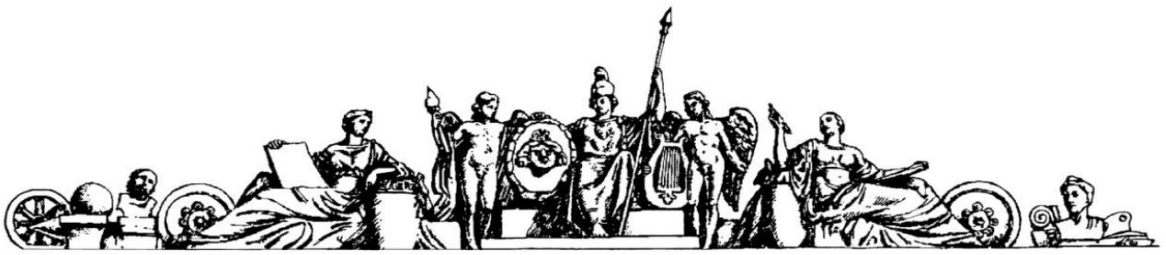

Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра : Информационная безопасность (ИУ-8)

Алгоритмические языки программирования

Курсовая работа

Тема: Корпоративный P2P чат

Руководитель: Бородин А.А
Выполнил : Уханов А.В.
Группа ИУ8-31

г. Москва, 2017

Содержание

Цель

Система предназначена для защищенного обмена информацией между двумя клиентами посредством шифрования трафика.

Основные определения

P2P сети - это компьютерные сети, основанные на равноправии участников, в которых каждый узел может одновременно выступать как в роли клиента ,так и в роли сервера.

Гибридная P2P сеть - это сеть, в которой существует сервер, используемый для координации работы, поиска или предоставления информации о существующих машинах этой сети.

End-to-end – это система, в рамках которой, зашифрованная информация передается от устройства к устройству напрямую, без посредников, так как правила закрытого ключа не позволяют расшифровать информацию никому, кроме её получателя.

Пир – равноправный участник (пользователь) одноранговой сети.

Qt – кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++.

JSON – текстовый формат обмена данными, основанный на JavaScript.

SQLite – компактная встраиваемая реляционная база данных.

Сокет – абстрактный объект, представляющий конечную точку соединения.

Прокси – промежуточный сервер, выполняющий роль посредника между пользователем и целевым сервером.

Введение

Сегодня трудно найти человека, который не пользуется интернет-мессенджерами. Один только WhatsApp установлен на сотнях миллионов устройств по всему миру и пропускает через себя в общей сложности десятки миллиардов сообщений в день. А ведь еще есть Skype, Viber, Messenger от Facebook, ВКонтакте, Telegram и пр. Впрочем, вместе с ростом популярности сервисов обмена сообщениями все чаще поднимается вопрос конфиденциальности переписки. Несмотря на то, что люди ежедневно оставляют в интернете большое количество информации о себе, чаще всего им нужно пообщаться с собеседником даже без намека на возможное появление вольных и невольных свидетелей.

Приложение позволит пользователям общаться с уверенностью в том, что их переписка не будет доступна третьим лицам. Эта проблема очень актуальна и важна в настоящее время, данный проект решает её и поможет пользователям не переживать о безопасности. Чат планируется как корпоративное средство общения - он обеспечит безопасность в важной деловой переписке.

Созданный проект будет прост в использовании, а также предоставит защищенный обмен информацией. Он будет полезен компаниям в которых недопустимо использование публичных систем (например, Skype), в связи с ограниченным доступом к Интернету или корпоративными требованиями к безопасности для предотвращения утечки информации (к примеру, для того же Skype, в мае 2011 года

Microsoft запатентовала технологию «законного вмешательства» в работу VoIP для правоохранительных органов)

Требования к проекту

- Защищенный обмен информацией;
- Выработка общего ключа по алгоритму Диффи-Хеллмана
- Шифрование информации осуществляется по алгоритму **ГОСТ** Р 34.12-2015**;**
- Установка канала связи типа P2P;
- Клиент ПО разрабатывается под ОС MacOS ;

Описание работы системы

Общая информация о системе

Система представляет собой гибридную p2p сеть, в которой пиры также взаимодействуют между собой без каких-либо посредников, но найти друг друга они могут используя некий “Информационный сервер” который только хранит информацию о пирах (их никнеймы и IP-адреса) и может отослать эту информацию любому пиру в этой сети. Информационный сервер никаким образом не участвует в отправке сообщений.

