

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа программной инженерии

**Курсовая работа**  
**Разработка системы активации программного продукта**  
по дисциплине «Защита информации»

Выполнил студент  
гр. 3530904/60105



Стойкоски Н.С.

Руководитель

Б.М. Медведев

Санкт-Петербург  
2019 г.

## Оглавление

Введение.....	3
Постановка задачи .....	4
Secure Sockets Layer (SSL) .....	5
OpenSSL .....	6
Генерация SSL сертификатов .....	7
LicenseServer.....	8
Протокол взаимодействия с сервером .....	9
LicenseClient .....	10
LicenseCreator .....	11
Пример использования .....	12
Заключение .....	15
Список использованной литературы .....	16
Приложение 1. Текст программы LicenseServer .....	17
Приложение 2. Текст программы LicenseClient.....	23
Приложение 3. Текст программы LicenseCreator .....	32

## Введение

Активация продукта - это процедура проверки лицензии, требуемая некоторыми проприетарными компьютерными программами. Активация продукта предотвращает неограниченное бесплатное использование скопированного или реплицированного программного обеспечения. Неактивированное программное обеспечение отказывается полностью функционировать до тех пор, пока не определит, разрешено ли ему полноценно функционировать. Активация позволяет программному обеспечению прекратить блокировать его использование. Активация может длиться «вечно» или может иметь ограничение по времени, требующее продления или повторной активации для дальнейшего использования.

В одной форме активация продукта относится к методу, изобретенному Риком Ричардсоном и запатентованному (патент США 5 490 216) компанией Uniloc, где прикладная программа хеширует серийные номера оборудования и идентификационный номер, относящийся к лицензии продукта (ключ продукта), для создания уникальной установки. Этот идентификатор установки отправляется производителю, чтобы проверить подлинность ключа продукта и убедиться, что ключ продукта не используется для нескольких установок.

В качестве альтернативы поставщик программного обеспечения отправляет пользователю уникальный серийный номер продукта. Когда пользователь устанавливает приложение, он запрашивает, чтобы пользователь ввел серийный номер своего продукта, и сверяет его с системами поставщика через Интернет. Приложение получает лицензионные ограничения, которые применяются к лицензии этого пользователя, такие как ограничение по времени или включение функций продукта, из системы поставщика и, при необходимости, также блокирует лицензию для системы пользователя. После активации лицензия продолжает работать на компьютере пользователя, и дальнейшая связь с системами поставщика не требуется. Некоторые системы активации также поддерживают активацию в пользовательских системах без подключений к Интернету; общий подход заключается в обмене зашифрованными файлами в интернет-терминале.

## Постановка задачи

1. Написать приложение с графическим интерфейсом которое предоставляет возможности пользователю:

- Выбрать на диске любой файл (приложение, документ, картинка...)
- Сгенерировать автоматически или ввести самому список серийных кодов - лицензии, которые можно будет использовать для освобождения (расшифровка) выбранного файла.

- Зашифровать файл и заключить его для использование пользователем.

\* Приложение должно работать и по аргументам командной строки (без использование графического интерфейса) для это использование в автоматические сборки при создание ПО.

2. Написать приложение с графическим интерфейсом которое будет хранить вложенный файл из (1) и предоставляет возможности пользователю:

- Ввести серийный номер (лицензия), который будет отправляться на сервер (по защищенному каналу) для проверки. При успешного ввода (успешной проверки на сервер), вложенный файл расшифруется и пользователь получает к нему доступ.

3. Написать серверное приложение которое будет получать (по защищенному каналу) запросы на проверки серийных номеров и будет обратно отправлять ответ в виде да/нет (\*или в виде кода для расшифровки вложенного файла). Локально будет отмечать уже использованные серийные коды.

## Secure Sockets Layer (SSL)

SSL (англ. Secure Sockets Layer — уровень защищённых сокетов) — криптографический протокол, который подразумевает более безопасную связь. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Протокол широко использовался для обмена мгновенными сообщениями и передачи голоса через IP (англ. Voice over IP — VoIP) в таких приложениях, как электронная почта, интернет-факс и др. В 2014 году правительство США сообщило об уязвимости в текущей версии протокола. SSL должен быть исключён из работы в пользу TLS (см. CVE-2014-3566).

SSL изначально разработан компанией Netscape Communications для добавления протокола HTTPS в свой веб-браузер Netscape Navigator. Впоследствии на основании протокола SSL 3.0 был разработан и принят стандарт RFC, получивший имя TLS.

Протокол SSL размещается между двумя протоколами: протоколом, который использует программа-клиент (HTTP, FTP, LDAP, TELNET и т.д.) и транспортным протоколом TCP/IP. SSL защищает данные, выступая в роли фильтра для обеих сторон и передаёт их далее на транспортный уровень.

Работу протокола можно разделить на два уровня:

- Слой протокола подтверждения подключения (Handshake Protocol Layer)
- Слой протокола записи

Первый слой, в свою очередь, состоит из трёх подпротоколов:

- Протокол подтверждения подключения (Handshake Protocol)
- Протокол изменения параметров шифра (Cipher Spec Protocol)
- Предупредительный протокол (Alert Protocol)

Протокол подтверждения подключения используется для согласования данных сессии между клиентом и сервером. К данным сессии относятся:

- Идентификационный номер сессии
- Сертификаты обеих сторон
- Параметры алгоритма шифрования
- Алгоритм сжатия информации
- «Общий секрет» применён для создания ключей; открытый ключ

## **OpenSSL**

OpenSSL — полноценная криптографическая библиотека с открытым исходным кодом, широко известна из-за расширения SSL/TLS, используемого в веб-протоколе HTTPS.

Поддерживает почти все низкоуровневые алгоритмы хеширования, шифрования и электронной подписи, а также реализует большинство популярных криптографических стандартов, в том числе: позволяет создавать ключи RSA, DH, DSA, сертификаты X.509, подписывать их, формировать CSR и CRT, шифровать данные и тестировать SSL/TLS соединения.

Доступна в виде пакетов для большинства UNIX-подобных операционных систем (включая Solaris/OpenSolaris, Linux, Mac OS X, QNX4, QNX6 и четырёх операционных систем BSD с открытым исходным кодом), а также для OpenVMS и Microsoft Windows.

OpenSSL поддерживает разные алгоритмы шифрования и хеширования:

### **Симметричные**

Blowfish, Camellia, DES, RC2, RC4, RC5, IDEA, AES, ГОСТ 28147-89

### **Хеш-функции**

MD5, MD2, SHA, MDC-2, ГОСТ Р 34.11-94

### **Асимметричные**

RSA, DSA, Diffie-Hellman key exchange, ГОСТ Р 34.10-2001 (34.10-94)

Поддержка алгоритмов ГОСТ появилась в версии 1.0.0, выпущенной 29 марта 2010 года, и была реализована сотрудниками фирмы «Криптоком».

