Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

|  |
| --- |
| Gerb-BMSTU_01 |

Факультет

Кафедра ИУ - 8 «Компьютерная Безопасность»

**КУРСОВОЙ ПРОЕКТ**

РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ:

ЧАТ С ИСПОЛЬЗОВАНИЕМ КРИПТОГРАФИЧЕСКОГО АЛГОРИТМА RSA

Студенты: Шапран Алексей Викторович, ИУ8-34

Косых Александр Михайлович, ИУ8-34

Горбачев Александр Максимович, ИУ8-34

Оглавление

[1. Введение 3](#_Toc501921656)

[2. Постановка задачи 3](#_Toc501921657)

[3. Обзор аналогов 4](#_Toc501921658)

[4 Анализ предметной области 5](#_Toc501921663)

[4.1 Что такое чат 5](#_Toc501921664)

[4.2 Виды чатов 5](#_Toc501921665)

[5 Решение 6](#_Toc501921666)

[5.1 Архитектура системы 6](#_Toc501921667)

[5.2 Реализация 6](#_Toc501921668)

[5.2.1Реализация Qt(С++) 6](#_Toc501921669)

[5.2.2 Реализация алгоритма RSA 6](#_Toc501921670)

[5.2.3 Cерверная часть 6](#_Toc501921671)

[5.2.4 Среды разработки 6](#_Toc501921672)

[5.2 Ход работы 6](#_Toc501921673)

[5.2.1 Проблемы и решения 7](#_Toc501921674)

[6 Результат 7](#_Toc501921675)

[7 Выводы 7](#_Toc501921676)

[8 Список литературы 7](#_Toc501921677)

[9 Приложения 7](#_Toc501921678)

# Введение

В настоящих реалиях общение on-line занимает огромную часть в жизни каждого человека, поэтому сейчас существует множество продуктов, обеспечивающих эту возможность.

Так же, с развитием информационного пространства, возникает вопрос организации безопасности личной информации: ее хранения и передача.

Поэтому основной задачей перед собой мы поставили:

Разобраться в существующих реализациях и попробовать реализовать свой проект, содержащий некоторые функции гигантов этой области, например, таких как Telegram.

# Постановка задачи

Перед нами была поставлена задача:

* Реализовать чат в режиме реального времени..
* Обеспечить ему безопасность передачи и хранения данных.
* По возможности реализовать хранение истории сообщений.

И были выдвинуты следующие критерии:

* В нашем проекте обязательно должен присутствовать язык программирования С++.
* Проект должен соответсвовать протоколу UDP.
* Очевидно, что чат должен быть асинхронным.

# 3. Обзор аналогов

Нами были рассмотрены несколько реализованных чатов, относительно которых мы можем построить свой проект.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Продукт | Поддержка сохранения сообщений | Шифрование данных | Android приложение | Desktop версия |
| Telegram | Да | **AES**  **RSA**  **Метод Диффи-Хеллмана**  **SHA-1, MD5** | Да | Да |
| WhatsUP | Да | SSL и TLS | Да | Да |
| Viber | Да | salsa20 | Да | Да |
| Veon | Да | Не найдено | Да | Нет |
| IUGram  (Наш проект) | Да | RSA | Да | Да |

*\*информация не подтверждена, так как была найдена в интернет-источниках*

Даные аналоги были рассмотрены как самые популярные, следовательно из них мы могли узнать о самом удачном способе реализации нашего проекта.

# 4 Анализ предметной области

## Что такое чат

**Чат**- средство обмена сообщениями по компьютерной сети в режиме реального времени, а также [программное обеспечение](https://ru.wikipedia.org/wiki/Программное_обеспечение), позволяющее организовывать такое общение. Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает чат от [форумов](https://ru.wikipedia.org/wiki/Веб-форум) и других «медленных» средств. То есть, если на форуме можно написать вопрос и ждать, пока кто-нибудь посчитает нужным на него ответить (в то же время можно получить и несколько ответов сразу от разных пользователей), то в чате общение происходит только с теми, кто присутствует в нём в настоящий момент, а результаты обмена сообщениями могут и не сохраняться.

## Виды чатов

Существует несколько разновидностей программной реализации чатов:

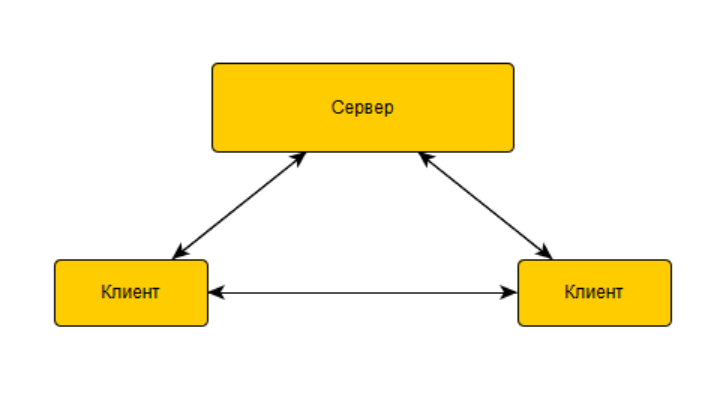
* HTTP или веб-чаты. Такой чат выглядит как обычная веб-страница, где можно прочесть последние несколько десятков фраз, написанные участниками чата и модераторами. Страница чата автоматически обновляется с заданной периодичностью.
* Чаты, использующие технологию Adobe Flash. Вместо периодической перезагрузки страницы, между клиентом и сервером открывается сокет, что позволяет моментально отправлять или получать сообщения, расходуя меньше трафика.
* IRC, специализированный протокол для чатов.
* Программы-чаты для общения в локальных сетях. Часто есть возможность передачи файлов.
* Чаты, реализованные поверх сторонних протоколов.
* Чаты, работающие по схеме клиент-сервер, это позволяет использовать их в сетях со сложной конфигурацией, а также управлять клиентскими приложениями

По количеству лиц, участвующих в коммуникации, чаты делятся на:

* all2all — групповые;
* p2p — персональные — личное общение

# Решение

## Архитектура системы



## 5.2 Реализация

### 5.2.1Реализация Qt(С++)

Было реализовано приложение, состоящее из 4 форм, обеспечивающий базовый функционал чата:

* Chatwindow.ui
* Iugram.ui
* Secondwindow.ui
* Settingswindow.ui

Их реализация представлена в приложении.

Для обмена сообщениями использовались UDP сокеты, реализация которых входила в стандарт данной версии Qt Creator.

Все нижепредставленные функции находятся в исходном файле формы chatwindow.

Функции:

* Открытие соединения и отправка первого сообщения - Создание UDP-чата - void ChatWindow::**UdpChat**(QString nick, int port)
* Отправка сообщения - void ChatWindow::**send**(QString str, MessageType type)
* Получение сообщения - void ChatWindow::**read**()

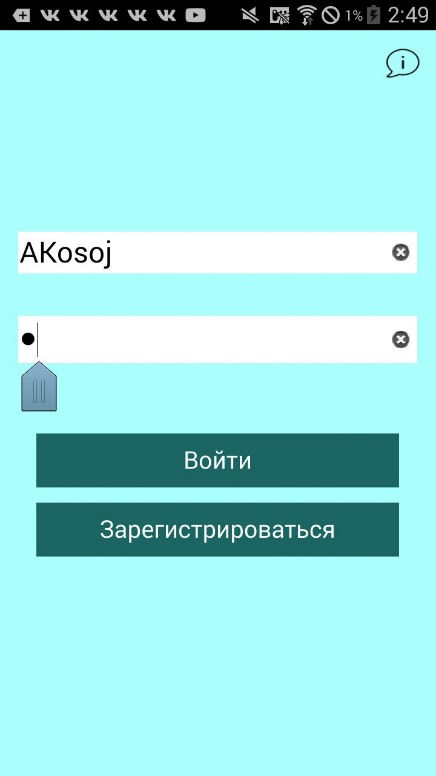
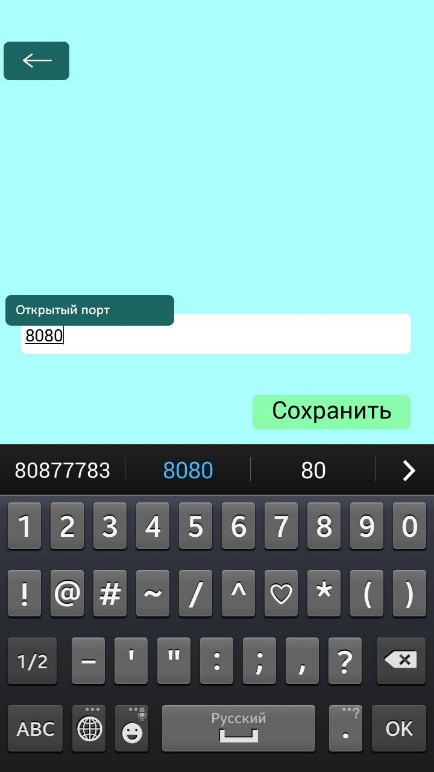
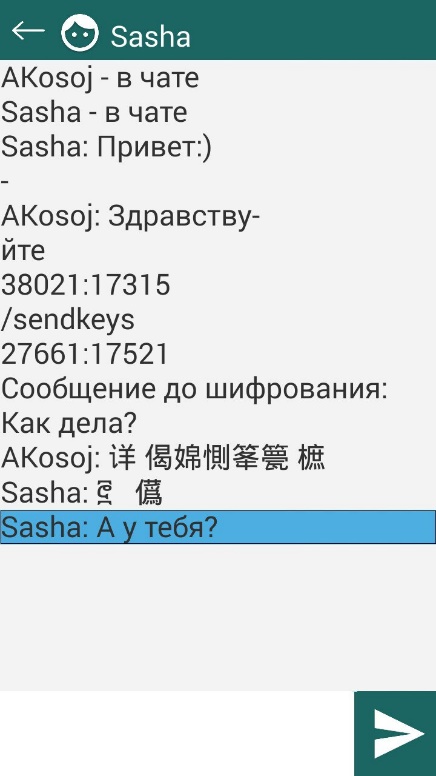
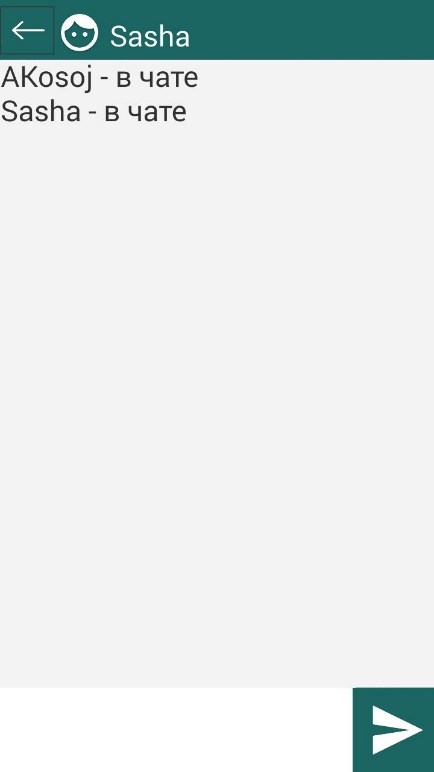
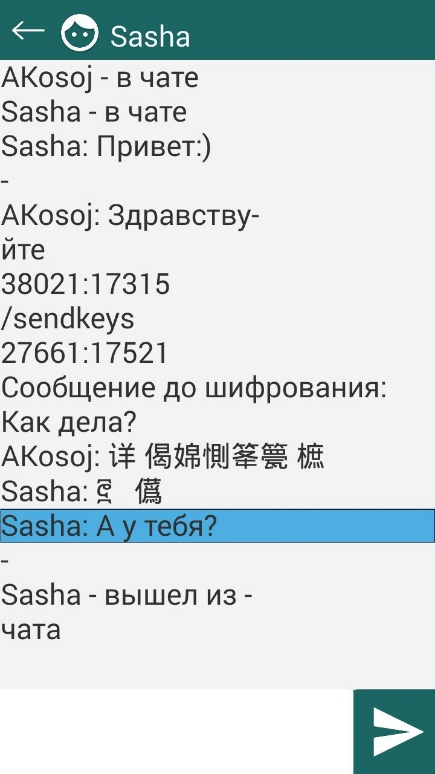
Связь с модулем шифрования RSA осуществлена в следующих функциях:

* Нажатие на кнопку ОК - void ChatWindow::**returnPressed**()
* Выбор сообщения для расшифровки - void ChatWindow::**on\_listWidget\_itemClicked**(QListWidgetItem \*item)
* Создание ключей, при создании формы chatwindow - keys = rsa.produce\_keys();

Код всех функций находится в приложении(п 9).

Сама реализация модуля RSA представлена в пункте 5.2.2.

Скриншоты работы программы:

### 5.2.2 Реализация алгоритма RSA

В проекте было реализовано шифрование сообщений с помощью криптографического алгоритма RSA. В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать, кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными.

Структура алгоритма

В текущей реализации RSA-ключи генерируются следующим образом (вызов функции produce\_keys():

* + - С помощью функции produce\_prime() генерируется 2 различных простых числа, заданной длины (p1,p2).
    - С помощью функции produce\_pkey(prime1,prime2) (p = p1\*p2). вычисляется их произведение, которое называется модулем.
    - Вычисляется значение функции Эйлера orla с помощью функции produce\_orla(prime1,prime2) по формуле: 𝜑(𝑛)=(𝑝1−1)∗(𝑝2−1)
    - С помощью функции produce\_ekey(orla) выбирается число e, взаимно простое со значением функции Эйлера.
    - С помощью функции produce\_dkey(e,orla) вычисляется число d, являющееся секретной экспонентой по формуле 𝑑=𝑒−1 𝑚𝑜𝑑 𝜑(𝑛).
    - ekey, pkey, dkey сохраняются в структуру Keys.

Предположим, что Боб хочет послать Алисе сообщение m.

* + - Алиса генерирует у себя ключи функцией keys\_produce().
    - Компоненты ekey и pkey (открытый ключ) отправляются Бобу.
    - Боб, получив эти компоненты, вызывает функцию endecrypt(m, ekey, pkey) и получает зашифрованное с, которое он отправляет Алисе. Шифрование происходит по формуле: с=𝑚𝑒𝑘𝑒𝑦 𝑚𝑜𝑑 𝑝𝑘𝑒𝑦.
    - Чтобы расшифровать полученное сообщение c, Алиса вызывает функцию endecrypt(c, ekey, dkey). Дешифровка происходит по формуле: 𝑚=𝑐𝑑𝑘𝑒𝑦 𝑚𝑜𝑑 𝑝𝑘𝑒𝑦.

