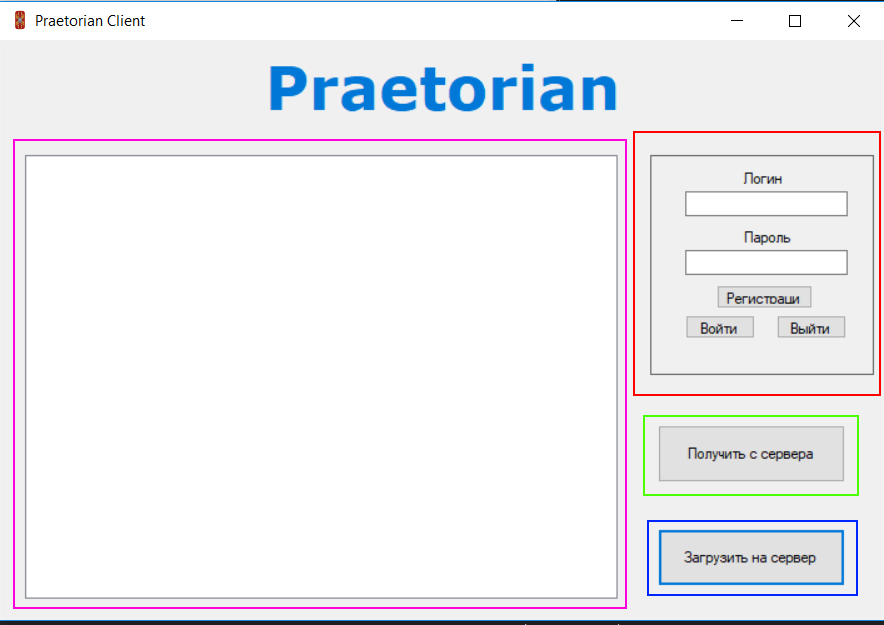


Рис 6.1 – Скриншот окна главного меню

Красной областью выделена панель авторизации, регистрации и выхода. Для регистрации необходимо заполнить поля ввода и нажать на кнопку «регистрация». Для авторизации необходимо заполнить поля ввода и нажать на кнопку «Войти». Результат авторизации будет показан в виде всплывающего окна, с текстом, содержащим результат действия.

Зелёной областью выделена кнопка, по нажатию которой происходит загрузка файла с сервера на клиент в выбранную пользователем директорию. Для того, чтобы выбрать необходимый файл – выберите его в списке, который обозначен розовым цветом.

Синей областью обозначена кнопка, нажатие которой приводит к открытию диалогового окна следующего вида

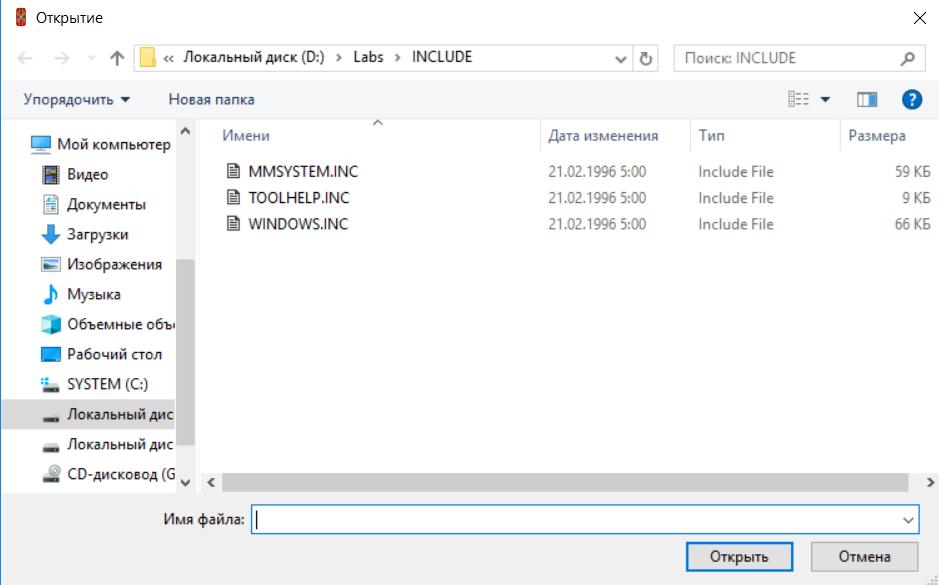


Рисунок 6.2 – Диалоговое окно выбора файла

В случае ошибки, пользователю будет выведено сообщение об ошибке, в котором будет описана суть произошедшей ошибки

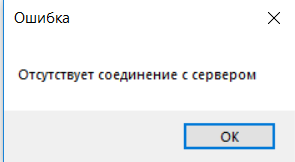


Рисунок 6.3 – Диалоговое окно ошибки

Пользовательский интерфейс клиента Praetorian достаточно простой и интуитивно понятный, что позволяет пользователю быстро в нём освоиться и начать использовать программное средство.

**Заключение**

В ходе выполнения курсовой работы были проанализированы и рассмотрены значительные объёмы текстовой, графической и видеоинформации по теме анализа, планирования и конструирования программных средств. Были изучены примеры коммерчески успешных проектов похожей тематики, выявлены их преимущества и недостатки.

На основании анализа полученной информации был сформулирован ряд общих требований к программному средству, выполнение которых должно было обеспечить пользователю комфортное использование приложения Praetorian.

Сформулирован ряд функциональных требований к программному средству, выполнение которых обеспечило надёжность и безопасность функционирования и работы с программным средством.

Произведено конструирование программного средства и реализация спроектированных алгоритмов в программном коде. Некоторые алгоритмы были подвергнуты переработке и включены в программное средство в улучшенном варианте. Реализован графический пользовательский интерфейс.

Проведено тестирование программного средства на предмет ошибок и недочётов, в ходе которого были выявлены и устранены существовавшие проблемы. Проведён анализ ошибок, с целью использования данного опыта в будущих проектах.

Создано руководство пользователя, для обеспечения его информацией, необходимой для удобного использования игрового проекта.

Данный проект может быть улучшен, путём добавления в него новых возможностей, таких как синхронизация файловых каталогов на стороне клиента и сервера, а также написанием мобильного приложения для взаимодействия с сервером.

**Список использованной литературы**

[1] Шарп. Д. Microsoft Visual C#. Подробное руководство/ Д.Шарп – СПб. : Питер, 2017. – 848 с.

[2] Шилдт. Г. С# 4.0. Полное руководство/ Г.Шилдт – Москва: Издательский дом «Вильямс», 2015. – 1056 с.

[3] Ярмолик С.В. Криптография и охрана коммерческой информации / С.В.Ярмолик, В.Н.Ярмолик. – Минск: БГУИР, 2010. – 35 с.

[4] Ахо А. Структуры данных и алгоритмы/ А. Ахо, Д. Хопкрофт, Д. Ульман. – Москва: Издательский дом «Вильямс», 2003. – 382 с.

[5] Sharp [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/.

[6] Google Drive [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

[7] Google Drive – безопасность [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://support.google.com/drive/answer/141702?hl=ru&co=GENIE.Platform%3DDesktop>.;

[8] Google Drive – взлом [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://safe.cnews.ru/news/top/najden_legkij_sposob_vzloma_akkuantov>.;

[9] FTP/FTPS [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

[10] SSH [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wikipedia.org/.;

**Приложение А**

**Исходный код программного средства**