Соединение пиров в сети

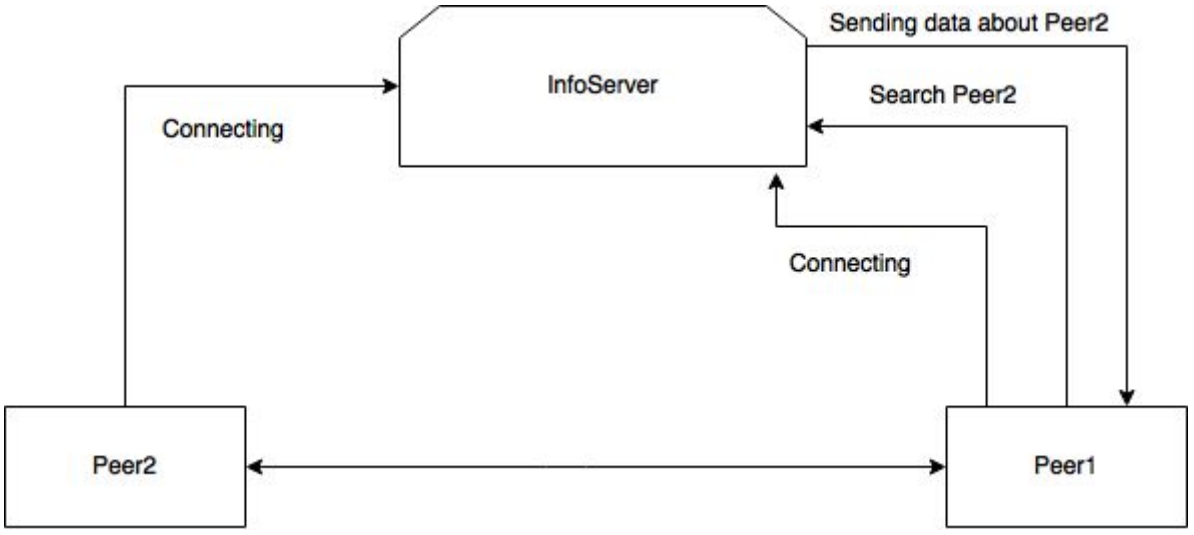

Рисунок 1 Схема подключения одного пира к другому

При входе в сеть, каждый пир автоматически отсылает свои данные информационному серверу. Если мы захотим найти некоторого пользователя в этой сети, зная его никнейм (к примеру, Peer2), мы делаем запрос к информационному серверу (Search Peer2). Инфосервер отсылает данные пиру 1 о пире 2 (Sending data about Peer2), после чего, имея адрес пира 2, пир 1 сохраняет его данные у себя(чтобы каждый раз не запрашивать их с сервера) и может написать ему сообщение. При получении сообщения пир 2 так же сохраняет адрес и никнейм пира 1 у себя и может ему ответить и писать в дальнейшем. Если же искомый пользователь не найден - сервер просто отправит ошибку с соответствующим текстом пиру 1.

Информационный сервер

Как уже говорилось выше, он служит только для поиска одного пира другим и хранения достоверной информации о пирах. Сервер, при входе пира в сеть, сохраняет его адрес и имя у себя (как было сказано выше, пир автоматически отсылает эти данные серверу при подключении к сети). Данные хранятся в базе данных.