## Генерация SSL сертификатов

Создаем два сертификата СА, красный и синий. Красный и синий - это просто произвольные имена для различения сертификатов. На сервере будет использоваться blue\_ca.pem, а на клиенте - red\_ca.pem.

```
openssl req -out blue_ca.pem -new -x509 -nodes
mv privkey.pem blue_privkey.pem
openssl req -out red_ca.pem -new -x509 -nodes
mv privkey.pem red_privkey.pem
```

Затем создаем два файла с именами blue\_index.txt и red\_index.txt. Открываем их в текстовом редакторе и помещаем две нулевые цифры в начале. Далее создаем локальные пары сертификат / ключ, полученные из сертификатов СА. При вводе информации для red\_local.req убедитесь, что полное доменное имя соответствует IP-адресу / имени хоста сервера. В примере используется локальный хост 127.0.0.1

```
openssl genrsa -out blue_local.key 1024
openssl req -key blue_local.key -new -out blue_local.req
openssl x509 -req -in blue_local.req -CA blue_ca.pem -CAkey
blue_privkey.pem -CAserial blue_index.txt -out blue_local.pem
```

```
openssl genrsa -out red_local.key 1024
openssl req -key red_local.key -new -out red_local.req
openssl x509 -req -in red_local.req -CA red_ca.pem -CAkey
red_privkey.pem -CAserial red_index.txt -out red_local.pem
```

blue\_local.pem и red\_local.pem - локальные сертификаты, а red\_local.key и blue\_local.key - связанные с ними закрытые ключи. blue\_local будет использоваться на клиенте, а red\_local будет использоваться на сервере.

## LicenseServer

Было реализовано серверное приложение под фреймворк Qt на языке C++, позволяющее по уникальным ключам для приложения конечного пользователя, вводить список возможных лицензий, ввода код для дешифровки конечного приложения, проверка лицензии и получения кода для дешифрования.

Был реализован класс **SslServer** который предоставляет возможность передавать информации двухсторонно с клиентской части. Его функциональность основывается на использовании класса QSslSocket библиотеки Qt.

QSslSocket устанавливает безопасное зашифрованное TCP-соединение, которое можно использовать для передачи зашифрованных данных. Он может работать как в режиме клиента, так и в режиме сервера, и поддерживает современные протоколы SSL, включая SSL 3 и TLS 1.2.

QSslSocket имеет API для добавления обоих типов сертификатов, CA и локальных, и локального сертификата, связанного с ключом.

addCaCertificate() для сертификата CA.

setLocalCertificate() для локального сертификата.

setPrivateKey() для закрытого ключа для локального сертификата.

Основной класс **LicenseServer** который унаследован от SslServer (для передача информации) локально хранит список лицензии и кодов для дешифрования. Эти данные загружаются в памяти при его инициализации, и последующие изменения обратно записываются на диске. Были реализованы функции:

void saveStateToDisk()

void addLicense(string fileId, string license)

void setDecryptionCode(string fileId, string decryptionCode)

string getDecryptionCode(string fileId, string license)

bool isLicenseValid(string fileId, string license)

void lockFile(string fileId)



## Протокол взаимодействия с сервером

1. Добавление лицензия в списке для файл с идентификатором ID:

**ADD <ID> <LICENSE>**

2. Установка кода для дешифрования файл с идентификатором ID:

**SET <ID> <LICENSE>**

3. Заблокировать дальнейшее добавление лицензии для файл с идентификатором ID:

**LOCK <ID>**

4. Проверить, действительна ли лицензия для файл с идентификатором ID:

**CHECK <ID> <LICENSE>**

Выдается обратное сообщение в виде “**true**” или “**false**” в зависимости от результата проверки.

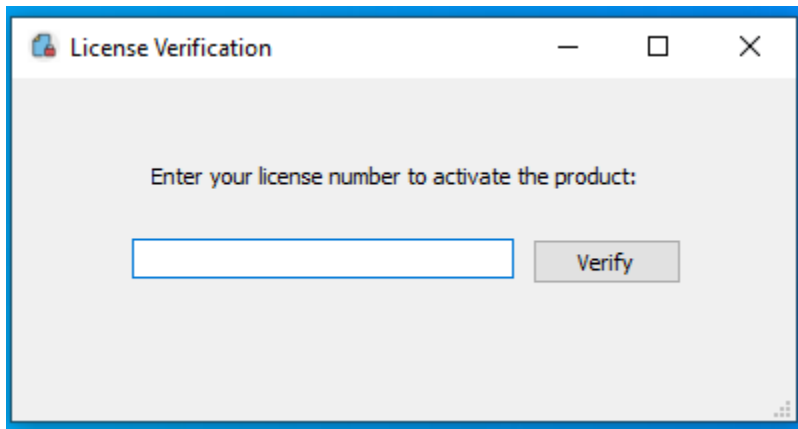
5. Получить код дешифрования для файл с идентификатором ID:

**GET <ID> <LICENSE>**

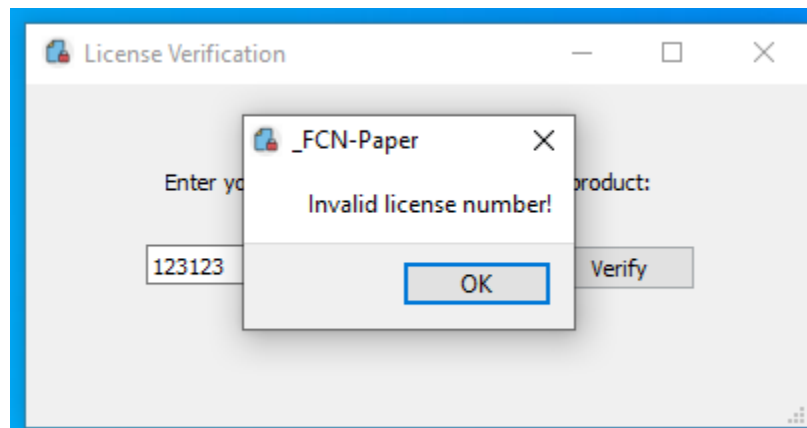
Если задана действительна лицензия, то она отмечается как уже использованная и выдается обратное сообщение которое содержит код дешифрования, в противном случае выдается сообщение “**-1**”

## LicenseClient

Было реализовано приложение с графическим интерфейсом которое может хранить вложенный зашифрованный файл и предоставляет возможность пользователю ввести номер лицензии. Лицензия проверяется на сервер, и при успешной проверке обратно получается код который используется для расшифрования и запуска вложенного файла.