### 5.2.3 Cерверная часть

Серверная часть была реализована с помощью с сервера Mongo, в нем хранения данных происходит в json коллекциях, а не в таблицах, как например в MySQL. Поделючение к Qt осуществлялось через официальные драйвера (mongo-cxx) и библиотеки boost.

(Надо будет вставить как будут выглядеть наши коллекции, если они, конечно, будут)

### 5.2.4 Среды разработки

* Qt Creator
* Mongo shell
* Visual Studio 2017

## 5.2 Ход работы

* Определение предметной области
* Обзор аналогов, для поиска оптимального решения
* Реализация Qt приложения с базовым функционалом
* Реализация алгоритма RSA
* Подключение модуля RSA к Qt приложению
* Подключение сервера для хранения даных

### Проблемы и решения

* Оптимизация UI под различные разрешения экраны
* Подключение сервера и обзор аналогов
* Малая скорость работы программы при больших ключах шифрования

# Результат

Было реализовано приложение, содержащие базовые функции чата, обеспечивающие безопасность передачи и хранения данных, с помощью алгоритма RSA.

# Выводы

В ходе проделанной работы:

* Произведена исследовательская работа в сфере мобильных приложений.
* Произведена работа с источниками, содержащими информацию о языке С++, фреймворке Qt, алгоритме шифрования RSA
* Разобраны способы подключения и возможности Android SDK, Android NDK в среду разработки QtCreator.
* Определенно на практике преимущества использования сетевого протокола пользовательских датаграмм UDP в легковесном мобильном приложении.
* На практике определенно, что для реализации безопасного алгоритма RSA необходимы большие вычислительные мощности.
* Проведены тесты в среде непрерывной Travis-CI.

# Список литературы

* John Torjo «Boost.Asio C++ Network Programming»
* С. Коутинхо. Введение в теорию чисел. Алгоритм RSA.
* М. Вельшенбах. Криптография на Си и С++ в действии.
* GOT для профессионального программиста
* <https://docs.mongodb.com/> - документация по Mongo
* <http://doc.qt.io/> - документация по Qt

# Приложения

*Исходные файлы Qt проекта:*

Chatwindow.h

#ifndef CHATWINDOW\_H

#define CHATWINDOW\_H

#include <QMainWindow>

#include <QListWidgetItem>

#include <QMessageBox>

#include <QObject>

#include <QUdpSocket>

const unsigned int width\_widget = 19;

enum MessageType {

USUAL\_MESSAGE, // Обычный текст

PERSON\_ONLINE, // Сообщение "Я-онлайн"

// WHO\_IS\_ONLINE, // Запрос о статусе пользователей

KEYS\_MESSAGE, // сообщение с ключами

};

namespace Ui {

class ChatWindow;

}

class ChatWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit ChatWindow(QWidget \*parent = 0);

~***ChatWindow***();

public:

Ui::ChatWindow \*ui;

private slots:

void **returnPressed**();

void **on\_pushButton\_clicked**();

void **on\_pushButton\_3\_clicked**();

private:

// Для работы по UDP-протоколу в

// локальной сети

QUdpSocket\* socket;

void **UdpChat**(QString nick, int port);

void **send**(QString str, MessageType type);

private slots:

void **read**();

void **on\_listWidget\_itemClicked**(QListWidgetItem \*item);

};

#endif // CHATWINDOW\_H

Chatwindow.cpp

#include "ui\_chatwindow.h"

#include "secondwindow.h"

#include "rsa.h"

#include <QTimer>

#include <QTime>

Rsa rsa;

Key keys;

long pub\_key\_otheruser;

long exp\_key\_otheruser;

extern QString new\_user\_log;

extern int port;

extern QListWidgetItem \* item\_;

ChatWindow::**ChatWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::ChatWindow)

{

ui->setupUi(this);

keys = rsa.produce\_keys(); // создание ключей

ui->lineEdit->setText(item\_->text());

socket = nullptr;

UdpChat(item\_->text(), port);

}

ChatWindow::~***ChatWindow***()

{

delete ui;

}

// Кнопка назад, ведущая к поисковой странице

void ChatWindow::**on\_pushButton\_clicked**()

{

send(new\_user\_log + " - вышел из чата", USUAL\_MESSAGE);

socket->*close*();

SecondWindow \* second\_window = new SecondWindow(this);

this->close();

second\_window->showFullScreen();

}

// Открытие соединения и отправка первого сообщения - Создание UDP-чата

void ChatWindow::**UdpChat**(QString nick, int port)

{

// Если соединение уже открыто, то закрываем его

if(socket != nullptr)

{

socket->*close*();

delete socket;

socket = nullptr;

}

// UDP - обмен сообщениями внутри локальной сети

// TCP/IP

// 1. Создание сокета для работы с сетью

socket = new QUdpSocket(this);

// QHostAddress::Any позволяет принимать сообщения со всех IP адресов

// Занятие определённого порта - все входящие сообщения на этот порт будут поступать в текущий диалог

if(socket->bind(QHostAddress::Any, port))