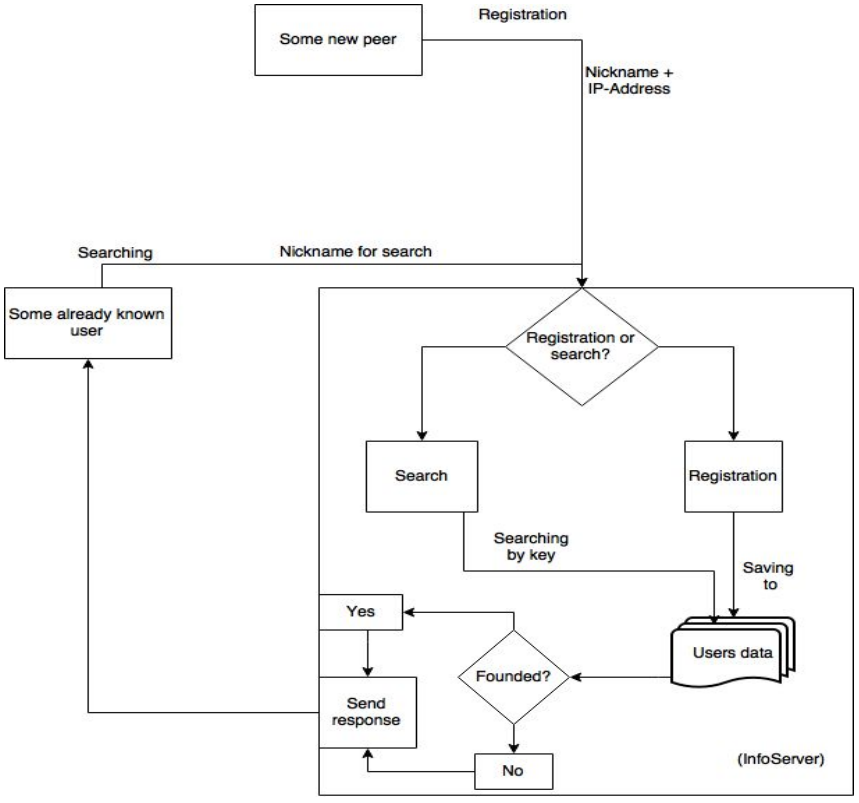

Рисунок 2 Схема обработки запросов от пиров сервером и отправка им ответа

Если пир ищет другого пира - сервер отправит ответ “ищущему” пиру зависящий от того найден ли искомый им пользователь (отправит адрес пользователя), или же нет (ошибка: пользователь не найден).

Инфосервер состоит из класса ServerWindow, который представляет собой:

1. Слот onStartingClicked(), слушающую сокет этого сервера, и подключающей слот ConnectClient() при обнаружении нового соединения к серверу.
2. Слот ConnectClient(), обрабатывает новые подключения, создает сокет для подключившегося клиента и передает управление в слот ListeningClient().
3. Слот ListeningClient() слушает соединение конкретного клиента с сервером и отправляет ему при необходимости информацию об искомом пире, либо сообщение об ошибке.
4. Функция AddNewUser - добавляет пользователя в базу данных.
5. Функция SearchUser - выполняет поиск в базе данных. Возвращает информацию о пользователе (никнейм, адрес, порт).
6. Функция SendAllUsers - отправляет пиру список всех пиров в сети.
7. Функция Resolver - служит для распознавания типа запроса по флагу. Возвращает число - код запроса.
8. Слот on_Stopping_clicked() останавливает сервер по нажатию кнопки Stop

Клиент p2p

При входе в клиентское приложение пользователю будет предложено выбрать никнейм и статус пользователя (публичный или приватный) после чего, его данные (статус, никнейм, адрес) автоматически отправятся на информационный сервер. Пользователь может запросить у сервера информацию о интересующем его пире, после чего клиентское приложение пользователя автоматически инициализирует новый сокет полученным адресом. Данная информация сохранится у этого пользователя это сделано для того, чтобы не требовалось все время отправлять запрос поиска одного и того же пира на сервер в объекте класса Peer, который создан для хранения информации о пире (сеансовый

ключ, имя пира, сокет, инициализированный адресом этого пира). После сохранения искомого пира пользователь отправляет некий запрос (ConnectRequest) другому пиру в котором он сообщает адрес своего “слушающего” сокета. Также он отсылает пиру свой публичный ключ, таким образом пиры обмениваются ключами, после чего генерируют один - сеансовый ключ. Для обмена ключами используется алгоритм Диффи-Хеллмана.[5] Этот запрос помечается особым флагом, поэтому клиент адресата распознает эту информацию и сохранит её у себя. Отправление этого запроса (ConnectRequest) необходимо вследствие того, что сообщение адресату будет отправляться с некоторого сокета (SendSocket) который клиент не будет слушать. Слушающий сокет у клиента только один, а отправляющих столько же, сколько и “знакомых” ему пиров. Соответственно, производительность клиента будет выше, если ему не придется слушать много адресов, а слушать только один, конкретный, на который и будут приходить все сообщения. Все “услышанные” данные передаются в функцию (Resolver), определяющую вид данных по особым флагам (запрос на соединение (от другого пира), запрос на обмен ключами, ответ от сервера и т.д.).

