**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Кафедра информатики и компьютерных систем**

ДАНИЛИН

Даниил Алексеевич

**РАЗРАБОТКА БЕЗОПАСНОГО КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ ОБМЕНА ИНФОРМАЦИИ НА ПРЕДПРИЯТИИ**

Дипломная работа

Научный руководитель:   
старший преподаватель,   
Ю.А. Бондаренко

Допущена к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Зав. кафедрой информатики и компьютерных систем,

доцент И.П. Стецко

Минск, 2022

****ОГЛАВЛЕНИЕ****

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 3](#_Toc105061892)

[РЕФЕРАТ 5](#_Toc105061893)

[ВВЕДЕНИЕ 8](#_Toc105061896)

[ГЛАВА 1 АНАЛИЗ КРИПТОГРАФИЧЕСКИХ МЕТОДОВ ЗАЩИТЫ ИНФОРМАЦИИ И ИССЛЕДОВАНИЕ ДОСТУПНЫХ ПРОГРАММНЫХ РЕШЕНИЙ 10](#_Toc105061897)

[1.1 Назначение и структура криптографических методов защиты 10](#_Toc105061898)

[1.2 Анализ криптографических методов 12](#_Toc105061899)

[1.3 Алгоритмы шифрования 13](#_Toc105061900)

[1.3.1 Симметричные алгоритмы 14](#_Toc105061901)

[1.3.2 Асимметричные алгоритмы 16](#_Toc105061902)

[1.3.3 Стойкость шифра 17](#_Toc105061903)

[1.4 Описание гибридных систем шифрования 18](#_Toc105061904)

[1.4.1 Пример работы гибридного алгоритма 18](#_Toc105061905)

[1.4.2 Слабости гибридного шифрования 19](#_Toc105061906)

[1.5 Сравнительный обзор программных средств для безопасного обмена информации 20](#_Toc105061907)

[ГЛАВА 2 РАЗРАБОТКА ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ И ВЫБОР ТЕХНОЛОГИЙ ДЛЯ РАЗРАБОТКИ 22](#_Toc105061908)

[2.1 Диаграмма вариантов использования 22](#_Toc105061909)

[2.1.1 Выделение необходимых требований для клиента 22](#_Toc105061910)

[2.1.2 Выделение необходимых требований для сервера 23](#_Toc105061911)

[2.2 Анализ требований к шифрованию 24](#_Toc105061912)

[2.3 Анализ требований к стороннему тестированию и наличию открытого кода 24](#_Toc105061913)

[2.4 Анализ требований к сбору и хранению данных пользователей 25](#_Toc105061914)

[2.5 Постановка задачи 26](#_Toc105061915)

[ГЛАВА 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРИЛОЖЕНИЯ, ПОСТРОЕННОГО НА АЛГОРИТМАХ ШИФРОВАНИЯ 27](#_Toc105061916)

[3.1 Выбор технологий и среды разработки 27](#_Toc105061917)

[3.1.1 Обзор и анализ технологий для клент-серверной архитектуры 27](#_Toc105061918)

[3.1.2 Обзор и анализ алгоритмов шифрования 27](#_Toc105061919)

[3.2 Описание архитектурной модели приложения (про ключи) 28](#_Toc105061920)

[ГЛАВА 4 РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ С# 29](#_Toc105061921)

[4.1 Общая структура приложения 29](#_Toc105061922)

[4.1.1 Разработка симметричного алгоритма шифрования 29](#_Toc105061923)

[4.1.2 Разработка серверной части 29](#_Toc105061924)

[4.1.3 Разработка клиентской части 29](#_Toc105061925)

[4.1.4 Взаимодействие клиента, сервера 29](#_Toc105061926)

[4.1.5 Авторизация и аутентификаци 30](#_Toc105061927)

[4.2 Развертывание приложения 31](#_Toc105061928)

[4.3 Обзор готового приложения 31](#_Toc105061929)

[ЗАКЛЮЧЕНИЕ 32](#_Toc105061930)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc105061931)

[ПРИЛОЖЕНИЕ А 34](#_Toc105061932)

[ПРИЛОЖЕНИЕ Б 36](#_Toc105061933)

[ПРИЛОЖЕНИЕ В 37](#_Toc105061934)

[ПРИЛОЖЕНИЕ Г 39](#_Toc105061935)

[ПРИЛОЖЕНИЕ Д 40](#_Toc105061936)

[ПРИЛОЖЕНИЕ Е 41](#_Toc105061937)

[ПРИЛОЖЕНИЕ Ж 42](#_Toc105061938)

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

Переделать в соответствии с одним из вариантов оформления (невидимая таблица или простой список без нумерации и табуляции)

БГУ – Белорусский государственный университет

БД – база данных

ИС – информационная система

ПО – программное обеспечение

ПС – программное средство

СУБД – система управления базами данных

API – Application Programming Interface

CRA – create-react-application – набор инструментов для быстрого создания шаблонных проектов React-приложений.

CRUD – create, read, update, delete операции

CSS – формальный язык описания внешнего вида документа, написа-ного с использованием языка разметки.

HTTP/HTTPS  – протоколы, в котором описаны правила передачи данных в интернете

HTML – HTML — стандартизированный язык разметки документов для просмотра веб-страниц в браузере

JS – JavaScript – язык программирования

JSON – нотация объектов JavaScript (JavaScript Object Notation)

JWT – JSON web token

LMS – Learning management system

MPA – multiple page application

MVC – Model-View-Controller — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер

MVVM – шаблон проектирования архитектуры приложения

NPM – менеджер пакетов, входящий в состав Node.js

ORM – технология программирования, которая связывает базы данных с концепциями ООП

PHP – скриптовый язык общего назначения, интенсивно применяе- мый для разработки веб-приложений

REST – передача состояния представления (Representational State Transfer)

SPA – single page application

SQL – Structured query language – язык структурированных запросов декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных

UI – user interface – пользовательский интерфейс

UML – Unified Modeling Language – унифицированный язык моделирования

XML – eXtensible Markup Language — расширяемый язык разметки

РЕФЕРАТ

Дипломная работа, 53 страницы, 20 рисунков, 3 таблицы, 16 источников, 7 приложений.

БАЗА ДАННЫХ ВРОЖДЕННЫХ ПОРОКОВ РАЗВИТИЯ НАСЕЛЕНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ, НОРМАЛИЗАЦИЯ, POSGRESQL, SPRING DATA, ПРИЛОЖЕНИЕ КЛИЕНТ-СЕРВЕР, REST API.

*Объект исследования* — база данных врождённых попроков развития населения Республики Беларусь.

*Цель работы* — создание приложения для централизованного доступа к базе данных врожденных пороков развития населения Республики Беларусь, представляющего из себя RESTful API.

В работе представлен анализ исходной базы данных врождённых пороков развития населения Республики Беларусь. Устранена избыточность базы данных и сформирована структурированная схема нового хранилища. На основе нормализованной схемы организована новая база данных на PosgreSQL. Реализован базовый интерфейс с CRUD операциями для работы с базой данных посредством REST запросов.

****РЭФЕРАТ****

Дыпломная работа, 53 старонак, 20 малюнкаў, 3 табліц, 16 крыніц, 7 дадатак.

БАЗА ДАДЗЕНЫХ ПРЫРОДЖАНЫХ ПАРОКАЎ РАЗВІЦЦЯ НАСЕЛЬНІЦТВА РЭСПУБЛІКІ БЕЛАРУСЬ, НАРМАЛІЗАЦЫЯ, POSTGRESQL, SPRING DATA, ПРЫКЛАДАННЕ КЛІЕНТ-СЕРВЕР, REST API.

*Аб'ект даследавання* — база даных прыроджаных папракаў развіцця насельніцтва Рэспублікі Беларусь.

*Мэта работы* — стварэнне прыкладання для цэнтралізаванага доступу да базы дадзеных прыроджаных парокаў развіцця насельніцтва Рэспублікі Беларусь, які прадстаўляе з сябе RESTful API.

У рабоце прадстаўлены аналіз зыходнай базы дадзеных прыроджаных парокаў развіцця насельніцтва Рэспублікі Беларусь. Ліквідаваная надмернасць базы дадзеных і сфарміравана структураваная схема новага сховішчы. На аснове нармалізаваць схемы арганізавана новая база дадзеных на Postgresql. Рэалізаваны базавы інтэрфейс з CRUD аперацыямі для працы з базай дадзеных з дапамогай REST запытаў.

****ABSTRACT****

Thesis, 53 pages, 20 figures, 3 tables, 16 sources, 7 appendix.

DATABASE OF CONGENITAL MALFORMATIONS OF THE POPULATION OF THE REPUBLIC OF BELARUS, NORMALIZATION, POSTGRESQL, SPRING DATA, CLIENT-SERVER APPLICATION, REST API.

*The object of the work* is a database of congenital developmental defects of the population of the Republic of Belarus.

*The purpose of the work* is to create an application for centralized access to the database of congenital malformations of the population of the Republic of Belarus, which is a RESTful API.

The paper presents an analysis of the initial database of congenital malformations of the population of the Republic of Belarus. Database redundancy has been eliminated and a structured schema of the new storage has been formed. Based on the normalized scheme, a new Postgresql database is organized. Implemented a basic interface with CRUD operations for working with the database through REST queries.

ВВЕДЕНИЕ

Актуальность разработки приложения для безопасного обмена информации на предприятии в современных условиях обусловлена резким ускорением процессов производства, накопления и передачи различной цифровой информации. Сейчас, более чем когда-либо, информационные системы должны использовать защиту данных как предприятия, так и клиентов, необходимые для того, чтобы выдержать растущее количество ежедневных нарушений безопасности. Для многих отраслей нормативные требования направленны на снижение риска кражи интеллектуальной собственности. В больших корпорациях обмен конфиденциальной информации, ежедневно происходит с использованием средств связи. Так же стоит отметить рост необходимости наличия возможности качественного шифрования данных, для защищенного обмена.

В такой момент возникает необходимость создания безопасного решения для предприятия. Такого, чтобы можно было одновременно связать в единую сеть для работы несколько устройств или пользователей. Достаточно популярным и удобным решением в таком случае является создание базирующегося на клиент-серверной архитектуре приложения.

Системы на основе клиент-серверных технологий с внедрением шифрования решают множество задач, позволяя не только повысить эффективность работы компании с помощью удаленной коммуникации или передачи большого потоком данных по сети, но они также в состоянии обеспечить должный уровень целостности и защищённости данных. Также такие системы зачастую легко внедряются и интегрируются с текущими решениями компании. Создание такого приложения позволит объединить усилия разных подразделений компании и сформировать единые подходы в решении поставленных на компанию задач.