{

// При получении данных (сигнал readyRead) вызывается метод (слот) read, который читает и обрабатывает сообщение

connect(socket, SIGNAL(readyRead()), this, SLOT(read()));

}

else

{

// Какая-то программа на этом устройстве уже заняла порт port

QMessageBox::critical(this, "Ошибка!", "Заданный порт уже занят другой программой. Выберите другой порт (1024 - ...) и попробуйте подключиться еще раз.");

return;

}

send(new\_user\_log + " - в чате", USUAL\_MESSAGE);

QTime now = QTime::currentTime();

QString nowStr = now.toString("hh:mm:ss");

QString str = nowStr + " " + new\_user\_log;

send(str, PERSON\_ONLINE);

}

// Отправка сообщения

void ChatWindow::**send**(QString str, MessageType type)

{

// Полный пакет данных будет в массиве data

QByteArray data; // Массив данных для отправки

// Последовательный вывод байт в data

QDataStream out(&data, QIODevice::WriteOnly);

out << qint8(type); // Тип сообщения

out << str; // Само сообщение

// Отправка полученного массива данных всем в локальный сети на порт указанный в интерфейсе

socket->writeDatagram(data, QHostAddress::Broadcast, port);

}

// Получение сообщения

void ChatWindow::**read**()

{

while (socket->hasPendingDatagrams())

{

// Массив (буфер) для полученных данных

QByteArray buf;

// Установка массиву размера, соответствующего размеру полученного пакета данных

buf.resize(socket->pendingDatagramSize());

QHostAddress\* address = new QHostAddress();

// Принятие данных, пришедших по сети

socket->readDatagram(buf.data(), buf.size(), address);

// Разбор полученного пакета

QDataStream in(&buf, QIODevice::ReadOnly);

// Получение типа пакета

qint8 type = 0;

in >> type;

QString str;

in >> str;

if (str.length() == 0)

{

return;

}

unsigned int size = 0;

if (str.length() > width\_widget)

{

size = str.length() / width\_widget + 1;

for (unsigned int i = 0; i < size; ++i)

{

str.insert(width\_widget \* i, "\n");

}

}

if (type == USUAL\_MESSAGE)

{

// Отображение строки в интерфейсе

ui->listWidget->addItem(str);

}

else if (type == KEYS\_MESSAGE)

{

QString temp\_pub\_key;

for (auto it: str)

{

if (it != ':')

{

temp\_pub\_key.push\_back(it);

}

else break;

}

QString temp\_exp\_key;

bool after = false;

for (auto it: str)

{

if (it != ':' && !after)

continue;

if (it == ':')

{

after = true;

continue;

}

if (after)

{

temp\_exp\_key.push\_back(it);

}

}

if (temp\_exp\_key.toLong() != keys.ekey &&

temp\_pub\_key.toLong() != keys.pkey)

{

exp\_key\_otheruser = temp\_exp\_key.toLong();

pub\_key\_otheruser = temp\_pub\_key.toLong();

}

ui->listWidget->addItem(str);

}

}

}

// Кнопка отправить сообщение

void ChatWindow::**on\_pushButton\_3\_clicked**()

{

returnPressed();

}

// Нажатие на кнопку ОК

void ChatWindow::**returnPressed**()

{

QString text = ui->lineEdit\_2->text();

if (text.isEmpty())

{

return;

}

// Если введен запрос на отправку ключей

if (text.contains("/sendkeys"))

{

ui->listWidget->addItem(text);

ui->lineEdit\_2->clear();

QString public\_key = QString::number(keys.pkey);

QString exp\_key = QString::number(keys.ekey);

send(public\_key + ":" + exp\_key, KEYS\_MESSAGE);

return;

}

// Шифрование

QList<QListWidgetItem \*> found = ui->listWidget->findItems("/sendkeys", Qt::MatchContains);

if (found.count() > 0)

{

ui->listWidget->addItem("Сообщение до шифрования: \n" + text);

for (auto & it : text)

{

it = QChar((int)rsa.endecrypt((long)it.unicode(), exp\_key\_otheruser, pub\_key\_otheruser));

}

}

send(new\_user\_log + ": " + text, USUAL\_MESSAGE);

ui->lineEdit\_2->clear();

QTime now = QTime::currentTime();

QString nowStr = now.toString("hh:mm:ss");

send(nowStr + " " + new\_user\_log, PERSON\_ONLINE);

}

// Выбор сообщения для расшифровки

void ChatWindow::**on\_listWidget\_itemClicked**(QListWidgetItem \*item)

{

int index = item->text().indexOf(":");

QString temp;

for (int i = 0; i < index; ++i)

{

temp.push\_back(item->text()[i]);

}

if (temp == new\_user\_log)

{

QMessageBox::information(this, "Информация", "Собственные сообщения не подлежат расшифровки.");

return;

}

QString toDecrypt = item->text();

toDecrypt.remove(0, ui->lineEdit->text().size() + 1);

for (auto& it: toDecrypt)

