МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет приборостроения, информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 100503 —	Информационная безопасность автоматизированных систем
Специализация —	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МАКЕТА УДОСТОВЕРЯЮЩЕГО ЦЕНТРА С ИСПОЛЬЗОВАНИЕМ КРИПТОСИСТЕМЫ НИДЕРРАЙТЕРА НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ С/С++

Студент _____ Ротанин Илья Андреевич

Руководитель _____ Акмаев А. Ж.

Нормконтроллер _____ Иванов А. П.

Рецензент зам. ген. директора АО «ПНИЭИ» по спецвопросам

_____ Сомкин С. А.

Работа допущена к защите (протокол заседания кафедры от 200617 № 14)

Заведующий кафедрой _____ С. Л. Зефиров

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 230617 № 13)

Секретарь ГЭК _____ А. П. Иванов

Пенза 2017

РЕФЕРАТ

Пояснительная записка содержит 123 с., 25 рис., 2 табл., 11 источников, 11 прил.

ЦЕНТР СЕРТИФИКАЦИИ, ПОСТКВАНТОВАЯ КРИПТОГРАФИЯ, СЕРТИФИКАТ ОТКРЫТОГО КЛЮЧА, ЭЛЕКТРОННАЯ ПОДПИСЬ, ИНФРАСТРУКТУРА ОТКРЫТОГО КЛЮЧА

Объектом разработки данного дипломного проекта является макет удостоверяющего центра.

Цель дипломного проекта — разработка макета удостоверяющего центра с использованием криптосистемы Нидеррайтера.

В процессе работы были расмотрены инфраструктура открытых ключей, квантовые атаки на алгоритмы открытого распределения ключей и постквантовые алгоритмы открытого распределения ключей, был разработан алгоритм удостоверяющего центра и выполнена его программная реализация, так же была выполнена программная реализация формирования цифровой подписи по криптосистеме Нидеррайтера.

В результате дипломного проектирования был разработан макет удостоверяющего центра по модели клиент-сервер с использованием криптосистемы Нидеррайтера.

СОДЕРЖАНИЕ

Введение 5
1 Обзор инфраструктуры открытых ключей 6
1.1 Понятие инфраструктуры открытых ключей 6
1.2 Принцип работы инфраструктуры открытых ключей 7
1.3 Варианты архитектурного построения инфраструктуры открытых ключей 8
1.4 Структура электронного сертификата ключа проверки электронной подписи 14
1.5 Использование инфраструктуры открытых ключей 19
2 Обзор квантовых атак на алгоритмы, лежащие в основе открытого распределения ключей 21
3 Обзор постквантовых алгоритмов открытого распределения ключей 23
3.1 Основные подходы, решающие проблему квантовых атак . . 23
3.2 Криптосистема McEliece 24
3.3 Криптосистема Нидеррайтера 26
4 Разработка алгоритма работы макета удостоверяющего центра 31
4.1 Разработка алгоритма работы серверной части макета удостоверяющего центра 31
4.2 Разработка алгоритма работы клиентской части макета удостоверяющего центра 40
5 Программная реализация алгоритма макета удостоверяющего центра 43
5.1 Программная реализация алгоритма серверной части макета удостоверяющего центра 43
5.2 Программная реализация алгоритма клиентской части макета удостоверяющего центра 50
6 Руководство по использованию макета удостоверяющего центра . . . 55
6.1 Руководство по использованию серверной части 55
6.2 Руководство по использованию клиентской части 60

Заключение 64
Список использованных источников 65
Приложение А Содержимое файла servermain.cpp 67
Приложение Б Содержимое файла TcpServer.h 69
Приложение В Содержимое файла TcpServer.cpp 71
Приложение Г Содержимое файла x509cert.h 82
Приложение Д Содержимое файла x509cert.cpp 85
Приложение Е Содержимое файла structkey.h 96
Приложение Ж Содержимое файла niedercrypt.h 98
Приложение И Содержимое файла niedercrypt.cpp 101
Приложение К Содержимое файла clientmain.cpp 112
Приложение Л Содержимое файла mainwindow.h 114
Приложение М Содержимое файла mainwindow.cpp 116

ВВЕДЕНИЕ

Инфраструктура открытых ключей широко используется для обеспечения надежной работы информационных систем и для безопасного обмена данными между пользователями. Но многие используемые асимметричные криптографические алгоритмы являются потенциально уязвимыми к атакам, реализуемым на квантовых компьютерах.

Учитывая, что с каждым годом исследователи объявляют о новых достижениях в области квантовых технологий, то вероятность создания квантовых компьютеров, превосходящих по производительности классические, все больше. В связи с этим стоит задуматься о переводе инфраструктуры открытых ключей на постквантовые криптографические алгоритмы.

Данная выпускная квалификационная работа посвящена разработке макета удостоверяющего центра с использованием криптосистемы Нидеррайтера.

Первый этап дипломного проекта посвящен обзору инфраструктуры открытых ключей.

Во втором этапе выполнен обзор квантовых атак на асимметричные криптографические алгоритмы.

Третий этап посвящен обзору постквантовых алгоритмов открытого распределения ключей.

На четвертом этапе был разработан алгоритм работы макета удостоверяющего центра.

На пятом этапе выполнена программная реализация алгоритма работы макета удостоверяющего центра.

На шестом этапе было написано руководство по использованию макета удостоверяющего центра.

1 Обзор инфраструктуры открытых ключей

1.1 Понятие инфраструктуры открытых ключей

Большие возможности использования в информационных технологиях криптографии представляет криптосистема с открытым ключом. В криптосистеме такого типа используется ключевая пара: секретный ключ и открытый ключ. Ключи из ключевой пары являются комплементарными друг к другу: зашифрованное сообщение открытым ключом можно расшифровать только секретным ключом из этой же ключевой пары, а электронная подпись, созданную с помощью секретного ключа (ключа электронной подписи), можно проверить, используя соответствующий открытый ключ (ключ проверки электронной подписи). Инфраструктурой открытых ключей обеспечиваются технологии и процессы создания ключевой пары, их использования, мониторинга жизненного цикла, а так же исключения из употребления [1].

Инфраструктура открытых ключей реализуется по модели клиент-сервер, то есть проверка или предоставление какой-либо информации происходит только по инициативе конечного пользователя.

Основными объектами инфраструктуры открытых ключей являются [2]:

- удостоверяющий центр — это базовый управляющий компонент ИОК, выпускающий цифровые сертификаты подчиненных центров и конечных пользователей. Удостоверяющий центр является доверенной третьей стороной и осуществляет управление выданными им сертификатами;

- сертификат ключа проверки электронной подписи (сертификат открытого ключа) — данные конечного пользователя сертификата и его открытый ключ, подписанные секретным ключом удостоверяющего центра. Выпуская сертификат, удостоверяющий центр подтверждает, что субъект, указанный в сертификате, владеет секретным ключом, соответствующим этому открытому ключу;

- реестр выданных сертификатов — хранилище всех когда-либо выданных данным удостоверяющим центром сертификатов;

- реестр аннулированных сертификатов — списки аннулированных сертификатов, а так же содержащие дату и обоснование аннулирования;

- центр запросов — компонент системы, с помощью которого конечные пользователи могут запросить или отозвать сертификат;

- конечные пользователи — пользователи, являющиеся владельцами сертификата ключа проверки электронной подписи и использующие ИОК.

1.2 Принцип работы инфраструктуры открытых ключей

Инфраструктура открытых ключей представляет собой систему, где пользователи взаимодействуют между собой используя сертификаты открытого ключа, выданные удостоверяющим центром.

Удостоверяющий центр осуществляется следующую деятельность [2]:

- создает сертификаты ключей проверки электронных подписей и выдает такие сертификаты лицам, обратившимся за их получением (заявителям), при условии установления личности заявителя;

- устанавливает сроки действия сертификатов ключей проверки электронных подписей;

- аннулирует ранее выданные сертификаты ключей проверки электронных подписей;

- выдает по обращению заявителя ключ электронной подписи и ключ проверки электронной подписи;

- ведет реестры выданных и аннулированных сертификатов;

- проверяет уникальность ключей проверки электронных подписей в реестре сертификатов;

- осуществляет по обращениям клиентов проверку электронных подписей.

При получении удостоверяющим центром запроса на выдачу сертификата ключа проверки подписи, удостоверяющий центр генерирует

ключевую пару и создает сертификат ключа проверки электронной подписи. Если личность получателя сертификата подтверждена, то ему будут выданы сертификат и ключевая пара. Открытый ключ может быть известен неограниченному кругу лиц и распространяется в виде сертификата открытого ключа, что позволяет проверить его подлинность и целостность. А секретный ключ владелец должен хранить в тайне. Сертификат может быть аннулирован удостоверяющим центром в следующих случаях [2]:

- компрометация ключа электронной подписи;
- изменение информации о владельце сертификата;
- обращение владельца сертификата.

1.3 Варианты архитектурного построения инфраструктуры открытых ключей

Архитектура ИОК определяет структуру отношений между удостоверяющий центр и другими субъектами ИОК. Могут выделяться следующие удостоверяющие центры по признаку доверия [1]:

- корневые центры сертификации — удостоверяющие центры, которым доверяет изначально все сообщество пользователей;
- доверенные центры сертификации — удостоверяющие центры, которым доверяют владельцы сертификатов.

Доверие в системах удостоверяющий центров определяется путем доверия. Путь доверия — цепочка сертификатов, которая позволяет убедиться, что последний из них был выдан доверенным удостоверяющим центром [1]. Пример пути доверия показан на рисунке 1. Последний сертификат в пути доверия является предъявленным сертификатом; начальное звено — сертификат корневого УЦ; промежуточные звенья — сертификаты, выданные промежуточными УЦ. При потери доверия к корневому центру сертификации, теряется доверие ко всей цепочки сертификатов.

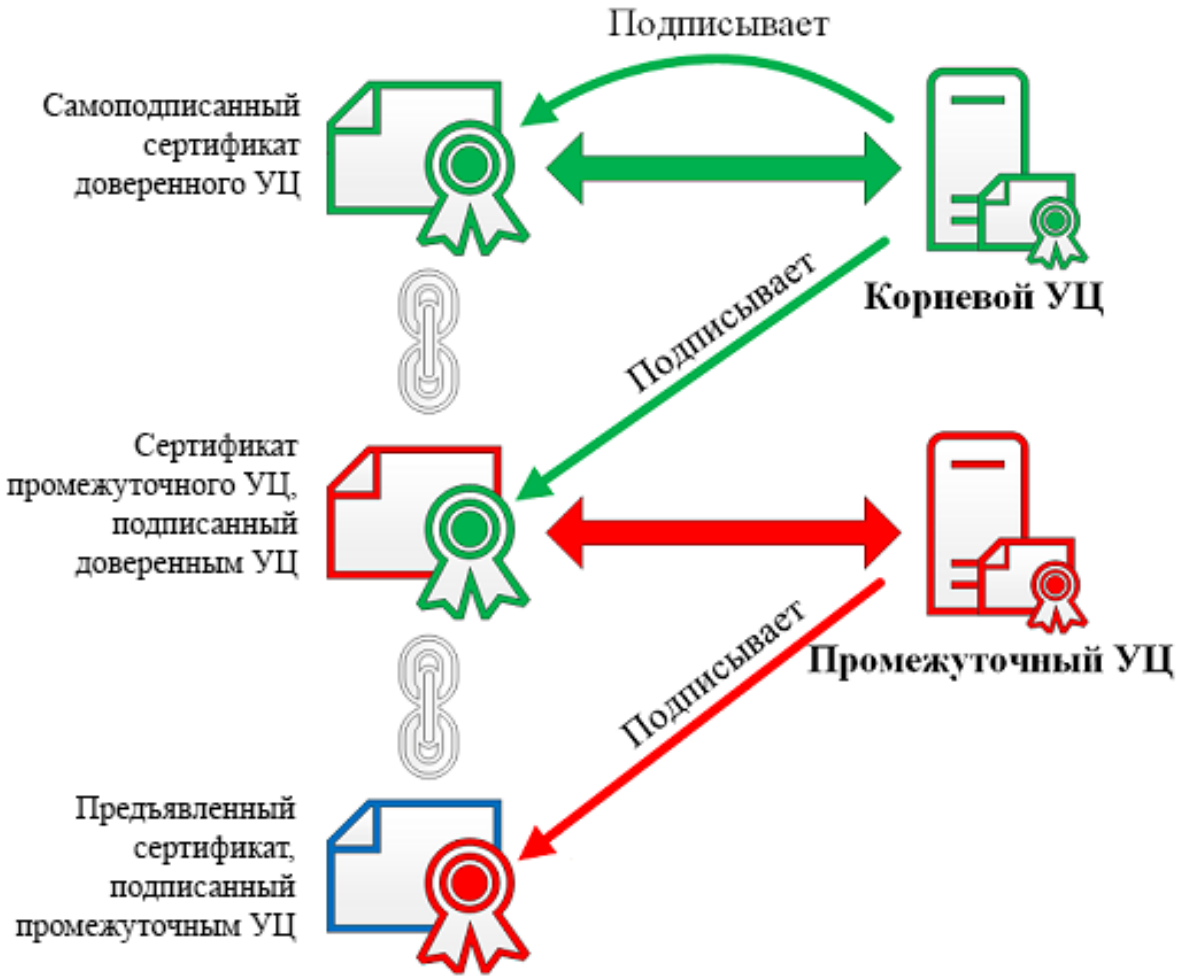
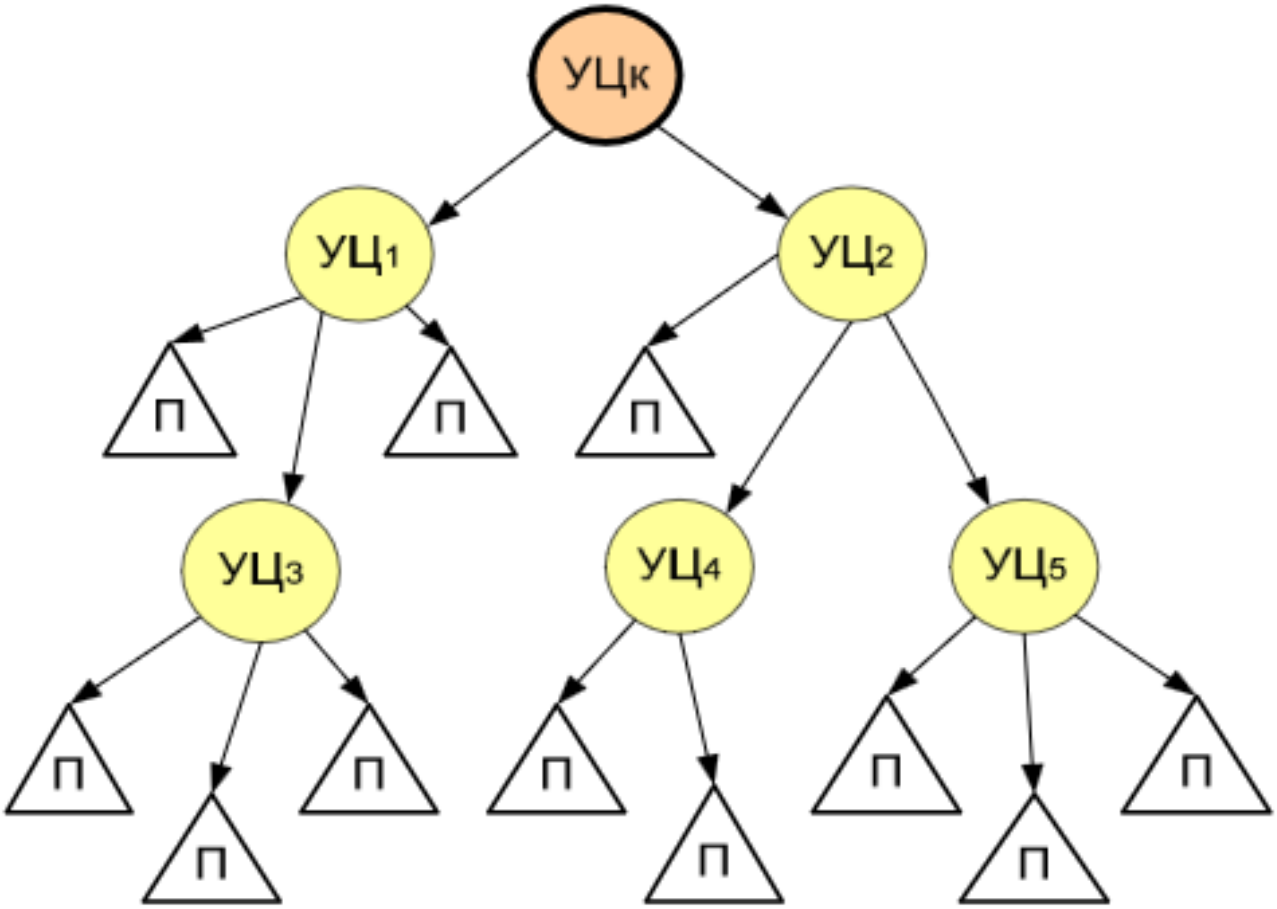


Рисунок 1 — Путь доверия

Обычно выделяются следующие виды архитектур ИОК [1]:

- простая архитектура ИОК. Все пользователи доверяют одному удостоверяющему центру и при взаимодействии друг с другом пользуются только выданными им сертификатами. Если злоумышленник выдаст себя за УЦ, то потребуется перевыпустить все выданные сертификаты;

- иерархическая архитектура ИОК является наиболее часто встречающейся архитектурой ИОК. Структурная схема иерархической ИОК представлена на рисунке 2.

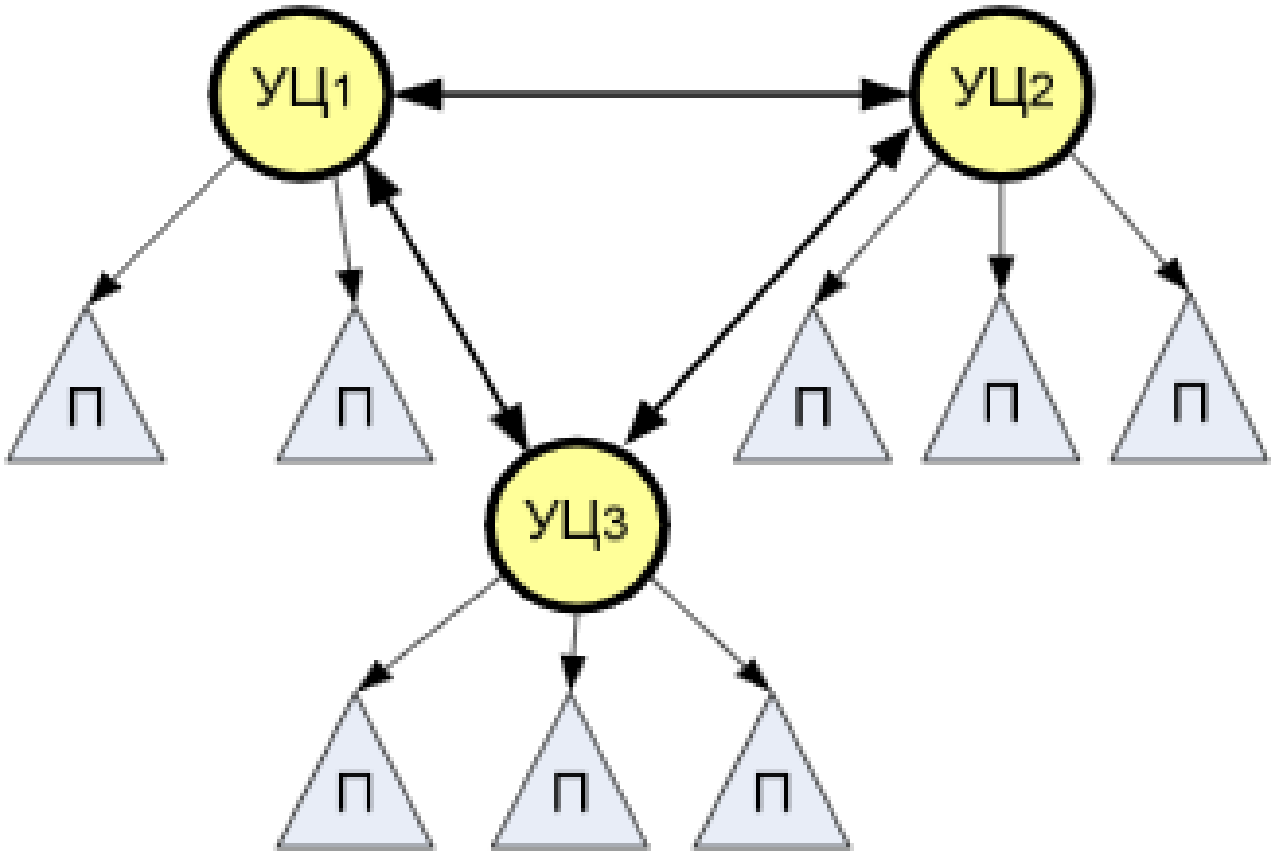


УЦк — корневой удостоверяющий центр; УЦi — подчиненные удостоверяющие центры; П — пользователи; ⟶ — отношения подчинения

Рисунок 2 — Структурная схема иерархической архитектуры ИОК

Во главе структуры стоит один корневой центр сертификации, которому доверяет все сообщество пользователей. Так же присутствуют другие УЦ, которые подчиняются вышестоящему. К каждому УЦ приписано некоторое подмножество пользователей или нижестоящих УЦ. Достоинством данной архитектуры является простота и однозначность построения пути доверия. Из недостатков можно выделить сложность внесения изменений в структуру. В иерархической архитектуре ИОК, если злоумышленник выдал себя за какой-то УЦ, система продолжает работать без данного центра сертификации до восстановления его штатной работоспособности;

- сетевая архитектура ИОК строится как сеть доверительных отношений. Структурная схема иерархической ИОК представлена на рисунке 3.