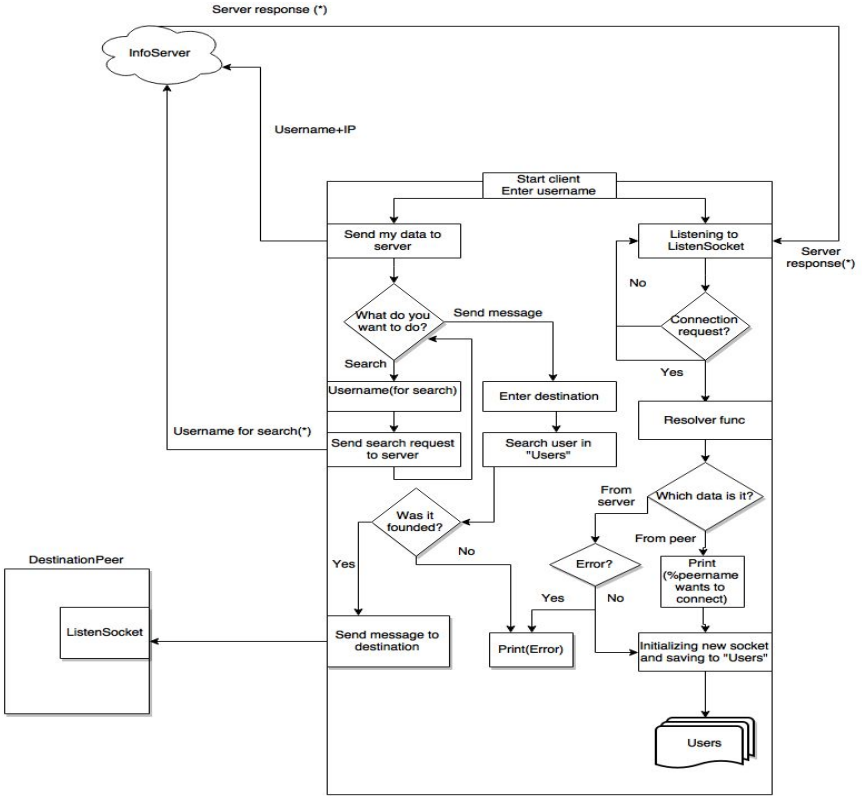

*Рисунок 3 Визуализация работы клиента (прим. * - визуализация механизма поиска пира с помощью сервера)*

Клиент состоит из нескольких взаимодействующих друг с другом классов:

1. Класс ClientWindow - основной интерфейс клиента. Содержит в себе все необходимые для взаимодействия пиров функции и переменные
2. Класс Peer служащий только для хранения информации о знакомых пирах. Содержит публичные поля, из-за чего не требуется get-функций.
3. Класс IssueCreator предоставляющий интерфейс для создания Github Issues.
4. Класс Kuznyechik предоставляющий интерфейс для шифрования и дешифровки сообщений по заданному ключу. Описание методов класса есть в официальной документации, ссылка на которую приведена в источниках.
5. Класс MyCrypto предоставляющий интерфейс для работы с блоками байт, которые требуются для класса Kuznyechik. Описание методов класса есть в официальной документации, ссылка на которую приведена в источниках.

Рассмотрим отдельно основные методы классов:

- ClientWindow
    - onRead - Слот, прослушивающий сокет. Получив сообщение, он направляет его в функцию Resolver для определения типа информации. После ответа от Resolver функция определяет что делать с данными (вывести если это сообщение, сохранить пользователя если это коннект-запрос, обменяться ключами и т.п.)
    - ConnDetector - Слот, обнаруживающий входящие соединения.
    - on_SearchLine_returnPressed - Отправляет на сервер запрос о получении адреса некоего пользователя (имя искомого пользователя передается как аргумент) . Также функция добавляет нового пользователя с полученными от сервера данными в структуру для хранения знакомых пиров (Peers).
    - on_pushButon_clicked - Слот, запускающий IssueCreator по нажатию соответствующей кнопки.

- Resolver - Функция для определения типа данных. На вход подается аргумент в виде строки, тип которой определяется по флагу. Прочитав флаг, Resolver возвращает целое значение (код), говорящее о типе полученных данных.
- SendMessageToPeer - Функция для отправки сообщения пиру. Входной параметр - имя адресата.
- ConnectToPeer - Функция отправляющая запрос на соединение другому пиру. После соединения оба пира автоматически добавят друг друга в список знакомых пиров.
- Encrypt/Decrypt - Функции для шифрования и дешифровки сообщений по ключу соответственно. Подробную документацию по ним можно найти на сайте разработчика. Ссылка приведена в источниках.
- GenKeyParams - Функция для генерации параметров ключей (Prime, Generator, PrivNumb, PublicNumb) и самой пары ключей. Подробнее об этих параметрах можно узнать из описания алгоритма Диффи-Хеллмана, а также из официальной документации к библиотеке CryptoPP. Ссылка приведена в источниках
- SearchPeerByName - Функция для поиска пира в структуре, хранящейся в классе.