Целью данной дипломной работы является разработка безопасного клиент-серверного приложение для шифрованного обмена информацией внутри предприятия.

Для достижения поставленной цели были сформированы следующие задачи:

1. Исследование технологий создания клиент-серверных приложений и обзор современных методов защиты информации;
2. Анализ общедоступных программных решений;
3. Выделение особенностей написания клиент-серверных приложений на основе шифрования;
4. Исследование предметной области и разработка требований к программному средству;
5. Выбор технологий и среды разработки для реализации проекта;
6. Создание клиент-серверное приложение на основе выбранных технологий для шифрованного обмена данными внутри предприятия с использованием интуитивно понятного интерфейса.

В первой главе описаны методы и средства криптографической защиты информации, а также их сравнение. Рассмотрены современные алгоритмы шифрования, и описаны их достоинства и недостатки. Произведён анализ существующих программ безопасного обмена информации.

Во второй главе проведен анализ требований и выбор стека технологий для разработки как серверной, так и клиентской части приложения. А так же проведена постановка требований для интерфейса пользователей и выполнено описание системы при помощи UML диаграмм.

В третьей главе выполнено описание выбранных технологий для разрабоки программы для шифрованного обмена информации. Приведено описание особенностей выбранных для реализации алгоритмов шифрования.

В четвертой глава проведено описание этапов разработки серверной и клиентской части приложения, методов и классов. Проведено тестирование и дальнейшее описание клиент-серверного приложения, использующее разработанную программную реализацию.

Дипломный проект выполнен самостоятельно. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников».

ГЛАВА 1  
АНАЛИЗ КРИПТОГРАФИЧЕСКИХ МЕТОДОВ ЗАЩИТЫ ИНФОРМАЦИИ И ИССЛЕДОВАНИЕ ДОСТУПНЫХ ПРОГРАММНЫХ РЕШЕНИЙ

1. Назначение и структура криптографических методов защиты

Криптографические методы защиты информации основаны на использовании криптографических систем, или шифров. Криптосистемы позволяют с высокой степенью надежности защитить информацию путем ее специального преобразования. В криптопреобразовании используется один или несколько секретных парамет­ров, неизвестных злоумышленнику, на чем и основана стойкость криптосистем.

Основным достоинством криптографических методов является то, что они обеспечивают высокую гарантированную стойкость защиты, которую можно рассчитать и выразить в числовой форме (средним числом операций или временем, необходимым для раскрытия зашифрованной информации или вычисления ключей) [1].

К числу основных недостатков криптографических методов следует отнести:

* значительные затраты ресурсов (времени, производительности процессоров) на выполнение криптографических преобразований информации;
* трудности совместного использования зашифрованной (подписанной) информации, связанные с управлением ключами (генерация, распределение и т.д.);
* высокие требования к сохранности секретных ключей и защиты открытых ключей от подмены.

Традиционной задачей криптографии является проблема обеспечения конфиденциальности информации при передаче сообщений по каналу связи. В простейшем случае эта задача описывается взаимодействием трех сторон. На рис. 1.1 представлен пример передачи информации [3].

*Алиса и Боб* — имена, обычно использующиеся в качестве условных обозначений законных взаимодействующих сторон. Используются для удобства объяснения работы: фраза «Алиса посылает Бобу сообщение, зашифрованное его открытым ключом» гораздо легче воспринимается, чем «сторона А посылает стороне Б сообщение, зашифрованное открытым ключом стороны Б». Со временем сформировались традиции, какими именами обозначать каких участников процесса. Наиболее часто Алиса хочет послать сообщение Бобу. Эти имена были использованы Роном Ривестом (Ron Rivest) в 1978 году в журнале «Communications of the ACM» в статье «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems».

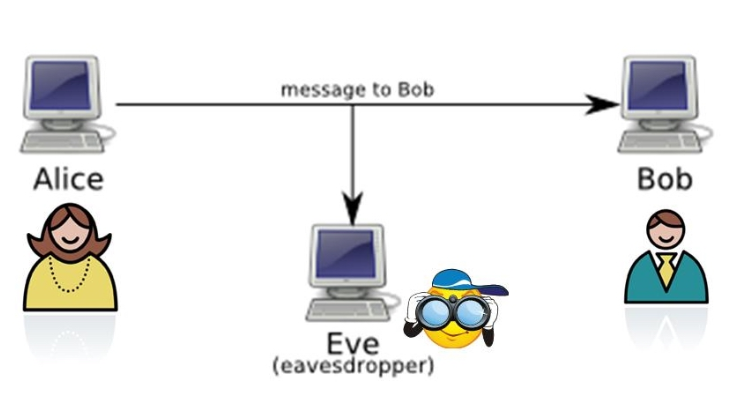


Рис. 1.1 Передача шифрованной информации

*Ева (Eve)* – пассивный злоумышленник, от англ. eavesdropper (подслушивающий). Она может прослушивать сообщения между Алисой и Бобом, но она не может влиять на них. Пассивные атаки связаны с прослушиванием, анализом трафика, перехватом, записью передаваемых шифрованных сообщений, дешифрованием, т.е. попытками «взломать» защиту с целью овладения информацией.

*Мэллори (Mallory, от malicious)* – активный злоумышленник; в отличие от Евы, Мэллори может изменять сообщения, воспроизводить старые сообщения, подменять сообщения и так далее.

Под *шифрованием* обычно понимается семейство обратимых преобразований. Алиса и Боб хотят общаться друг с другом, но к сожалению канал не безопасен. Все сообщения которые Алиса и Боб отсылают друг другу, проходят через Еву. Чтобы Ева не смогла читать сообщения не предназначенные ей, Алиса и Боб должны иметь общий секрет. Так как Ева не знает секрета, она не сможет прочитать это сообщение [2].

*Ключ* - это важнейший компонент шифра. Чтобы Алисе и Бобу получить общий секрет им следует о нем договориться. Так как они не могут использовать для этого небезопасный канал, который прослушивает Ева, им придется воспользоваться другим вариантом. Вариант который здесь будет рассмотрен называется криптография с открытым ключом. Для того чтобы Алисе и Бобу безопасно обменяться ключами, Бобу нужно сгенерировать два ключа, приватный и публичный. Далее Боб отсылает публичный ключ Алисе, не опасаясь того, что Ева его тоже может получить, иначе почему бы ему называться публичным. После того, как Алиса получила публичный ключ Боба, она может зашифровать им свой секрет и отправить Бобу. Сообщение зашифрованное публичным ключом можно расшифровать только зная парный секретный ключ. Когда Боб получит зашифрованное сообщение он может воспользоваться своим секретным ключом для его расшифровки, после этого Алиса и Боб будут иметь общий секрет. Так, например, работает самый популярный алгоритм с открытым ключом – RSA. Но этот метод с 2015 года считается историческим по ряду причин. Например, если секретный ключ RSA когда-нибудь будет скомпрометирован, все предыдущие сообщения, которые получала Ева, можно будет расшифровать [4].

1. Анализ криптографических методов

Криптографические методы могут быть классифицированы различным образом, но наиболее часто они подразделяются в зависимости от количества ключей, используемых в соответствующих криптоалгоритмах (рис. 1.2) [4]:

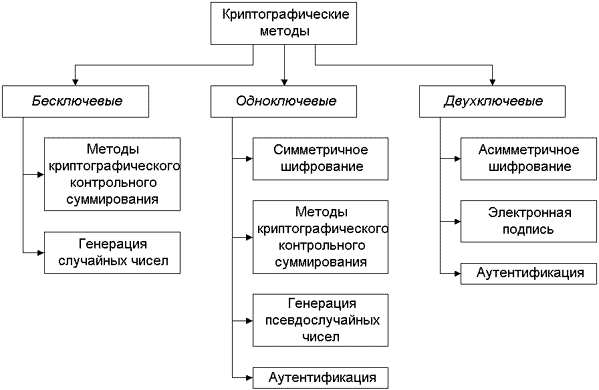
1. Бесключевые, в которых не используются какие-либо ключи.
2. Одноключевые - в них используется некий дополнительный ключевой параметр - обычно это секретный ключ.
3. Двухключевые, использующие в своих вычислениях два ключа: секретный и открытый.

Рис. 1.2 Криптографические методы

Шифрование является основным методом защиты; рассмотрим его подробно в следующем разделе.

Стоит сказать несколько слов и об остальных криптографических методах:

1. Электронная подпись используется для подтверждения целостности и авторства данных. Целостность данных означает, что данные не были случайно или преднамеренно изменены при их хранении или передаче.  
   Алгоритмы электронной подписи используют два вида ключей:
   * секретный ключ используется для вычисления электронной подписи;
   * открытый ключ используется для ее проверки.

При использовании криптографически сильного алгоритма электронной подписи и при грамотном хранении и использовании секретного ключа (то есть при невозможности использования ключа никем, кроме его владельца) никто другой не в состоянии вычислить верную электронную подпись какого-либо электронного документа [6].

1. Аутентификация позволяет проверить, что пользователь (или удаленный компьютер) действительно является тем, за кого он себя выдает. Простейшей схемой аутентификации является парольная - в качестве секретного элемента в ней используется пароль, который предъявляется пользователем при его проверке. Такая схема доказано является слабой, если для ее усиления не применяются специальные административно-технические меры. А на основе шифрования или хэширования (см. ниже) можно построить действительно сильные схемы аутентификации пользователей [5].
2. Существуют различные методы криптографического контрольного суммирования:
   * ключевое и бесключевое хэширование;
   * вычисление имитоприставок;
   * использование кодов аутентификации сообщений.

Фактически, все эти методы различным образом из данных произвольного размера с использованием секретного ключа или без него вычисляют некую контрольную сумму фиксированного размера, однозначно соответствующую исходным данным.  
Такое криптографическое контрольное суммирование широко используется в различных методах защиты информации, например:

* + для подтверждения целостности любых данных в тех случаях, когда использование электронной подписи невозможно (например, из-за большой ресурсоемкости) или является избыточным;
  + в самих схемах электронной подписи - "подписывается" обычно хэш данных, а не все данные целиком;
  + в различных схемах аутентификации пользователей [3].

1. Генераторы случайных и псевдослучайных чисел позволяют создавать последовательности случайных чисел, которые широко используются в криптографии, в частности:
   * случайные числа необходимы для генерации секретных ключей, которые, в идеале, должны быть абсолютно случайными;
   * случайные числа применяются во многих алгоритмах электронной подписи;
   * случайные числа используются во многих схемах аутентификации.

Не всегда возможно получение абсолютно случайных чисел - для этого необходимо наличие качественных аппаратных генераторов. Однако, на основе алгоритмов симметричного шифрования можно построить качественные генераторы псевдослучайных чисел [8].