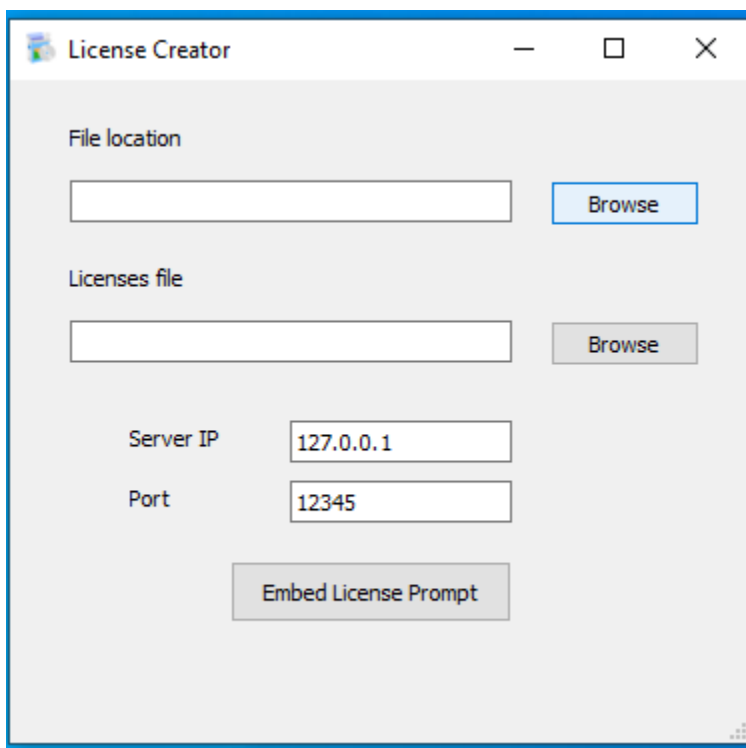
При неуспешного ввода лицензии выдается ошибка:



При усшеного ввода лицензии, сохраняется код для расшифровки в registry при помощи QSettings, создается ункиальная временная директория где вставляется расшифрованный файл и запускается с использованием QProcess или QDesktopServices::openUrl (если это не исполняемый файл). После запуска, эта директория удаляется вместе с расшифрованным файлом.

## LicenseCreator

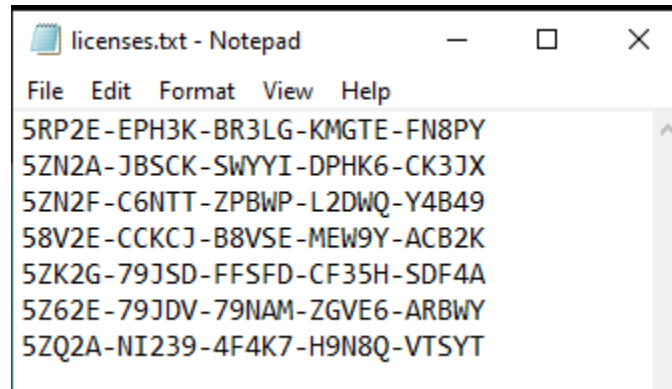
Было реализовано приложение с графическим интерфейсом которое позволяет пользователю заблокировать для использования собственный файл и внести список лицензии для его разблокировки которые отправляются на сервер вместе с кодом для расшифровки файла. Это делается по соблюдением протокола взаимодействия с сервером.



Для корректного использования, данное приложение при запуске должно находиться в одной и той же директории с приложением LicenseClient. В результате использования этого файла создается новый файл который по сути есть LicenseClient с вложенным зашифрованным пользовательским файлом.

## Пример использования

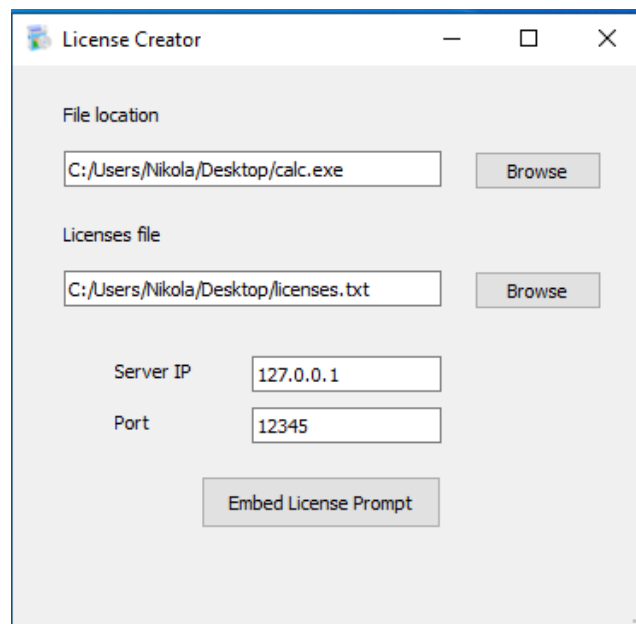
Создаем список лицензии



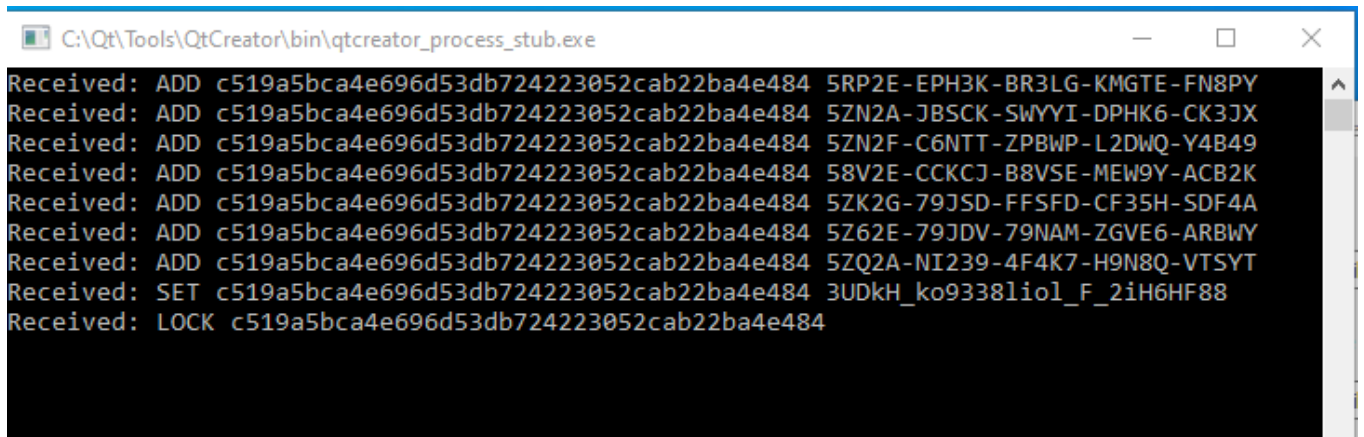
Файлы LicenseCreator и LicenseClient должны находится в одной директории:



Запускаем LicenseCreator.exe и вводим параметры:

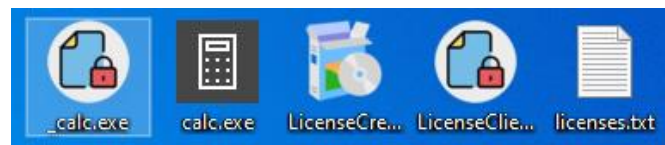


Нажимаем кнопку Embed License Prompt, при этом сервер получает следующие запросы:

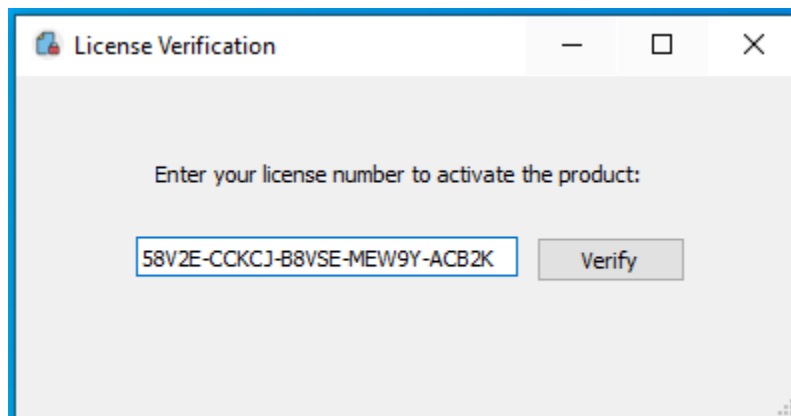


```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5RP2E-EPH3K-BR3LG-KMGTE-FN8PY
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5ZN2A-JBSCK-SWYYI-DPHK6-CK3JX
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5ZN2F-C6NTT-ZPBWP-L2DWQ-Y4B49
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 58V2E-CCKCJ-B8VSE-MEW9Y-ACB2K
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5ZK2G-79JSD-FFSFD-CF35H-SDF4A
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5Z62E-79JDV-79NAM-ZGVE6-ARBWY
Received: ADD c519a5bca4e696d53db724223052cab22ba4e484 5ZQ2A-NI239-4F4K7-H9N8Q-VTSYT
Received: SET c519a5bca4e696d53db724223052cab22ba4e484 3UDkH_ko9338liol_F_2iH6HF88
Received: LOCK c519a5bca4e696d53db724223052cab22ba4e484
```

И соответственно создается заблокированный файл:



При его запуске, открывается окно для ввода лицензии. Вводим одна из списке:



При нажатие кнопки Verify, сервер получает следующие запросы:

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Received: CHECK c519a5bca4e696d53db724223052cab22ba4e484 58V2E-CCKCJ-B8VSE-MEW9Y-ACB2K
license is valid
Received: GET c519a5bca4e696d53db724223052cab22ba4e484 58V2E-CCKCJ-B8VSE-MEW9Y-ACB2K
Client Disconnected
```

Вложенный файл дешифруется и запускается:



При последующего запуска, больше не будет требоваться ввести номер лицензии.

## Заключение

В данной курсовой работе была реализована система активации программного продукта которая удобна и довольно простая для использования. При этом использовался криптографический протокол SSL для обеспечения безопасную связь между клиент для ввода лицензии и сервером для ее подтверждения.

Было реализовано серверное приложение которое хранит список лицензии и кодов для расшифровки для разных файлов с уникальным идентификатором. Это приложение может получать запросы на добавление лицензии и кодов для расшифровки, а так же и запросы на проверки лицензии, и обратное получения кода для расшифровки по номера лицензии.

Было реализовано приложение с графическим интерфейсом которое может хранить вложенный зашифрованный файл и предоставляет возможность пользователю ввести номер лицензии. Лицензия проверяется на сервер, и при успешной проверки обратно получается код который используется для расшифрования и запуска вложенного файла.

Было реализовано приложение с графическим интерфейсом которое позволяет пользователю заблокировать для использования собственный файл и внести список лицензии для эго разблокировки которые отправляются на сервер вместе с кодом для расшифровки файла.

## Список использованной литературы

1. Описание протоколов SSL/TLS // 3. ООО "КРИПТО-ПРО"., 2002. — P. 49.
2. *P. Karlton*. The Secure Sockets Layer (SSL) Protocol Version 3.0 // 1-st. — RTFM, Inc., August 2011. — № 1. — P. 67.
3. <https://doc.qt.io/qt-5/qtnetwork-index.html>
4. <https://doc.qt.io/qt-5/qcryptographicichash.html>
5. <https://doc.qt.io/qt-5/qsslsocket.html#details>
6. [http://www.infidigm.net/articles/qsslsocket\\_for\\_ssl\\_beginners/](http://www.infidigm.net/articles/qsslsocket_for_ssl_beginners/)
7. <https://github.com/jbagg/QSslSocket-example>
8. <https://www.openssl.org/docs/>
9. [https://en.wikipedia.org/wiki/Product\\_activation](https://en.wikipedia.org/wiki/Product_activation)
10. <https://ru.wikipedia.org/wiki/SSL>
11. <https://ru.wikipedia.org/wiki/OpenSSL>



# Приложение 1. Текст программы LicenseServer

## LicenseServer.h

```
#ifndef LICENSESERVER_H
#define LICENSESERVER_H

#include "SslServer.h"
#include <map>
#include <string>

class LicenseServer: public SslServer
{
    Q_OBJECT

public:
    LicenseServer(QObject *parent = nullptr);
    virtual ~LicenseServer() override = default;

protected slots:
    virtual void rx() override;

private:
    const std::string licensesFilename_ = "licenses.dat";
    const std::string codesFilename_ = "codes.dat";
    const std::string locksFilename_ = "locks.dat";

    std::map<std::string, std::map<std::string, bool>> licenses_;
    std::map<std::string, std::string> decryptionCodes_;
    std::map<std::string, bool> locks_;

    void saveStateToDisk();
    void addLicense(const std::string& fileId, const std::string& license);
    void setDecryptionCode(const std::string& fileId, const std::string&
decryptionCode);
    std::string getDecryptionCode(const std::string& fileId, const
std::string& license);
    bool isLicenseValid(const std::string& fileId, const std::string&
license);
    void lockFile(const std::string& fileId);

};

#endif // LICENSESERVER_H
```

## SslServer.h

```
#ifndef SSLSERVER_H
```

```

#define SSLSERVER_H

#include <QTcpServer>
#include <QSslKey>
#include <QSslCertificate>

class SslServer : public QTcpServer
{
    Q_OBJECT

public:
    SslServer(QObject *parent = nullptr);
    virtual ~SslServer() = default;

protected slots:
    virtual void rx();

protected:
    void incomingConnection(qintptr socketDescriptor);

private slots:
    void sslErrors(const QList<QSslError> &errors);
    void link();
    void disconnected();

private:
    QSslKey key;
    QSslCertificate cert;

};

#endif // SSLSERVER_H

```

## LicenseServer.cpp

```

#include "LicenseServer.h"
#include <QFile>
#include <QTcpSocket>
#include <sstream>
#include <string>
#include <fstream>

#include <iostream>

LicenseServer::LicenseServer(QObject *parent):
    SslServer(parent)
{
    //Load licenses file, entries in file have format "<ID> <LICENSE> [0|1]"
    std::ifstream inpLicenses(licensesFilename_);
    std::string line;
    while (std::getline(inpLicenses, line))
    {

```