- Peer
  - Конструктор, принимающий имя пира и указатель на его сокет.
  - Конструктор, принимающий имя пира, указатель на его сокет и сеансовый ключ.
  - SetSessionKey - Set-функция для инициализации сеансового ключа
  -
- IssueCreator
  - Конструктор, принимающий выделенный текст. Этот текст заносится в окно Description.
  - Read/Write-TokenToFile - Функции для чтения/записи в Config-файл токена.

    - ParseToken - Функция для парсинга полученных данных от Github. Возвращает токен.
    - GetGithubToken - Функция отправки запроса на Github для получения токена.
    - on_Send_clicked - Слот, отвечающий за отправку запроса на Github с целью создания Issue.
- Peers - Структура для хранения данных о знакомых пирах. Данная структура является вектором, элементы которого - объекты класса Peer.

Выбор технологий

Выбор языка программирования

- Python**.** Код на Python достаточно прост и понятен. За счёт простоты кода, дальнейшее сопровождение программ, написанных на Python, становится легче и приятнее по сравнению с Java или C++. Из минусов - скорость работы языка. Он значительно медленнее C/C++, Java. Также, в данном проекте динамическая типизация языка будет скорее вредна из-за обилия используемых в проекте типов данных.
- Swift**.** Достаточно молодой язык. Отсутствие необходимых библиотек, а также стабильности. Также большой проблемой сейчас является отсутствие поддержки рефакторинга со стороны Xcode. Скорость работы тоже проигрывает C++, но в меньшей степени чем Python.
- C++**.** На языке C++ разрабатывают программы для самых различных платформ и систем. Возможность работы на низком уровне с памятью, адресами, портами. Возможность создания обобщенных алгоритмов для разных типов данных, их специализация, и вычисления на этапе компиляции, используя шаблоны. Для языка написано множество библиотек, фреймворков, позволяющих удобно создавать GUI. Скорость его работы выше аналогов. Все вышеописанное позволяет сделать однозначный выбор в пользу C++.

Выбор библиотек и фреймворков

- WxWidgets - Кроссплатформенная библиотека для реализации графического интерфейса. Менее популярна чем Qt, имеет меньше модулей.
- Qt - Помимо удобного создания интерфейса предоставляет большое количество модулей для работы с JSON, SQL, сокетами, системными процессами, имеет свои аналоги стандартных контейнеров, предоставляет удобное взаимодействие с пользователем на уровне сигналов и слотов. Также Qt имеет хорошо структурированную и подробную документацию. Выбор пал на именно на Qt вследствие вышеперечисленных плюсов.

Также, необходимо выбрать библиотеки, реализующие шифрование по алгоритму “Кузнечик” и генерацию пары ключей для каждого пользователя.

- В качестве библиотеки для “Кузнечика” была выбрана реализация этого алгоритма Сергеем Конюховым.[1] Реализация оказалась довольно проста в использовании, но пришлось частично модифицировать код проекта для использования этой библиотеки.
- В качестве библиотеки для генерации ключей выбор пал на CryptoPP. CryptoPP является хорошо документированной библиотекой с большим количеством примеров.[2]

Описание технических решений

Шифрование

Реализация алгоритма “Кузнечик” имела один недостаток, функция зашифровки сообщения шифровала только блоки по 16 байт. Если размер блока не был кратен 16 байтам - эта функция генерировала необработанное исключение, из-за чего программа прекращала свою работу.

Было принято решение разбивать сообщение на блоки по 16 байт и каждый шифровать отдельно, складывая новое сообщение из этих

блоков. Если же размер последнего блока был не кратен 16 байтам - блок дополнялся пустыми символами которые никак не влияли на получившийся шифротекст.