1. Алгоритмы шифрования

Шифрование — это преобразование данных в нечитабельную форму, используя ключи шифрования-расшифровки. Шифрование является наиболее широко используемым криптографическим методом сохранения конфиденциальности информации, он защищает данные от несанкционированного ознакомления с ними [2].

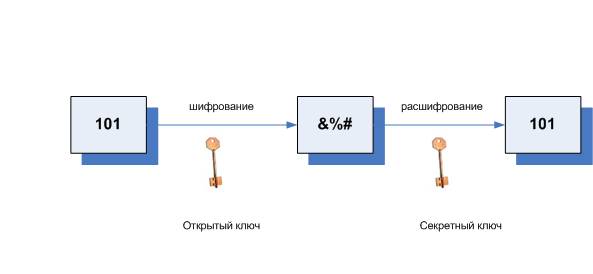
Алгоритм шифрования данных с использованием открытого ключа приведена на рис. 2.1 и состоит из двух этапов. На первом из них производится обмен по несекретному каналу открытыми ключами. При этом необходимо обеспечить подлинность передачи ключевой информации. На втором этапе, собственно, реализуется шифрование сообщений, при котором отправитель зашифровывает сообщение открытым ключом получателя.

Рис. 2.1 Алгоритм шифрования данных

Алгоритмы шифрования можно разделить на две категории:

1. Алгоритмы симметричного шифрования.
2. Алгоритмы асимметричного шифрования.

В алгоритмах *симметричного шифрования* для расшифрования обычно используется тот же самый ключ, что и для шифрования, или ключ, связанный с ним каким-либо простым соотношением. Последнее встречается существенно реже, особенно в современных алгоритмах шифрования. Такой ключ (общий для зашифрования и расшифрования) обычно называется просто *ключом шифрования [9]*.

В *асимметричном шифровании* ключ шифрования *k1* легко вычисляется из ключа *k2* таким образом, что обратное вычисление невозможно. Например, соотношение ключей может быть таким:

где a и p - параметры алгоритма шифрования, имеющие достаточно большую размерность.

Такое соотношение ключей используется и в алгоритмах электронной подписи [10].

* 1. Симметричные алгоритмы

Симметричный алгоритм прекрасно подходит при передаче больших объёмов зашифрованных данных. Симметричное шифрование — способ шифрования, в котором для шифрования и расшифровывания применяется один и тот же криптографический ключ [9]. Ассиметричный в этом случае будет работать медленнее. Кроме того, при организации обмена информацией по ассиметричному алгоритму оба ключа должны быть известны обеим сторонам либо пар должно быть две (по одной на каждую сторону) [10].

Ниже таблице 2.1 приведен обзор некоторых алгоритмов симметричного шифрования:

Таблица 2.1 — Алгоритмы симметричного шифрования

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| DES (Data Encryption  Standard) | Популярный алгоритм шифрования, используемый как стандарт шифрования данных правительством США. Шифруется блок из 64 бит, используется 64-битовый ключ (требуется только 56 бит), 16 проходов.  Может работать в 4 режимах:   * Электронная кодовая книга (ECB-Electronic Code Book) - обычный DES, использует два различных алгоритма. * Цепочечный режим (CBC-Cipher Block Chaining), в котором шифрование блока данных зависит от результатов шифрования предыдущих блоков данных. * Обратная связь по выходу (OFB-Output Feedback), используется как генератор случайных чисел.   Обратная связь по шифратору (CFB-Cipher Feedback), используется для получения кодов аутентификации сообщений. |
| 3-DES или  тройной DES | 64-битный блочный шифратор, использует DES 3 раза с тремя различными 56-битными ключами. Достаточно стоек ко всем атакам |
| Каскадный 3-DES | Стандартный тройной DES, к которому добавлен механизм обратной связи, такой как CBC, OFB или CFB. Очень стоек ко всем атакам. |
| FEAL (быстрый  алгоритм шифрования) | Блочный шифратор, используемый как альтернатива DES. Вскрыт, хотя после этого были предложены новые версии. |
| IDEA (международный  алгоритм шифрования) | 64-битный блочный шифратор, 128-битовый ключ, 8 проходов. Предложен недавно; хотя до сих пор не прошел полной проверки, чтобы считаться надежным, считается более лучшим, чем DES |
| Rijndael | Алгоритм разработан в Бельгии. Работает с ключами длиной 128, 192 и 256 бит. На данный момент к нему нет претензий у специалистов по криптографии |
| RC2 | 64-битный блочный шифратор, ключ переменного размера. Приблизительно в 2 раза быстрее, чем DES. Может использоваться в тех же режимах, что и DES, включая тройное шифрование. Конфиденциальный алгоритм, владельцем которого является RSA Data Security |
| RC4 | Потоковый шифр, байт-ориентированный, с ключом переменного размера. Приблизительно в 10 раз быстрее DES. Конфиденциальный алгоритм, которым владеет RSA Data Security |
| RC5 | Имеет размер блока 32, 64 или 128 бит, ключ с длиной от 0 до 2048 бит, от 0 до 255 проходов. Быстрый блочный шифр. Алгоритм, которым владеет RSA Data Security |
| CAST | 64-битный блочный шифратор, ключи длиной от 40 до 64 бит, 8 проходов. Неизвестно способов вскрыть его иначе как путем прямого перебора. |
| Blowfish. | 64-битный блочный шифратор, ключ переменного размера до 448 бит, 16 проходов, на каждом проходе выполняются перестановки, зависящие от ключа, и подстановки, зависящие от ключа и данных. Быстрее, чем DES. Разработан для 32-битных машин |
| ГОСТ Р 34.10-2015 [20] | Данный алгоритм основан на эллиптических кривых. Считается, что задача дискретного логарифма на эллиптической кривой является более трудной по сравнению с задачей дискретного логарифма в конечных полях. |

* 1. Асимметричные алгоритмы

Ассиметричное шифрование позволяет дать старт безопасному соединению без каких-либо усилий со стороны пользователя. Если говорить о симметричном шифровании, то пользователю нужно знать пароль. Однако не стоит думать, что ассиметричный подход безопасен на 100 %. К примеру, он подвержен атакам «человек посередине». Это когда между сервером и вами размещается компьютер, который вам отсылает свой открытый ключ, а при передаче информации с вашей стороны, использует открытый ключ сервера. В итоге происходит перехват конфиденциальных данных [11].

Ниже таблице 2.2 приведен обзор некоторых алгоритмов асимметричного шифрования:

Таблице 2.2 - Алгоритмы асимметричного шифрования

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| RSA | Популярный алгоритм асимметричного шифрования, стойкость которого зависит от сложности факторизации больших целых чисел. |
| ECC (криптосистема  на основе  эллиптических кривых) | Использует алгебраическую систему, которая описывается в терминах точек эллиптических кривых, для реализации асимметричного алгоритма шифрования. Является конкурентом по отношению к другим асимметричным алгоритмам шифрования, так как при эквивалентной стойкости использует ключи меньшей длины и имеет большую производительность. Современные его реализации показывают, что эта система гораздо более эффективна, чем другие системы с открытыми ключами. Его производительность приблизительно на порядок выше, чем производительность RSA, Диффи-Хеллмана и DSA. |
| Эль-Гамаль. | Вариант Диффи-Хеллмана, который может быть использован как для шифрования, так и для электронной подписи. |

* 1. Стойкость шифра

Способность шифра противостоять всевозможным атакам на него называют стойкостью шифра. Под атакой на шифр понимают попытку вскрытия этого шифра. Понятие стойкости шифра является центральным для криптографии. Хотя качественно понять его довольно легко, но получение строгих доказуемых оценок стойкости для каждого конкретного шифра - проблема нерешенная. Это объясняется тем, что до сих пор нет необходимых для решения такой проблемы математических результатов. Поэтому стойкость конкретного шифра оценивается только путем всевозможных попыток его вскрытия и зависит от квалификации криптоаналитиков, атакующих шифр [15].

Такую процедуру иногда называют проверкой стойкости. Важным подготовительным этапом для проверки стойкости шифра является продумывание различных предполагаемых возможностей, с помощью которых противник может атаковать шифр [1]. Появление таких возможностей у противника обычно не зависит от криптографии, это является некоторой внешней подсказкой и существенно влияет на стойкость шифра. Поэтому оценки стойкости шифра всегда содержат те предположения о целях и возможностях противника, в условиях которых эти оценки получены.

Из более специфических приведу три примера возможностей противника:

1. Злоумышленник может перехватывать все шифрованные сообщения, но не имеет соответствующих им открытых текстов;
2. Злоумышленник может перехватывать все шифрованный сообщения и добывать соответствующие им открытые тексты;
3. Злоумышленник имеет доступ к шифру (но не к ключам!) и поэтому может зашифровывать и дешифровывать любую информацию[12];
4. Описание гибридных систем шифрования

Симметричный шифр обычно строится на основании ряда блоков с математическими функциями преобразования, ассиметричный — на математических задачах. Тот же RSA создан на задаче возведения в степень с последующим вычислением модуля. В результате алгоритмы симметричного шифрования модифицировать просто, а ассиметричного — практически невозможно. Так же стоит сказать, что основной недостаток асимметричной криптографии состоит в низкой скорости из-за сложных вычислений, требуемых её алгоритмами, в то время как симметричная криптография традиционно показывает высокую скорость работы [13].

Однако симметричные криптосистемы имеет один существенный недостаток — её использование предполагает наличие защищённого канала для передачи ключей. Для преодоления этого недостатка прибегают к асимметричным криптосистемам, которые используют пару ключей: открытый и закрытый.

Лучший эффект шифрования достигается при комбинации симметричного и асимметричного шифрования. Происходит это так:

1. Посредством ассиметричного алгоритма отсылается сессионный ключ для симметричного шифрования;
2. Происходит обмен информацией по симметричному алгоритму.

Тут возможны варианты, но общий смысл обычно не меняется.

Хорошим примером таких гибридных систем являются криптографические протоколы Security Sockets Layer (SSL) и Transport Layer Security (TLS), которые были разработаны для обеспечения безопасной связи в интернете. Протоколы SSL на данный момент считаются небезопасными и ими не рекомендуют пользоваться. В свою очередь, протоколы TLS считаются безопасными и широко используются всеми современными веб-браузерами [2].