```

        std::istringstream iss(line);

        std::string fileId, license, isValid;
        iss >> fileId >> license >> isValid;

        licenses_[fileId][license] = (isValid == std::string("1"));
    }

    //Load decryption codes file, entries in file have format "<ID> <CODE>"
    std::ifstream inpCodes(codesFilename_);
    while(std::getline(inpCodes, line))
    {
        std::istringstream iss(line);

        std::string fileId, decryptionCode;
        iss >> fileId >> decryptionCode;

        decryptionCodes_[fileId] = decryptionCode;
    }

    //Load fileId locks
    std::ifstream inpLocks(locksFilename_);
    while(std::getline(inpLocks, line))
    {
        std::istringstream iss(line);

        std::string fileId;
        bool isLocked;

        iss >> fileId >> isLocked;

        locks_[fileId] = isLocked;
    }
}

void LicenseServer::rx()
{
    QTcpSocket* clientSocket = qobject_cast<QTcpSocket*>(sender());

    while(clientSocket->canReadLine())
    {
        std::string line = clientSocket->readLine().toStdString();
        std::cout << "Received: " << line;

        std::istringstream iss(line);

        std::string command, fileId;
        iss >> command >> fileId;

        //std::cout << "command = " << command << std::endl;
        //std::cout << "fileId = " << fileId << std::endl;

        if(command == "ADD")
        {
            std::string license;
            iss >> license;

```

```

        //std::cout << "license = " << license << std::endl;
        addLicense(fileId, license);
    }
    else if(command == "SET")
    {
        std::string decryptionCode;
        iss >> decryptionCode;

        //std::cout << "decryptionCode = " << decryptionCode <<
std::endl;
        setDecryptionCode(fileId, decryptionCode);
    }
    else if(command == "CHECK")
    {
        std::string license;
        iss >> license;

        //std::cout << "license = " << license << std::endl;

        if(isLicenseValid(fileId, license))
        {
            std::cout << "license is valid" << std::endl;
            clientSocket->write("true\n");
        }
        else
        {
            std::cout << "license is not valid" << std::endl;
            clientSocket->write("false\n");
        }
    }
    else if(command == "GET")
    {
        std::string license;
        iss >> license;

        //std::cout << "license = " << license << std::endl;

        std::string decryptionCode = getDecryptionCode(fileId, license);

        //std::cout << "m_decryptionCode = " << decryptionCode <<
std::endl;
        clientSocket->write(std::string(decryptionCode + "\n").c_str());
    }
    else if(command == "LOCK")
    {
        lockFile(fileId);
    }
    else
    {
        clientSocket->write("invalid request\n");
    }
}

}

void LicenseServer::saveStateToDisk()
{

```

```

//write licenses file
std::ofstream outLicenses(licensesFilename_);
for(auto it = licenses_.begin(); it != licenses_.end(); it++)
{
    for(auto it2 = it->second.begin(); it2 != it->second.end(); it2++)
    {
        outLicenses << it->first << " " << it2->first << " " << it2-
>second << std::endl;
    }
    outLicenses.flush();
    outLicenses.close();

//write decryption codes file
std::ofstream outCodes(codesFilename_);
for(auto it = decryptionCodes_.begin(); it != decryptionCodes_.end();
it++)
{
    outCodes << it->first << " " << it->second << std::endl;
}
outCodes.flush();
outCodes.close();

//write file locsk
std::ofstream outLocks(locksFilename_);
for(auto it = locks_.begin(); it != locks_.end(); it++)
{
    outLocks << it->first << " " << it->second << std::endl;
}
outLocks.flush();
outLocks.close();
}

void LicenseServer::addLicense(const std::string &fileId, const std::string
&license)
{
    if(!locks_[fileId])
    {
        licenses_[fileId][license] = true;
    }
}

void LicenseServer::setDecryptionCode(const std::string &fileId, const
std::string &decryptionCode)
{
    if(!locks_[fileId])
    {
        decryptionCodes_[fileId] = decryptionCode;
    }
}

std::string LicenseServer::getDecryptionCode(const std::string &fileId, const
std::string &license)
{
    if(isLicenseValid(fileId, license))
    {

```

```

        //invalidate license
        licenses_[fileId][license] = false;

        return decryptionCodes_[fileId];
    }
    return std::string("-1");
}

bool LicenseServer::isLicenseValid(const std::string &fileId, const
std::string &license)
{
    return licenses_[fileId][license];
}

void LicenseServer::lockFile(const std::string &fileId)
{
    locks_[fileId] = true;
    saveStateToDisk();
}

```

## SslServer.cpp

```

#include "SslServer.h"
#include <QSslSocket>
#include <QFile>

SslServer::SslServer(QObject *parent) : QTcpServer(parent)
{
    QFile keyFile("C:/ssl_certificates/red_local.key");
    keyFile.open(QIODevice::ReadOnly);
    key = QSslKey(keyFile.readAll(), QSsl::Rsa);
    keyFile.close();

    QFile certFile("C:/ssl_certificates/red_local.pem");
    certFile.open(QIODevice::ReadOnly);
    cert = QSslCertificate(certFile.readAll());
    certFile.close();

    if (!listen(QHostAddress("127.0.0.1"), 12345))
    {
        // FQDN in red_local.pem is set to 127.0.0.1. If you change this, it
        will not authenticate.
        qCritical() << "Unable to start the TCP server";
        exit(0);
    }
    connect(this, &SslServer::newConnection, this, &SslServer::link);
}

void SslServer::incomingConnection(qintptr socketDescriptor)
{
    QSslSocket *sslSocket = new QSslSocket(this);

    connect(sslSocket, SIGNAL(sslErrors(QList<QSslError>)), this,
    SLOT(sslErrors(QList<QSslError>)));
    sslSocket->setSocketDescriptor(socketDescriptor);
    sslSocket->setPrivateKey(key);
}

```

```

        sslSocket->setLocalCertificate(cert);
        sslSocket->addCaCertificates("C:/ssl_certificates/blue_ca.pem");
        sslSocket->setPeerVerifyMode(QSslSocket::VerifyPeer);
        sslSocket->startServerEncryption();

        addPendingConnection(sslSocket);
    }

void SslServer::sslErrors(const QList<QSslError> &errors)
{
    foreach (const QSslError &error, errors)
        qDebug() << error.errorString();
}

void SslServer::link()
{
    QTcpSocket *clientSocket;

    clientSocket = nextPendingConnection();
    connect(clientSocket, &QTcpSocket::readyRead, this, &SslServer::rx);
    connect(clientSocket, &QTcpSocket::disconnected, this,
&SslServer::disconnected);
}

void SslServer::rx()
{
}

void SslServer::disconnected()
{
    qDebug("Client Disconnected");
    QTcpSocket* clientSocket = qobject_cast<QTcpSocket*>(sender());
    clientSocket->deleteLater();
}

```

## Main.cpp

```

#include <QCoreApplication>
//#include "licenseserver.h"
#include "LicenseServer.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    LicenseServer server;

    return a.exec();
}

```

## Приложение 2. Текст программы LicenseClient

## LicenseManager.h