УЦi — удостоверяющие центры; П — пользователи; ⟶ — отношения подчинения; ⟷ — отношения кросс-сертификации

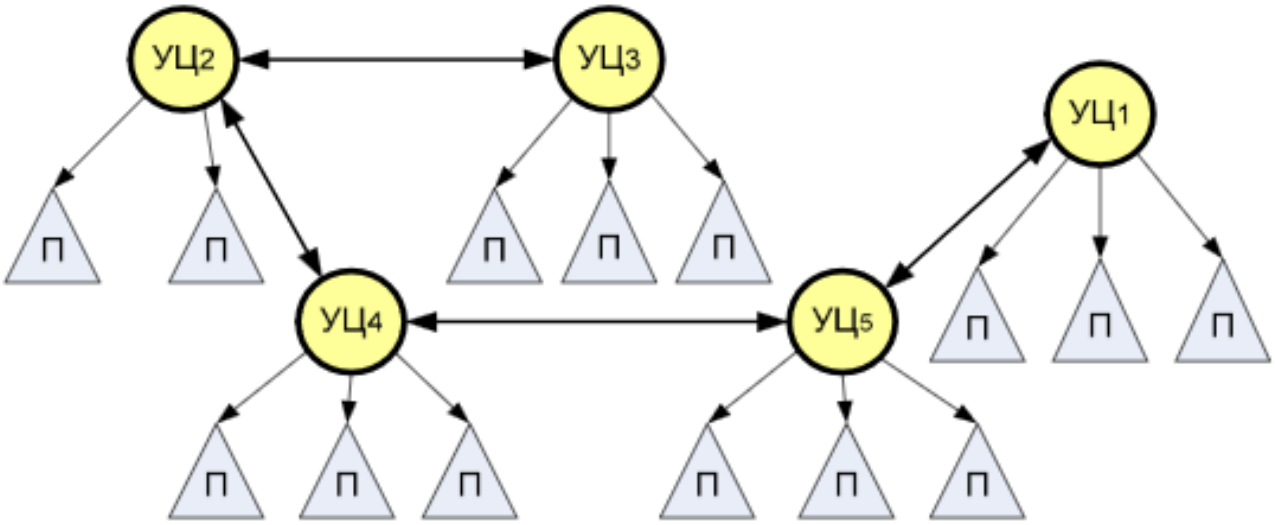
Рисунок 3 — Структурная схема сетевой архитектуры ИОК

Некоторое множество УЦ, связанные равноправными отношениями (отношениями кросс-сертификации), образуют сеть. Корневой центр сертификации не выделяется. В сетевой архитектуре все УЦ доверяют всем УЦ, с которыми установлены равноправные отношения доверия. Для установления равноправных отношений между двумя УЦ, они выпускают сертификаты друг для друга. Каждый пользователь ИОК доверяет только тому УЦ, который выдал ему сертификат. В сетевую архитектуру ИОК легко добавить новый УЦ: ему необходимо обменяться сертификатами хотя бы с одним УЦ из сети. Компрометация одного удостоверяющего центра не повлияет на работу ИОК в целом. УЦ, выпустившие сертификаты

скомпрометированного УЦ, аннулируют их, таким образом исключив из ИОК ненадежный центр сертификации. Таким образом компрометация одного из УЦ на работе пользователей, связанных с другими УЦ, никак не отразится.

Достоинством сетевой архитектуры ИОК является простота внесения изменений в архитектуру, а основной недостаток — сложность и неоднозначность путей доверия;

- кросс-сертифицированная архитектура ИОК. Структурная схема кросс-сертифицированной архитектуры ИОК представлена на рисунке 4.

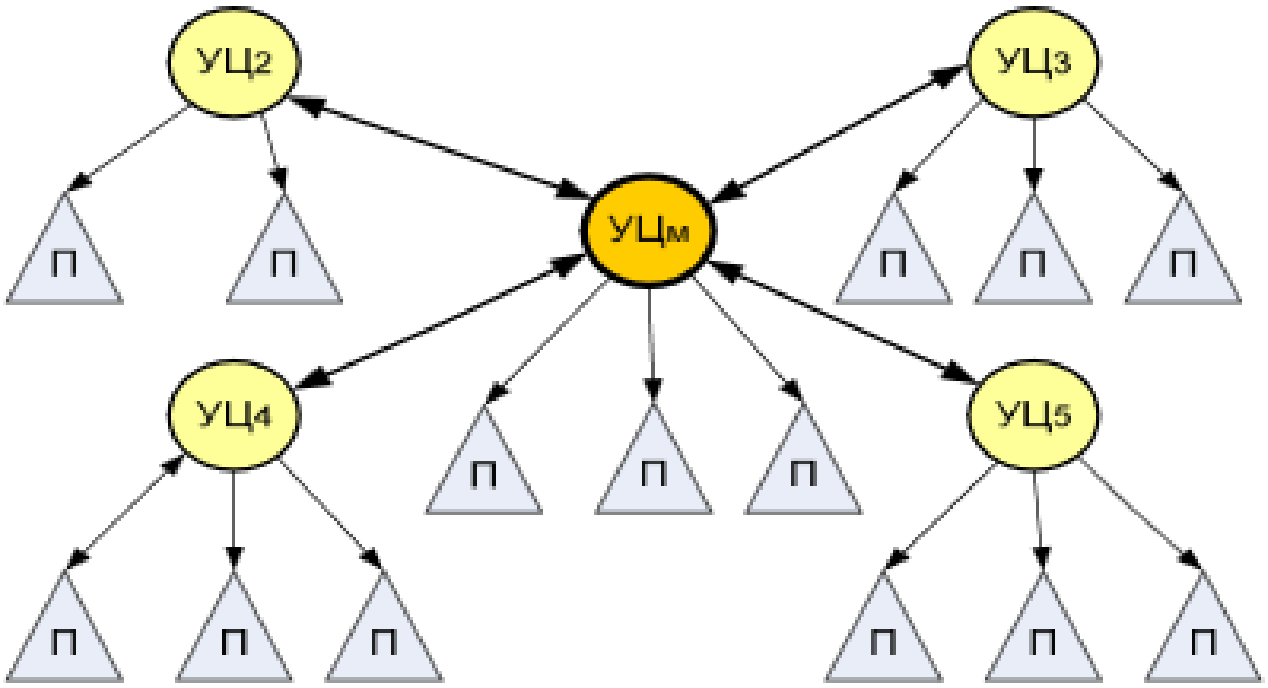


УЦi — удостоверяющие центры; П — пользователи; ⟶ — отношения подчинения; ⟷ — отношения кросс-сертификации

Рисунок 4 — Структурная схема кросс-сертифицированной архитектуры ИОК

Данная архитектура представляет собой смешанный вид сетевой и иерархической архитектур. Архитектура включает в себя локальные иерархические системы ИОК нескольких предприятий. На основе кросс-сертифицированных связей между корневыми центрами локальных систем инфраструктуры открытых ключей предприятий образуется связующая система, позволяющая взаимодействовать пользователями одной локальной системы с пользователями других;

- мостовая архитектура ИОК. Структурная схема мостовой архитектуры ИОК представлена на рисунке 5.



УЦм — мостовой удостоверяющий центр; УЦi — удостоверяющие центры; П — пользователи; ⟶ — отношения подчинения; ⟷ — отношения кросс-сертификации

Рисунок 5 — Структурная схема мостовой архитектуры ИОК

Мостовая архитектура ИОК позволяет убрать недостатки сложного процесса сертификации в кросс-сертифицированной архитектуре. Идея данной архитектуры заключается в том, что удостоверяющие центры предприятий доверяют одному определенному мостовому корневому УЦ, то есть он является их общим головным УЦ. Мостовой удостоверяющий центр может не быть основным пунктом доверия, он, лишь играет роль посредника между другими УЦ.

Достоинства мостовой архитектуры ИОК: простота включения новых элементов, не входящих в иерархию; возможна организация гибких взаимосвязей, которые могут быть изменены без ущерба для поддерживаемых ИОК. Недостатками мостовой архитектуры являются: возможность организации связи только между корневыми УЦ.

1.4 Структура электронного сертификата ключа проверки электронной подписи

Структура электронного сертификата открытого ключа задается стандартом X.509. Структура сертификата, определяемая различными версиями стандарта X.509 показана в таблице 1.

Таблица 1 — Структура электронного сертификата открытого ключа

<table>
<tr><td>version</td><td rowspan="8">X.509 v1</td><td rowspan="10">X.509 v2</td><td rowspan="11">X.509 v3</td></tr>
<tr><td>serialNumber</td></tr>
<tr><td>signatureInfo</td></tr>
<tr><td>issuer</td></tr>
<tr><td>validity</td></tr>
<tr><td>subject</td></tr>
<tr><td>subjectPublicKeyInfo</td></tr>
<tr><td colspan="2">issuerUniqueIdentifier</td></tr>
<tr><td colspan="2">subjectUniqueIdentifier</td></tr>
<tr><td colspan="3">extensions</td></tr>
<tr><td>signature</td><td colspan="3">все версии</td></tr>
</table>

Поля, представленные в таблице 1 имеют следующее предназначение [3]:

- version определяет версию стандарта X.509 данного сертификата;

- serialNumber — серийный номер данного сертификата — представляет собой целое число, присваиваемое УЦ каждому сертификату. Значение serialNumber должно быть уникальным для каждого сертификата, выданного заданным УЦ (таким образом имя выдавшего органа и порядковый номер однозначно определяют сертификат);

- signatureInfo содержит идентификатор алгоритма цифровой подписи и хэш-функции, используемой УЦ при подписании сертификата (например, md5WithRSAEncryption, sha-1WithRSAEncryption, id-dsa-with-sha1 и т. д.);

- issuer содержит ифнформацию об УЦ, подписавшем и выдавшем данный сертификат;

- validity представляет собой интервал времени, в течение которого УЦ гарантирует, что он будет поддерживать информацию о статусе сертификата;

- subject содержит информацию о субъекте, которому был выдан данный сертификат;

- subjectPublicKeyInfo используется для хранения сертифицируемого открытого ключа и для определения алгоритма, примером которого является данный открытый ключ (например, rsaEncryption, dhpublicnumber, id-dsa и т. д.);

- issuerUniqueIdentifier используется для однозначного определения выдавшего органа в случае повторного использования имени;

- subjectUniqueIdentifier используется для однозначного определения субъекта в случае повторного использования имени;

- extensions является необязательным полем и дает возможность добавлять, при необходимости, дополнительные поля в структуру сертификата;

- signature содержит цифровую подпись и идентификатор алгоритма цифровой подписи и хэш-функции, используемой УЦ при подписании сертификата.

Поля issuer и subject включают в себя стандартные атрибуты имени [3], далее представлены некоторые из атрибутов:

- commonName (общее имя, идентификатор: 2.5.4.3) должен иметь значение в виде текстовой строки, содержащей имя, фамилию и отчество (если имеется) — для физического лица, или наименование — для юридического лица;

- countryName (наименование страны, идентификатор: 2.5.4.6) задает двухсимвольный код страны;

- stateOrProvinceName (наименование региона, идентификатор: 2.5.4.8) должен иметь значение в виде текстовой строки, содержащей

наименование соответствующего субъекта Российской Федерации;

- localityName (наименование населенного пункта, идентификатор: 2.5.4.7) должен иметь значение в виде текстовой строки, содержащей наименование соответствующего населенного пункта;

- organizationName (наименование организации, идентификатор: 2.5.4.10) должен иметь значение в виде текстовой строки, содержащей наименование юридического лица;

- organizationUnitName (подразделение организации, идентификатор: 2.5.4.11) должен иметь значение в виде текстовой строки, содержащей наименование подразделения юридического лица.

Для хранения сертификата в файле его структура представляется по правилам ASN.1 и кодируется.

Для описания структуры цифрового сертификата используются следующие типы данных языка ASN.1 [3]:

- SEQUENCE — используется для описания структуры данных, включающих различные типы данных;

- INTEGER — целочисленный тип данных;

- OBJECT IDENTIFIER — последовательность целых чисел;

- UTCTime — временной тип данных, содержащий по 2 цифры для обозначения года, месяца, числа, часа, минут и секунд;

- GeneralizedTime — расширенный временной тип данных, содержащий 4 цифры для обозначения года и по 2 цифры для обозначения месяца, числа, часа, минут и секунд;

- SET — тип данных, описывающий структуру данных разных типов;

- UTF8String — тип данных, для хранения символьной строки, использующей кодировку UTF8;

- NULL — тип данных, обозначающий, что значение поля не определено;

- BIT STRING — тип данных для хранения последовательности бит.

Пример сертификата, представленного по правилам ASN.1 показан на рисунке 6.

```
SEQUENCE(3 elem)
    SEQUENCE(6 elem)
        INTEGER(64 bit) 12077081205577649624
        SEQUENCE(2 elem)
            OBJECT IDENTIFIER 1.2.840.113549.1.1.5 sha1WithRSAEncryption(PKCS #1)
            NULL
        SEQUENCE(1 elem)
            SET(1 elem)
                SEQUENCE(2 elem)
                    OBJECT IDENTIFIER 2.5.4.3 commonName(X.520 DN component)
                    UTF8String CA name
        SEQUENCE(2 elem)
            UTCTime2016-12-23 19:11:41 UTC
            UTCTime2036-12-23 19:11:41 UTC
        SEQUENCE(1 elem)
            SET(1 elem)
                SEQUENCE(2 elem)
                    OBJECT IDENTIFIER 2.5.4.3 commonName(X.520 DN component)
                    UTF8String Subject name
        SEQUENCE(2 elem)
            SEQUENCE(2 elem)
                OBJECT IDENTIFIER 1.2.840.113549.1.1.1 rsaEncryption(PKCS #1)
                NULL
            BIT STRING(1 elem)
                SEQUENCE(2 elem)
                    INTEGER(2048 bit) 28196046850336403296993969892683...
                    INTEGER 65537
    SEQUENCE(2 elem)
        OBJECT IDENTIFIER 1.2.840.113549.1.1.5 sha1WithRSAEncryption(PKCS #1)
        NULL
    BIT STRING(2048 bit) 000000111110001010101111011000101011001...
```

Рисунок 6 — Пример сертификата, представленного по правилам ASN.1

Согласно стандарту X.509 подписывается определенная часть цифрового сертификата, которая называется TSB-сертификат. К TSB-сертификату относятся данные, относящиеся к идентификатору второго уровня SEQUENCE (уровни видны на рисунке 6). Таким образом X.509 сертификат получается путем добавления к TSB-сертификату его электронной подписи по правилам ASN.1.

Для кодирования используется стандарт X.690. Правила кодирования, описанные в X.690, служат для представления структур данных, описанных по правилам ASN.1, в виде последовательностей байт.

Структура сертификата, представленная по правилам ASN.1 кодируется в виде блока, имеющего три части [4]:

- идентификатор — один или байт, содержащий закодированный тип данных;

- длина блока — один или несколько байт, содержащие количество байт, которое используется для закодированных данных в этом блоке;

- последовательность байт, содержащая закодированные данные.

Сопоставление типов данных и их идентификаторов, представлено в таблице 2.

Таблица 2 — Идентификаторы типов

Наименование типа	Идентификатор
SEQUENCE	30
INTEGER	02
OBJECT IDENTIFIER	06
UTCTime	17
GeneralizedTime	18
SET	31
UTF8String	0C
NULL	05
BIT STRING	03

Длина блока кодируется следующим образом [4]:

- если количество байт, используемых для закодированных данных в этом блоке не превышает 127 байт, то их количество записывается в соответствующий октет;

- если количество байт, используемых для закодированных данных в этом блоке больше 127 байт, то в первый байт записывается количество дополнительно выделяемых байт для записи длины, при этом восьмой бит первого байта устанавливается в 1. Во второй и последующие байты записывается значение длины блока данных.

Согласно стандарту RFC 2459 файл цифрового сертификата могут иметь следующие расширения:

- .CER — сертификат, или набор сертификатов, закодированных по стандарту CER;

- .DER — сертификат, закодированный по стандарту DER;

- .PEM — PEM-сертификат, закодированный по стандарту DER с использованием Base64 и помещенный между строками «----- BEGIN CERTIFICATE -----» и «----- END CERTIFICATE -----»;

- .P12 — PKCS #12 содержит блок, хранящий одновременно и закрытый ключ, и сертификат (в зашифрованном виде).

1.5 Использование инфраструктуры открытых ключей

На базе ИОК могут быть реализованы такие информационные технологии как:

- электронная подпись. Сторона А вычисляет хэш-функцию документа, результатом зашифрования полученного значения с помощью секретного ключа является электронная подпись. Сторона Б, получив документ и сертификат открытого ключа стороны А, верифицирует сертификат в удостоверяющем центре. Если сертификат действителен, расшифровывает полученную ЭЦП с помощью открытого ключа и вычисляет хеш-функцию от полученного документа. Если вычисленное и расшифрованное значения хэш-функций равны, то принимается, что полученный документ был подписан стороной А и не был искажен после подписания;

- шифрование сообщений. Сторона Б получает от стороны А сертификат открытого ключа. Полученный сертификат сторона Б верифицирует в удостоверяющем центре, если сертификат действителен, то сторона Б зашифровывает сообщение открытым ключом. После получения стороной А зашифрованного сообщения, она его сможет расшифровать соответствующим секретным ключом;

- авторизация пользователей. Сертификаты могут использоваться для подтверждения личности пользователя и установки прав на доступ и изменение каких-либо данных.

2 Обзор квантовых атак на алгоритмы, лежащие в основе открытого распределения ключей

Так как по скорости вычисления квантовые компьютеры значительно будут превосходить классические компьютерные архитектуры, то многие криптографические алгоритмы, основанные на сложности решения задачи дискретного логарифма на эллиптической кривой, на сложности разложения числа на простые множители, на сложности взятия логарифмов в конечных, полях станут уязвимыми к квантовым криптографическим атакам.

В 1994 году Питер Шор предложил квантовый алгоритм разложения числа на простые множители (факторизации), который позволяет разложить на множители число N за время $O((\log\ N)^3)$ [5].

Например, RSA использует открытый ключ n, который является произведением двух больших простых чисел. Для реализации атаки на ключ необходимо найти простые множители числа n. Так как число n является достаточно большим, используя известные классические алгоритмы, подобрать простые множители практически невозможно. Лучшие из известных классических алгоритмов факторизации чисел — метод квадратичных форм Шенкса и алгоритм Полларда — Штрассена, позволяют раскладывать на простые множители число за время порядка $O(n^{1/4})$. Среди вероятностных алгоритмов лучшим считается специальный метод решета числового поля, способный с вероятностью 1/2 вычислить простой делитель за время равное формуле (1) [5].

$$e^{(32/9\cdot\log M)^{1/3}\cdot(\log\log M)^{2/3}} \tag{1}$$

Квантовый алгоритм Шора способен разложить число на простые множители за время немного большее времени умножения целых чисел.

Алгоритм Шора имеет вероятностный характер. Первый источник случайности встроен в классическое вероятностное сведения факторизации числа к нахождению периода некоторой функции; второй — результат считывания случайных чисел из квантовой памяти.

Квантовый алгоритм Шора основан на способность кубитов принимать несколько значений одновременно. Алгоритм работы можно разделить на две части:

- классическое сведение факторизации числа к нахождению периода некоторой функции;
- квантовое нахождение периода этой функции.

В 2016 году физики из Массачусетского технологического института и Инсбрукского университета спроектировали квантовый компьютер, реализующий масштабируемую версию алгоритма Шора: число 15 было разложено на множители 3 и 5 с использованием квантового компьютера с 5 кубитами — информационными единицами квантовых компьютеров. Для реализации атаки на криптографические алгоритмы открытого распределения ключей необходимо использовать квантовый компьютер с несколькими сотнями кубитов, но ученные отмечают, что система допускает увеличение количества кубитов [6].

3 Обзор постквантовых алгоритмов открытого распределения ключей

3.1 Основные подходы, решающие проблему квантовых атак

Постквантовые алгоритмы открытого распределения ключей — алгоритмы, которые остаются актуальными при появлении квантовых компьютеров и, соответственно, квантовых атак. Так как большинство традиционных алгоритмов опираются на проблемы разложения целых чисел на простые множители, которые будут легко решаемы с помощью квантовых компьютеров, использующих алгоритм Шора, то сейчас ведется разработка и усовершенствование криптографических алгоритмов, устойчивых к квантовым атакам.

В постквантовой криптографии с открытым распределением ключей выделяются несколько различных подходов, которые решают проблему квантовых атак [7]:

- криптография, основанная на хэш-функциях. Примером данного подхода является подпись Меркла с открытым ключом на основе хэш-дерева. Данный алгоритм предложен в 1979 году как альтернатива RSA. Основным недостатком данного алгоритма является ограничение на количество подписей, полученных из набора ключей;

- криптография, основанная на решетках. Сложность взлома алгоритмов шифрования, основанных на решетках, может достигать экспоненциального времени. Примерами таких криптосистем является RLWE, NTRUEncrypt, NTRUSign. Компания Google в июле 2016 в качестве эксперимента добавила постквантовое шифрование в браузер Chrome Canary, используя алгоритм New Hope, который основан на проблеме RLWE [8];

- криптография, основанная на кодах исправления ошибок является одним из наиболее перспективных кандидатов постквантовой криптографии благодаря высокой скорости работы при эффективной реализации. Наиболее

известными такими криптосистемами являются McEliece и криптосистема Нидеррайтера, которые будут более подробно рассмотрены далее.

3.2 Криптосистема McEliece

Криптосистема McEliece, основанная на теории алгебраического кодирования, разработана в 1978 году Робертом Мак-Элисом. Данная криптосистема была первой, в которой использовалась рандомизация в процессе шифрования. Криптосистема McEliece широкого распространения в криптографии не получила из-за своих недостатков, но всё же является кандидатом для постквантовой криптографии и на ее основе создано множество модификаций.

Алгоритм шифрования основан на сложности декодирования полных линейных кодов. Существует некоторое количество вариантов криптосистемы с использованием различных типов кодов, но наилучшим кодом для криптосистемы McEliece считается коды Гоппы по следующим причинам:

- существование нескольких быстрых алгоритмов декодирование кода за полиномиальное время;

- код Гоппы легко задать, но тяжело распознать, так как порождающая матрица кода является почти случайной, для ее составления подойдет любой неприводимый многочлен над полем $GF(2^m)$;

- на данный момент криптосистема McEliece на основе кодов Гоппы не поддается криптоанализу [9].

В данной криптосистеме используются следующие параметры безопасности: n, k, t. Изначально были предложены следующие значения параметров: $n = 1024$, $k = 524$, $t = 50$, но для устойчивости против квантового компьютера следует использовать следующие значения этих параметров: $n = 6960$, $k = 5413$, $t = 119$.

Алгоритм генерации ключевой пары:

- выбирается (n, k)-линейный код C, который исправляет t ошибок и вычисляется порождающая $k \times n$ матрица G;

- генерируется случайная $k \times k$ невырожденная матрица S;

- генерируется случайная $n \times n$ матрица перестановки P;

- вычисляется $k \times n$ матрица G' по формуле (2).

$$G' = SGP \tag{2}$$

В результате открытым ключом является пара (G', t), секретным ключом — (S, G, P). При использовании рекомендованных параметров размер публичного ключа составляет до 8 373 911 бит.

Для зашифрования сообщения m используется следующий алгоритм:

- сообщение m представляется в виде последовательности двоичных символов длины k;

- генерируется случайный вектор z длины n и имеющий расстояние Хэмминга t;

- вычисление шифротекста c осуществляется по формуле (3).

$$c = mG' + z \tag{3}$$

Расшифрование шифротекста c производится по следующему алгоритму:

- вычисляется обратная матрица P^{-1};

- вычисляется c' по формуле (4).

$$c' = cP^{-1} \tag{4}$$

- используя быстрый алгоритм декодирования кода C, вычисляется m' из c';

- вычисляется исходное сообщение m по формуле (5).

$$m = m'S^{-1} \tag{5}$$

Криптосистема McEliece обладает следующими недостатками:

- очень большой размер открытого ключа;

- зашифрованное сообщение гораздо длиннее исходного;

- криптосистема не может быть использована для построения цифровой подписи;

3.3 Криптосистема Нидеррайтера

Криптосистема Нидеррайтера — криптосистема с открытым распределением ключей, основанная на теории алгебраического кодирования. Криптосистема была предложена в 1986 году Харальдом Нидеррайтером. Отличием от криптосистемы McEliece является то, что в криптосистеме Нидеррайтера используется проверочная матрица кода. А так же с помощью криптосистемы Нидеррайтера возможно создание цифровой подписи.