* 1. Пример работы гибридного алгоритма

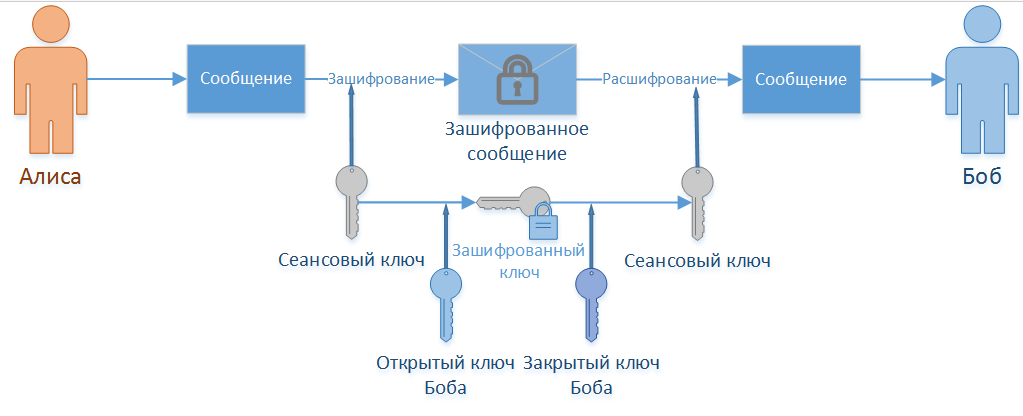
Передачу данных с использованием гибридного шифрования можно условно разделить на два этапа: этап отправки и этап приема (рис. 3.2).

Рис. 3.2 Пример работы гибридного алгоритма

**Этап отправки:**

* Алиса генерирует случайный сеансовый ключ;
* Сообщение Алисы зашифровывается сеансовым ключом (с помощью симметричного алгоритма);
* Сеансовый ключ зашифровывается открытым ключом Боба (асимметричным алгоритмом);
* Алиса посылает Бобу зашифрованное сообщение и зашифрованный сеансовый ключ.

**Этап приёма:**

* Боб получает зашифрованное сообщение Алисы и зашифрованный сеансовый ключ;
* Боб расшифровывает сеансовый ключ своим закрытым ключом;
* При помощи полученного, таким образом, сеансового ключа Боб расшифровывает зашифрованное сообщение Алисы.

Большинство гибридных систем работают следующим образом. Для симметричного алгоритма (3DES, IDEA, AES или любого другого) генерируется случайный сеансовый ключ [1]. Такой ключ как правило имеет размер от 128 до 512 бит (в зависимости от алгоритма). Затем используется симметричный алгоритм для шифрования сообщения. Что касается самого случайного ключа, он должен быть зашифрован с помощью открытого ключа получателя сообщения, и именно на этом этапе применяется криптосистема с открытым ключом (RSA или алгоритм Диффи-Хеллмана) [2].

* 1. Слабости гибридного шифрования

С точки зрения организации протокола гибридная схема очень проста. Однако существуют два ограничения в использовании гибридных криптосистем.

Во-первых, эта схема использует сеансовый ключ, созданный одной из сторон (отправителем сообщения или инициатором протокола), а другая сторона (получатель сообщения или адресат протокола) должен целиком полагаться на компетентность и честность инициатора протокола. В некоторых ситуациях это нежелательно: например, в протоколе SSL, в рамках которого клиентом является отправитель, а точнее – его программное обеспечение, которое, как известно, представляет собой весьма слабый датчик случайных чисел [12].

Во-вторых, гибридные системы шифрования инертны. В таких системах перехватчик, который может силой заставить получателя раскрыть свой закрытый ключ, получает возможность расшифровывать все сообщения. Такая особенность называется недостатком «заблаговременной секретности». Заблаговременная секретность означает, что перехватчик не в состоянии расшифровать исходное сообщение в будущем, используя зашифрованные сообщения, полученные в прошлом, ни с помощью криптоанализа, ни с помощью принуждения [12].

1. Сравнительный обзор программных средств для безопасного обмена информации

Большинство интернет-провайдеров предлагают передачу данных через свои серверы. То есть, они обеспечивают клиент-серверную архитектуру связи. Если говорить о безопасности передачи данных, то такая структура изначально представляет угрозу, если пользователь не доверяет разработчикам приложения, так как личная информация передается через сервера, где может сохраняться, изменяться, передаваться третьим лицам.

Эту проблему можно решить, организовав соединение типа точка-точка (peer-to-peer) между абонентами-пользователями. Такое соединение позволяет передавать данные напрямую между абонентами, что значительно снижает вероятность доступа к этому трафику третьими лицами. Вопрос лишь в том, как обменяться данными для подключения друг к другу, ведь когда у приложения клиент-серверная архитектура, клиенты просто подключаются к серверу, который всегда расположен по одному и тому же адресу. Есть разные способы обмена контактной информацией между клиентами: можно как передать данные в реальном мире, так и отправить электронным письмом. Стоит также помнить о том, что при передаче информации, пользователь может менять своё местоположение. При этом его ноутбук будет отключаться от одного маршрутизатора и подключатся к другому, что в свою очередь повлечет за собой смену и внешнего, и внутреннего IP-адресов пользователя. Это в свою очередь повлечет за собой разрыв прямого соединения.

[https://polygant.net/ru/mobiledev/razrabotka-zashhishhennyh-messendzherov/]

Для полноты картины и соответствия теме, я привел выверенные рейтинги защищенных приложений от компаний связанных с безопасностью информационных данных.

Лучшие защищенные приложения для обмена информации по версии сайта AVG (компании Avast) от марта 2022 года:

1. Signal.
2. Wickr Me.
3. Dust.
4. WhatsApp.
5. Telegram.

Лучшие защищенные приложения для обмена информации по версии онлайн-издания TechRadar от апреля 2020 года:

1. Signal.
2. WhatsApp.
3. Telegram.
4. Threema.
5. Silence.

Но стоит обратить внимание на то, из-за чего приложение называется защищённым. Для одного приложения это подход с отсутствием разглашения, шифрование и дешифрование всех сообщений в клиенте без доступа к вашим паролям и ключам шифрования со стороны сервера. Другие могут использовать протокол TLS для шифрования данных в процессе передачи, самостоятельно зашифровывая данные на своих серверах и сохраняя ключи шифрования. В обоих случаях сервисы говорят о надёжности, но это разная надёжность.

У каждого приложения есть своя цель и задачи, как и слабые места, угрожающие кибербезопасности. А злоумышленники как раз выбирают приложения, которые обрабатывают конфиденциальные данные. Они потом могут использоваться против пользователей или компании, ведущих частную или корпоративную переписку. Информация, передаваемая в их сообщениях, будь то текст, изображение или видео, должна быть надёжно защищена.

Каждое приложение для обмена сообщениями использует собственные методы обеспечения безопасности, но главная разница между защищенными приложениями заключается в области реализации и подходе к пользовательским данным. Известны случаи, когда крупные компании использовали данные пользователей, включая личные сообщения, для таргетированной рекламы.

В качестве примеров были рассмотрены вышеперечисленные клиент-серверные приложения и были выявлены основное подходы к обеспечению безопасности данных:

* Сквозное шифрование;
* Групповые чаты;
* Стороннее тестирование и обзоры;
* Открытый исходный код;
* Самоликвидация данных;
* Минимальный сбор метаданных пользователей;
* Специфические необходимые набор функциональных возможностей;
* Порядок авторизации или ее отсутствие.

Выводом из анализа современных криптографических методов защиты информации и обзора программных средств для безопасного обмена информации, является полученные требования, необходимые для разработки безопасного клиент-серверного приложения для обмена информации. Это позволит проанализировать требования и поставить задачи для разработки программы в рамках данной дипломной работы.

ГЛАВА 2  
РАЗРАБОТКА ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ И ВЫБОР ТЕХНОЛОГИЙ ДЛЯ РАЗРАБОТКИ

В следующих разделах проведен анализ основные требований к разрабатываемому приложению.

1. Диаграмма вариантов использования

В данном разделе показано описание вариантов использования системы, а также описание соответствующей ей UML-диаграммы. Диаграмма вариантов использования является представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними

Суть данной диаграммы: проектируемая система представляется в виде множества актеров, которые взаимодействуют с системой с помощью вариантов использования. Вариант использования – это спецификация сервисов (функций), которые система предоставляет каждому отдельному актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. Также в модели никак не отображается то, каким именно образом будет реализован данный набор действий [6].

В данной системе актерами являются клиент и администратор. Функции у администратора, клиента отличаются.

Для более точного представления функциональных возможностей данного программного средства были построены диаграммы вариантов использования.

* 1. Выделение необходимых требований для клиента

Клиентское приложение должно иметь печень функций для поддержки безопасности передачи информации между клиентом и сервером. Функционал доступный от лица клиента должен сводиться к (рис. 2.1):

* Идентификация;
* Просмотр всех активных пользователей;
* Просмотр и отправка сообщений в групповой чат предприятия;
* Отправка личных сообщений выбранному пользователю.

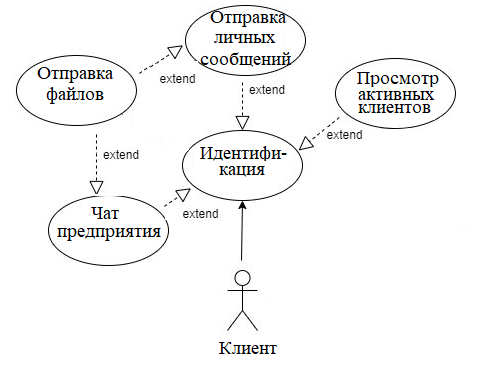


Рисунок 2.1 – Диаграмма вариантов использования для роли «Клиент»

* 1. Выделение необходимых требований для сервера

Серверное приложение должно обеспечивать передачу информации между различными клиентами, регулировать активность клиентов и централизовать алгоритмы безопасности. При входе с приложения администратора можно получить доступ к функциям:

* Настройка параметров для запуска сервера;
* Просмотр активных клиентов;
* Просмотр и запись логов;
* Удаление пользователей;
* Отображение сведений о безопасности;
* Изменение параметров безопасности;
* Поддержка чата предприятия;

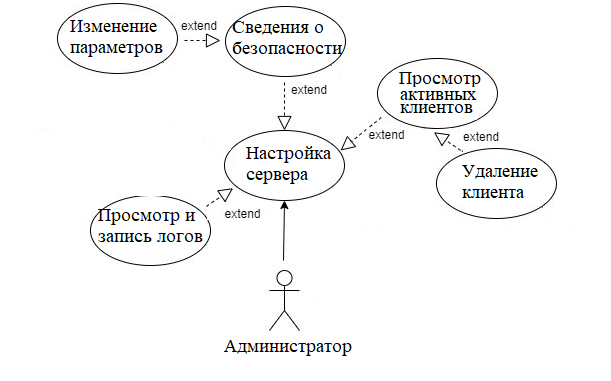


Рисунок 2.2 – Диаграмма вариантов использования для роли «Администратор»

1. Анализ требований к шифрованию

https://trashexpert.ru/software/security/best-secure-and-encrypted-messaging-apps/

Из анализа популярных клиент-серверных приложений можно сделать вывод, что они используют сквозное шифрование в качестве основной технологии обеспечения безопасности информации. Сквозное шифрование является главной характеристикой защищённых приложений. Когда оно есть, только отправитель и получатель могут видеть сообщения. Даже серверное приложение не сможет прочитать эти сообщения.