{

it = QChar((int)rsa.endecrypt((long)it.unicode(),keys.dkey,keys.pkey));

}

toDecrypt.remove(0, 1);

unsigned int size = 0;

if (toDecrypt.length() > width\_widget)

{

size = toDecrypt.length() / width\_widget + 1;

for (unsigned int i = 0; i < size; ++i)

{

toDecrypt.insert(width\_widget \* i, "\n");

}

}

item->setText(ui->lineEdit->text() + ": " + toDecrypt);

}

Iugram.h

#ifndef IUGRAM\_H

#define IUGRAM\_H

#include <QMainWindow>

#include <QtAlgorithms>

#include <QString>

#include <QLinkedList>

#include <QMultiMap>

#include <QPair>

#include <QVector>

#include <QMessageBox>

// Для post() и get() запросов

//#include <QNetworkAccessManager>

//#include <QNetworkRequest>

//#include <QNetworkReply>

//#include <QUrl>

static QString ip\_address;

static QString port;

static bool bad\_log = false;

void **set\_list\_of\_users**();

namespace Ui {

class IUGram;

}

class IUGram : public QMainWindow

{

Q\_OBJECT

public:

explicit IUGram(QWidget \*parent = 0);

~***IUGram***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_pushButton\_3\_clicked**();

void **on\_lineEdit\_2\_returnPressed**();

void **on\_lineEdit\_returnPressed**();

void **on\_lineEdit\_2\_selectionChanged**();

void **on\_lineEdit\_selectionChanged**();

private:

Ui::IUGram \*ui;

};

#endif // IUGRAM\_H

Iugram.cpp

QString new\_user\_pass;

QString new\_user\_log;

QLinkedList<QPair<QString, QString>> list\_of\_users;

IUGram::**IUGram**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::IUGram)

{

ui->setupUi(this);

}

IUGram::~***IUGram***()

{

delete ui;

}

void **set\_list\_of\_users**()

{

// Получение списка зерегистрированных пользователей (TODO: get() запросы)

// manager = new QNetworkAccessManager(); // Инициализация менеджера

// QUrl url("http://www..."); // Инициализация URL, с которого будут получаться данные

// QNetworkRequest request; // Отправляемый запрос

// request.setUrl(url); // Устанавлвиваем URL в запрос

// manager->get(request); // Выполняем запрос

// QNetworkReply \*reply;

// QString temp\_list\_of\_users;

// temp\_list\_of\_users.push\_back(reply->readAll());

QVector<QPair<QString, QString>> local\_users;

local\_users.push\_back(QPair<QString, QString>("S", "Sasha"));

local\_users.push\_back(QPair<QString, QString>("A", "AKosoj"));

for (auto it : local\_users)

{

list\_of\_users.push\_back(qMakePair(it.first,it.second));

}

}

// Кнопка настройки - шестеренка

void IUGram::**on\_pushButton\_clicked**()

{

QString temp = "Чат с шифрованием RSA. Все ваши сообщения будут защищены и надежно сохранены! Приятного пользования:)";

QMessageBox::information(this, "Информация", temp);

}

// Кнопка войти

void IUGram::**on\_pushButton\_2\_clicked**()

{

// Считывание пароля нового user-а

new\_user\_pass = ui->lineEdit->text();

// Считывание логина нового user-а

new\_user\_log = ui->lineEdit\_2->text();

// Если логин и пароль уже зарегистрированы для данного пользователя

if (qFind(list\_of\_users.begin(), list\_of\_users.end(), QPair<QString, QString>(new\_user\_pass, new\_user\_log)) != list\_of\_users.end())

{

// Создание второго окна со списком сообщений

SecondWindow \* second\_window = new SecondWindow(this);

this->close();

second\_window->showFullScreen();

}

else

{

// Сообщение о том, что вход не удался, нужно зарегистрироваться

QString temp = "Неправильный логин или пароль!";

QMessageBox::information(this, "Ошибка авторизации!", temp);

}

}

// Кнопка зарегистрироваться

void IUGram::**on\_pushButton\_3\_clicked**()

{

// Считывание пароля нового user-а

new\_user\_pass = ui->lineEdit->text();

// Считывание логина нового user-а

new\_user\_log = ui->lineEdit\_2->text();

// Если существует зарегистрированный пользователь с таким же логином

for (auto it : list\_of\_users)

{

if (it.second == new\_user\_log)

{

bad\_log = true;

}

}

if (bad\_log)

{

// Сообщение о том, что регистрация не удалась, потому что существует пользователь с таким логином

QString temp = "Данный логин уже зарегистрирован! Пожалуйста, придумайте другой логин.";

QMessageBox::information(this, "Ошибка регистрации!", temp);

bad\_log = false;

return;

}

// Если одно из полей логин или пароль не заполнено

if (new\_user\_log == "" || new\_user\_pass == "" || new\_user\_log == " Логин" || new\_user\_pass == " Пароль")