Классическую криптосистему Нидеррайтера на сегодняшний день нельзя считать стойкой, но существует несколько криптостойких модификаций, из которых можно выделить три подхода [10]:

- зашумление проверочной матрицы;

- использование метрики, отличной от классической хэмминговой (например ранговой);

- использование кодов с набором специфических свойств (кодов Гоппы).

Алгоритм генерации ключевой пары:

- выбирается (n, k)-код C над полем Галуа $GF(q)$, исправляющий t ошибок с эффективным алгоритмом декодирования;

- генерируется $(n - k) \times n$ проверочная матрица H кода C;

- генерируется случайная $(n - k) \times (n - k)$ невырожденная матрица S над полем $GF(q)$;

- генерируется случайная $n \times n$ матрица перестановки P;

- вычисляется $(n - k) \times n$ матрица H' по формуле (6).

$$H' = SHP \tag{6}$$

В результате открытым ключом является пара (H', t), секретным ключом — набор (S^{-1}, H, P^{-1}).

Алгоритм шифрования сообщения:

- представить сообщение в виде двоичной последовательности m длины n, имеющей вес t, то есть сообщения — это всевозможные ошибки, которые код C способен исправить;

- шифротекст c в криптосистеме Нидеррайтера представляет собой зашумленный синдром шифруемого вектора ошибки и вычисляется по формуле (7).

$$c = mH'^T \tag{7}$$

Расшифрование шифротекста c производится по следующему алгоритму:

- вычисляется синдром s по формуле (8).

$$s = c(S^T)^{-1} \tag{8}$$

- используя быстрый алгоритм декодирования кода C, находится m' из синдрома s;

- вычисляется исходное сообщение m по формуле (9).

$$m = m'P^{-1} \tag{9}$$

В 2001 году было предложено использовать криптосистему Нидеррайтера для создания электронной подписи. Алгоритм был разработан Николя Куртуа, Мэттью Финиазом и Николя Сандриером [11].

Алгоритм подписи документа D выглядит следующим образом:

- выбирается хеш-функция h с размером хеша $n - k$ символов, который можно представить в виде синдрома и попытаться декодировать;

- вычислить хеш s по формуле (10).

$$s = h(D) \tag{10}$$

- для каждого $i = 0, 1, 2, ...$ вычислить s_i по формуле (11).

$$s_i = h(s|i) \tag{11}$$

- найти наименьший i_0 такой, что s_{i_0} возможно декодировать, тогда z — результат декодирования этого синдрома.

В результате подписью документа D является конкатенация $(z|i_0)$.

Алгоритм проверки подписи:

- из подписи $(z|i_0)$ документа D получить z и i_0;

- вычислить s_1 по формуле (12).

$$s_1 = zH'^T \tag{12}$$

- вычислить s_2 по формуле (13).

$$s_2 = h(h(D)|i_0) \tag{13}$$

- если $s_1 = s_2$, то подпись верна.

По сравнению с McEliece, криптосистема Нидеррайтера имеет ряд преимуществ: во-первых, имеется возможность использование криптосистемы для создания цифровой подписи; во вторых, размер открытого ключа в $\frac{n-k}{n}$ раз меньше. А так же по сравнению с RSA, скорость зашифрования выше в 50 раз, а расшифрования — в 100 раз.

Недостатками криптосистемы Нидеррайтера являются:

- необходимость представить сообщение, подлежащего шифрованию, в виде q-арного вектора длиной n и веса не более t;

- размер ключей больше, чем в классических криптосистемах с открытым распределением ключей;

- размер шифротекста намного превышает размер открытого текста приблизительно в 2 раза.

4 Разработка алгоритма работы макета удостоверяющего центра

4.1 Разработка алгоритма работы серверной части макета удостоверяющего центра

Серверная часть макета удостоверяющего центра должна выполнять следующие функции:

- выдачу цифровых сертификатов открытого ключа по запросу клиента;
- формирование ключевой пары криптосистемы Нидеррайтера;
- отзыв ранее выданных цифровых сертификатов;
- ведение реестра выданных и отозванных сертификатов;
- проверку подлинности выданных цифровых сертификатов по запросу клиента.

Основываясь на данных функциях, был разработан алгоритм работы серверной части макета удостоверяющего центра. Блок-схема алгоритма приведена на рисунке 7.

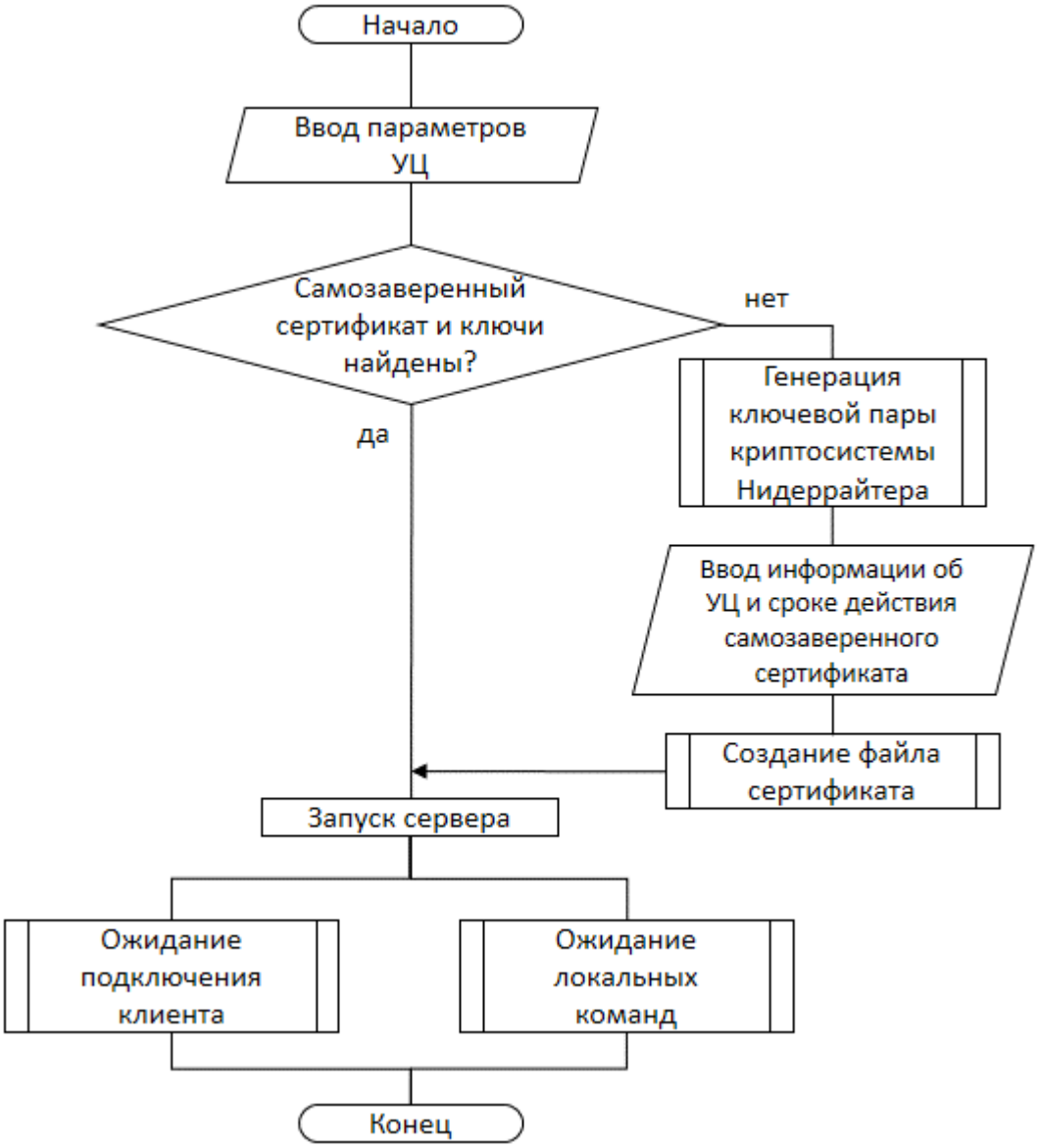


Рисунок 7 — Блок-схема алгоритма работы серверной части макета УЦ

После запуска серверной части макета УЦ происходит считывание параметров УЦ из файла конфигурации. Параметры УЦ включают в себя:

- номер порта, который прослушивает сервер;
- путь к файлу самозаверенного сертификата;
- путь к файлу секретного ключа УЦ;
- путь к каталогу, куда будут сохраняться клиентские сертификаты;
- путь к каталогу, куда будут созраняться клиентские ключи;
- параметры, задающие размер клиентских ключей;
- путь к БД УЦ.

Если по указанным путям не обнаружены секретный ключ или самозаверенный сертификат, то программа ожидает ввода с клавиатуры данных об УЦ (наименование УЦ, страна, регион, населенный пункт, организация, подразделение организации). После получения всех данных, создается новый сертификат.

Далее запускается сервер и программа переходит в режим ожидания подключения клиента или ввода локальной команды от администратора сервера.

Алгоритм ожидания и обработки команд представлен на рисунке 8.

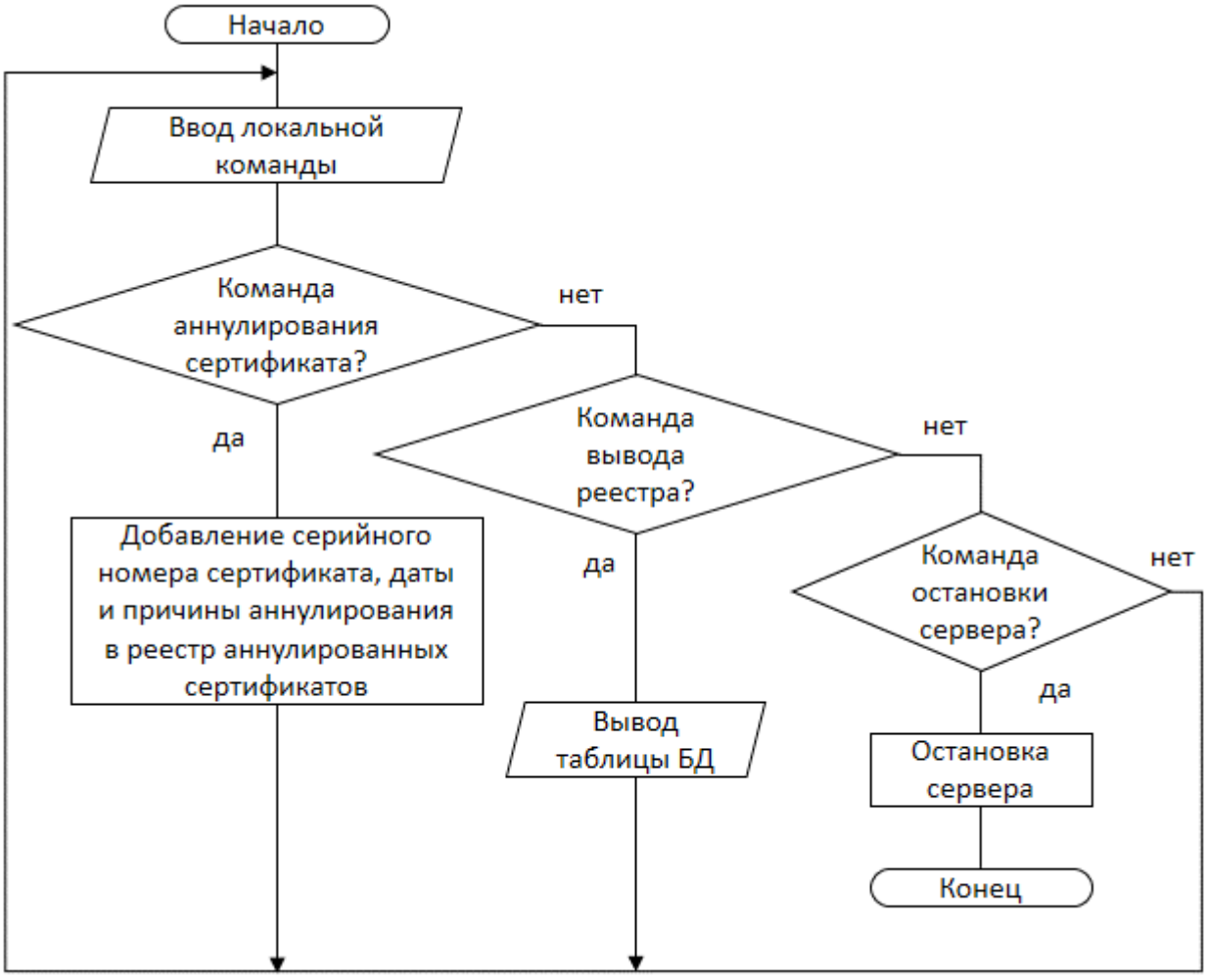


Рисунок 8 — Блок-схема алгоритма ожидания и обработки команд

Сервер может исполнять следующие локальные команды:

- аннулирование сертификата, результатом исполнения которой является добавление серийного номера сертификата, текущей даты и времени и причины аннулирования в реестр аннулированных сертификатов;

- вывод реестра выданных или аннулированных сертификатов на экран;

- остановка сервера.

Алгоритм ожидания подключения клиента и взаимодействия с ним показан на рисунке 9.

После установления соединения с клиентом и получения от него данных, программа определяет являются ли принятые данные запросом на выдачу сертификата открытого ключа или запросом на проверку сертификата.

Если принят запрос на выдачу сертификата, то из запроса выделяются данные о субъекте и сроке действия. Далее генерируется и сохраняется на сервере ключевая пара криптосистемы Нидеррайтера и создается файл сертификата открытого ключа. Выполняется проверка на уникальность ключа проверки электронной подписи, если ключ не уникальный, генерируется новая пара. Сертификат добавляется в реестр выданных сертификатов.

Если принят запрос на проверку сертификата, то сначала проверяется электронная подпись сертификата. Если подпись верна, осуществляется сравнение срока действия сертификата с текущими датой и временем. Затем осуществляется поиск проверяемого сертификата в реестре аннулированных сертификатов. По результатам проверки сертификата формируется и отправляется ответ клиенту и сервер снова переходит в режим ожидания.

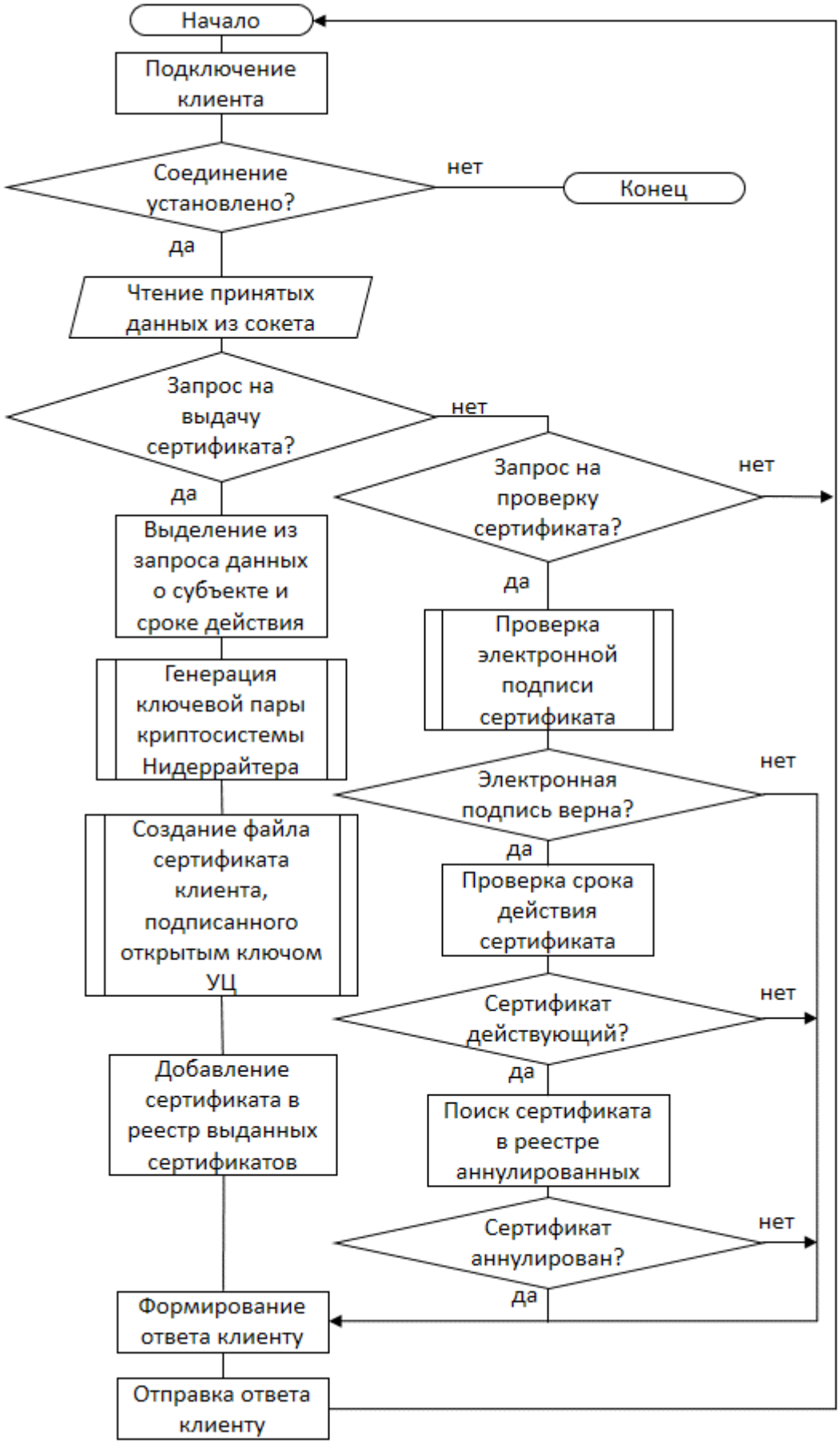


Рисунок 9 — Блок-схема алгоритма ожидания подключения клиента и взаимодействия с ним

Блок-схема алгоритма создания цифрового сертификата открытого ключа представлен на рисунке 10.

Сначала формируется TSB-сертификат, содержащий в себе данные о субъекте сертификата и выпускающего удостоверяющего центра. Далее формируется электронная подпись TSB-сертификата. В результате добавления к TSB-сертификату его электронной подписи и информации об алгоритме подписи будет создан X.509-сертификат.

Рисунок 10 — Блок-схема алгоритма создания цифрового сертификата открытого ключа

В качестве линейного кода был выбран (n, k)-код Рида — Соломона над полем Галуа $GF(q)$, исправляющий t ошибок. Проверочная матрица кода имеет вид, показанный в формуле (14).

$$H = \begin{pmatrix} 1 & \alpha & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \dots & \alpha^{(n-k)(n-1)} \end{pmatrix}, \tag{14}$$

где α — примитивный элемент поля $GF(q)$.

Для декодирование вектора $s = (s_0, s_1, ..., s_{n-k-1})$, необходимо использовать алгоритм быстрого декодирования. В данной реализации был выбран алгоритм Алгоритм Берлекемпа — Мэсси, согласно которому сначала находится многочлен локаторов ошибок $L(x)$. Коэффициенты многочлена локаторов ошибок находятся по формуле (15), $L_0 = 0$.

$$\begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_t \end{pmatrix} = M^{-1}V = \begin{pmatrix} s_{t-1} & s_{t-2} & \dots & s_0 \\ s_t & s_{t-1} & \dots & s_1 \\ \vdots & \vdots & \ddots & \vdots \\ s_{2t-2} & s_{2t-3} & \dots & s_t \end{pmatrix}^{-1} \begin{pmatrix} s_t \\ s_{t+1} \\ \vdots \\ s_{2t-2} \end{pmatrix} \tag{15}$$

Далее вычисляется многочлен ошибок $W(x)$ по формуле (16).

$$W(x) = L(x) \cdot S(x) \tag{16}$$

Коэффициенты многочлена $W(x)$ старшие чем $n - k$ обнуляются.

Далее следует вычислить производную $L'(x)$ многочлена локаторов ошибок $L(x)$. Производная вычисляется следующим образом: для чётных степеней производная равна нулю, для нечётных — степени уменьшенной

на 1.

Затем с помощью процедуры Ченя осуществляется поиск корней X многочлена $L(x)$ и значений ошибок Y. Блок-схема процедуры Ченя показана на рисунке 11.

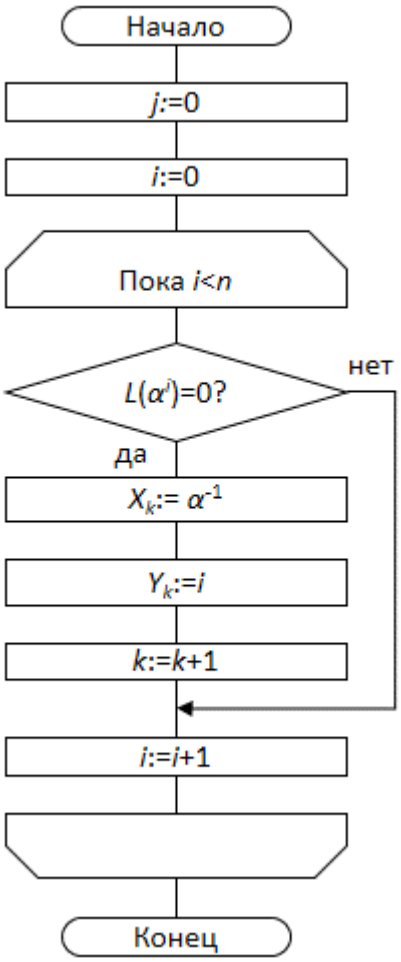


Рисунок 11 — Блок-схема процедуры Ченя

Теперь можно найти результат декодирования $E(x)$ синдрома. Коэффициенты $E(x)$ вычисляются с помощью алгоритма Форни по формуле (17).

$$E_{Y_i} = \frac{W(X_i)}{L'(X_i)}; i = \overline{0, t-1}, \tag{17}$$

где $i = \overline{0, t-1}$.

Сложение в поле Галуа $GF(2^n)$ с примитивным элементом 2 по модулю неприводимого многочлена $p(x)$ представляет собой побитовое

сложение по модулю 2. В таком случае, операция сложения является обратной самой себе. Для реализации умножения и деления необходимо сгенерировать таблицу быстрого умножения. Таблица должна состоять из двух полей alpha_of и index_of. Заполнение таблицы осуществляется по формулам (18) и (19).

$$\text{alpha_of}_i = 2^i \mod p(x) \tag{18}$$

$$\text{index_of}_{2^i \mod p(x)} = i, \tag{19}$$

где $i = \overline{0, 2^n - 1}$.

После заполнения таблицы умножение чисел a и b в поле Галуа $GF(2^n)$ осуществляется по формуле (20).

$$a \cdot b = \text{index_of}_{(\text{alpha_of}_a + \text{alpha_of}_b) \mod (2^n - 1)} \tag{20}$$

Деление большего числа a на меньшее b в поле Галуа $GF(2^n)$ осуществляется по формуле (21).

$$a/b = \text{index_of}_{\text{alpha_of}_a - \text{alpha_of}_b} \tag{21}$$

Возведение числа a в степень b в поле Галуа определяется как результат b-кратного умножения числа a на себя.