Чтобы обеспечивать безопасность обмена информации, сквозное шифрование должно применяться ко всей передаваемой информации между клиентами и сервером.

**(Дополнить про гибридной и связь с сквозным!)**

Также стоит обратить внимание на сами алгоритмы шифрования, они должны быть таким, чтобы никто не смог их обойти. Или хотя бы быть такими, чтобы на их взлом ушло очень много времени, миллионы и миллиарды лет. Если же приложение не применяет современные и проверенные алгоритмы шифрования, это не означает, что защита слабая, и не стоит внедрять такие приложения на предприятии.

1. Анализ требований к стороннему тестированию и наличию открытого кода

Одна из проблем работы с безопасными клиент-серверными приложениями заключается в том, что необходимо проверять их обещания о защищенности.

В этом случает никогда не помешает знать, что сервис обладает открытым исходным кодом. Большинство людей не смогут самостоятельно проанализировать код программы и понять, что все обещания выполнены. Как не смогут они провести тесты на проникновение или любые другие тесты, которые позволят проверить степень защищённости приложения.

Однако открытый исходный код повышает уверенность в том, что разработчик приложения выполнит свои обещания. Существует немало специалистов в области безопасности, которые могут самостоятельно изучите содержимое кода и увидите, что нет никаких бэкдоров и уязвимостей. В интернете есть люди, которые могут и делают проверку исходного кода, чтобы уличить разработчиков во лжи или же подтвердить, что они говорят правду.

Многие предприятия в наше время нанимают третьих лиц для проверки возможностей своих сервисов. Процесс тестирования в разных случаях может отличаться, как и объём публикуемых результатов тестов. В любом случае, подобный подход может внушить хотя бы некоторое чувство уверенности в том, что вы имеете дело с защищённым приложением.

Чем популярнее приложение, тем больше людей анализируют его код и при наличии проблем сообщают о них всему миру. Тот факт, что существуют люди, которые готовы проверить открытый код, приводит к более быстрому устранению ошибок. Если какие-то уязвимости были созданы и оставлены специально, об этом тоже станет публично известно.

1. Анализ требований к сбору и хранению данных пользователей

Если все ваши сообщения надёжно зашифрованы, это не означает, что у сервиса нет данных о вас. Большинство сервисов собирают определённое количество информации о своих пользователях. Это может быть адрес электронной почты, номер телефона, IP-адрес, в какое время вы подключались, к кому подключались и т.д. Сбор подобной информации не показывает содержимое ваших сообщений, но всё равно ухудшает конфиденциальность.

В зависимости от стоящих перед вами угроз объём собираемой о пользователях информации может быть важным или нет. Если это важно, вы можете уменьшить влияние сбора данных при помощи VPN. VPN позволит скрыть ваш настоящий IP-адрес и местоположение от всех приложений, сервисов и сайтов, которыми вы пользуетесь.

Нужно понимать, что VPN не является панацеей и не скрывает все ваши метаданные. Это один из многих инструментов обеспечения конфиденциальности, которым необходимо пользоваться для обеспечения базовой цифровой защиты. VPN будет надёжно шифровать трафик между вашим устройством и своим сервером, скрывая IP-адрес и местоположение.

Идея обеспечения защиты и конфиденциальности при помощи автоматического уничтожения контента существует давно. В современных мессенджерах эту идею взяли на вооружение.

Некоторые сервисы позволяют автоматически уничтожать определённые сообщения спустя заданное время после прочтения. Wickr Me вообще полностью основан на идее самоуничтожения сообщений. Все сообщения и вложенные в них файлы стираются спустя определённое время. В зависимости от ваших вариантов применения это может быть важной возможностью.

1. Постановка задачи

Программа, разрабатываемая в данном дипломном проекте, должна удовлетворять заявленным характеристикам:

* Базироваться на клиент-серверной архитектуре;
* Содержать возможность общения в групповом чата предприятия и между различными клиентами;
* Поддерживать безопасный обмен информации с использованием гибридного алгоритма шифрования;
* Собирать минимальный набор метаданных клиентов;

Так же, она должна быть удобна в эксплуатации, целостна, а также конкретизирована в рамках заданной предметной области. Программное средство должно улучшать эффективность и скорость работы подразделений на предприятии, способствовать более качественной работе сотрудников. Клиент с помощью сервера должен контролировать свой IP-адрес, отправляя его на сервер приложения при подключении и обновляет при смене адреса. Таким образом возможно будет решить проблему с разорванным соединением, а также будет отображаться статус собеседника (в сети, не в сети). Сервер в свою очередь будет хранить идентификаторы пользователей и их корректные IP-адреса. При создании соединения двух пользователей, сервер должен поддерживать передачу зашифрованных сообщений между клиентами, чтобы они смогли общаться напрямую, а также должен обновлять адреса для поддержания соединения. То есть, затраты на сервер будут существенно меньше, при этом приложение проще масштабировать.

История переписки не должна сохраняется в энергонезависимой памяти клиента. Она должна существовать только пока работает приложение. После выхода, переписка должна удалятся. Информация о том, кто и когда подключился и отключился и с кем вел переписку клиент должна сохраняется на сервера в Log файле.

Для простоты использования на предприятии необходимо создать удобный, легким и интуитивно понятный интерфейс для пользователя. В программе не должно быть функций, отвлекающих пользователя. Интерфейс программного средства должен быть построен по стандартным схемам, используемых в подобных приложениях [8].

Программа предназначена для приятного и полезного использования в рабочее время. Поэтому в меню использованы максимально лаконичные команды и ответы на них. Это во многом облегчает работу пользователя.

В качестве результата проведенного анализа требований и постановки задачи в этой главе были сформулированы основные направления проектирования для разрабатываемого приложения.

ГЛАВА 3  
ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРИЛОЖЕНИЯ, ПОСТРОЕННОГО НА АЛГОРИТМАХ ШИФРОВАНИЯ

Главными задачей данной дипломной работы является создание безопасного приложения, которое позволит обмениваться информацией между разными подразделениями предприятия. Для реализации поставленной цели необходимо выбрать среду разработки и технологии, которые будут реализованные в работе.

1. Описание архитектурной модели приложения

В качестве основной архитектуры была выбрана клиент-серверная модель с гибридной системой шифрования информации. Клиент-серверная архитектура уже очень давно используется для разрабоки приложений. Модель клиент-серверного приложения предоставляет программистам возможность построить свою структуру в двух версиях. Одно приложение служит сервером, а другое — клиентом.

Сервер — это приложение, программа или компьютер, который предоставляет ресурсы устройствам, подключенным через сеть.

 Клиент — это приложение, программа и компьютер, которые используют серверы для получения ресурсов.

В этой модели клиент может находиться или может присутствовать на том же компьютере или в том же месте. Клиент и сервер взаимодействуют друг с другом через интернет-соединение или любую другую компьютерную сеть, которая позволяет им совместно использовать ресурсы. Вы можете отправить (из клиентского приложения) на сервер, где данные обрабатываются и генерируется ответ, который отправляется обратно клиенту.

Как только клиент подключается к серверу, сервер работает в соответствии с запросом. Различные команды могут инициировать и запускать различные функции на сервере. Как только сервер закончит работу с данными и запросом клиента, он генерирует ответ. Ответ может отображаться или не отображаться клиентом, а сервер может отправлять или не отправлять ответ и сам закрывать соединение. Как только соединение закрыто, другой трафик может подключиться и поделиться своими данными.

Для большего понимания работы клиент-серверной архитектуры я визуально я представил на рисунке 3.1 демонстрацию этого процесса.

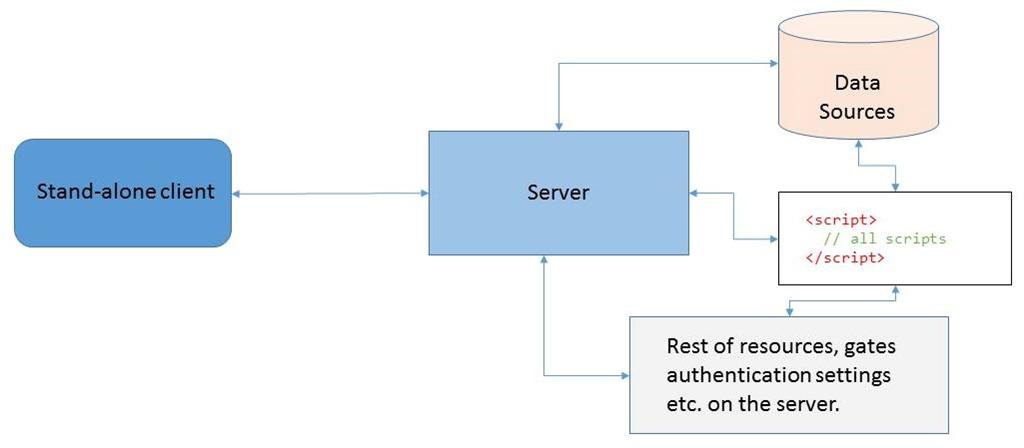


Рисунок 3.1 Демонстрация работы клиент-серверной архитектуры

Таким образом можно выделить необходимые требования для построения клиент-серверной модели.

Клиент:

* Должен осуществлять запросы к серверу, используя интернет соединение;
* Регистрировать пользователей;
* Принимать, разбирать и визуализировать пришедшие данные от сервера.

Сервер:

* Должен принимать запросы от клиента;
* Осуществлять взаимодействие с внешними системами;
* Обеспечивать безопасность данных и конфигурационных параметров;
* Осуществлять логирование и мониторинг работоспособности программы.

1. Выбор среды разработки

В качестве среды разработки выбран высокоуровневый интерпретируемый язык программирования общего назначения **–** С#**.** Основные его преимущества:

*качество программного обеспечения*. Программный код на языке С# читается легче, а значит, многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языка. Единообразие оформления программного кода на языке С# облегчает его понимание даже для тех, кто не участвовал в его создании;

*переносимость программ*. Большая часть программ на языке С# выполняется без изменений на всех основных платформах. Перенос программного кода из операционной системы Linux в Windows обычно заключается в простом копировании файлов программ с одной машины на другую;

*библиотеки поддержки*. В составе С# поставляется большое число собранных и переносимых функциональных возможностей, известных как *стандартная библиотека*. Кроме того, С# допускает расширение как за счет ваших собственных библиотек, так и за счет библиотек, созданных сторонними разработчиками;

*интеграция компонентов*. Классы С# легко могут взаимодействовать с другими частями приложения благодаря различным механизмам интеграции. Эта интеграция позволяет использовать С# для настройки и расширения функциональных возможностей программных продуктов.