```
#ifndef LICENSEMANAGER_H
#define LICENSEMANAGER_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <QProcess>
#include "SslClient.h"

class LicenseManager
{
public:
    LicenseManager(const std::string& selfPath);

    bool isAlreadyVerified();
    bool verifyLicense(const std::string& license);
    std::string getDecryptionCode(const std::string& license);
    std::string getSavedDecryptionCode();
    void decryptAndRun(const std::string& decryptionCode);

private:
    SslClient client_;
    QProcess *process;

    std::string selfPath_;
    std::string fileId_;

    std::string getFileId(const std::string& filepath);
    std::string getFileNameFromPath(const std::string& filepath);
    std::string getFileExtension(const std::string& filepath);
    std::vector<char> readAllBytes(const std::string& filename);
    void decryptRange(std::vector<char> &file, const std::string
&decryptionKey, size_t startPos, size_t endPos);

};

#endif // LICENSEMANAGER_H
```

## MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLineEdit>
#include <QTextEdit>
#include "licensemanager.h"

QT_BEGIN_NAMESPACE
```



```

namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(LicenseManager &licenseManager, QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

    LicenseManager &licenseManager_;

    QLineEdit *licenseLineEdit;
    QPushButton *verifyPushButton;

    QPushButton *queryServerPushButton;

};
#endif // MAINWINDOW_H

```

## SslClient.h

```

#ifndef SSLCLIENT_H
#define SSLCLIENT_H

#include <QSocket>

//http://www.infidigm.net/articles/qsslsocket_for_ssl_beginners/
class SslClient: public QObject
{
    Q_OBJECT

public:
    SslClient();
    void connectToServer();
    void sendData(const QString &data);
    QString readData();

private:
    QSocket server;

Q_SIGNALS:
    void disconnected(void);

private slots:
    void sslErrors(const QList<QSocketError> &errors);
    void rx(void);
    void serverDisconnect(void);

};

```

```
#endif // SSLCLIENT_H
```

## LicenseManager.cpp

```
#include "licensemanager.h"
#include <QTemporaryDir>
#include <QProcess>
#include <QString>
#include <QFile>
#include <QFileInfo>
#include <QUrl>
#include <QDesktopServices>
#include <QCryptographicHash>
#include <fstream>
#include <QSettings>
#include "windows.h"

LicenseManager::LicenseManager(const std::string &selfPath):
    selfPath_(selfPath)
{
    fileId_ = getFileId(selfPath);
    client_.connectToServer();
    process = new QProcess();
}

bool LicenseManager::isAlreadyVerified()
{
    if(getSavedDecryptionCode().size() > 1)
    {
        return true;
    }
    return false;
}

bool LicenseManager::verifyLicense(const std::string &license)
{
    QString query = "CHECK %1 %2\n";
    client_.sendData(query.arg(fileId_.c_str()).arg(license.c_str()));

    QString ans = client_.readData();
    return ans == "true\n";
}

std::string LicenseManager::getDecryptionCode(const std::string &license)
{
    QString query = "GET %1 %2\n";
    client_.sendData(query.arg(fileId_.c_str()).arg(license.c_str()));
    std::string decryptionCode = client_.readData().toStdString();
    decryptionCode = decryptionCode.substr(0, decryptionCode.size() - 1);

    qDebug() << "got decryption code = " << decryptionCode.c_str();

    //Add decryptionCode to registry
    QSettings settings("LicenseManager", fileId_.c_str());
    settings.setValue("DecryptionCode",
        QString::fromStdString(decryptionCode));
}
```

```

        return decryptionCode;
    }

std::string LicenseManager::getSavedDecryptionCode()
{
    std::string fileId = getFileId(selfPath_);
    QSettings settings("LicenseManager", fileId.c_str());
    return settings.value("DecryptionCode").toString().toStdString();
}

void LicenseManager::decryptAndRun(const std::string& decryptionCode)
{
    std::ofstream log("licenseClient_log.txt");

    std::string filename = getFileNameFromPath(selfPath_);
    log << "filename = " << filename << std::endl;

    //Read file into array
    std::vector<char> fileMe = readAllBytes(selfPath_);
    const size_t mySize = fileMe.size();
    log << "mySize = " << mySize << std::endl;

    //Get embedded file start position and size
    size_t startPos;
    memcpy(&startPos, &fileMe[mySize - sizeof(size_t)], sizeof(size_t));
    size_t embeddedFileSize = mySize - sizeof(size_t) - startPos;
    log << "startPos = " << startPos << std::endl;
    log << "embeddedFileSize = " << embeddedFileSize << std::endl;

    //Get embedded file extension
    size_t fileExtSize;
    memcpy(&fileExtSize, &fileMe[mySize - 2*sizeof(size_t)], sizeof(size_t));
    log << fileExtSize << std::endl;
    std::vector<char> fileExtension_c(fileExtSize);
    memcpy(&fileExtension_c[0], &fileMe[mySize - 2*sizeof(size_t) -
fileExtSize], fileExtSize);
    std::string fileExtension(fileExtension_c.begin(),
fileExtension_c.end());
    log << "fileExtension = " << fileExtension << std::endl;

    //Decrypt file
    decryptRange(fileMe, decryptionCode, startPos, startPos +
embeddedFileSize);
    log << "decrypted" << std::endl;

    //Write file to temp location
    QTemporaryDir dir;
    std::string tempDirPath = dir.path().toStdString();
    std::string outFilePath = tempDirPath + "/" + filename;
    log << "tempDirPath = " << tempDirPath << std::endl;
    log << "outFilePath = " << outFilePath << std::endl;
    outFilePath = outFilePath.substr(0, outFilePath.size()-3) +
fileExtension;
    log << "changed ext outFilePath = " << outFilePath << std::endl;
}

```

```

std::ofstream fout(outFilePath, std::ios::binary | std::ios::out);
fout.write((char*)&fileMe[startPos], embeddedFileSize);
fout.flush();
fout.close();
log << "file written to disk" << std::endl;

//Run file
if(fileExtension == "exe")
{
    int ret = process->execute(QString::fromStdString(outFilePath));
    log << "outFilePath parameter = " << outFilePath << std::endl;
    log << "process->execute ret code = " << ret << std::endl;
    process->waitForFinished(-1);
}
else
{
    QDesktopServices::openUrl(QUrl::fromUserInput(QString::fromStdString(outFilePath)));
    Sleep(10000);
}

//Delete decrypted file and its temp directory
QFile file(outFilePath.c_str());
file.remove();
dir.remove();

log.flush();
log.close();

//Quit qt application
exit(0);
}

std::string LicenseManager::getFileId(const std::string &filepath)
{
    QFile f(QString::fromStdString(filepath));
    if (f.open(QFile::ReadOnly)) {
        QCryptographicHash hash(QCryptographicHash::Algorithm::Sha1);
        if (hash.addData(&f)) {
            return hash.result().toHex().toStdString();
        }
    }
    return QByteArray().toHex().toStdString();
}

std::vector<char> LicenseManager::readAllBytes(const std::string &filename)
{
    std::ifstream ifs(filename, std::ios::binary|std::ios::ate);
    std::ifstream::pos_type pos = ifs.tellg();

    std::vector<char> result(pos);

    ifs.seekg(0, std::ios::beg);