Алгоритмы генерации ключевой пары, формирования и проверки электронной подписи представлены в подразделе 3.3.

4.2 Разработка алгоритма работы клиентской части макета удостоверяющего центра

Клиентская часть макета удостоверяющего центра должна выполнять следующие функции:

- отправку серверной части запроса на выдачу цифрового сертификата;
- получение от серверной части цифрового сертификата
- отправку запроса на проверку подлинности цифрового сертификата;
- получение от серверной части результата проверки подлинности цифрового сертификата.

Основываясь на данных функциях, был разработан алгоритм работы клиентской части макета удостоверяющего центра. Блок-схема алгоритма приведена на рисунке 12.

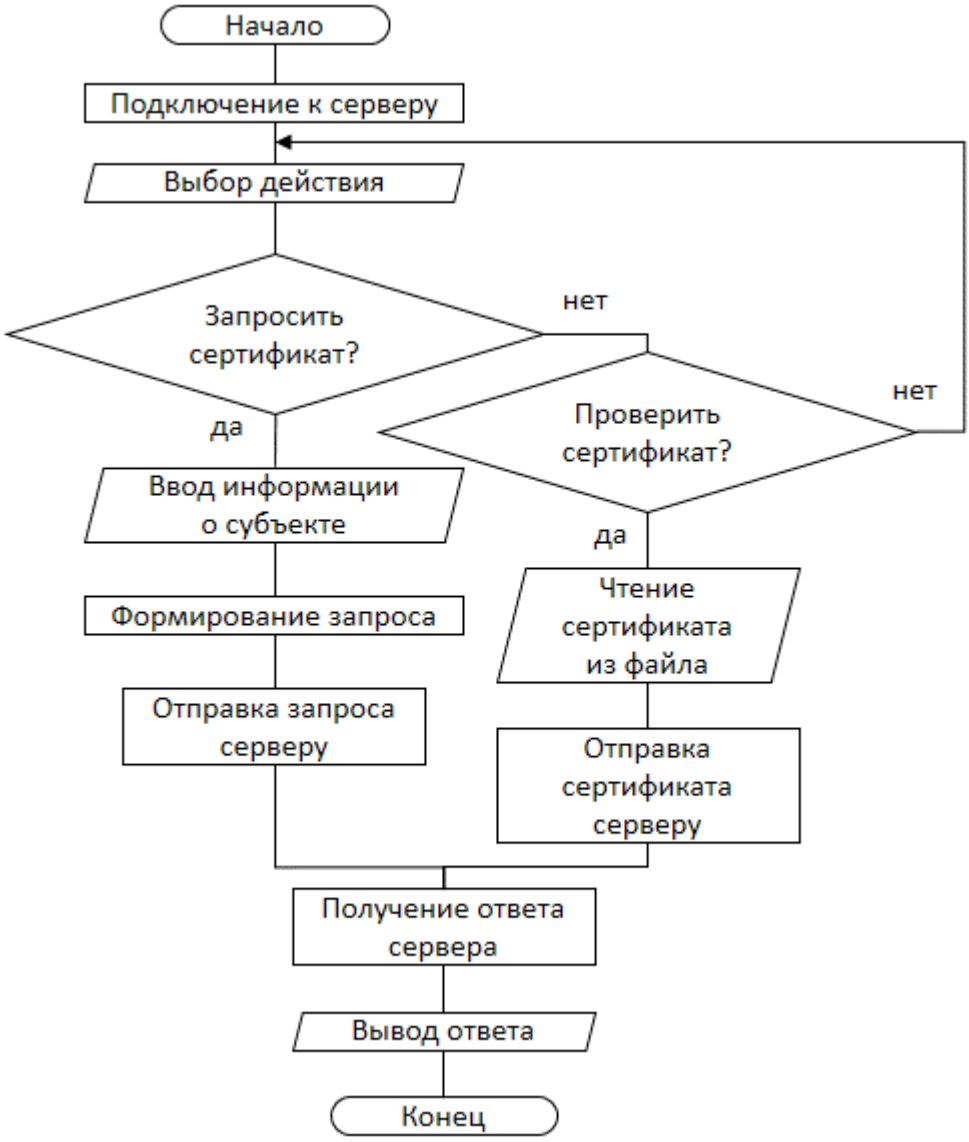


Рисунок 12 — Блок-схема алгоритма работы клиентской части макета УЦ

После запуска клиентской программы выполняется подключение к серверу УЦ. Далее пользователь должен выбрать одно из действий: запрос на выдачу сертификата или проверка сертификата на подлинность.

Для формирования запроса на выдачу сертификата открытого ключа пользователь должен ввести данные о субъекте сертификата, содержащие:

- общее имя;
- страна;
- регион;
- наименование населенного пункта;
- наименование организации;
- наименование подразделения организации.

После ввода всех данных, формируется запрос и отправляется серверной части. После чего ожидается ответ сервера, при получении которого он выводится пользователю.

При выборе пользователем функции проверки сертификата на подлинность, пользователь должен указать путь к файлу сертификата открытого ключа, который необходимо проверить. После указания пути программа передает файл сертификата серверу и ожидает ответ.

5 Программная реализация алгоритма макета удостоверяющего центра

5.1 Программная реализация алгоритма серверной части макета удостоверяющего центра

Макет удостоверяющего центра программно реализован на языке высокого уровня C++ с использованием кроссплатформенный инструментарий разработки ПО Qt.

Серверная часть макета удостоверяющего центра реализована в виде консольного приложения. Исходный код программы состоит из следующий файлов:

- servermain.cpp — является точкой входа в приложение, содержимое файла показано в приложении А;
- TcpServer.h — является заголовочным файлом для класса TcpServer, содержимое файла показано в приложении Б;
- TcpServer.cpp — является файлом реализации класса TcpServer, содержимое файла показано в приложении В
- x509cert.h — является заголовочным файлом для класса X509Cert, содержимое файла показано в приложении Г;
- x509cert.cpp — является файлом реализации класса X509Cert, содержимое файла показано в приложении Д;
- structkey.h — объявляет структуру для хранения ключей, содержимое файла показано в приложении Е;
- niedercrypt.h — является заголовочным файлом для класса NiederCrypt, содержимое файла показано в приложении Ж;
- niedercrypt.cpp — является файлом реализации класса NiederCrypt, содержимое файла показано в приложении И.

Класс TcpServer обеспечивает сетевое взаимодействие с клиентской частью удостоверяющего центра через TCP-сокеты, обрабатывает запросы, полученные от клиентов, и локальные команды. Класс TcpServer включает в

себя следующие функции:

- функция writeDefaultParam при отсутствии в корневом каталоге программы файла serverSetting.ini с параметрами работы сервера, создает его и записывает в него параметры по умолчанию. Содержимое файла по умолчанию представлено на рисунке 13.

```
[General]
port=2323

[PATH]
selfSignedCert=selfSignedCert.crt
privKeyCA=privKeySelfSignedCert.key
clientCerts=clientCerts
clientKeys=clientKeys

[keyParam]
n=20
k=16

[BD]
path=CA.db3
```

Рисунок 13 — Содержимое файла serverSetting.ini

- функция createConnectionBD считывает параметры из файла serverSetting.ini в корневом каталоге программы и устанавливает соединение с БД SQLite. Функция возвращает «истину» при успешном соединении с БД или «ложь» при возникновении ошибки;

- функция startServer проверяет наличие файла самозаверенного сертификата, файла секретного ключа, базы данных, при их отсутствии запрашивает необходимые данные для создания сертификата удостоверяющего центра и создает файлы цифрового сертификата, ключа и базу данных SQLite, структура которой показана на рисунке 14. После чего начинает прослушивать порт. Функция возвращает 0 при успешном запуске сервера и 1 при возникновении какой-либо ошибки;

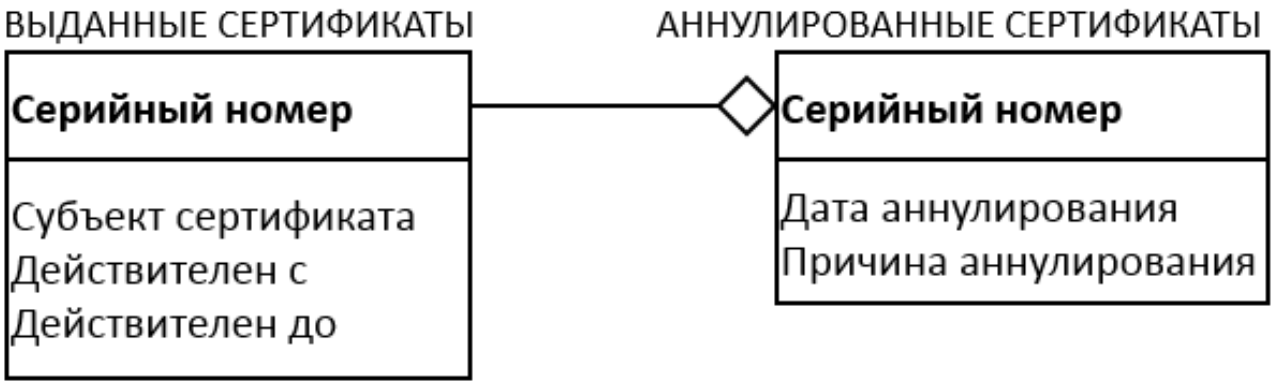


Рисунок 14 — Структура базы данных сервера

- функция slotNewConnection вызывается при подключении нового клиента к серверу;

- функция slotReadClient вызывается при получении данных от клиента, осуществляет обработку запроса клиента, формирование и отправку ответа;

- функция saveToFile записывает массив байтов в указанный файл. Функция принимает в качестве аргументов путь записи файла и массив байт, предназначенный для записи; возвращает — 0 при успешной записи файла и 1 при возникновении ошибки;

- функция openFile считывает данные из указанного файла в двоичном виде. Функция принимает в качестве аргумента путь к файлу, который необходимо прочитать; возвращает — массив байт, прочитанный из файла;

- функция readCommand считывает и распознает локальные команды полученные с клавиатуры. Обрабатываются следующие команды: revoke <серийный №> <причина> — аннулирует сертификат с указанным серийным номером; select <issued | revoked> — выводит в консоль реестр выданных (issued) или аннулированных (revoked) сертификатов;

- print — функция вывода в консоль. В качестве аргумента функция принимает символьную строку, предназначенную для вывода;

- функция commandRevoke реализует аннулирование сертификата — добавляет сертификат в реестр аннулированных сертификатов. В качестве

аргумента функция принимает символьную строку, в которой указан серийный номер сертификата, подлежащего аннулированию, и причина аннулирования; возвращает — 0 при успешном аннулировании сертификата и 1 при возникновении ошибки;

- функция commandSelect выводит в консоль содержимое указанного реестра. В качестве аргумента функция принимает символьную строку, содержащую название реестра для вывода в консоль; возвращает — 0 при успешной печати реестр и 1 при возникновении ошибки.

Класс X509Cert реализует создание, декодирование и проверку цифровых X.509-сертификатов открытого ключа, закодированных по стандарту DER и использующиех криптосистему Нидеррайтера. Класс X509Cert включает в себя следующие функции:

- функция getCert формирует сертификат открытого ключа, подписанный указанным ключом. Функция принимает в качестве аргументов открытый ключ, запрос в виде массива байт, закрытый ключ УЦ; возвращает — цифровой сертификат представленный в виде массива байт;

- функция checkCert проверяет подпись сертификата указанным ключом. Функция в качестве аргументов сертификат, подлежащий проверке, предствленный в виде массива байт и открытый ключ; возвращает — 0 — если подпись верна, 1 — если подпись не верна и 2 при возникновении какой-либо ошибки в процессе проверки;

- функция openCert декодирует сертификат открытого ключа. Функция принимает в качестве аргумента цифровой сертификат, представленный в виде массива байт; возвращает — 0 при успешном декодировании цифрового сертификата и 1 при возникновении ошибки;

- функция genTSB формирует TSB-сертификат. Функция принимает в качестве аргументов открытый ключ, представленный в виде массива байт, и целочисленное число — срок действия сертификата; возвращает TSB-

сертификат, представленный в виде массива байт;

- функция KeyToByte кодирует ключ по стандарту DER. Функция принимает на вход открытый или секретный ключ, а возвращает принятый ключ, закодированный по стандарту DER и представленный в виде массива байт;

- функция X509NameToDer кодирует поля сертификата issuer и subject по стандарту DER. Функция возвращает массив байт;

- функция getSizeInt расчитывает размер (в байтах) целого числа, закодированного по стандарту DER. Функция в качестве аргумента принимает целочисленное неотрицательное число, возвращает — минимальное количество байт, которое требуется для хранения этого числа;

- функция sizeToDer записывает по стандарту DER размер данных в массив байт. Функция принимает на вход целое неотрицательное число и массив байт, в который нужно записать это число;

- функция intToDer записывает по стандарту DER целое число в массив байт. Функция принимает на вход целое неотрицательное число и массив байт, в который нужно записать это число;

- функция sizeDecoder декодирует размер данных в массиве байт, закодированного по стандарту DER. Функция принимает в качестве аргумента массив байт, возвращает целочисленное неотрицательное число;

- функция getData возвращает массив байт, несущий «полезные данные» (без обозначения типа данных и их размера);

- функция derToInt декодирует целое число, закодированное по стандарту DER. Функция принимает в качестве аргумента массив байт, возвращает целочисленное неотрицательное число;

- функция DerToX509Name декодирует поля сертификата issuer и subject, закодированные по стандарту DER. Функция принимает в качестве аргумента массив байт, возвращает структуру, содеражащую

декадированные поля issuer и subject сертификата открытого ключа;

- функция keyDecoder декодирует ключ, закодированный по стандарту DER. Функция принимает в качестве аргумента массив байт, возвращает структуру, содержащую открытый или секретный ключ.

Класс NiederCrypt реализует криптосистему Нидеррайтера и позволяет генерировать ключевую пару, создавать и проверять цифровую подпись для массива байт. Класс NiederCrypt включает в себя следующие функции:

- функция gen_table формирует таблицу умножения в поле Галуа;
- функция gf_mul реализует умножение двух чисел в поле Галуа. Функция в качестве аргументов принимает два числа и возвращает число, являющееся результатом умножения в поле Галуа этих двух чисел;
- функция gf_div реализует деление двух чисел в поле Галуа. Функция в качестве аргументов принимает два числа и возвращает число, являющееся результатом деления в поле Галуа первого числа на второе;
- функция gf_pow реализует возведение числа в степень в поле Галуа. Функция в качестве аргументов принимает два числа и возвращает число, являющееся результатом возведения первого числа в степень второго в поле Галуа;
- функция vec_mul реализует умножение двух полиномов в поле Галуа. Функция в качестве аргументов принимает два полинома, представленные в виде вектора, и возвращает вектор, являющийся результатом умножения этих двух полиномов в поле Галуа;
- функция matrix_Transp транспонирует матрицу. Функция в качестве аргумента принимает матрицу и возвращает матриуц, являющуюся транспонированной по отношению к принятой матрице;
- функция matrix_mul реализует умножение матриц в поле Галуа. Функция в качестве аргументов принимает две матрицы и возвращает матрицу, являющуюся результатом умножения двух принятых матриц в

поле Галуа;

- функция Get_matr реализует нахождение дополнительного минора произвольного элемента матрицы. Функция принимает на вход матрицу, номер строки и номер столбца; возвращает матрицу, являющуюся дополнительным минором элемента, находящегося на пересечении указанных столбца и строки;

- функция Det реализует нахождение определителя матрицы в поле Галуа. Функция принимает в качестве аргумента матрицу и возвращает целое число, равное определителю этой матрицы;

- функция Obr реализует нахождение обратной матрицы в поле Галуа с помощью матрицы алгебраических дополнений. Функция принимает в качестве аргумента матрицу и возвращает обратную ей матрицу;

- функция f_x вычисляет значение математической функции $f(x)$ в поле Галуа. Функция в качестве аргумента принимает полином, представленный в виде вектора, и целое число. Возвращает — целое число, являющееся значением этой функции;

- функция decode реализует алгоритм быстрого декодирования Берлекемпа — Мэсси, для кода Рида — Соломона. Функция принимает в качестве аргумента синдром, представленный в виде массива байт и возвращает результат декодирования данного синдрома, представленный в виде вектора;

- функция gen_S генерирует случайную матрицу S, являющуюся частью секретного ключа;

- функция gen_P генерирует матрицу перестановки P, являющуюся частью секретного ключа;

- функция genKey формирует ключевую пару криптосистемы Нидеррайтера. Функция принимает в качестве аргументов два целочисленных неотрицательных числа, являющиеся параметрами кода Рида — Соломона;

- функция getPubKey возвращает открытый ключ для дальнейшего его использования вне класса;

- функция getPrivKey возвращает секретный ключ для дальнейшего его использования вне класса;

- функция setPubKey устанавливает сгенерированный ранее открытый ключ для дальнейшего его использования в классе;

- функция setPrivKey устанавливает сгенерированный ранее секретный ключ для дальнейшего его использования в классе;

- функция hash рассчитывает хеш по алгоритму SHA-256 для указанного массива байт. Функция принимает в качестве аргумента массив байт для которого необходимо вычислить хеш;

- функция signature формирует электронную подпись указанного массива байт. Функция принимает в качестве аргумента массив байт к которому необходимо сформировать электронную подпись, которая будет возвращена функциие в виде массива байт;

- функция checkSign проверяет подпись указанного массива байт. Функция принимает в качестве аргумента массив байт и его электронную подпись, так же представленную в виде массива байт, возвращает «истину» если подпись верна, иначе «ложь».

5.2 Программная реализация алгоритма клиентской части макета удостоверяющего центра

Клиентская часть макета удостоверяющего центра реализована в виде оконного приложения. Исходный код программы состоит из следующий файлов:

- clientmain.cpp — является точкой входа в приложение, содержимое файла показано в приложении К;

- mainwindow.h является заголовочным файлом для класса MainWindow, содержимое файла показано в приложении Л;

- mainwindow.cpp — является файлом реализации класса MainWindow, содержимое файла показано в приложении М

- x509cert.h — является заголовочным файлом для класса X509Cert. Файл аналогичен используемому в серверной части и его содержимое показано в приложении Г;

- x509cert.cpp — является файлом реализации класса X509Cert. Файл аналогичен используемому в серверной части и его содержимое показано в приложении Д;

- structkey.h — объявляет структуру для хранения ключей. Файл аналогичен используемому в серверной части и его содержимое показано в приложении Е.

Класс MainWindow обеспечивает взаимодействие с графическим интерфейсом пользователя, сетевое взаимодействие с серверной частью удостоверяющего центра через TCP-сокеты, формирует и отправляет запросы серверу и принимает от него ответы.

С помощью Qt Designer разработан графический интерфейс клиентской программы. Главное окно включает в себя две вкладки:

- вкладка «Получить сертификат», показанная на рисунке 15, предназначена для отправки серверу запроса на получение сертификата, содержит поля для ввода информации о соискателе сертификата;

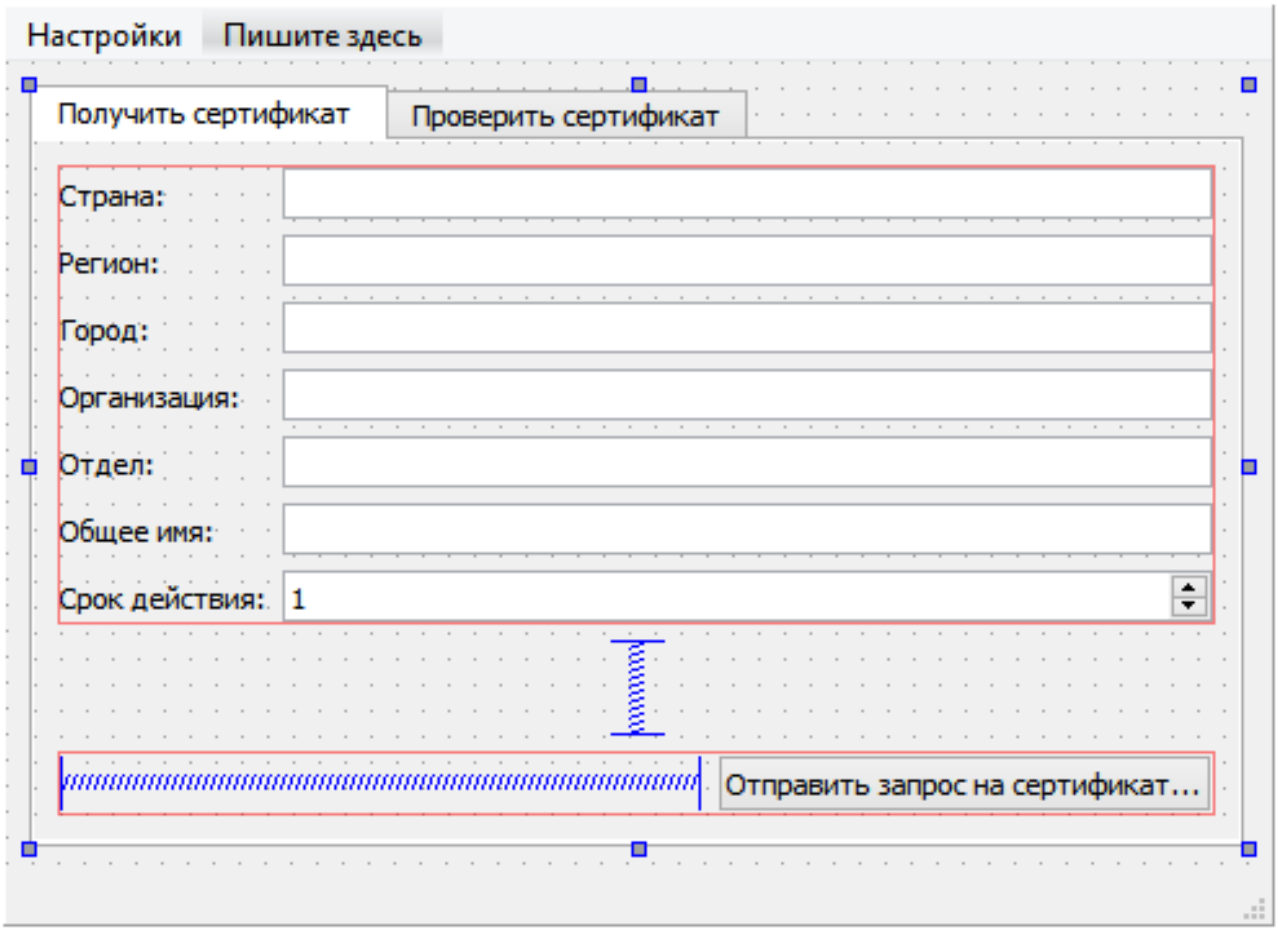


Рисунок 15 — Вкладка «Получить сертификат»

- вкладка «Проверить сертификат», показанная на рисунке 16, предназначена для отправки серверу запроса на проверку сертификата, содержит поле для выбора файла сертификата открытого ключа и поля для вывода атрибутов выбранного сертификата.