{

// Сообщение о том, что регистрация не удалась, потому что нельзя оставлять поля логин или пароль пустыми

QString temp = "Оба поля (логин и пароль) должны быть заполнены!";

QMessageBox::information(this, "Ошибка регистрации!", temp);

return;

}

if (new\_user\_pass.size() < 8)

{

// Сообщение о том, что регистрация не удалась, потому что пароль очень короткий

QString temp = "Минимальная длина пароля - 8 символов!";

QMessageBox::information(this, "Ошибка регистрации!", temp);

return;

}

// Если регистрация прошла успешно, то добавляем логин и пароль нового user-а в список (TODO post() запросы)

list\_of\_users.push\_back(qMakePair(new\_user\_pass,new\_user\_log));

// Создание второго окна со списком сообщений

SecondWindow \* second\_window = new SecondWindow(this);

this->close();

second\_window->showFullScreen();

}

// Нажатие на кнопку ввод/enter на клавиатуре при нахождении курсора в поле с логином

void IUGram::**on\_lineEdit\_2\_returnPressed**()

{

on\_pushButton\_2\_clicked();

}

// Нажатие на кнопку ввод/enter на клавиатуре при нахождении курсора в поле с паролем

void IUGram::**on\_lineEdit\_returnPressed**()

{

on\_pushButton\_2\_clicked();

}

// Двойное нажатие по слову логин приводит к его исчезновению

void IUGram::**on\_lineEdit\_2\_selectionChanged**()

{

ui->lineEdit\_2->setText("");

}

// Двойное нажатие по слову пароль приводит к его исчезновению

void IUGram::**on\_lineEdit\_selectionChanged**()

{

ui->lineEdit->setText("");

ui->lineEdit->setEchoMode(QLineEdit::Password);

}

Secondwindow.h

#ifndef SECONDWINDOW\_H

#define SECONDWINDOW\_H

#include <QMainWindow>

#include <QListWidgetItem>

namespace Ui {

class SecondWindow;

}

class SecondWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit SecondWindow(QWidget \*parent = 0);

~***SecondWindow***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_listWidget\_itemClicked**(QListWidgetItem \*item);

void **on\_lineEdit\_returnPressed**();

private:

Ui::SecondWindow \*ui;

};

#endif // SECONDWINDOW\_H

Secondwindow.cpp

#include "secondwindow.h"

#include "ui\_secondwindow.h"

#include "iugram.h"

#include "chatwindow.h"

#include "settingswindow.h"

extern QLinkedList<QPair<QString, QString>> list\_of\_users;

extern QString new\_user\_log;

QListWidgetItem \* item\_;

SecondWindow::**SecondWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::SecondWindow)

{

ui->setupUi(this);

}

SecondWindow::~***SecondWindow***()

{

delete ui;

}

// Кнопка поиска собеседника - лупа

void SecondWindow::**on\_pushButton\_clicked**()

{

ui->listWidget->clear();

// Вектор найденных юзеров

QVector<QString> found\_users;

// Логин user-а, которого ищет текущий user

QString user\_find = ui->lineEdit->text();

// Если в поле поиск ничего не введено

if (user\_find == "")

{

// Сообщение о том, что не был введен логин для поиска user-a

QString temp = "Вы не ввели логин для поиска пользователя!";

QMessageBox::information(this, "Не был введен логин для поиска пользователя!", temp);

return;

}

for (auto it : list\_of\_users)

{

// Если среди списка зарегистрированных user-ов есть тот, которого ищут

// (не учитывается регистр, поиск возможен даже для неполного логина, тогда выводятся

// все логины, в которые входит ищущийся логн)

if (it.second.contains(user\_find, Qt::CaseInsensitive))

{

found\_users << (it.second);

}

}

if (!found\_users.isEmpty())

{

// Вывод всех найденных user-ов

for (auto item : found\_users)

{

ui->listWidget->addItem(item);

}

}

else

{

// Сообщение о том, что не удалось найти такого user-а

QString temp = "Ни одного юзера не найдено:(";

QMessageBox::information(this, "Поиск не дал результатов!", temp);

}

}

// Кнопка выхода - крестик

void SecondWindow::**on\_pushButton\_2\_clicked**()

{

IUGram \* first\_window = new IUGram(this);

this->close();

first\_window->showFullScreen();

}

// Выбор пользователя для открытия чата с ним

void SecondWindow::**on\_listWidget\_itemClicked**(QListWidgetItem \*item)

{

item\_ = item;

SettingsWindow \* settings\_window = new SettingsWindow(this);

this->close();

settings\_window->showFullScreen();

}

// Нажатие кнопки ОК

void SecondWindow::**on\_lineEdit\_returnPressed**()

{

on\_pushButton\_clicked();

}

Settingswindow.h

#ifndef SETTINGSWINDOW\_H

#define SETTINGSWINDOW\_H

#include <QMainWindow>

namespace Ui {

class SettingsWindow;

}

class SettingsWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit SettingsWindow(QWidget \*parent = 0);