```

```

        ifs.read(&result[0], pos);

        return result;
    }

void LicenseManager::decryptRange(std::vector<char> &file, const std::string
&decryptionKey,
                                size_t startPos, size_t endPos)
{
    for(size_t i = startPos; i < endPos; i++)
    {
        for(int j = decryptionKey.size() - 1; j >= 0; j--)
        {
            file[i] ^= decryptionKey[j];
        }
    }
}

std::string LicenseManager::getFileNameFromPath(const std::string &filepath)
{
    std::string filename = filepath;
    const size_t last_slash_idx = filename.find_last_of("\\\\/");
    if (std::string::npos != last_slash_idx)
    {
        filename.erase(0, last_slash_idx + 1);
    }
    return filename;
}

std::string LicenseManager::getFileExtension(const std::string &filepath)
{
    if(filepath.find_last_of(".") != std::string::npos)
        return filepath.substr(filepath.find_last_of(".") + 1);
    return "";
}

```

## MainWindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QDesktopServices>
#include <QDebug>
#include <QFile>
#include <string>
#include <windows.h>
#include "SslClient.h"

MainWindow::MainWindow(LicenseManager &licenseManager, QWidget *parent):
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    licenseManager_(licenseManager)
{
    ui->setupUi(this);
}

```

```

setWindowTitle("License Verification");
setWindowIcon(QIcon("license_icon.ico"));

verifyPushButton = findChild<QPushButton*>("verifyPushButton");
licenseLineEdit = findChild<QLineEdit*>("licenseLineEdit");

connect(verifyPushButton, &QPushButton::clicked, [&]() {

    std::string license = licenseLineEdit->text().toStdString();
    if(licenseManager.verifyLicense(license))
    {
        std::string decryptionCode =
licenseManager_.getDecryptionCode(license);
        hide();
        licenseManager_.decryptAndRun(decryptionCode);
    }
    else
    {
        QMessageBox messageBox;
        messageBox.setText("Invalid license number!");
        messageBox.exec();
    }
});
}

MainWindow::~MainWindow()
{
    delete ui;
}

```

## SslClient.cpp

```

#include "SslClient.h"
#include <QSSLKey>
#include <QSSLCertificate>

SslClient::SslClient()
{
    //connect(&server, &QSslSocket::readyRead, this, &SslClient::rx);
    connect(&server, &QSslSocket::disconnected, this,
&SslClient::serverDisconnect);
    connect(&server, SIGNAL(sslErrors(QList<QSslError>)), this,
SLOT(sslErrors(QList<QSslError>)));
    server.addCaCertificates("C:/ssl_certificates/red_ca.pem");
    server.setPrivateKey("C:/ssl_certificates/blue_local.key");
    server.setLocalCertificate("C:/ssl_certificates/blue_local.pem");
    server.setPeerVerifyMode(QSslSocket::VerifyPeer);
}

void SslClient::connectToServer()
{
    // FQDN in red_local.pem is set to 127.0.0.1.
    //If you change this, it will not authenticate.

    server.connectToHostEncrypted("127.0.0.1", 12345);
}

```

```

        if (server.waitForEncrypted(5000))
        {
            qDebug() << "Authentication Succeeded";
        }
        else
        {
            qDebug() << "Unable to connect to server";
            exit(1);
        }
    }

void SslClient::sendData(const QString &data)
{
    server.write(data.toStdString().c_str());
}

QString SslClient::readData()
{
    if(server.waitForReadyRead())
        return server.readAll();
    return QString();
}

void SslClient::sslErrors(const QList<QSslError> &errors)
{
    foreach (const QSslError &error, errors)
        qDebug() << error.errorString();
}

void SslClient::serverDisconnect(void)
{
    qDebug("Server disconnected");
    exit(0);
}

void SslClient::rx(void)
{
    //rx_ready = true;
    //server.write("ADD 12345-ABCDEF-GHIJK-67890");
    //qDebug() << server.readAll();
}

```

## main.cpp

```

#include "mainwindow.h"

#include <QApplication>
#include <string>
#include "licensemanager.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    LicenseManager licenseManager(argv[0]);
}

```

```

MainWindow w(licenseManager);

if(licenseManager.isAlreadyVerified())
{
    std::string decryptionCode = licenseManager.getSavedDecryptionCode();
    licenseManager.decryptAndRun(decryptionCode);
}
else
{
    w.show();
}

return a.exec();
}

```

## Приложение 3. Текст программы LicenseCreator

### MainWindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

#include <QPushButton>
#include <QLineEdit>
#include <string>
#include <vector>
#include "SslClient.h"

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

    SslClient client_;

    QString filePath_;
    QString licensesPath_;
    std::string fileId_;
    std::string encryptionKey_;
}

```



```

QPushButton* filePathButton;
QPushButton* licensesPathButton;
QPushButton* embedButton;
QLineEdit* filePathLineEdit;
QLineEdit* licensesPathLineEdit;
QLineEdit* serverIpLineEdit;
QLineEdit* serverPortLineEdit;

void registerFileLicenses();
std::vector<char> readAllBytes(const std::string& filename);
void encrypt(std::vector<char> &vec, const std::string& encryptionKey);
std::string getNameFromPath(const std::string& filepath);
std::string getFileId(const std::string& filepath);
std::string generateRandomEncryptionKey();
std::string getFileExtension(const std::string &filepath);

};
#endif // MAINWINDOW_H

```

## SslClient.h

```

#ifndef SSLCLIENT_H
#define SSLCLIENT_H

#include <QSocket>

//http://www.infidigm.net/articles/qsslsocket_for_ssl_beginners/
class SslClient: public QObject
{
    Q_OBJECT

public:
    SslClient();
    void connectToServer();
    void sendData(const QString &data);
    QString readData();

private:
    QSocket server;

Q_SIGNALS:
    void disconnected(void);

private slots:
    void sslErrors(const QList<QSocketError> &errors);
    void rx(void);
    void serverDisconnect(void);

};

#endif // SSLCLIENT_H