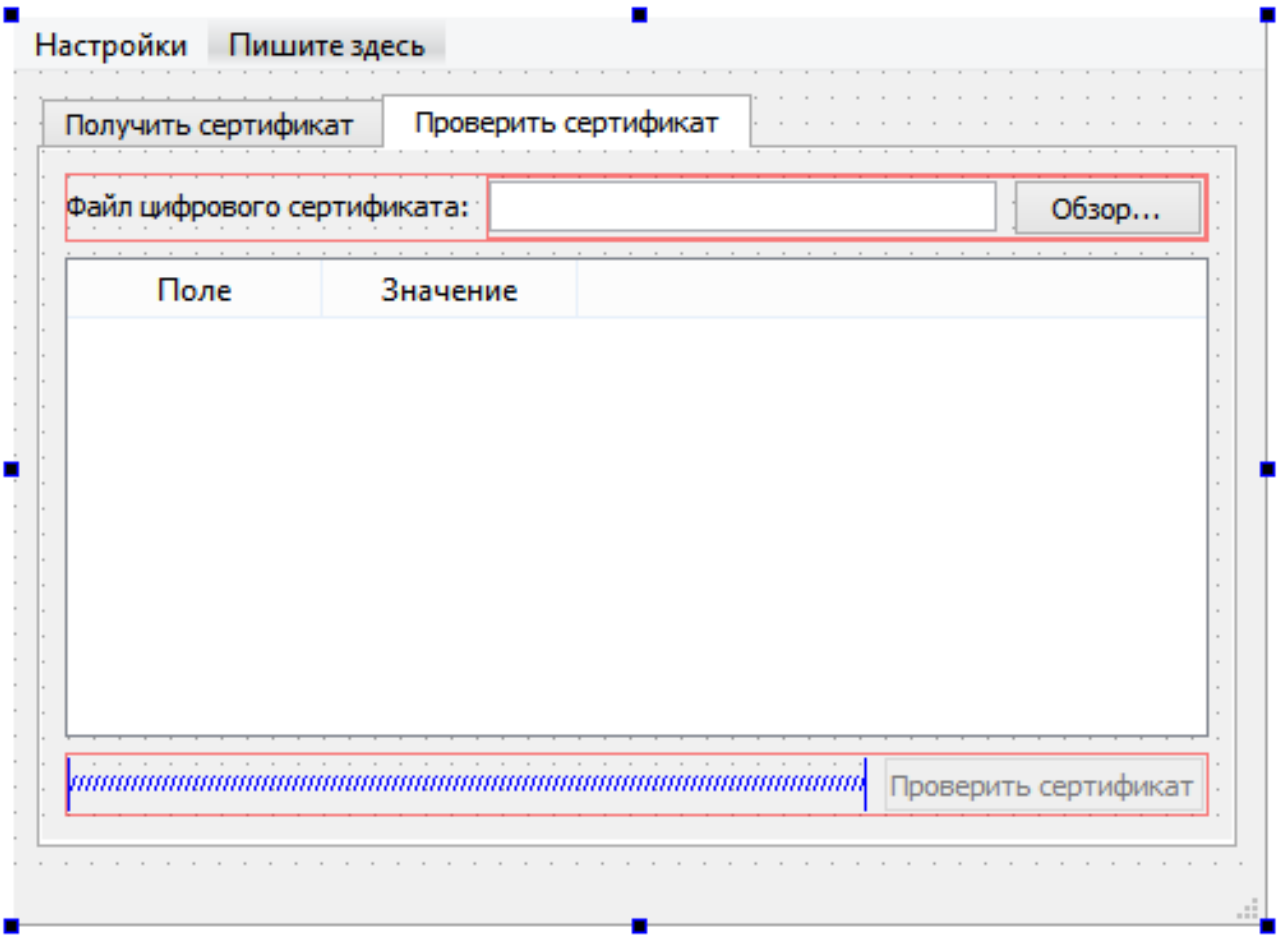


Рисунок 16 — Вкладка «Проверить сертификат»

Так же главное окно включает строку состояния, куда выводится статус соединение с сервером, и строку меню, где задается IP-адрес сервера, к которому будет выполнено подключение.

Класс MainWindow включает в себя следующие функции:

- функция connectToServ выполняет подключение к серверу;

- функция sendToServer предназначена для отправки серверу массива байт через TCP-сокет. Функция принимает в качестве аргумента массив байт, предназначенный для отправки; возвращает: 0 — при успешной отправке, 1 — при возникновении какой-либо ошибки;

- функция readData считывает из TCP-сокета массив байт, полученный от серверной части УЦ. Функция возвращает массив считанных байт;

- функция readFile считывает данные из указанного файла в двоичном виде. Функция в качестве аргумента принимает символьную строку,

содержащую путь к файлу, и массив байт, куда нужно произвести запись. При успешной записи функция возвращает 0, иначе 1;

- функция slotReadyRead выполняется при наличии данных в сокете, готовых для чтения;

- функция slotConnected выполняется после установления соединения с сервером;

- функция slotDisconnected выполняется после разрыва соединения с сервером;

- функция on_getCert_Button_clicked выполняется после нажатия кнопки «Отправить запрос на сертификат». Функция проверяет все ли поля заполнены, при заполнении всех полей формирует и отправляет серверу запрос иначе информирует пользователя о необходимости заполнения пустых полей;

- функция on_cert_Button_clicked выполняется после нажатия кнопки выбора файла сертификата для проверки. Функция открывает диалоговое окно для выбора файла и после считывания выбранного файла заполняет таблицу атрибутами сертификата;

- функция on_checkCert_Button_clicked выполняется после нажатия кнопки «Проверить сертификат». Функция формирует запрос на проверку выбранного цифрового сертификата и отправляет запрос серверу;

- функция on_action_clicked выполняется после выбора в строке меню пункта «Настройка — Адрес сервера». Функция вызывает диалоговое окно, в котором вводится IP-адрес и номер порта серверной части УЦ.

6 Руководство по использованию макета удостоверяющего центра

6.1 Руководство по использованию серверной части

Настройки удостоверяющего центра хранятся в файле serverSetting.ini в корневом каталоге программы. Содержимое файла по умолчанию представлено на рисунке 13.

Параметры имееют следующее значение:

- General/port — номер порта, который будет прослушивать сервер;
- PATH/selfSignedCert — путь к файлу самозаверенного сертификата;
- PATH/privKeyCA — путь к файлу секретного ключа УЦ;
- PATH/clientCerts — путь к каталогу, куда будут сохраняться файлы сертификатов клиентов;
- PATH/clientKeys — путь к каталоку, куда будут сохраняться файлы ключей клиентов;
- keyParam — размер ключе клиентов, причем значение keyParam/n должно быть больше значения keyParam/k минимум на 4;
- BD/path — путь к базе данных, содержащей реестры выданных и аннулированных сертификатов.

Первый запуск программы показан на рисунке 17.

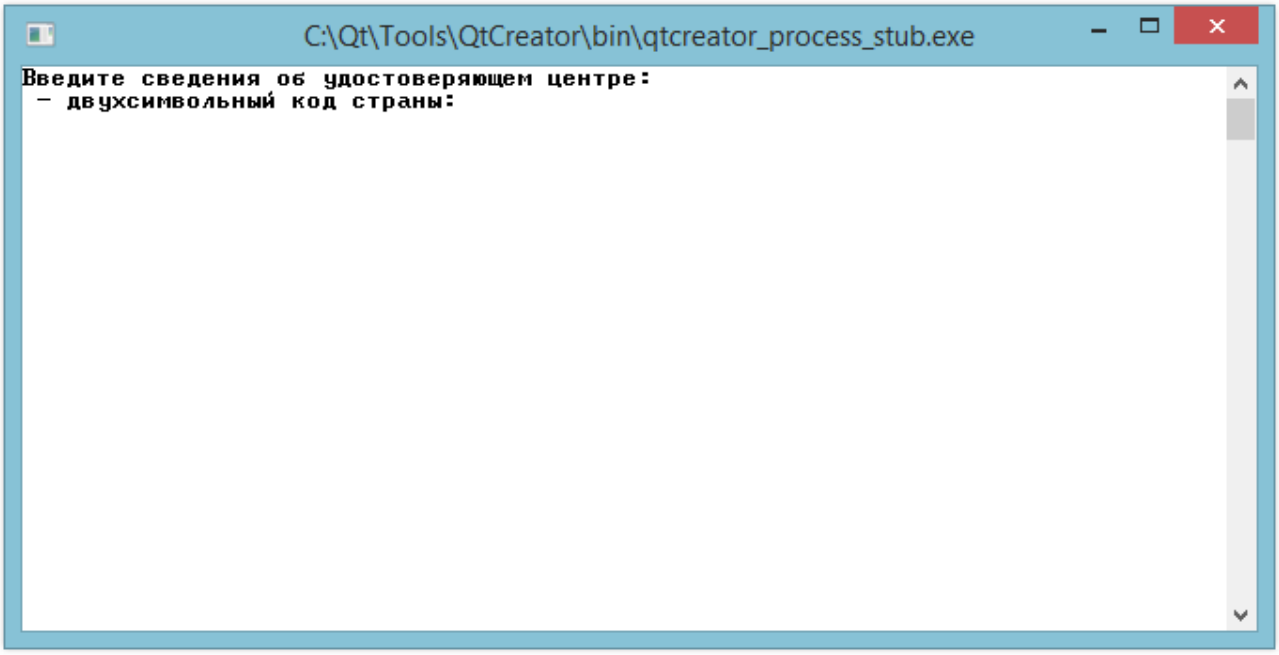


Рисунок 17 — Первый запуск программы

При первом запуске программа запросит пользователя указать сведения об УЦ:

- двухсимвольный код страны;
- регион;
- населенный пункт;
- наименование организации;
- наименование подразделения организации;
- общее название УЦ.

При указании сведений об УЦ можно использовать латинские буквы, цифры, пробелы и знаки препинания. Далее необходимо указать рамеры ключевой матрицы. При указании размера число n должно быть больше числа k минимум на 4.

После ввода необходимых данных будут созданы необходимые для работы удостоверяющего центра файлы:

- самозаверенный сертификат открытого ключа УЦ;
- файл секретного ключа УЦ для подписи сертификатов клиентов;
- файл, содержащий реестры выданных этим удостоверяющим центром и аннулированных сертификатов (представленный в виде базы

данных SQLite).

Расположение файлов будет соответсвовать указанному в файле serverSetting.ini.

После чего в консоль будет выведены сообщение об успешном запуске сервера, что говорит о готовности УЦ к обработке запросов пользователей и вводе локальных команд. При повторных запусках программы, УЦ будет сразу готов к обработке запросов при наличии необходимых вышеперечисленных файлов. Запущенный сервер показан на рисунке 18.

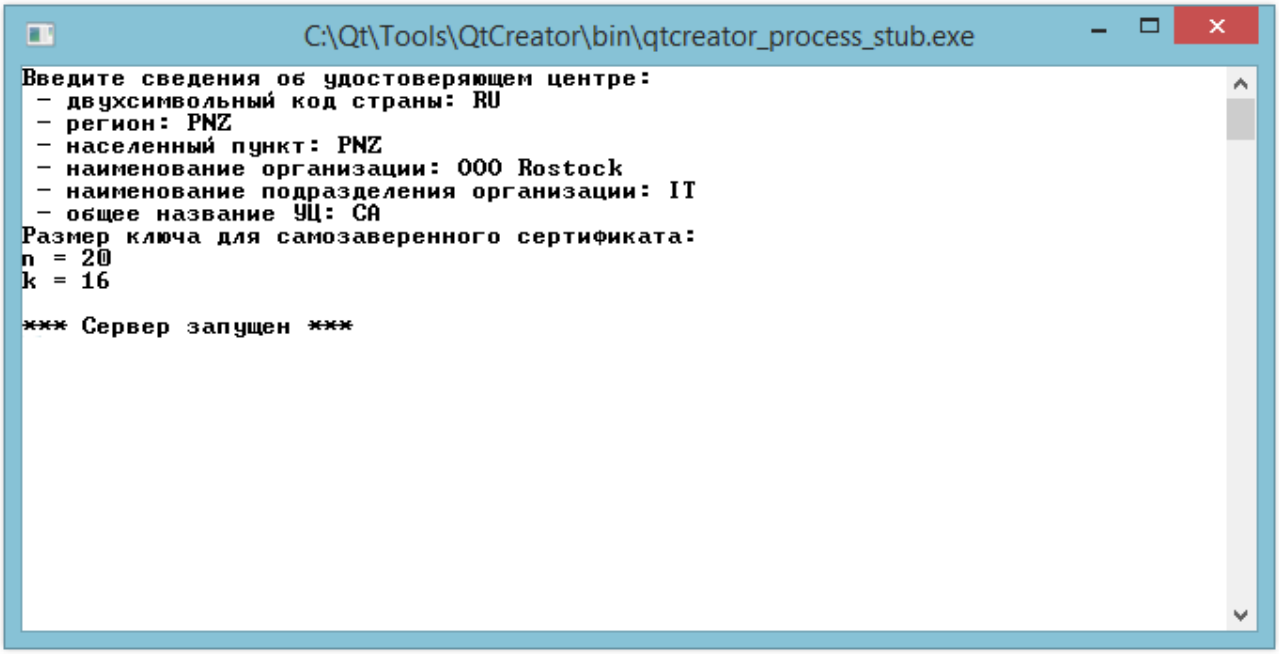


Рисунок 18 — Готовый к обработке клиентских запросов сервер УЦ

Для получения помощи по обрабатываемым локальным командам необходимо ввести команду help. Результат выполнения команды показан на рисунке 19.

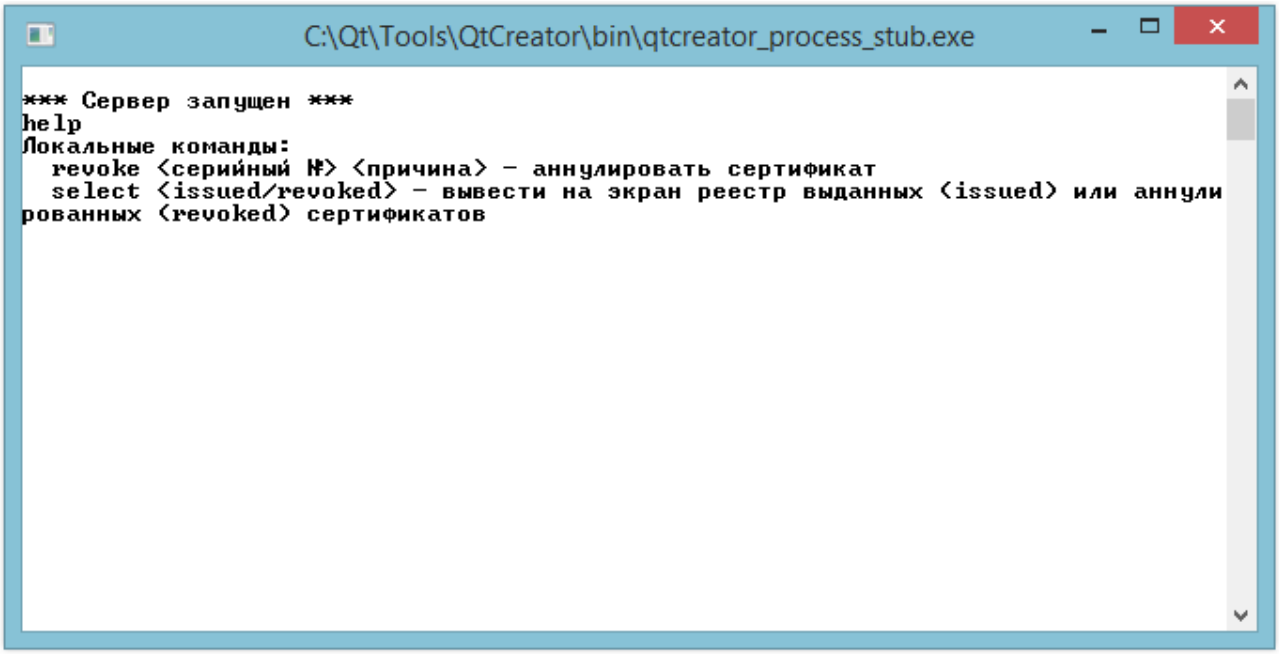


Рисунок 19 — Результат выполнения команды help

Для вывода в консоль реестра выданных сертификатов необходимо ввести комманду select issued. Выводеденный реестр будет представлен в виде таблицы со следующими полями:

- № — серийный номер сертификата;

- дествителен с — дата и время, начиная с которого сертификат будет считаться действительным;

- дествителен до — дата и время, после которого сертификат будет считаться не действительным;

- субъект — сведения о владельце сертификата.

Пример выполнения команды показан на рисунке 20.

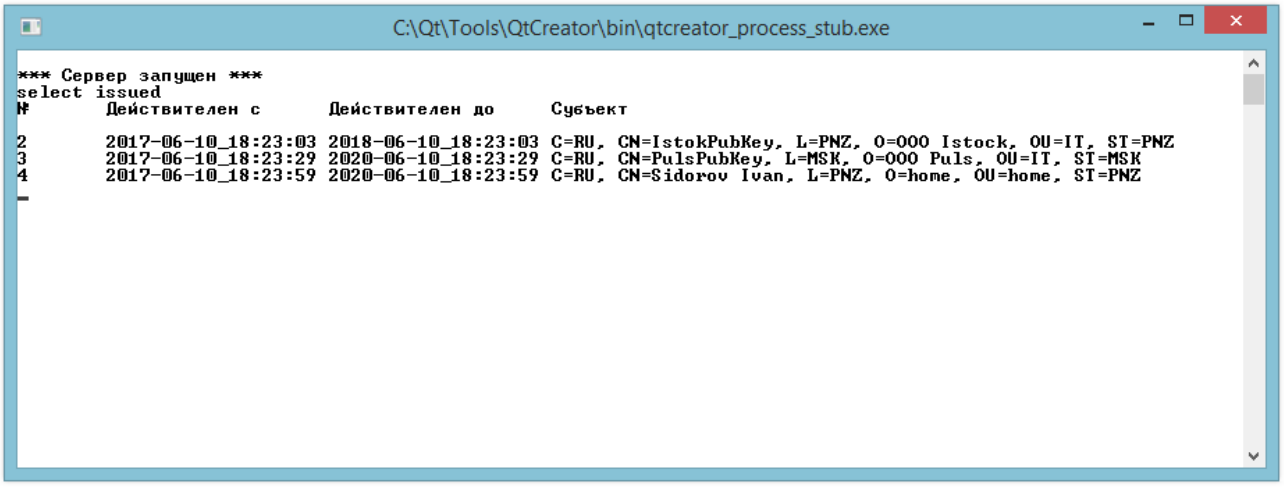


Рисунок 20 — Выведенный в консоль реестр выданных сертификатов

Для аннулирования сертификата необходимо ввести комманду revoke <серийный №> <причина>. Серийный номер указывается в виде шестнадцатиричного числа; для указания причины аннулирования сертификата можно использовать латинские буквы, цифры и знаки препинания. При успешном аннулировании сертификата будет выведено сообщение «Сертификат отозван». Пример выполнения команды показан на рисунке 21.

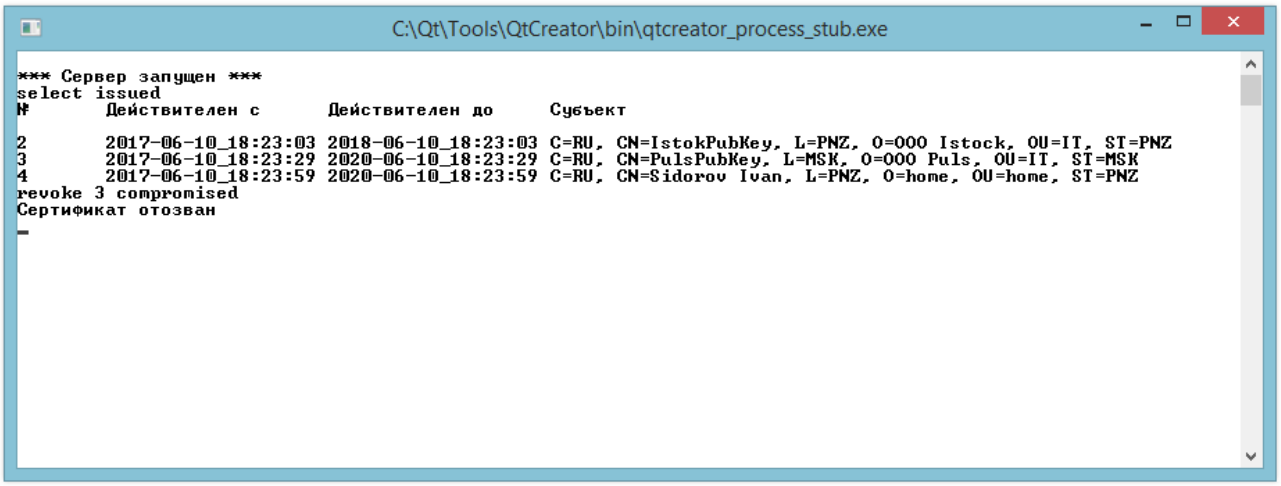


Рисунок 21 — Результат выполнения команды аннулирования сертификата

Для вывода в консоль реестра аннулированных сертификатов необходимо ввести комманду select revoked. Выводеденный реестр будет представлен в виде таблицы со следующими полями:

- № — серийный номер сертификата;
- дата аннулирования — дата и время аннулирования сертификата;
- причина аннулировнаия — причина, по которой сертификат был аннулирован.

Пример выполнения команды показан на рисунке 22.