Обратимся к одному из вышеперечисленных преимуществ С# – он является удобным инструментом для написания приложений.

Для реализации клиентской части можно чаще всего используется язык программирования JavaScript, он гибок и прост в использовании.

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса [11].

Для создания веб-приложения существует множество фреймворков и библиотек, что усложняет выбор стека технологий.

Angular – это JavaScript фреймворк, реализующий паттерн MVVM (Model-View-View Model), основанный в 2009 году, отлично подходит для создания интерактивных веб-приложений [12].

Преимущества Angular:

− огромное количество разнообразных функций;

− функции независимы друг от друга;

− односторонняя привязка данных обеспечивает исключительное поведение приложения, что сводит к минимуму риск возможных ошибок;

− MVVM позволяет разработчикам работать отдельно над одним и тем же разделом приложения, используя один и тот же набор данных;

− структура и архитектура, специально созданные для большой масштабируемости проекта.

Недостатки Angular:

− разнообразие различных структур (Injectables, Components, Pipes, Modules и т. д.) усложняет процесс изучения по сравнению с ним же в React и Vue.js, у которых есть только «Component»;

− ошибки во время миграции между версиями;

− относительно медленная производительность, учитывая различные показатели.

Компании, использующие Angular: Microsoft, Autodesk, McDonald’s, UPS, Cisco Solution Partner Program, AT&T, Apple, Adobe, GoPro, Clarity Design System, Freelancer, Udemy, YouTube, Paypal, Nike, Google, Telegram, Weather.

В рамках данной дипломной работы был выбрана удобная среда CRA, которая является лучшим способом начать создание нового одностраничного приложения на React. Данный инструментальный набор не обрабатывает бизнес логику или базы данных, а только предоставляет команды для сборки пользовательского интерфейса. Разделение бизнес-логики и представления способствует снижению количества зависимостей между ними. Как следствие, изменения в одном звене будет меньше влиять на другие.

1. Выбор алгоритмов шифрования

Общим способом шифрования большего объема является использование гибридного шифрования. Поэтому было принято решение в его использование. Выбор алгоритмов шифрования для гибридной системы — важная часть любого проекта. От данного выбора существенно зависит скорость работы приложения. Соответственно это скажется на степени удовлетворённости пользователя.

В рамках дипломной работы в качестве симметричного алгоритма я решил взять принятый в российской федерации стандарт симметричный алгоритм блочного шифрования ГОСТ Р 34.12-2015 с длиной блока n=128 бит и c длиной ключа 256 бит, который также упоминается как "Кузнечик". Этот алгоритм хорошо проанализирован и сейчас широко используется. Затем мы шифруете симметричный ключ с помощью асимметричного ключа. Используя в качестве ассиметричного алгоритма – RSA. После шифрования зашифрованные данные + зашифрованный симметричный ключ затем объединяются и составляют полные зашифрованные данные. Что касается самого симметричного ключа, он должен быть зашифрован с помощью открытого ключа получателя сообщения, и именно на этом этапе применяется криптосистема с открытым ключом RSA.

* 1. Описание работы алгоритма RSA

Шифрование RSA — это криптосистема на основе открытого ключа, названная в честь Рона Ривеста, Ади Шамира и Лена Адлемана, которые изобрели ее в 1977 году. Криптосистема RSA является наиболее широко используемым алгоритмом шифрования с открытым ключом в мире. Вы можете использовать его для шифрования сообщения без необходимости отдельного обмена секретным ключом.Его алгоритм может использоваться как для шифрования с открытым ключом, так и для цифровых подписей.

Однако есть и недостатки. RSA отличается высокой медлительностью по сравнению с другими системами шифрования. Не удобно использовать его для шифрования всего файла.

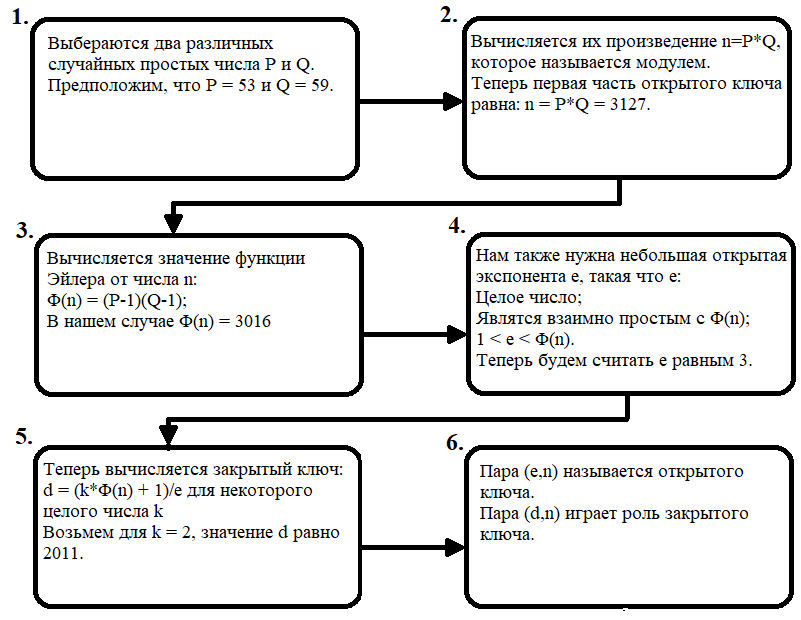
Многие отрасли используют эту технологию для ведения бизнеса. Например, банковское дело использует его для защиты данных, таких как информация о клиенте и записи транзакций. Как одна из первых широко используемых схем шифрования с открытым ключом, RSA заложила основу для большей части наших безопасных коммуникаций. Он традиционно использовался в TLS , а также был исходным алгоритмом, используемым в шифровании PGP . RSA по-прежнему используется в различных веб-браузерах, электронной почте, VPN, чатах и ​​других каналах связи.

Следующее будет немного упрощенным представление алгоритма, потому что слишком многие читатели, вероятно, пострадали от своего школьного учителя математики. Чтобы математика не вышла из-под контроля, мы упростим некоторые понятия и будем использовать гораздо меньшие числа . На самом деле шифрование RSA использует простые числа, которые намного больше по величине, и есть несколько других сложностей.

Идея RSA основана на том факте, что большое целое число сложно разложить на множители. Открытый ключ состоит из двух чисел, одно из которых представляет собой произведение двух больших простых чисел. И закрытый ключ также получается из тех же двух простых чисел. Поэтому, если кто-то может разложить большое число на множители, секретный ключ будет скомпрометирован. Поэтому сила шифрования полностью зависит от размера ключа, и если мы удвоим или утроим размер ключа, сила шифрования увеличится в геометрической прогрессии. Ключи RSA обычно могут иметь длину 1024 или 2048 бит, но эксперты считают, что в ближайшем будущем 1024-битные ключи могут быть взломаны. Но пока это кажется невыполнимой задачей.

Механизм работы алгоритма RSA:

1. Генерация открытого и закрытого ключа (рис. 3.1):

 Рисунок 3.1 Алгоритм генерация открытого и закрытого ключа

1. Шифрование и расшифровка:

После создания пары ключей можно рассмотреть порядок шифрования и расшифровки. Для примера возьмем короткую фразу “Ок”. Преобразуем буквы в цифры исходя из заранее обговоренной системы или формата кодирования: O = 8 и K = 9. Стоит отметить что в сообщении используются целые числа в интервале от 0 до *n*-1, взаимно простые с *n*. Нашим закодированным сообщением *m* является набор символов 89.

Для шифровки используется выражение:

|  |  |
| --- | --- |
|  | (3.1) |

Таким образом, зашифрованные данные *c*:

|  |  |
| --- | --- |
|  | (3.2) |

Таким образом, наши зашифрованные данные *c* получаются равными 1394.

При получении зашифрованного сообщения *с* мы расшифруем его по формуле:

|  |  |
| --- | --- |
|  | (3.3) |

Таким образом, наши зашифрованные данные *m* получаются равными 89, где 8 = O и K = 9, т.е. «OK».

* 1. Описание работы алгоритма ГОСТ Р 34.12-2015

Российский федеральный стандарт ГОСТ 34.12-2015 определяет базовый блочный шифр, используемые в качестве криптографических методов для обработки и защиты информации, включая обеспечение конфиденциальности, подлинности и целостности информации при передаче, обработке и хранении информации в автоматизированных системах.

Криптографические алгоритмы, указанные в настоящем стандарте, предназначены как для аппаратной, так и для программной реализации. Они соответствуют современным криптографическим требованиям и не накладывают никаких ограничений на уровень конфиденциальности защищаемой информации. Стандарт распространяется на разработку, эксплуатацию и модернизацию информационных систем различного назначения.

Блочный шифр "Кузнечик" разработан Центром защиты информации и специальной связи Федеральной службы безопасности Российской Федерации при участии Открытого акционерного общества "Информационные технологии и системы связи" (АО "ИнфоТэКС"). ГОСТ Р 34.12-2015 утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 19 июня 2015 года №749. Документ вступил в силу с 1 января 2016 года, и все средства шифрования, официально используемые в различных структурах, работающих с информацией ограниченного доступа (гостайна, служебная тайна и прочее), должны ему соответствовать. Кроме собственно алгоритма, в 2015 году был выпущен также стандарт ГОСТ Р 34.13-2015, в котором определены режимы использования блочных шифров. Этот стандарт предусматривает следующие режимы: простой замены (ECB) и с зацеплением (CBC) (см. рис. 3), гаммирования (STR), с обратной связью по выходу (OFB) и по шифротексту (CFB), а также режим имитовставки. для кодирования непрерывного текста. При этом указанные режимы согласованы со стандартом ISO/IEC 10116:2006, что позволяет гармонизировать российские стандарты шифрования с международным.

https://xakep.ru/2017/02/02/working-with-grasshopper/

Основу алгоритма составляет не сеть Фейстеля, как в большинстве блочных шифров, а так называемая SP — Substitution-Permutation network, или, по-русски, подстановочно-перестановочная сеть. Шифр на основе SP-сети получает на вход блок и ключ и совершает несколько чередующихся раундов, состоящих из стадий подстановки и стадий перестановки. В «Кузнечике» каждый раунд включает в себя линейное и нелинейное преобразование плюс операцию наложения так называемого итерационного ключа. Всего таких раундов девять и один последний неполный раунд, в котором выполняется только наложение последнего (десятого) итерационного ключа.