```

## MainWindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

```

```

#include <QFileDialog>
#include <QMessageBox>
#include <QStringList>
#include <QCryptographicHash>
#include <QDebug>
#include <QTime>
#include <fstream>
#include <sstream>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    setWindowTitle("License Creator");
    setWindowIcon(QIcon("license_icon.ico"));

    filePathButton = findChild<QPushButton*>("filePathButton");
    licensesPathButton = findChild<QPushButton*>("licensesPathButton");
    embedButton = findChild<QPushButton*>("embedButton");
    filePathLineEdit = findChild<QLineEdit*>("filePathLineEdit");
    licensesPathLineEdit = findChild<QLineEdit*>("licensesPathLineEdit");
    serverIpLineEdit = findChild<QLineEdit*>("serverIpLineEdit");
    serverPortLineEdit = findChild<QLineEdit*>("serverPortLineEdit");

    filePathLineEdit->setReadOnly(true);
    licensesPathLineEdit->setReadOnly(true);
    serverIpLineEdit->setReadOnly(true);
    serverPortLineEdit->setReadOnly(true);

    connect(filePathButton, &QPushButton::clicked, [&]() {
        QFileDialog dialog;
        dialog.setFileMode(QFileDialog::AnyFile);
        QStringList fileNames;
        if(dialog.exec())
        {
            fileNames = dialog.selectedFiles();
            filePath_ = fileNames.first();
            filePathLineEdit->setText(filePath_);
        }
    });

    connect(licensesPathButton, &QPushButton::clicked, [&]() {
        QFileDialog dialog;
        dialog.setFileMode(QFileDialog::AnyFile);
        QStringList fileNames;
        if(dialog.exec())
        {
            fileNames = dialog.selectedFiles();
            licensesPath_ = fileNames.first();
            licensesPathLineEdit->setText(licensesPath_);
        }
    });

    connect(embedButton, &QPushButton::clicked, [&]() {

```

```

        std::string licenseClientFilePath = "LicenseClient.exe";
        std::string userFilePath = filePath_.toStdString();
        encryptionKey_ = generateRandomEncryptionKey();

        std::vector<char> licenseClientFile =
readAllBytes(licenseClientFilePath);
        std::vector<char> userFile = readAllBytes(userFilePath);
        encrypt(userFile, encryptionKey_);

        std::string ext = getFileExtension(userFilePath);
        std::vector<char> fileExtension(ext.begin(), ext.end());
        size_t fileExtSize = fileExtension.size();
        std::vector<char> fileExtSize_c(sizeof(size_t));
        memcpy(&fileExtSize_c[0], &fileExtSize, sizeof(size_t));

        size_t startPos = licenseClientFile.size();
        std::vector<char> startPos_c(sizeof(size_t));
        memcpy(&startPos_c[0], &startPos, sizeof(size_t));

        std::string outFileName = "_" + getFileNameFromPath(userFilePath);
        std::string outExt = getFileExtension(outFileName);
        if(outExt != "exe")
        {
            outFileName = outFileName.substr(0, outFileName.size() -
outExt.size()) + "exe";
        }

        std::ofstream fout(outFileName, std::ios::binary | std::ios::out);
        fout.write((char*) &licenseClientFile[0], licenseClientFile.size());
        fout.write((char*) &userFile[0], userFile.size());
        fout.write((char*) &fileExtension[0], fileExtension.size());
        fout.write((char*) &fileExtSize_c[0], fileExtSize_c.size());
        fout.write((char*) &startPos_c[0], startPos_c.size());
        fout.close();

        fileId_ = getFileId(outFileName);
        licensesPath_ = licensesPathLineEdit->text();
        registerFileLicenses();

        QMessageBox messageBox;
        messageBox.setText("Sucess!");
        messageBox.exec();
    });
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::registerFileLicenses()
{
    client_.connectToServer();

    //Set licenses
    std::ifstream inpLicenses(licensesPath_.toStdString());

```

```

std::string line;
while (std::getline(inpLicenses, line))
{
    QString query = "ADD %1 %2\n";
    client_.sendData(query.arg(fileId_.c_str()).arg(line.c_str()));
}

//Set decryption code
QString query = "SET %1 %2\n";
client_.sendData(query.arg(fileId_.c_str()).arg(encryptionKey_.c_str()));

//Lock file
query = "LOCK %1\n";
client_.sendData(query.arg(fileId_.c_str()));
}

std::vector<char> MainWindow::readAllBytes(const std::string &filename)
{
    std::ifstream ifs(filename, std::ios::binary|std::ios::ate);
    std::ifstream::pos_type pos = ifs.tellg();

    std::vector<char> result(pos);

    ifs.seekg(0, std::ios::beg);
    ifs.read(&result[0], pos);

    return result;
}

void MainWindow::encrypt(std::vector<char> &vec, const std::string
&encryptionKey)
{
    for(size_t i = 0; i < vec.size(); i++)
    {
        for(size_t j = 0; j < encryptionKey.size(); j++)
        {
            vec[i] ^= encryptionKey[j];
        }
    }
}

std::string MainWindow::getFileNameFromPath(const std::string &filepath)
{
    std::string filename = filepath;
    const size_t last_slash_idx = filename.find_last_of("\\\\/");
    if (std::string::npos != last_slash_idx)
    {
        filename.erase(0, last_slash_idx + 1);
    }
    return filename;
}

std::string MainWindow::getFileId(const std::string &filepath)
{
    QFile f(QString::fromStdString(filepath));
    if (f.open(QFile::ReadOnly)) {

```

```

        QCryptographicHash hash(QCryptographicHash::Algorithm::Sha1);
        if (hash.addData(&f)) {
            return hash.result().toHex().toStdString();
        }
    }
    return QByteArray().toHex().toStdString();
}

std::string MainWindow::generateRandomEncryptionKey()
{
    qsrand(QTime::currentTime().msec());

    std::string seed = "X683nFHF8_9iF83kDUI_HoHl2a";
    int keyLength = 10 + qrand() % 20;
    std::string encryptionKey;
    for(int i = 0; i < keyLength; i++)
    {
        int idx = qrand() % (seed.length() - 1);
        encryptionKey += seed[idx];
    }
    return encryptionKey;
}

std::string MainWindow::getFileExtension(const std::string &filepath)
{
    if(filepath.find_last_of(".") != std::string::npos)
        return filepath.substr(filepath.find_last_of(".") + 1);
    return "";
}

```

## SslClient.cpp

```

#include "SslClient.h"
#include <QSslKey>
#include <QSslCertificate>

SslClient::SslClient()
{
    //connect(&server, &QSslSocket::readyRead, this, &SslClient::rx);
    connect(&server, &QSslSocket::disconnected, this,
    &SslClient::serverDisconnect);
    connect(&server, SIGNAL(sslErrors(QList<QSslError>)), this,
    SLOT(sslErrors(QList<QSslError>)));
    server.addCaCertificates("C:/ssl_certificates/red_ca.pem");
    server.setPrivateKey("C:/ssl_certificates/blue_local.key");
    server.setLocalCertificate("C:/ssl_certificates/blue_local.pem");
    server.setPeerVerifyMode(QSslSocket::VerifyPeer);
}

void SslClient::connectToServer()
{
    server.connectToHostEncrypted("127.0.0.1", 12345);

    if (server.waitForEncrypted(5000))
    {
        qDebug() << "Authentication Succeeded";
    }
}

```

```

    }
    else
    {
        qDebug() << "Unable to connect to server";
        exit(1);
    }
}

void SslClient::sendData(const QString &data)
{
    server.write(data.toStdString().c_str());
}

QString SslClient::readData()
{
    if(server.waitForReadyRead())
        return server.readAll();
    return QString();
}

void SslClient::sslErrors(const QList<QSslError> &errors)
{
    foreach (const QSslError &error, errors)
        qDebug() << error.errorString();
}

void SslClient::serverDisconnect(void)
{
    qDebug("Server disconnected");
    exit(0);
}

void SslClient::rx(void)
{
}

```

## Main.cpp

```

#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```