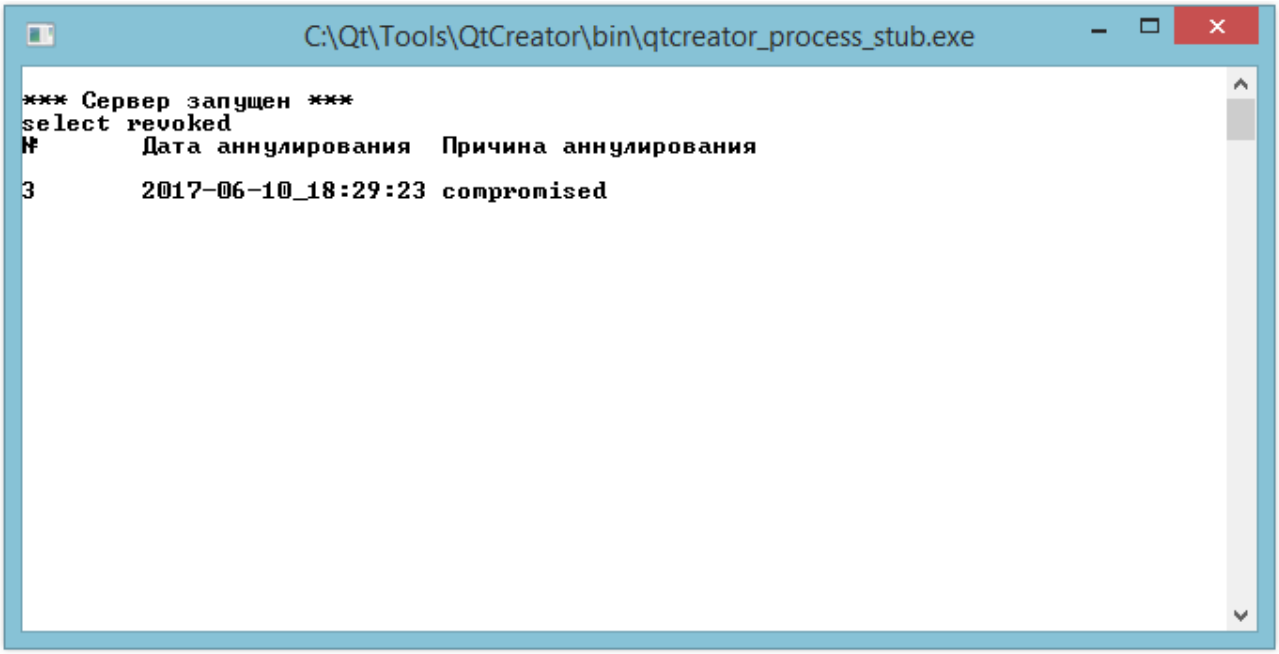


Рисунок 22 — Выведенный в консоль реестр аннулированных сертификатов

Все созданные удостоверяющим центром файлы цифровых сертификатов и ключей сохраняются по путям, указанным в файле serverSetting.ini.

6.2 Руководство по использованию клиентской части

Для подключения к серверной части УЦ необходимо указать его IP-адрес. Для этого необходимо в строке меню выбрать пункт «Настройка — Адрес сервера» и в появившемся диалоговом окне указать IP-адрес и номер порта в формате «255.255.255.255:9999», где до двоеточия указывается IPv4-адрес, а после номер порта. Деалоговое окно показано на рисунке 23.

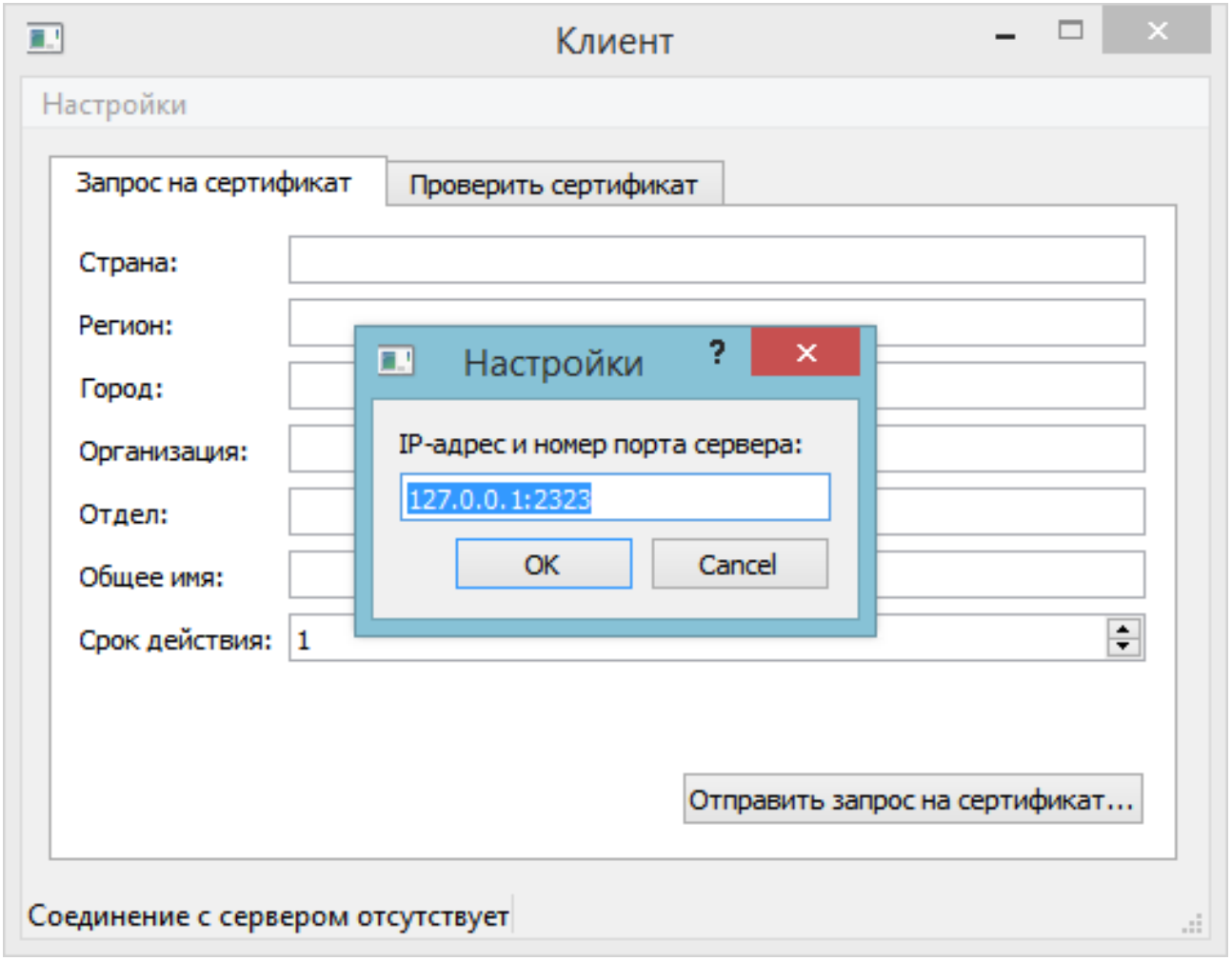


Рисунок 23 — Диалоговое окно настройки адреса серверной части УЦ

После нажатия кнопки «OK» в диалоговом окне настройки будет выполнено подключение к серверной части УЦ. Текущее состояние соединения можно проверить в строке состояния.

Для формирования и отправки запроса на выдачу сертификата необходимо во вкладке «Запрос на сертификат» указать в соответсвующих полях информацию о соискателе цифрового сертификата открытого ключа. При заполнении полей можно использовать латинские буквы, цифры, пробелы и знаки препинания, поле «Страна» должно содержать двухбуквенный идентификатор страны. Пример заполнения представлен на рисунке 24.

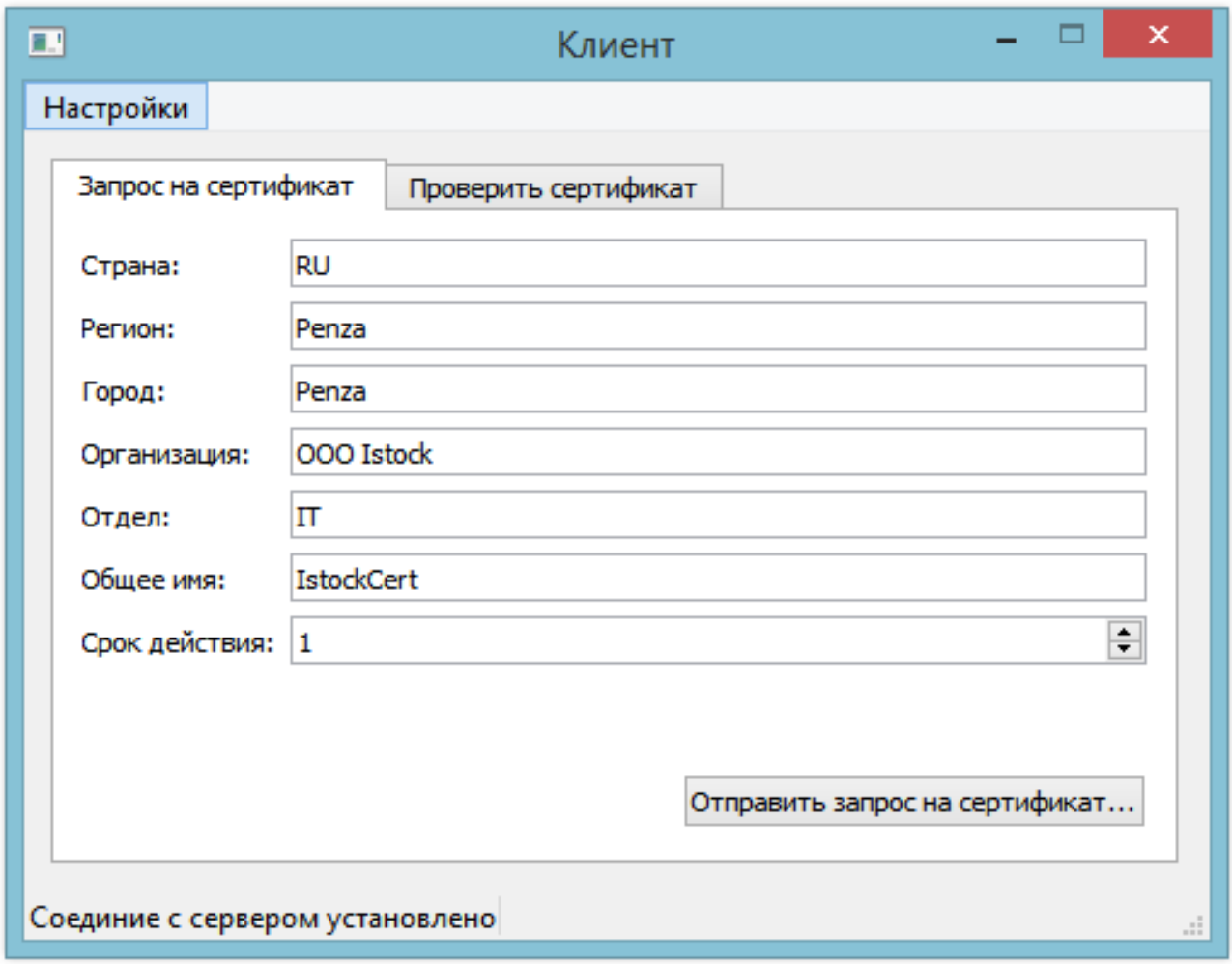


Рисунок 24 — Пример заполнения полей для отправки запроса на получение сертификата

После заполнения всех полей нужно нажать на кнопку «Отправить запрос на сертификат...», при успешном формировании и отправки запроса появится сообщение.

Для проверки сертификата на подлинность необходимо открыть вкладку «Проверить сертификат». Далее необходимо указать путь к файлу сертификата открытого ключа в соответствующей строке или, нажав на кнопку «Обзор...», выбрать файл сертификата в появившемся диалоговом окне. После этого отобразится таблица с полями выбранного сертификата и их значениями. Пример открытого файла сертификата показан на рисуке 25.

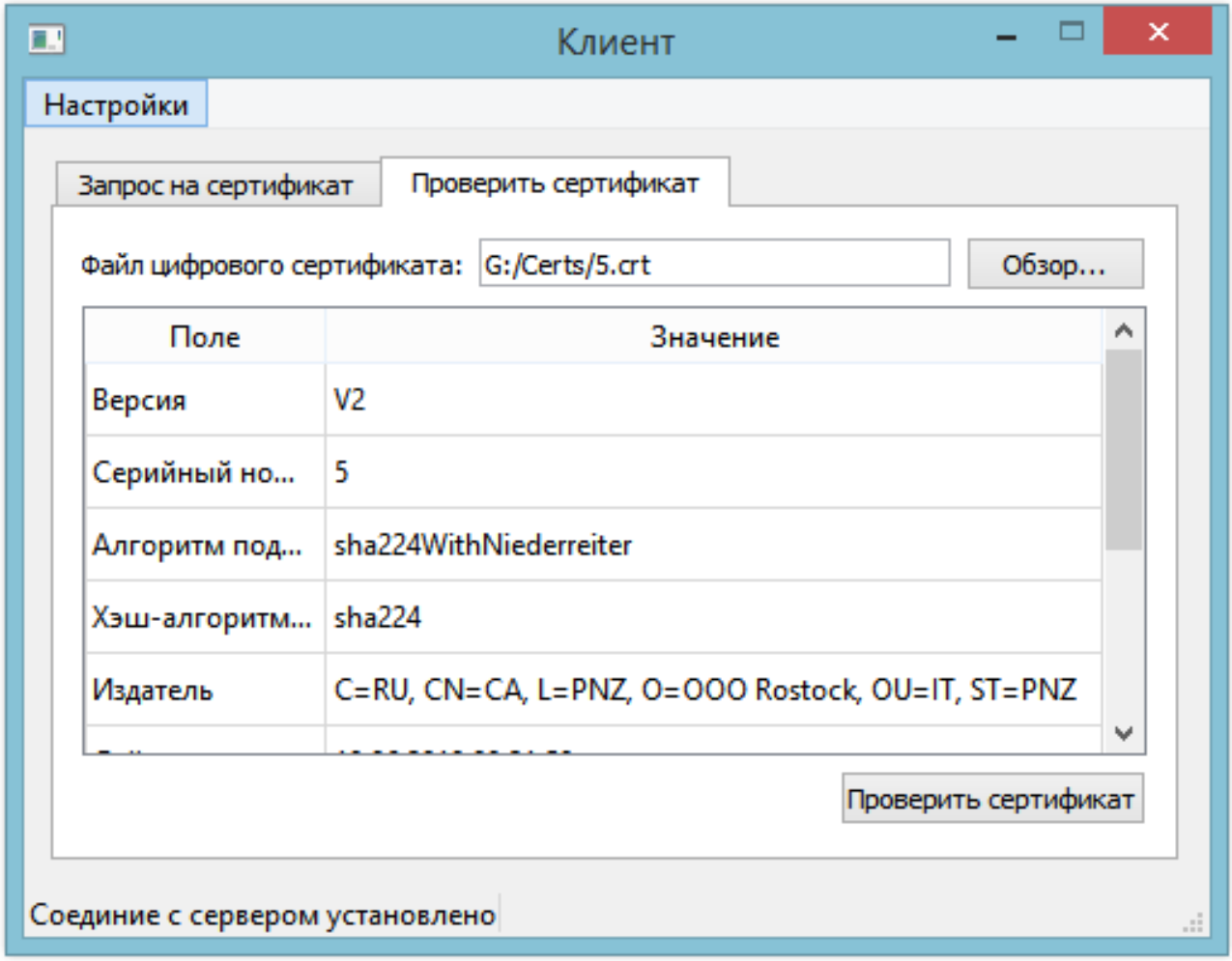


Рисунок 25 — Пример открытого файла цифрового сертификата

После выбора сертификата становится доступной кнопка «Проверить сертификат», после ее нажатия сертификат будет передан серверной части УЦ для проверки. Результат проверки отобразится в виде диалогового сообщения.

Возможны следующие варианты ответов:

- сертификат действителен;
- срок дейтсвия сертификата истек или еще не наступил;
- целостность сертификата нарушена;
- сертификат аннулирован удостоверяющим центром.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы были рассмотрены инфраструктура открытых ключей и постквантовые асимметричные криптографические алгоритмы.

В результате выполнения дипломного проектирования был разработан алгоритм работы макета удостоверяющего центра по модели клиент-сервер, использующего для формирования электронной подписи криптосистему Нидеррайтера, и осуществлена его программная реализация. Макет удостоверяющего центра обеспечивает выдачу, проверку и аннулирование цифровых сертификатов открытого ключа.

Таким образом, задание на выпускную квалификационную работу выполнено в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Королёв В. И. Архитектурное построение инфраструктуры открытых ключей интегрированного информационного пространства [Электронный ресурс]. — Режим доступа: http://bit.mephi.ru/wp-content/uploads/2015-3/part_08.pdf;

2 Федеральный закон «Об электронной подписи». — М. : КонсультантПлюс, 2016. — 16 с. — (Закон Российской Федерации);

3 Рекомендация МСЭ-Т X.509 : ИСО/МЭК 9594-8. — Сектор стандартизации электросвязи МСЭ, 2005, 174 с;

4 Рекомендация МСЭ-Т X.690 : ИСО/МЭК 8825-1:2015. — Сектор стандартизации электросвязи МСЭ, 2015, 38 с;

5 Валиев К. А. Квантовые компьютеры: надежды и реальность / К. А. Валиев, А. А. Кокин. — И.: Изд-во НИЦ, 2001. — 352 с;

6 Создана первая масштабируемая реализация квантового алгоритма [Электронный ресурс]. — Режим доступа: https://lenta.ru/news/2016/03/04/algorithm/;

7 D. J. Bernstein Introduction to post-quantum cryptography // Department of Computer Science, University of Illinois at Chicago. — 2009;

8 Experimenting with Post-Quantum Cryptography [Электронный ресурс]. — Режим доступа: https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html;

9 A. Valentijn Goppa Codes and Their Use in the McEliece Cryptosystems / A. Valentijn, M. Finiasz, N. Sendrier. — Syracuse University SURFACE. — 2015;

10 Самохина М. А. Модификации криптосистемы Нидеррайтера, их стойкость и практические применения // Труды МФТИ. — 2009. — Том 1, № 2;

11 N. Courtois How to achieve a McEliece-based Digital Signature Scheme // Advances in Cryptology — ASIACRYPT 2001.

Приложение А

(обязательное)

Содержимое файла servermain.cpp

```
#include <QCoreApplication>
#include "niedercrypt.h"
#include "x509cert.h"
#include "structkey.h"
#include "TcpServer.h"
#include <QtDebug>
#include "QtSql/QSqlDatabase"
#include "QSqlRecord"

int servermain(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    TcpServer Server ;
    Server.startServer();

    return a.exec();
}
```

Приложение Б

(обязательное)

Содержимое файла TcpServer.h

```
#ifndef TCPSERVER_H
#define TCPSERVER_H
#include <QTcpServer>
#include <QTcpSocket>
#include <QAbstractSocket>
#include <QtDebug>
#include <x509cert.h>
#include "QtSql/QSqlDatabase"
#include "QSqlRecord"
#include <QtConcurrentRun>
class TcpServer : public QObject
{
    Q_OBJECT
private:
    QTcpServer* m_ptcpServer;
    quint16     m_nNextBlockSize;
    int nPort;
    X509Cert cert;
    QString SelfSignedCertName, KeySelfSignedCertName, pathKey,
   pathCert;
    unsigned char param_k, param_n;
private:
    void writeDefaultParam();
    bool createConnectionBD();
    int saveToFile(QString &, QByteArray &);
    QByteArray openFile(QString &);
    void readCommand();
    void print(QString string);
    int commandRevoke(QString command);
    int commandSelect(QString table);
public:
    TcpServer(QObject* pwgt = 0);
    int startServer();
public slots:
    virtual void slotNewConnection();
            void slotReadClient   ();
};
#endif // TCPSERVER_H
```

Приложение В

(обязательное)