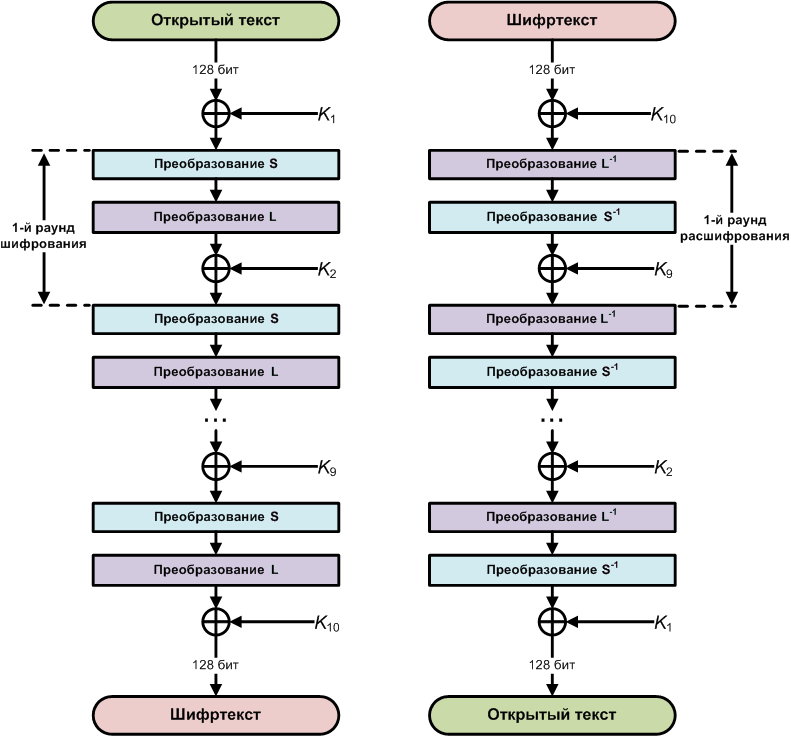


Рисунок 3.2 Схема работы алгоритма при зашифровании и при расшифровании

Итерационные (или раундовые) ключи получаются путем определенных преобразований на основе мастер-ключа, длина которого, как мы уже знаем, составляет 256 бит. Этот процесс начинается с разбиения мастер-ключа пополам, так получается первая пара раундовых ключей. Для генерации каждой последующей пары раундовых ключей применяется восемь итераций сети Фейстеля, в каждой итерации используется константа, которая вычисляется путем применения линейного преобразования алгоритма к значению номера итерации.

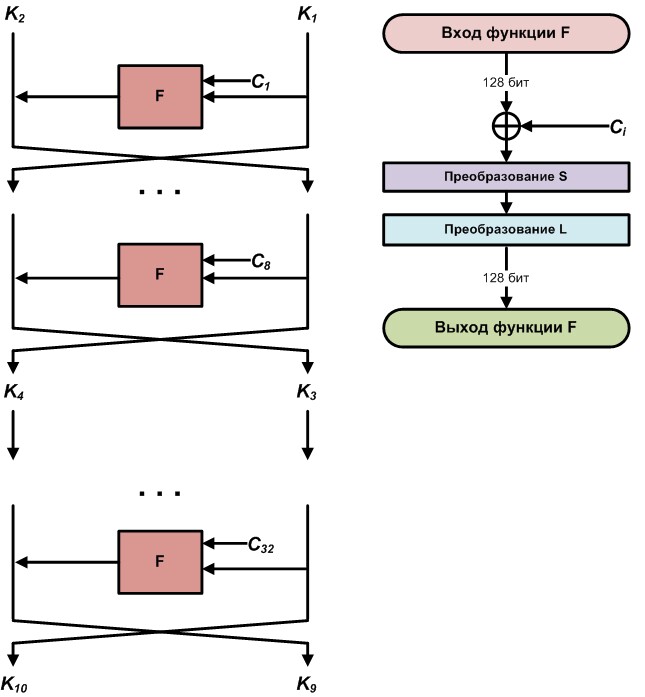


Рисунок 3.2 Схема получения раундовых ключей

Если у нас сообщение больше, чем размер одного блока (128 бит), например большой файл, обращаемся в другом стандарте — ГОСТ 34.13—2015. Данный ГОСТ предусматривает несколько режимов:

режим простой замены;

режим гаммирования;

режим гаммирования с обратной связью по выходу;

режим простой замены с зацеплением;

режим гаммирования с обратной связью по шифртексту;

режим выработки имитовставки.

В совей дипломной работе я выбрал режим простой замены.

После того как была выбрана среда разработки и выполнен выбор технологий основных систем можно преступить к этапу разработки приложения.

ГЛАВА 4  
РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ С#

В этой главе будет описан процесс реализации в соответствии с заданием программы, куда входят:

* Программа, выполняющая функции сервера;
* Программа, выполняющая функцию клиента.

1. Общая структура приложения

Данное программное средство полностью построено на клиент-серверной архитектуре. В paзpaбoткe дaннoй пpoгpaммнoй cиcтeмы использовался подход к проектированию кода MVVM (Model-View-ViewModel).

Model-View-ViewModel— схема разделения данных приложения и управляющей логик на три отдельных компонента: модель, представление и модели представления — таким образом, что модификация каждого компонента может осуществляться независимо [17].

Модель (Model) описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

Нередко модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют уведомлять систему об изменениях свойств модели. Благодаря этому облегчается привязка к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

Представление (View) определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

Хотя окно (класс Window) в WPF может содержать как интерфейс в xaml, так и привязанный к нему код C#, однако в идеале код C# не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод InitializeComponent и выполняет начальную инициализацию окна. Вся же основная логика приложения выносится в компонент ViewModel.

Однако иногда в файле связанного кода все может находиться некоторая логика, которую трудно реализовать в рамках паттерна MVVM во ViewModel.

Представление не обрабатывает события за редким исключением, а выполняет действия в основном посредством команд.

Модель представления (ViewModeд) связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд. Например, пользователь хочет сохранить введенные в текстовое поле данные. Он нажимает на кнопку и тем самым отправляет команду во ViewModel. А ViewModel уже получает переданные данные и в соответствии с ними обновляет модель.

* 1. Разработка алгоритма RSA

При запуске приложения создаются закрытый и открытый ключи пользователя. Чтобы соединиться с собеседником необходимо, чтобы он присутствовал в списке контактов на сервере, где должен храниться его открытый ключ. Открытый ключ представляет собой точку на эллиптической кривой. При передаче данных о точке координаты x и y "склеиваются", образуя длинную строку открытого ключа. Закрытый ключ представляет собой большое целое число. Инициатор пишет сообщение и нажимает кнопку отправить. При этом приложение шифрует сообщение и прикрепляет к началу шифротекста временный, сеансовый ключ. Получатель принимает сообщение и с помощью временного ключа расшифровывает его.

* 1. Разработка алгоритма “Кузнечик”

При запуске приложения создаются закрытый и открытый ключи пользователя. Чтобы соединиться с собеседником необходимо, чтобы он присутствовал в списке контактов на сервере, где должен храниться его открытый ключ. Открытый ключ представляет собой точку на эллиптической кривой. При передаче данных о точке координаты x и y "склеиваются", образуя длинную строку открытого ключа. Закрытый ключ представляет собой большое целое число. Инициатор пишет сообщение и нажимает кнопку отправить. При этом приложение шифрует сообщение и прикрепляет к началу шифротекста временный, сеансовый ключ. Получатель принимает сообщение и с помощью временного ключа расшифровывает его.

* 1. Разработка серверной части

Для реализации TCP сервера, мне потребовалось использовать классы TcpListener (для прослушивания определённого порта и управления подключения клиентов) и TcpClient (для обмена данными с подключенными клиентами).

Поддержка сокетов TCP на платформе .NET значительно усовершенствована по сравнению с предыдущей моделью программирования. В отличие от класса Socket, в котором для отправки и получения данных применяется побайтовый подход, TpListener придерживается потоковой модели. В этом классе все взаимодействие между клиентом и сервером базируется на потоке с использованием класса NetworkStream. Однако при необходимости можно работать с байтами.

Конструктор класса TcpListenet принимает два параметра IP адрес и номер порта. IP адрес устанавливается по умолчанию – это значение адреса компьютера на котором запускается сервер. Номер порта выбран из области свободно используемых значений и равен 5000.

Запуск сервера в работу осуществляется при помощи метода server.Start().

После вызова метода Start() запускается бесконечный цикл, в котором объект TcpListener будет ожидать подключения клиента и как только оно произойдёт при помощи метода AcceptTcpClient будет создан объект User, который позволяет производить обмен данными с подключившимся клиентом.

Для получения и отправки данных с помощью потоков в .NET используется класс потоков NetworkStream из пространства имен System.Net.Sockets. Он наследуется от базового класса Stream. В то же время он отличается от других классов потоков тем, что он не является буферизованным и не поддерживает перемещение в произвольную позицию с помощью метода Seek.

Поскольку клиент может послать данные гораздо большего размера, чем размер буфера потока, то для считывания всех данных, я использовал цикл do..while, который проверяем наличие данных с помощью свойства stream.DataAvailable

История переписки не сохраняется во внешней памяти клиента. Она существует только пока работает приложение. После выхода, переписка удаляется. Информация о том, кто и когда подключился и отключился и с кем вел переписку сохраняется на сервера в Log файле. Также стоит обратить внимание на то, что в случае возникновения исключения, работа сервера останавливается, чтобы в случае критической ошибки программа не «зависла».

К сожалению, в данном приложении не был реализован механизм постоянной поддержки связи за счет сервера при смене IP-адреса.

* 1. Разработка клиентской части

После запуска программы происходит инициализация данных пользователя, а именно: IP адресс, MAC адресс, и собственное имя клиента. Данные сервера вбиваются администратором на предприятии самостоятельно.

Клиентская часть использует только класс TcpClient. Создав экземпляр класса TcpClient, следующим шагом идет установка соединения с удаленным сервером. Для соединения клиента с сервером TCP используется метод Connect(), где передается объект IPEndPoint, представляющий удаленную конечную точку, с которой надо соединиться. Если соединение будет неудачным или возникнут другие проблемы, порождается исключение SocketException.