Протокол передачи данных

В процессе проектирования взаимодействий между пользователем и между пользователем и сервером возникла необходимость создать свой протокол передачи данных.
В общем виде он выглядит так:

!*Некоторый флаг*! + Данные

За распознавание флагов отвечает функция Resolver (как у сервера, так и у клиента)

Перечень флагов при взаимодействии пользователя и сервера

Для сервера :

- !0! - Входящий запрос на регистрацию от пира
- !S! - Запрос на поиск приватного пира
- !UPD! - Запрос на обновление списка пиров в сети (сервер отправляет актуальные данные о пирах, находящихся в сети запросившему пиру)
- !OFF! - Сообщение пира об уходе из сети

Для пользователя:

- !CNCTD! - Успешное подключение к серверу
- !SMESS! - Сообщение от сервера (к примеру о том, что искомый пользователь не найден)
- !S! - Данные о пирах (либо адрес найденного приватного пира)

Примеры использования:

1) Пользователь входит в сети и отправляет запрос вида:

   !0!*Имя пользователя*, *IP-адрес*, *Порт*

   В ответ на это сервер отправляет ему два сообщения - об успешном соединении и список всех пиров :

   !CNCTD! и **!S!*Имя пира*, *IP-адрес*, *Порт***|

   Данные каждого пира отделены символом “|”.

2) Пользователь хочет найти приватного пира :

!S!*Имя искомого пира*

Сервер ответит в формате:

!S!*Имя найденного пира*, *IP-адрес*, *Порт*

3) Пользователь хочет запросить актуальную информацию о пирах, находящихся в сети: **!UPD!**

Сервер ответит в формате: **!S!*Имя пира*, *IP-адрес*, *Порт*|**

Данные каждого пира отделены символом “|”.

4) Пользователь уходит из сети: **!OFF!**

Перечень флагов при взаимодействии двух пользователей

- !C! - Запрос на установление соединения и обмен ключами
- !M! - Сообщение
- !A! - Ответ на обмен ключами (отправка публичного сгенерированного ключа)

Примеры использования:

1) Пользователь при получении данных о пире от сервера отправляет ему запрос на установление соединения и обмен ключами:

   !C!*Имя пользователя*,*Адрес слушающего сокета*, *Порт*,*Простое число(Prime)*, *Генератор(Generator)*, *Публичный ключ*

   Адресат сохранит его данные у себя, и сгенерирует сеансовый ключ на основе его переданных параметров, после чего ответит в следующей форме:

   !A!*Имя пользователя*,*Сгенерированный публичный ключ*

   Таким образом, обмен ключами будет завершен и пользователи смогут отправлять сообщения друг другу

2) Обычные же сообщения выглядят так:

   !M!*Имя пользователя*,*Зашифрованное сообщение*

Используя имя пользователя выполняется поиск сеансового ключа, соответствующего имени отправителя. Далее сообщение дешифруется этим ключом и заносится в историю сообщений, после чего выводится на экран

Прокси

Клиентское приложение также имеет возможность использовать системные настройки прокси. С помощью стандартной библиотеки Qt можно получить системный адрес прокси, и установить через него соединение с информационным сервером. Сначала клиент подключается к прокси-серверу, затем прокси-сервер подключается к указанному серверу и отсылает регистрационные данные ему. Прокси-сервер позволяет защищать компьютер клиента от некоторых сетевых атак, но так как как клиент отсылает свой настоящий адрес на инфосервер, то это не обеспечивает ему анонимности и ничто не мешает другим клиентам соединиться с ним при получении его адреса.

Заключение

Таким образом, удалось реализовать успешный проект, выполняющий все необходимые задачи: реализован безопасный обмен сообщениями, реализована архитектура P2P сети, и создано стабильно работающее приложение.

В дальнейшем планируется сделать удобную форму регистрации и входа, добавить возможность обмена файлами, стикеры, возможность голосовой связи.

Список использованных источников

1. Конюхов С. Реализация алгоритма “Кузнечик” [Электронный ресурс] . URL : https://github.com/KoSeAn97/Encryptor-With-Kuznyechik: репозиторий (дата обращения: 20.10.2017)
2. Документация: Библиотека для C++ с реализациями криптографических схем [Электронный ресурс]. URL: https://www.cryptopp.com (дата обращения: 27.10.2017)
3. Документация: Официальная документация по фреймворку QT [Электронный ресурс]. URL: http://doc.qt.io (дата обращения: 5.11.2017)
4. Руководство: Реализация блочного шифра “Кузнечик” на языке C++ [Электронный ресурс]. URL: https://habrahabr.ru/post/313932/ (дата обращения : 15.10.2017)
5. Описание протокола Диффи-Хеллмана [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана (дата обращения : 25.11.2017)