Содержимое файла TcpServer.cpp

```
#include "TcpServer.h"
#include <QtCore>
#include <QTextCodec>
#include <iostream>
#include <QTextStream>
QTextStream cout(stdout);
QTextStream cin(stdin);
TcpServer::TcpServer(QObject* pwgt /*=0*/) :
    QObject(pwgt), m_nNextBlockSize(0){;}
int TcpServer::startServer()
{
    if (!createConnectionBD())
    {
        qDebug() << "Cannot␣open␣database:";
        return 1;
    }
    if ((!QFile(SelfSignedCertName).exists()) ||
  (!QFile(KeySelfSignedCertName).exists()))
    {
        X509Cert CA;
        print("Введите␣сведения␣об␣удостоверяющем␣центре:\n");
        do
        {
            print("␣–␣двухсимвольный␣код␣страны:␣");
            CA.IssuerDN["C"] = cin.readLine().toUtf8();
        }
        while(CA.IssuerDN["C"].length() != 2);
        print("␣–␣регион:␣");
        CA.IssuerDN["ST"] = cin.readLine().toUtf8();
        print("␣–␣населенный␣пункт:␣");
        CA.IssuerDN["L"] = cin.readLine().toUtf8();
        print("␣–␣наименование␣организации:␣");
        CA.IssuerDN["O"] = cin.readLine().toUtf8();
        print("␣–␣наименование␣подразделения␣организации:␣");
        CA.IssuerDN["OU"] = cin.readLine().toUtf8();
        print("␣–␣общее␣название␣УЦ:␣");
        CA.IssuerDN["CN"] = cin.readLine().toUtf8();
        NiederCrypt crypt;
        print("Размер␣ключа␣для␣самозаверенного␣сертификата:\nn␣=␣
  ");
        int n, k;
        cin >> n;
        print("k␣=␣");
        cin >> k;
        crypt.genKey(n, k);
        QByteArray privKey = CA.KeyToByte(crypt.getPrivKey());
        saveToFile(KeySelfSignedCertName, privKey);
        saveToFile(SelfSignedCertName,
  CA.getSelfSignedCert(CA.KeyToByte(crypt.getPubKey()), privKey));
        QSqlQuery query;
```

```
        QString   str  = "CREATE␣TABLE␣issuedCertificates␣(␣"
                         "serialNo␣INTEGER␣PRIMARY␣KEY␣NOT␣
   NULL,␣"
                         "subjectDN␣␣␣TEXT␣NOT␣NULL,␣"
                         "NotAfter␣␣TEXT␣NOT␣NULL,␣"
                         "NotBefore␣␣TEXT␣NOT␣NULL␣);␣";
        if (!query.exec(str))
        {
            qDebug() << "Unable␣to␣create␣a␣table␣
   issuedCertificates";
            return 1;
        }
        str = "CREATE␣TABLE␣revokedCertificates␣(␣"
             "serialNo␣INTEGER␣PRIMARY␣KEY␣NOT␣NULL,␣"
             "date␣TEXT␣NOT␣NULL,␣"
             "comment␣TEXT␣NOT␣NULL␣);␣";
        if (!query.exec(str))
        {
            qDebug() << "Unable␣to␣create␣a␣table␣
   revokedCertificates";
            return 1;
        }
    }
    if (cert.openCert(openFile(SelfSignedCertName)) != 0)
    {
        print("Файл␣"+SelfSignedCertName.append("␣не␣является␣
   сертификатом␣открытого␣ключа"));
        return 1;
    }
    m_ptcpServer = new QTcpServer(this);
    if (!m_ptcpServer->listen(QHostAddress::Any, nPort))
    {
        qDebug() << "Unable␣to␣start␣the␣server:" <<
   m_ptcpServer->errorString() <<"\n";
        m_ptcpServer->close();
        return 1;
    }
    print("\n***␣Сервер␣запущен␣***\n");
    connect(m_ptcpServer, SIGNAL(newConnection()),
            this,         SLOT(slotNewConnection())
           );
    QFuture<void> future = QtConcurrent::run(this,
   &TcpServer::readCommand);
    return 0;
}

void TcpServer::print(QString string)
{
    QTextCodec *codec = QTextCodec::codecForName("CP866");
    QByteArray encodedString = codec->fromUnicode(string);
```

```
    std::cout << encodedString.data();
}

int TcpServer::commandRevoke(QString command)
{
    QString comment;
    QStringList commands = command.split('␣',
   QString::SkipEmptyParts);
    if (commands.size() < 3)
        return 1;
    unsigned serialNo;
    bool b;
    serialNo = commands[1].toInt(&b, 16);
    if ((commands[0] != "revoke") && (!b))
        return 1;
    if (commands.size() == 3)
        comment = commands[2];
    else
    {
        if (commands[2][0] != '"')
            return 1;
        if (command.count('"') != 2)
            return 1;
        comment = command.section('"', 1, 1);
    }
    QString strF = "SELECT␣*␣FROM␣issuedCertificates␣"
                   "WHERE␣serialNo=%1";
    strF = strF.arg(serialNo);
    QSqlQuery query;
    if (!query.exec(strF))
        return 1;
    QSqlRecord rec = query.record();
    query.next();
    if (query.isNull("serialNo"))
    {
        print("Неправильный␣серийный␣номер\n");
    }
    strF = "SELECT␣*␣FROM␣revokedCertificates␣"
                   "WHERE␣serialNo=%1";
    strF = strF.arg(serialNo);
    if (!query.exec(strF))
        return 1;
    rec = query.record();
    query.next();
    if (!(query.isNull("serialNo")))
    {
        print("Сертификат␣уже␣отозван\n");
    }
    else
    {
```

```
        strF = "INSERT␣INTO␣␣revokedCertificates␣(serialNo,␣date,␣
   comment)␣"
               "VALUES(%1,␣'%2',␣'%3');";
        strF = strF.arg(serialNo)
                .arg((QString)
   QDateTime::currentDateTime().toString("yyyy—MM—dd_hh:mm:ss"))
               .arg(comment);
        if (!query.exec(strF)) {
            qDebug() << "Unable␣to␣do␣insert␣opeation";
        }
        else
            print("Сертификат␣отозван\n");
    }
    return 0;
}

int TcpServer::commandSelect(QString table)
{
    QString strF = "SELECT␣*␣FROM␣";
    if (table == "issued")
    {
        strF.append("issuedCertificates␣");
        QSqlQuery query;
        if (!query.exec(strF))
            return 1;
        QSqlRecord rec = query.record();
        QStringList k;
        k << "№\t" << "Действителен␣с␣␣␣␣␣␣" << "Действителен␣до␣␣␣
   ␣␣" << "Субъект";
        for(int i = 0; i < k.length(); i++)
            print(k[i]);
        print("\n\n");
        while (query.next())
        {
            print(query.value(0).toString() + '\t');
            print(query.value(2).toString() + '␣');
            print(query.value(3).toString() + '␣');
            print(query.value(1).toString() + '\n');
        }
        return 0;
    }
    if (table == "revoked")
    {
        strF.append("revokedCertificates␣");
        QSqlQuery query;
        if (!query.exec(strF))
            return 1;
        QSqlRecord rec = query.record();
        QStringList k;
```

```
        k << "№\t" << "Дата␣аннулирования␣␣" << "Причина␣
   аннулирования\n\n";
        for(int i = 0; i < k.length(); i++)
            print(k[i]);
        while (query.next())
        {
            print(query.value(0).toString() + '\t');
            print(query.value(1).toString() + '␣');
            print(query.value(2).toString() +'\n');
        }
        return 0;
    }
    return 1;
}

void TcpServer::readCommand()
{
    QString command;
    while (1)
    {
        command = cin.readLine();
        QString comment;
        QStringList commands = command.split('␣',
   QString::SkipEmptyParts);
        if (commands.length() > 0)
        {
            if (commands[0] == "select")
            {
                if(commandSelect(commands[1]))
                    continue;
            }
            if (commands[0] == "revoke")
            {
                if(commandRevoke(command))
                    continue;
            }
            if (commands[0] == "help")
                print("Локальные␣команды:\n"
                  "␣␣revoke␣серийный<␣№>␣причина<>␣—␣аннулировать␣
   сертификат\n"
                  "␣␣select␣<issued/revoked>␣—␣вывести␣на␣экран␣
   реестр␣выданных␣(issued)␣или␣аннулированных␣(revoked)␣
   сертификатов\n");
        }
    }
}

/*virtual*/ void TcpServer::slotNewConnection()
{
```

```
    QTcpSocket* pClientSocket =
   m_ptcpServer->nextPendingConnection();
    connect(pClientSocket, SIGNAL(disconnected()),
            pClientSocket, SLOT(deleteLater())
           );
    connect(pClientSocket, SIGNAL(readyRead()),
            this,          SLOT(slotReadClient())
           );
}

void TcpServer::slotReadClient()
{
    QTcpSocket* pClientSocket = (QTcpSocket*)sender();
    QDataStream in(pClientSocket);
    in.setVersion(QDataStream::Qt_4_2);
    QByteArray data;
    for (;;) {
        if (!m_nNextBlockSize) {
            if (pClientSocket->bytesAvailable() < sizeof(quint16))
   {
                break;
            }
            in >> m_nNextBlockSize;
        }

        if (pClientSocket->bytesAvailable() < m_nNextBlockSize) {
            break;
        }
        m_nNextBlockSize = 0;
        in >> data;
    }
    switch ((unsigned char)data[0])
    {
    case 0x1:
    {
        data.remove(0, 1);
        QSqlQuery query;
        if (!query.exec("SELECT␣MAX(serialNo)␣AS␣max␣FROM␣
   issuedCertificates;")) {
            qDebug() << "Unable␣to␣execute␣query␣-␣exiting";
        }
        QSqlRecord rec = query.record();
        query.next();
        X509Cert certClient;
        certClient.SerialNum =
   query.value(rec.indexOf("max")).toInt() + 1;
        if (certClient.SerialNum == 1) certClient.SerialNum = 2;
        certClient.IssuerDN = cert.SubjectDN;
        NiederCrypt crypt;
        crypt.genKey(param_n, param_k);
```

```
     data = certClient.getCert(crypt.getPubKey(), data,
openFile(KeySelfSignedCertName));
     saveToFile((pathKey + "/%1.key").arg(certClient.SerialNum,
16), certClient.KeyToByte(crypt.getPrivKey()));
     saveToFile((pathCert +
"/%1.crt").arg(certClient.SerialNum, 16), data);
     QString strF =
             "INSERT␣INTO␣␣issuedCertificates␣(serialNo,␣
subjectDN,␣NotAfter,␣NotBefore)␣"
             "VALUES(%1,␣'%2',␣'%3',␣'%4');";
     QString subject;
     QMapIterator<QByteArray, QByteArray>
iter(certClient.SubjectDN);
     while (1)
     {
         iter.next();
         subject += iter.key() + "=";
         subject += iter.value();
         if (iter.hasNext())
             subject += ",␣";
         else break;
     }
     strF = strF.arg(certClient.SerialNum, 16)
             .arg(subject)
             .arg((QString)
certClient.NotAfter.toString("yyyy—MM—dd_hh:mm:ss"))
             .arg((QString)
certClient.NotBefore.toString("yyyy—MM—dd_hh:mm:ss"));
     if (!query.exec(strF)) {
         qDebug() << "Unable␣to␣do␣insert␣opeation";
     }
             data = "success";
     break;
 }
 case 0x2:
 {
     data.remove(0, 1);
     X509Cert certClient;
     if (certClient.openCert(data) != 0)
     {
         data = "err";
         break;
     }
     switch (certClient.checkCert(data, cert.PubKey))
     {
     case 0:
     {
         QDateTime dateToday = QDateTime::currentDateTime();
         if ((certClient.NotAfter < dateToday) &&
(certClient.NotBefore >= dateToday))
```

```
            {
                QString strF = "SELECT␣serialNo␣FROM␣
    revokedCertificates␣"
                               "WHERE␣serialNo=%1";
                strF = strF.arg(certClient.SerialNum);
                QSqlQuery query;
                if (!query.exec(strF)) {
                    qDebug() << "Unable␣to␣execute␣query␣—␣
    exiting";
                    data = "err";
                }
                else
                {
                    query.record();
                    query.next();

                    if (query.isNull("serialNo"))
                        data = "valid";
                    else
                        data = "revoked";
                }
            }
            else
                data = "expired";
            break;
        }
        case 1:
            data = "notValid";
            break;
        default:
            data = "err";
        }
        break;
    }
    default:
        data = "err";
        break;
    }
    QByteArray  arrBlock;
    QDataStream out(&arrBlock, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_2);
    out << quint16(0) << data;
    out.device()—>seek(0);
    out << quint16(arrBlock.size() — sizeof(quint16));
    pClientSocket—>write(arrBlock);
}

void TcpServer::writeDefaultParam()
{
    QSettings sett("serverSetting.ini", QSettings::IniFormat);
```

```
    sett.setValue("port", "2323");
    sett.setValue("PATH/selfSignedCert", "selfSignedCert.crt");
    sett.setValue("PATH/privKeyCA", "privKeySelfSignedCert.key");
    sett.setValue("PATH/clientCerts", "clientCerts");
    sett.setValue("PATH/clientKeys", "clientKeys");
    sett.setValue("keyParam/n", "20");
    sett.setValue("keyParam/k", "16");
    sett.setValue("BD/path", "CA.db3");
}

bool TcpServer::createConnectionBD()
{
    if (!QFile("serverSetting.ini").exists())
        writeDefaultParam();
    QSettings sett("serverSetting.ini", QSettings::IniFormat);
    SelfSignedCertName =
   sett.value("PATH/selfSignedCert").toString();
    KeySelfSignedCertName =
   sett.value("PATH/privKeyCA").toString();
    param_k = sett.value("keyParam/k").toUInt();
    param_n = sett.value("keyParam/n").toUInt();
    pathCert = sett.value("PATH/clientCerts").toString();
    pathKey = sett.value("PATH/clientKeys").toString();
    QDir().mkpath(pathCert);
    QDir().mkpath(pathKey);
    nPort = sett.value("port").toInt();
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName(sett.value("BD/path").toString());
    db.setUserName(sett.value("BD/userName").toString());
    db.setHostName(sett.value("BD/hostName").toString());
    db.setPassword(sett.value("BD/passwd").toString());
    if (!db.open())
        return false;
    return true;
}

int TcpServer::saveToFile(QString &path, QByteArray &data)
{
    QFile file(path);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Truncate))
        return 1;
    QDataStream outf(&file);
    for(int i = 0; i< data.size(); i++)
        outf << (unsigned char)data[i];
    if(outf.status() != QDataStream::Ok)
    {
        qDebug() << "Ошибка␣записи␣файла␣" << path;
        return 1;
    }
    file.close();
```

```
    return 0;
}

QByteArray TcpServer::openFile(QString &path)
{
    QByteArray data;
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly))
    {
        qDebug() << "Ошибка␣чтения␣файла␣" << path;
        return data;
    }
    data = file.readAll();
    file.close();
    return data;
}
```

Приложение Г

(обязательное)

Содержимое файла x509cert.h

```
#ifndef X509CERT_H
#define X509CERT_H
#include "structKey.h"
#include "niedercrypt.h"
#include <QtCore>
#include "QtSql/QSqlDatabase"
#include "QSqlQuery"
class X509Cert
{
public:
    unsigned __int32    SerialNum;
    PublicKey   PubKey;
    QMap<QByteArray, QByteArray> SubjectDN, IssuerDN;
    QByteArray
        SignatureAlgorithm = "",
        Signature = "";
    QDateTime NotAfter, NotBefore;
private:
    const QMap<QByteArray, unsigned __int32> X509NameConst = {
        { "C",  0x550406 },
        { "ST", 0x550408 },
        { "L",  0x550407 },
        { "O",  0x55040A },
        { "OU", 0x55040B },
        { "CN", 0x550403 }
    };
    const QMap<unsigned __int32, QByteArray> revX509NameConst = {
        { 0x550406, "C" },
        { 0x550408, "ST"},
        { 0x550407, "L"},
        { 0x55040A, "O"},
        { 0x55040B, "OU"},
        { 0x550403, "CN"}
    };
public:
    QByteArray X509NameToDer(QMap<QByteArray, QByteArray> &Name);
    QByteArray getCert(PublicKey &pubKey, QByteArray &req,
   QByteArray &prKey);
    QByteArray getSelfSignedCert(QByteArray &, QByteArray &);
    int checkCert(QByteArray &, PublicKey &);
    QByteArray KeyToByte(PublicKey key);
    QByteArray KeyToByte(PrivateKey key);
    int openCert(QByteArray &);
private:
    void addSignInfo(QByteArray &vec);
    unsigned char getSizeInt(unsigned __int32);
    void sizeToDer(QByteArray &, unsigned __int32);
    void intToDer(QByteArray &, unsigned __int32);
    void ByteToByte(QByteArray &, unsigned __int32, unsigned char);
    unsigned int sizeDecoder(QByteArray &);
```

```
    QByteArray genTSB(QByteArray &, int);
    unsigned __int32 derToInt(QByteArray &);
    QByteArray getData(QByteArray &);
    QMap<QByteArray, QByteArray> DerToX509Name(QByteArray &);
    PrivateKey keyDecoder(QByteArray &);
    PublicKey pubKeyDecoder(QByteArray &);
};
#endif // X509CERT_H
```

Приложение Д

(обязательное)

Содержимое файла x509cert.cpp

```
#include <x509cert.h>
#define SEQUENCE (unsigned char)0x30
#define INTEGER (unsigned char)0x02
#define OBJ_ID (unsigned char)0x06
#define UTCTime (unsigned char)0x17
#define GenTime (unsigned char)0x18
#define SET (unsigned char)0x31
#define UTF8String (unsigned char)0x0C
#define PrintString (unsigned char)0x13
#define NULLder (unsigned char)0x05
#define BIT_STR (unsigned char)0x03
void X509Cert::addSignInfo(QByteArray &vec)
{
    unsigned char a[] = { 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86,
   0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x05, 0x05, 0x00 };
    for (int i = 0; i < 15; i++)
        vec.push_back(a[i]);
}
QByteArray X509Cert::X509NameToDer(QMap<QByteArray, QByteArray>
   &Name)
{
    QByteArray vec;
    vec.push_back(SEQUENCE);
    QByteArray tmp;
    QMapIterator<QByteArray, QByteArray> i(Name);
    while (i.hasNext())
    {
        i.next();
        int size = i.value().size();
        if (size > 127)
            return NULL;
        tmp.push_back(SET);
        QByteArray tmp2;
        {
            tmp2.push_back(SEQUENCE);
            QByteArray tmp3;
            {
                tmp3.push_back(OBJ_ID);
                intToDer(tmp3, X509NameConst[i.key()]);
                if (i.key() == "C")
                    tmp3.push_back(PrintString);
                else
                    tmp3.push_back(UTF8String);
                tmp3.push_back((unsigned char)size);
                for (int k = 0; k < size; k++)
                    tmp3.push_back(i.value()[k]);
            }
            sizeToDer(tmp2, tmp3.size());
            tmp2.append(tmp3);
        }
```

```
        sizeToDer(tmp, tmp2.size());
        tmp.append(tmp2);
    }
    sizeToDer(vec, tmp.size());
    vec.append(tmp);
    return vec;
}

unsigned __int32 X509Cert::derToInt(QByteArray &arr)
{
    QByteArray data = getData(arr);
    unsigned __int32 m = 0;
    for(int i = 0; i < data.size(); i++)
        m |= (unsigned char)data[i] << (8*(data.size() - 1 - i));
    return m;
}

QByteArray X509Cert::getData(QByteArray &arr)
{
    QByteArray data;
    unsigned int size = sizeDecoder(arr);
    if (size > 127)
        arr.remove(2, (unsigned char)arr[1] ^ (unsigned char)0x80);
    arr.remove(0, 2);
    data = arr.left(size);
    arr.remove(0, size);
    return data;
}

QMap<QByteArray, QByteArray> X509Cert::DerToX509Name(QByteArray
   &name)
{
    QMap<QByteArray, QByteArray> r;
    while(name.size() > 0)
    {
        if ((unsigned char)name[0] == SET)
        {
            QByteArray set = getData(name);
            if ((unsigned char)set[0] == SEQUENCE)
            {
                QByteArray seq = getData(set);
                QByteArray key;
                if ((unsigned char)seq[0] == OBJ_ID)
                {
                    key = revX509NameConst[derToInt(seq)];
                }

                if (((unsigned char)seq[0] == UTF8String) ||
   ((unsigned char)seq[0] == PrintString))
                    r[key] = getData(seq);
```

```
            }
        }
    }
    return r;
}

unsigned char X509Cert::getSizeInt(unsigned __int32 n)
{
    unsigned __int32 mask = 0xFF000000;
    unsigned char c = 4;
    if (n == 0)
        return 1;
    while (1)
    {
        if ((mask & n) == 0)
        {
            c--;
            mask = mask >> 8;
        }
        else
            break;
    }
    return c;
}

void X509Cert::sizeToDer(QByteArray &vec, unsigned __int32 n)
{
    if (n < 127)
        vec.push_back(n);
    else
    {
        unsigned char countDop = getSizeInt(n);
        vec.push_back(countDop | (unsigned char)0x80);
        ByteToByte(vec, n, countDop);
    }
}

void X509Cert::ByteToByte(QByteArray &vec, unsigned __int32 n,
   unsigned char count)
{
    unsigned __int32 mask = 0xFF000000 >> (32 - count * 8);
    for (count; count > 0; count--)
    {
        vec.push_back((n & mask) >> ((count - 1) * 8));
        mask = mask >> 8;
    }
}

void X509Cert::intToDer(QByteArray &vec, unsigned __int32 n)
{
```

```
    unsigned char count = getSizeInt(n);
    sizeToDer(vec, count);
    ByteToByte(vec, n, count);
}

QByteArray X509Cert::KeyToByte(PublicKey key)
{
    QByteArray vec;
    vec.push_back(SEQUENCE);
    QByteArray tmp3;
    {
        tmp3.push_back(SEQUENCE);
        tmp3.push_back(10 + 3);
        tmp3.push_back(OBJ_ID);
        unsigned char a[] = { 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7,
   0x0D, 0x01, 0x01, 0x01 };
        for (int i = 0; i < 10; i++)
            tmp3.push_back(a[i]);
        tmp3.push_back(NULLder);
        tmp3.push_back((char)0);
        tmp3.push_back(BIT_STR);
        QByteArray tmp2;
        {
            tmp2.push_back(SEQUENCE);
            QByteArray tmp;
            {
                tmp.push_back(INTEGER);
                sizeToDer(tmp, key.H.size() + 1);
                tmp.push_back((char)0);
                tmp.append(key.H);
                tmp.push_back(INTEGER);
                intToDer(tmp, key.t);
            }
            sizeToDer(tmp2, tmp.size());
            tmp2.append(tmp);
        }
        sizeToDer(tmp3, tmp2.size() + 1);
        tmp3.push_back((char)0);
        tmp3.append(tmp2);
    }
    sizeToDer(vec, tmp3.size());
    vec.append(tmp3);
    return vec;
}

QByteArray X509Cert::KeyToByte(PrivateKey key)
{
    QByteArray vec;
    vec.push_back(SEQUENCE);
    QByteArray tmp3;
```

```
    {
        tmp3.push_back(SEQUENCE);
        tmp3.push_back(10 + 3);
        tmp3.push_back(OBJ_ID);
        unsigned char a[] = { 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7,
   0x0D, 0x01, 0x01, 0x01 };
        for (int i = 0; i < 10; i++)
            tmp3.push_back(a[i]);
        tmp3.push_back(NULLder);
        tmp3.push_back((char)0);
        tmp3.push_back(BIT_STR);
        QByteArray tmp2;
        {
            tmp2.push_back(SEQUENCE);
            QByteArray tmp;
            {
                tmp.push_back(INTEGER);
                sizeToDer(tmp, key.S.size());
                tmp.append(key.S);
                tmp.push_back(INTEGER);
                sizeToDer(tmp, key.H.size());
                tmp.append(key.H);
                tmp.push_back(INTEGER);
                sizeToDer(tmp, key.P.size());
                tmp.append(key.P);
            }
            sizeToDer(tmp2, tmp.size());
            tmp2.append(tmp);
        }
        sizeToDer(tmp3, tmp2.size() + 1);
        tmp3.push_back((char)0);
        tmp3.append(tmp2);
    }
    sizeToDer(vec, tmp3.size());
    vec.append(tmp3);
    return vec;
}

unsigned int X509Cert::sizeDecoder(QByteArray &arr)
{
    unsigned char count = arr[1];
    if(count > (unsigned char)0x80)
    {
        count ^= 0x80;
        unsigned int size = 0;
        for (int i = 0; i < count; i++)
        {
            size |= (unsigned char)arr[2 + i] << ((count - 1 -
   i)*8);
        }
```

```
        return size;
    }
    else
        return count;
}


PrivateKey X509Cert::keyDecoder(QByteArray &key)
{
    PrivateKey r;
    if ((unsigned char)key[0] == SEQUENCE)
    {
        key = getData(key);
        if ((unsigned char)key[0] == SEQUENCE)
        {
            getData(key);
            if((unsigned char)key[0] == BIT_STR)
            {
                key = getData(key);
                key.remove(0, 1);
                if((unsigned char)key[0] == SEQUENCE)
                {
                    key = getData(key);
                    r.S = getData(key);
                    r.H = getData(key);
                    r.P = getData(key);
                }
            }
        }
    }
    return r;
}

PublicKey X509Cert::pubKeyDecoder(QByteArray &key)
{
    PublicKey r;
    if ((unsigned char)key[0] == SEQUENCE)
    {
        key = getData(key);
        if ((unsigned char)key[0] == SEQUENCE)
        {
            getData(key);
            if((unsigned char)key[0] == BIT_STR)
            {
                key = getData(key);
                key.remove(0, 1);
                if((unsigned char)key[0] == SEQUENCE)
                {
                    key = getData(key);
                    r.H = getData(key);
```

```
                    r.H.remove(0, 1);
                    r.t = derToInt(key);
                }
            }
        }
    }
    return r;
}

QByteArray X509Cert::getSelfSignedCert(QByteArray &pKey,
   QByteArray &prKey)
{
    SerialNum = 1;
    NiederCrypt crypt;
    crypt.setPrivKey(keyDecoder(prKey));
    QByteArray cert;
    SubjectDN = IssuerDN;
    QByteArray TSB = genTSB(pKey, 10);
    QByteArray sign = crypt.signature(TSB);
    QByteArray tmp;
    addSignInfo(tmp);
    tmp.push_back(BIT_STR);
    sizeToDer(tmp, sign.size() + 1);
    tmp.push_back((char)0);
    cert.push_back(SEQUENCE);
    sizeToDer(cert, tmp.size() + sign.size() + TSB.size());
    cert.append(TSB);
    cert.append(tmp);
    cert.append(sign);
    return cert;
}

QByteArray X509Cert::getCert(PublicKey &pubKey, QByteArray &req,
   QByteArray &prKey)
{
    QByteArray cert;
    int period = req[0];
    req.remove(0, 1);
    if(req[0] == (char)SEQUENCE)
        SubjectDN = DerToX509Name(getData(req));
    else
        return cert;
    NiederCrypt crypt;
    crypt.setPrivKey(keyDecoder(prKey));
    QByteArray TSB = genTSB(KeyToByte(pubKey), period);
    QByteArray sign = crypt.signature(TSB);
    QByteArray tmp;
    addSignInfo(tmp);
    tmp.push_back(BIT_STR);
    sizeToDer(tmp, sign.size() + 1);
```

```
    tmp.push_back((char)0);
    cert.push_back(SEQUENCE);
    sizeToDer(cert, tmp.size() + sign.size() + TSB.size());
    cert.append(TSB);
    cert.append(tmp);
    cert.append(sign);
    return cert;
}

QByteArray X509Cert::genTSB(QByteArray &Pkey, int period)
{
    QByteArray TSB;
    TSB.push_back(SEQUENCE);
    QByteArray tmp;
    {
        tmp.push_back(0xA0);
        tmp.push_back(0x03);
        tmp.push_back(INTEGER);
        tmp.push_back(0x01);
        tmp.push_back(0x02);
        tmp.push_back(INTEGER);
        intToDer(tmp, SerialNum);
        addSignInfo(tmp);
        tmp.push_back(X509NameToDer(IssuerDN));
        NotAfter = QDateTime::currentDateTime();
        tmp.push_back(SEQUENCE);
        tmp.push_back(0x20);
        tmp.push_back(UTCTime);
        tmp.push_back(0x0D);
        tmp.push_back(NotAfter.toString("yyMMddhhmmss").toUtf8());
        tmp.push_back(0x5A);
        tmp.push_back(GenTime);
        tmp.push_back(0x0F);
        NotBefore = NotAfter.addYears(period);

   tmp.push_back(NotBefore.toString("yyyyMMddhhmmss").toUtf8());
        tmp.push_back(0x5A);
        tmp.push_back(X509NameToDer(SubjectDN));
        tmp.push_back(Pkey);
    }
    sizeToDer(TSB, tmp.size());
    TSB.append(tmp);
    return TSB;
}

int X509Cert::checkCert(QByteArray &cert, PublicKey &key)
{
    if((unsigned char)cert[0] == SEQUENCE)
    {
         cert = getData(cert);
```

```
        if((unsigned char)cert[0] == SEQUENCE)
        {
            unsigned int sizeTSB = sizeDecoder(cert) + 2;
            if (sizeTSB > 129)
                sizeTSB += (unsigned char)cert[1] ^ (unsigned
   char)0x80;
            QByteArray TSB = cert.left(sizeTSB);
            cert.remove(0, sizeTSB);
            if((unsigned char)cert[0] == SEQUENCE)
            {
                getData(cert);
                if((unsigned char)cert[0] == BIT_STR)
                {
                    QByteArray sign = getData(cert);
                    sign.remove(0, 1);
                    NiederCrypt crypt;
                    crypt.setPubKey(key);
                    if (crypt.checkSign(TSB, sign) == true)
                        return 0;
                    else
                        return 1;
                }
            }
        }
    }
    return 2;
}

int X509Cert::openCert(QByteArray &in)
{
    QByteArray cert = in;
    if((unsigned char)cert[0] == SEQUENCE)
    {
         cert = getData(cert);
        if((unsigned char)cert[0] == SEQUENCE)
        {
            QByteArray TSB = getData(cert);
            if ((unsigned char)TSB[0] == 0xA0)
            {
                getData(TSB);
                if ((unsigned char)TSB[0] == INTEGER)
                    SerialNum = derToInt(TSB);
                getData(TSB);
                if((unsigned char)TSB[0] == SEQUENCE)
                    IssuerDN = DerToX509Name(getData(TSB));

                if((unsigned char)TSB[0] == SEQUENCE)
                {
                    QByteArray period = getData(TSB);
                    QString tmp = getData(period);
```

```
                    tmp.resize(tmp.size() – 1);
                    NotAfter = QDateTime::fromString(tmp,
   "yyMMddhhmmss");
                    NotAfter = NotAfter.addYears(100);
                    tmp = getData(period);
                    tmp.resize(tmp.size() – 1);
                    NotBefore = QDateTime::fromString(tmp,
   "yyyyMMddhhmmss");
                }

                if((unsigned char)TSB[0] == SEQUENCE)
                    SubjectDN = DerToX509Name(getData(TSB));

                if((unsigned char)TSB[0] == SEQUENCE)
                    PubKey = pubKeyDecoder(TSB);
                return 0;
            }
        }
    }
    return 1;
}
```