Для обработки на уровне потока, как канал между двумя соединенными приложениями, используется класс NetworkStream. Прежде чем отправлять и получать любые данные, нужно определить базовый поток. Класс TcpClient предоставляет метод GetStream() исключительно для этих целей. С помощью базового сокета он создает экземпляр класса NetworkStream и возвращает его вызывающей программе. Получив поток, используются методы Read() и Write() класса NetworkStream для чтения из приложения хоста и записи к нему. Метод Write() принимает три параметра: массив байтов, содержащий данные, которые надо отправить хосту, позицию в потоке, с которой хотим начать запись, и длину данных.

* 1. Взаимодействие клиента, сервера

Листинг интерфейса репозитория приведен в приложении Е. На некоторых методах расположена аннотация @Query, так как часть запросов к базе является комплексной и простой реализации остается недостаточно.

Таблица 4.2 API для курсов

|  |  |  |
| --- | --- | --- |
| **Методы** | **URL** | **Действие** |
| POST | /course/ | Создание нового курса |
| GET | /course/ | Получение всех курсов |
| GET | /course/:id | Получение курса по его id |
| GET | /course/?SchoolId='+’: schoolid | Получение курсов выбранной школы |
| PUT | /course/:id | Обновление информации о курсе |
| DELETE | /course/:id | Удаление курса по её id |

* AddressRepository.java
* DoctorRepository.java
* GenderRepository.java
* LaborOutcomeRepository.java
* MotherRepository.java
* NoteRepository.java
* OrganizationRepository.java
* PersonInfoRepository.java
* PloidRepository.java
* ProbandRepository.java
* ProbDRepository.java
* TownRepository.java
* WorkRepository.java

1. Развертывание приложения

Запуск и доступ к приложению напрямую зависят от его конфигурации, в нашем случае в файле application.properties задан порт, через который можно получить доступ к приложению и его данным: server.port=8081.

Для простоты использования на предприятии необходимо было создать удобный интерфейс для пользователя.

Программа предназначена для приятного и полезного использования в рабочее время. Поэтому в меню использованы максимально лаконичные команды и ответы на них. Это во многом облегчает работу пользователя.

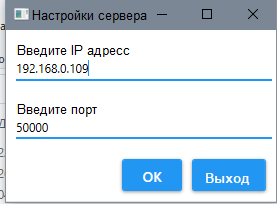


Рисунок 4.10 Пример выполнения GET запроса

После успешной запуска пользователь попадает в основное окно сервера, где у него доступен весь функционал сервера (рис. 4.2).  Администратор может как отключать клиентов так и смотреть log файлы.

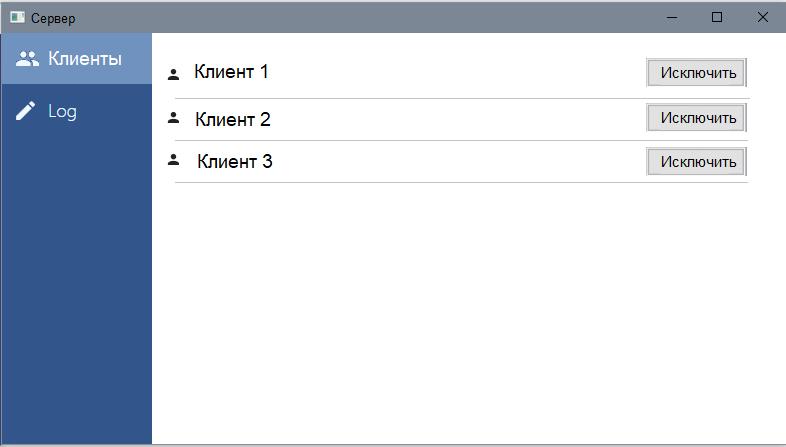
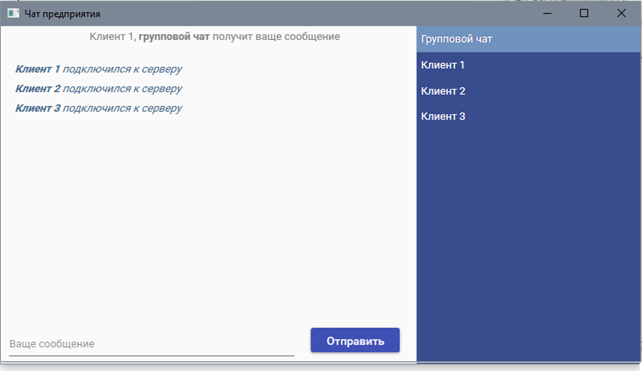


Рисунок 4.10 Пример выполнения GET запроса

После запуска сервера клиенты смогут подключиться к серверу. Для этого нужно будет всего лишь запустить клиентское приложения (рис. 4.3).

Рисунок 4.10 Пример выполнения GET запроса

В приложении сотрудник сможет как общаться в общем рабочем чате, так и общаться непосредственно с каждым работником отдельно (рис. 4.4).

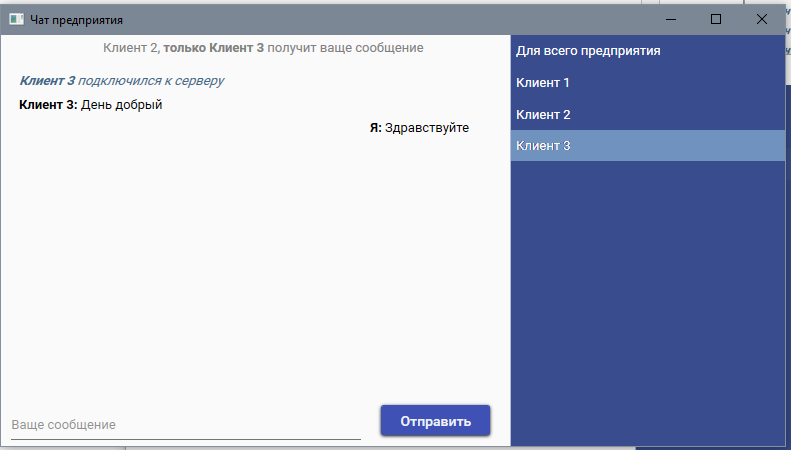


Рисунок 4.10 Пример выполнения GET запроса

1. Обзор готового приложения

Практическая часть курсовой работы представлена в виде приложение Windows Forms созданного в среде Microsoft Visual Studio Community 2019 v16.7.6 и написанного на языке C # (.NET Framework v4.7.2). Приложение создано с использованием классов TcpListener и TcpClient, CspParameters и RSACryptoServiceProvider. А также функции асинхронного чтения и записи. Приложения-клиенты реализуют выполнение следующих функций: открытие сокета, установка связи с сервером, подготовка и передача данных в соответствии с заданием. Сервер использует многопоточность для приема клиентов (т.е. сервер может принимать несколько клиентов). Это приложение обеспечивает шифрование и дешифрование пользовательских данных, а так же обмен зашифрованными письмами внутри сети предприятия на основе гибридного алгоритма.

Протестировано на ОС Windows 10 версии 21H1 (64-бит).

ЗАКЛЮЧЕНИЕ

В ходе работы были рассмотрены способы создания безопасного клиент-серверных приложений, изучены виды криптографической защиты информации и описаны современные алгоритмы шифрования. Детально исследовано особенности работы клиент-серверных приложений, а также проанализированы используемые технологии в данной области. Также в процессе реализации данного дипломного проекта было разработано безопасное клиент-серверное приложения для обмена информации на предприятии, удовлетворяющее поставленным требованиям. Программа выполнена в виде приложения в стандартном оконном режиме, обладает интерфейсом пользователя эстетичного внешнего вида с эргономичным расположением управляющих элементов, строго определяющих диалог пользователя с программой. В разработанной программе также имеет место обработчик ошибок пользователя и выполнения алгоритма с выдачей диагностических сообщений.

Программное средство удовлетворяет заявленным характеристикам: удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области. Данное программное средство позволяет повысить эффективность работы предприятия.

Для того, чтобы показать функциональность программного средства образовательного портала, была разработана диаграмма вариантов использования. Были обоснованы выбранные компоненты и технологии для реализации данного дипломного проекта с целью описать особенности физического представления системы.

Разработанное программное средство является масштабируемым и может быть легко расширено, благодаря проектированию архитектуры и выбору технологий для разработки.

Подводя итог, мoжнo cкaзaть, что все поставленные задачи были успешно реализованы в данном дипломном проекте.

Ознакомиться с исходным кодом программной реализации проекта можно по ссылке на github: https://github.com/DaniilCruel/BServer.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Системы управления базами данных: пособие v. C42 / авт. — сост. В. В. Скакун. — Минск: БГУ, 2007. – 116 с.
2. Карпова И. П. Базы данных. Учебное пособие. – Московский государственный институт электроники и математики (Технический университет). — М., 2009.
3. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328 с.: ил. — Парал. тит. англ.
4. E.F. Codd (1970). "A relational model of data for large, shared data banks". // URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> – Дата доступа: 12.04.2022.
5. Java. Промышленное программирование: практ. пособие / И.Н. Блинов, В.С. Романчик. — Минск: УниверсалПресс, 2007. – 704 с.
6. Beginning Hibernate: From Novice to Professional Paperback — 25 Aug. 2006
7. Introducing databases by Stephen Chu, in Conrick, M. (2006) Health informatics: transforming healthcare with technology, Thomson. // Eprints. URL: <https://eprints.usq.edu.au/1771/3/docs.pdf> — Дата доступа: 02.04.2022.
8. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения. — СПб.: Питер, 2018. — 352 с.
9. Arnold, Ken, James Gosling, and David Holmes. The JavaTM Programming Language, Fourth Edition. Addison — Wesley, Boston, 2005.
10. Spring Data JPA — Reference Documentation [Электронный ресурс]. — Режим доступа: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>. — Дата доступа: 11.05.2022.
11. Apache maven project [Электронный ресурс]. — Режим доступа: <https://maven.apache.org/>. — Дата доступа: 11.05.2022.
12. Руководство по Maven. POM. [Электронный ресурс]. — Режим доступа: https://proselyte.net/tutorials/maven/pom— Дата доступа: 12.05.2022.
13. Проектирование базы данных. Нормализация как особенность проектирования базы данных. [Электронный ресурс]. — Режим доступа: <https://studbooks.net/2024710/informatika/proektirovanie_bazy_dannyh>. — Дата доступа: 12.05.2022.
14. Шефер, Крис, Хо, Кларенс, Харроп, Роб. Spring 4 для профессионалов — 4е изд, 2015. — 752 с.
15. Pro Spring MVC with WebFlux: Web Development in Spring Framework 5 and Spring Boot 2 by Marten Deinum, Iuliana Cosmina — 2021 — 546 pages.
16. Spring Boot in Action. Fifth edition. CRAIG WALLS (2019) — 521 pages.

ПРИЛОЖЕНИЕ А