~***SettingsWindow***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_lineEdit\_returnPressed**();

void **on\_pushButton\_2\_clicked**();

private:

Ui::SettingsWindow \*ui;

};

#endif // SETTINGSWINDOW\_H

Settingswindow.cpp

#include <QNetworkConfiguration>

#include <QNetworkConfigurationManager>

int port;

SettingsWindow::**SettingsWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::SettingsWindow)

{

ui->setupUi(this);

}

SettingsWindow::~***SettingsWindow***()

{

delete ui;

}

// Кнопка назад, которая ведет к странице поиска пользователей

void SettingsWindow::**on\_pushButton\_clicked**()

{

SecondWindow \* second\_window = new SecondWindow(this);

this->close();

second\_window->showFullScreen();

}

// Кнопка ОК

void SettingsWindow::**on\_lineEdit\_returnPressed**()

{

on\_pushButton\_2\_clicked();

}

// Кнопка сохранить

void SettingsWindow::**on\_pushButton\_2\_clicked**()

{

if (ui->lineEdit->text().isEmpty())

{

QMessageBox::critical(this, "Ошибка!", "Поле открытый порт должно быть заполнено.");

return;

}

port = ui->lineEdit->text().toInt();

// Менеджер сети для определения подключения к WIFI

QNetworkConfigurationManager manager;

// Проверка наличия соединения

if (manager.isOnline())

{

ChatWindow \* chat\_window = new ChatWindow(this);

this->close();

chat\_window->showFullScreen();

}

else

{

QMessageBox::critical(this, "Ошибка подключения!", "Для перехода к чату необходимо подключиться к сети WIFI.");

}

}

*Исходные файлы реализации RSA:*

Rsa.h

#ifndef RSA\_H

#define RSA\_H

typedef struct key

{

long pkey;

long ekey;

long dkey;

} Key;

class Rsa

{

public:

Rsa();

virtual ~***Rsa***();

public:

Key **produce\_keys**();

long **endecrypt**(const long msg, const long key, const long pkey);

private:

long **produce\_pkey**(const long prime1, const long prime2);

long **produce\_ekey**(const long orla);

long **produce\_dkey**(const long ekey, const long orla);

long **produce\_prime**();

long **produce\_orla**(const long prime1, const long prime2);

long **produce\_gcd**(const long a, const long b);

bool **is\_prime**(const long digit);

};

#endif // RSA\_H

Rsa.cpp

#include "rsa.h"

#include <QTime>

using namespace std;

Rsa::Rsa()

{

}

Rsa::~Rsa()

{

}

long Rsa::endecrypt(const long msg, const long key, const long pkey)

{

long msg\_des = 1;

long root = msg;

long index = key;

while (index)

{

if (index & 1)

msg\_des = (msg\_des \* root) % pkey;

index >>= 1;

root = (root \* root) % pkey;

}

return msg\_des;

}

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

\* Генерация ключей

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

Key Rsa::produce\_keys()

{

long prime1 = produce\_prime();

long prime2 = produce\_prime();

while (prime2 == prime1)

prime2 = produce\_prime();

Key key;

long orla = produce\_orla(prime1, prime2);

key.pkey = produce\_pkey(prime1, prime2);

key.ekey = produce\_ekey(orla);

key.dkey = produce\_dkey(key.ekey, orla);

return key;

}

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

\* Генерация публичного ключа

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

long Rsa::produce\_pkey(const long prime1, const long prime2)

{

return prime1 \* prime2;

}

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

\* Генерация значения функции Эйлера

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

long Rsa::produce\_orla(const long prime1, const long prime2)

{

return (prime1 - 1) \* (prime2 - 1);

}

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

\* Генерация открытой экспоненты

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

long Rsa::produce\_ekey(const long orla)

{

long ekey;

while (true)

{

ekey = qrand() % orla;

if (ekey >= 2 && produce\_gcd(ekey, orla) == 1)

break;

}

return ekey;

}

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

\* Генерация приватного ключа (декриптора)

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

long Rsa::produce\_dkey(const long ekey, const long orla)

{

long dkey = orla / ekey;

while (true)

{

if (((dkey \* ekey) % orla) == 1)

break;

else

++dkey;

}

return dkey;

}

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

\* Генерация случайных чисел в промежутке

\* от 100 включительно до 200

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

long Rsa::produce\_prime()

{

long prime = 0;

qsrand(QTime::currentTime().msec());

while (true)

{

prime = qrand() % 100 + 100;

if (is\_prime(prime))

break;

}

return prime;

}

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

\* Вычисление НОД

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

long Rsa::produce\_gcd(const long a, const long b)

{

long dividend = a;

long divisor = b;

long residual = dividend % divisor;

while (residual)

{

dividend = divisor;

divisor = residual;

residual = dividend % divisor;

}

return divisor;

}

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

\* проверка числа на простоту

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

bool Rsa::is\_prime(const long digit)

{

int tmp = 2;

while (tmp < digit)

if (!(digit % tmp++))

break;

if (tmp == digit)

return true;

return false;

}