Приложение Е

(обязательное)

Содержимое файла structkey.h

```
#ifndef STRUCTKEY_H
#define STRUCTKEY_H
#include <QtCore>
struct PublicKey {
    QByteArray H;
    unsigned __int16 t;
};
struct PrivateKey {
    QByteArray S, H, P;
};
#endif // STRUCTKEY_H
```

Приложение Ж

(обязательное)

Содержимое файла niedercrypt.h

```
#ifndef NIEDERCRYPT_H
#define NIEDERCRYPT_H
#include "structKey.h"
#include <thread>
#include <QtCore>
#include <ctime>
#include <algorithm>
#include <functional>
#include <QtConcurrentRun>
#define hashSize 256
#define Algorithm QCryptographicHash::Sha224
#define p_x 0x12b

class NiederCrypt
{
private:
    unsigned __int16 st = 0, Q, N, K, d_min, t;
    unsigned char *alpha_of, *index_of;
    QVector<QVector<unsigned char>> Hpub, S, H, P;
public:
    NiederCrypt();
    void genKey(unsigned char, unsigned char);
    QByteArray signature(QByteArray &);
    bool checkSign(QByteArray &, QByteArray &);
    PublicKey getPubKey();
    PrivateKey getPrivKey();
    void setPubKey(PublicKey &);
    void setPrivKey(PrivateKey &);
private:
    void gen_table();
    unsigned char gf_mul(__int16, __int16);
    unsigned char gf_div(__int16, __int16);
    unsigned char gf_pow(unsigned char, __int16);
    QVector<unsigned char> vec_mul(QVector<unsigned char>&,
   QVector<unsigned char>&);
    QVector<QVector<unsigned char>>
   matrix_Transp(QVector<QVector<unsigned char>>&);
    QVector<QVector<unsigned char>>
   matrix_mul(QVector<QVector<unsigned char>> &,
   QVector<QVector<unsigned char>> &);
    QVector<unsigned char> matrix_mul(QVector<unsigned char> &,
   QVector<QVector<unsigned char>> &);
    QVector<unsigned char> matrix_mul(QVector<QVector<unsigned
   char>> &, QVector<unsigned char> &);
    QVector<unsigned char> matrix_mul(QVector<QVector<unsigned
   char>> &a, QByteArray &b);
    QVector<QVector<unsigned char>>
   Get_matr(QVector<QVector<unsigned char>> &, int, int);
    unsigned char Det(QVector<QVector<unsigned char>> &);
```

```
    QVector<QVector<unsigned char>> Obr(QVector<QVector<unsigned
   char>> &);
    void gen_S();
    void gen_P();
    unsigned char f_x(QVector<unsigned char> &, unsigned char);
    QVector<unsigned char> decoder(QByteArray &);
    QByteArray hash(QByteArray &);
    struct rezDecod {
        unsigned i;
        QByteArray syndr_i;
        QVector<unsigned char> Z;
    };
    QVector<unsigned char> callDecoder (QByteArray &, unsigned);
};
#endif // NIEDERCRYPT_H
```

Приложение И

(обязательное)

Содержимое файла niedercrypt.cpp

```
#include <niedercrypt.h>
NiederCrypt::NiederCrypt()
{
    for(int tmp = p_x; tmp != 1; tmp = tmp >> 1)
        st++;
    Q = 1 << st;
    alpha_of = new unsigned __int8[Q];
    index_of = new unsigned __int8[Q];
    gen_table();
    srand( time(0) );
}
PublicKey NiederCrypt::getPubKey()
{
    PublicKey key;
    for(int i = 0; i < Hpub.size(); i++)
        for(int k = 0; k < Hpub[0].size(); k++)
            key.H.push_back(Hpub[i][k]);
    key.t = t;

    return key;
}
PrivateKey NiederCrypt::getPrivKey()
{
    PrivateKey key;
    for(int i = 0; i < S.size(); i++)
        for(int k = 0; k < S[0].size(); k++)
            key.S.push_back(S[i][k]);

    for(int i = 0; i < H.size(); i++)
        for(int k = 0; k < H[0].size(); k++)
            key.H.push_back(H[i][k]);

    for(int i = 0; i < P.size(); i++)
        for(int k = 0; k < P[0].size(); k++)
            key.P.push_back(P[i][k]);

    return key;
}
void NiederCrypt::setPubKey(PublicKey &PubKey)
{
    t = PubKey.t;
    N = PubKey.H.size() / (t*2);
    K = N – 2*t;
    d_min = N – K + 1;
    Hpub.clear();
    Hpub.resize(N – K);
    for(int i = 0; i < Hpub.size(); i++)
    {
        Hpub[i].resize(N);
        for(int j = 0; j < Hpub[0].size(); j++)
```

```
            Hpub[i][j] = PubKey.H[i*N + j];
    }
    return;
}
void NiederCrypt::setPrivKey(PrivateKey &PrivKey)
{
    N = sqrt(PrivKey.P.size());
    K = N – PrivKey.H.size() / N ;
    d_min = N – K + 1;
    t = (d_min – 1) / 2;
    S.clear();
    S.resize(N – K);
    for(int i = 0; i < N – K; i++)
    {
        S[i].resize(N – K);
        for(int j = 0; j < N – K; j++)
            S[i][j] = PrivKey.S[i*(N – K) + j];
    }
    H.clear();
    H.resize(N – K);
    for(int i = 0; i < N – K; i++)
    {
        H[i].resize(N);
        for(int j = 0; j < N; j++)
            H[i][j] = PrivKey.H[i*N + j];
    }
    P.clear();
    P.resize(N);
    for(int i = 0; i < N; i++)
    {
        P[i].resize(N);
        for(int j = 0; j < N; j++)
            P[i][j] = PrivKey.P[i*N + j];
    }
    Hpub = matrix_mul(matrix_mul(S, H), matrix_Transp(P));
    return;
}
void NiederCrypt::genKey(unsigned __int8 n, unsigned __int8 k)
{
    N = n;
    K = k;
    d_min = N – K + 1;
    t = (d_min – 1) / 2;
    H.resize(N – K);
    for (int i = 0; i < H.size(); i++)
    {
        H[i].resize(N);
        H[i][0] = 1;
        for (int j = 1; j < N; j++)
            H[i][j] = gf_pow(2, j*(i + 1));
```

```
    }
    gen_S();
    gen_P();
    Hpub = matrix_mul(matrix_mul(S, H), P);
    P = matrix_Transp(P);
}
void NiederCrypt::gen_table()
{
    for (unsigned __int16 i = 0; i < Q; i++)
    {
        if (i < st)
        {
            alpha_of[1 << i] = i;
            index_of[i] = 1 << i;
        }
        else
        {
            unsigned __int16 k = 0, index = 1;
            while (k != i)
            {
                while ((k != i) && (index < Q))
                {
                    index = index << 1;
                    k++;
                }
                if (index >= Q)
                    index ^= p_x;
            }
            alpha_of[index] = i;
            index_of[i] = index;
        }
    }
}
unsigned __int8 NiederCrypt::gf_mul(__int16 a, __int16 b)
{
    if ((a == 0) || (b == 0))
        return 0;
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    return index_of[(alpha_of[a] + alpha_of[b]) % (Q - 1)];
}
unsigned __int8 NiederCrypt::gf_div(__int16 a, __int16 b)
{
    if ((a == 0) || (b == 0))
        return 0;
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    __int16 j = alpha_of[a] - alpha_of[b];
    while (j < 0)
        j += Q - 1;
```

```
    return index_of[j];
}
unsigned __int8 NiederCrypt::gf_pow(unsigned __int8 a, __int16 b)
{
    b %= N;
    if (b == 0)
        return 1;
    if (b == 1)
        return a;
    unsigned __int8 r = a;
    for (__int16 i = 1; i < b; i++)
        r = gf_mul(r, a);
    return r;
}
QVector<QVector<unsigned __int8>>
   NiederCrypt::matrix_Transp(QVector<QVector<unsigned __int8>> &a)
{
    QVector<QVector<unsigned __int8>> r(a[0].size());
    for (int i = 0; i < r.size(); i++)
    {
        r[i].resize(a.size());
        for (int j = 0; j < r[0].size(); j++)
            r[i][j] = a[j][i];
    }
    return r;
}
QVector<unsigned __int8> NiederCrypt::vec_mul(QVector<unsigned
   __int8> &a, QVector<unsigned __int8> &b)
{
    QVector<unsigned __int8> r(a.size() + b.size() - 1);

    for (int i = a.size() - 1; i >= 0; i--)
        for (int k = b.size() - 1; k >= 0; k--)
            r[i + k] ^= gf_mul(a[i], b[k]);
    return r;
}
QVector<QVector<unsigned __int8>>
   NiederCrypt::matrix_mul(QVector<QVector<unsigned __int8>> &a,
   QVector<QVector<unsigned __int8>> &b)
{
    if (a[0].size() != b.size())
    {
        QVector<QVector<unsigned __int8>> err;
        return err;
    }
    QVector<QVector<unsigned __int8>> r(a.size());
    for (int row = 0; row < a.size(); row++)
    {
        r[row].resize(b[0].size());
        for (int col = 0; col < b[0].size(); col++)
```

```
            for (int inner = 0; inner < b.size(); inner++)
                r[row][col] ^= gf_mul(a[row][inner],
   b[inner][col]);
    }
    return r;
}
QVector<unsigned __int8> NiederCrypt::matrix_mul(QVector<unsigned
   __int8> &a, QVector<QVector<unsigned __int8>> &b)
{
    if (a.size() != b[0].size())
    {
        QVector<unsigned __int8> err;
        return err;
    }
    QVector<unsigned __int8> r(b[0].size());
    for (int col = 0; col < b[0].size(); col++)
        for (int inner = 0; inner < b.size(); inner++)
            r[col] ^= gf_mul(a[inner], b[inner][col]);
    return r;
}
QVector<unsigned __int8>
   NiederCrypt::matrix_mul(QVector<QVector<unsigned __int8>> &a,
   QVector<unsigned __int8> &b)
{
    if (a[0].size() != b.size())
    {
        QVector<unsigned __int8> err;
        return err;
    }
    QVector<unsigned __int8> r(a.size());
    for (int row = 0; row < a.size(); row++)
        for (int inner = 0; inner < b.size(); inner++)
            r[row] ^= gf_mul(a[row][inner], b[inner]);
    return r;
}
QVector<unsigned __int8>
   NiederCrypt::matrix_mul(QVector<QVector<unsigned __int8>> &a,
   QByteArray &b)
{
    if (a[0].size() != b.size())
    {
        QVector<unsigned __int8> err;
        return err;
    }
    QVector<unsigned __int8> r(a.size());
    for (int row = 0; row < a.size(); row++)
        for (int inner = 0; inner < b.size(); inner++)
            r[row] ^= gf_mul(a[row][inner], (unsigned
   char)b[inner]);
    return r;
```

```
}
QVector<QVector<unsigned __int8>>
   NiederCrypt::Get_matr(QVector<QVector<unsigned __int8>> &Matr,
   int indRow, int indCol)
{
    int n = Matr.size(), ki = 0;
    QVector<QVector<unsigned __int8>> temp_matr(n – 1);
    for (int i = 0; i < n; i++)
    {
        if (i != indRow)
        {
            temp_matr[ki].resize(n – 1);
            for (int j = 0, kj = 0; j < n; j++)
                if (j != indCol)
                {
                    temp_matr[ki][kj] = Matr[i][j];
                    kj++;
                }
            ki++;
        }
    }
    return temp_matr;
}
unsigned __int8 NiederCrypt::Det(QVector<QVector<unsigned __int8>>
   &matr)
{
    int l, n = matr.size();
    unsigned __int8 sum11 = 1, sum12 = 0, sum21 = 1, sum22 = 0;
    if (n == 1)
        return matr[0][0];
    else if (n == 2)
        return gf_mul(matr[0][0], matr[1][1]) ^ gf_mul(matr[1][0],
   matr[0][1]);
    else
    {
        for (int i = 0; i < n; i++)
        {
            sum11 = 1;
            l = 2 * n – 1 – i;
            sum21 = 1;
            for (int j = 0; j < n; j++)
            {
                sum21 = gf_mul(sum21, matr[j][l % n]);
                l––;
                sum11 = gf_mul(sum11, matr[j][(j + i) % n]);
            }
            sum22 ^= sum21;
            sum12 ^= sum11;
        }
        return (sum12 ^ sum22);
```

```
    }
}
QVector<QVector<unsigned __int8>>
   NiederCrypt::Obr(QVector<QVector<unsigned __int8>> &Matrix)
{
    unsigned __int8 det = Det(Matrix);
    int n = Matrix.size();
    QVector<QVector<unsigned __int8>> obr_matr(n);
    if (det != 0)
    {
        for (int i = 0; i < n; i++)
        {
            obr_matr[i].resize(n);
            for (int j = 0; j < n; j++)
            {
                obr_matr[i][j] = gf_div(Det(Get_matr(Matrix, i,
   j)), det);
            }
        }
    }
    return matrix_Transp(obr_matr);
}
void NiederCrypt::gen_S()
{
    unsigned __int8 det = 0;
    S.resize(N – K);
    while (det == 0)
    {
        for (int row = 0; row < N – K; row++)
        {
            S[row].resize(N – K);
            for (int col = 0; col < N – K; col++)
                S[row][col] = (unsigned char) (rand() % Q);
        }
        det = Det(S);
    }
}
void NiederCrypt::gen_P()
{
    P.resize(N);
    QVector<unsigned __int8> Rand(N);
    for (int i = 0; i < N; i++)
        Rand[i] = i;
    std::random_shuffle(Rand.begin(), Rand.end());

    for (int i = 0; i < N; i++)
    {
        P[i].resize(N);
        P[i][Rand[i]] = 1;
    }
```

```
}
unsigned __int8 NiederCrypt::f_x(QVector<unsigned __int8> &f,
   unsigned __int8 x)
{
    unsigned __int8 sum = 0;
    for (int i = 0; i < f.size(); i++)
        sum ^= gf_mul(f[i], gf_pow(x, i));
    return sum;
}
QVector<unsigned char> NiederCrypt::decoder(QByteArray &syndr)
{
    QVector<unsigned char> E, s1;
    for(int i = 0; i < syndr.size(); i++)
        s1.push_back(syndr[i]);
    for(int v = t; v > 0; v--)
    {
        E.clear();
        QVector<QVector<unsigned __int8>> M(v);
        for (int i = 0; i < v; i++)
        {
            M[i].resize(v);
            for (int j = 0; j < v; j++)
                M[i][j] = s1[v - 1 - j + i];
        }
        if (Det(M) == 0)
            continue;
        M = Obr(M);
        QVector<unsigned __int8> L, V(v);
        for (int i = v; i < 2 * v; i++)
            V[i - v] = s1[i];

        L = matrix_mul(M, V);
        M.clear();
        L.insert(L.begin(), 1);
        QVector<unsigned __int8> X, Y;
        for (unsigned __int8 a = 1, i = 0; i < N; a = gf_mul(2,
   a), i++)
        {
            unsigned __int8 b = gf_div(1, a);
            if (f_x(L, b) == 0)
            {
                X.push_back(b);
                Y.push_back(i);
                if (X.size() == v) break;
            }
        }
        if (X.size() != v)
            continue;
        QVector<unsigned __int8> W(v);
        for (int i = 0; i < v; i++)
```

```
            for (int j = 0; j <= i; j++)
                W[i] ^= gf_mul(L[j], (unsigned char)s1[i - j]);
        QVector<unsigned __int8> L_der(L.size() - 1);
        for (int i = 0; i < L_der.size(); i += 2)
            L_der[i] = L[i + 1];
        E.resize(N);
        for (int i = 0; i < v; i++)
            E[Y[i]] = gf_div(f_x(W, X[i]), f_x(L_der, X[i]));
        if (matrix_mul(Hpub, E) != s1)
            continue;
        return E;
    }
    E.clear();
    return E;
}
QByteArray NiederCrypt::hash(QByteArray &D)
{
    QCryptographicHash hsh(Algorithm);
    hsh.addData(D);
    QByteArray rez = hsh.result();
    rez.resize(N - K);
    for(int i = 0; i < rez.size(); i++)
        rez[i] = (char)((unsigned)rez[i] % Q) ;
    return rez;
}
QVector<unsigned char> NiederCrypt::callDecoder (QByteArray &
   syndr, unsigned i)
{
    QByteArray syndr_i;
    QDataStream syndrStream(&syndr_i, QIODevice::WriteOnly);
    syndrStream << i;
    syndr_i = syndr + syndr_i;
    syndr_i = hash(syndr_i);
    return decoder(syndr_i);
}
QByteArray NiederCrypt::signature(QByteArray &D)
{
    QByteArray syndr;
    syndr = hash(D);
    unsigned i = 0;
    while (1)
    {
        QVector<QFuture<QVector<unsigned char>>>
   futures(QThread::idealThreadCount());

        for(unsigned k = 0; k < futures.size(); k++)
            futures[k] = QtConcurrent::run(this,
   &NiederCrypt::callDecoder, syndr, i + k);
        for(unsigned k = 0; k < futures.size(); k++)
        {
```

```
            QVector<unsigned char> z = futures[k].result();
            if (z.size() == N)
            {
                syndr.clear();
                QDataStream zStream(&syndr, QIODevice::WriteOnly);
                for(int j = 0; j < z.size(); j++)
                    zStream << z[j];
                zStream << i + k;
                return syndr;
            }
        }
        i += QThread::idealThreadCount();
    }
}
bool NiederCrypt::checkSign(QByteArray &D, QByteArray &sign)
{
    QByteArray s2 = sign;
    QVector<unsigned char> z;
    for(int i = 0; i < (N); i++)
        z.push_back(s2[i]);
    s2.remove(0, N);
    s2 = hash(D) + s2;
    s2 = hash(s2);
    QByteArray s1;
    z = matrix_mul(Hpub, z);
    for(int i = 0; i < z.size(); i++)
        s1.push_back(z[i]);
    if (s1 != s2)
        return false;
    return true;
}
```

Приложение К

(обязательное)

Содержимое файла clientmain.cpp

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w(0);
    w.show();
    return a.exec();
}
```

Приложение Л

(обязательное)

Содержимое файла mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QtCore>
#include <QTcpSocket>
#include <QtWidgets>
#include <ui_mainwindow.h>
#include "x509cert.h"
namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
    QTcpSocket* m_pTcpSocket;
    quint16     m_nNextBlockSize;
    QLabel *status_Label;
    QString host;
    int port;
    bool readyReadData = false;
    void nameToArray(QByteArray &, QMap<QByteArray, QByteArray> &);
    int readFile(QString &, QByteArray &);
    int writeFile(QString &nameWindows, QByteArray &, QString
   &mask);
    QByteArray readData();
    int sendToServer(QByteArray & arrBlock);
    void connectToServ();
private slots:
    void slotError       (QAbstractSocket::SocketError);
    void slotReadyRead   ();
    void slotConnected();
    void slotDisconnected();
    void on_getCert_Button_clicked();
    void on_cert_Button_clicked();
    void on_checkCert_Button_clicked();
    void on_action_clicked();
};
#endif // MAINWINDOW_H
```

Приложение М

(обязательное)

Содержимое файла mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    m_nNextBlockSize(0)
{
    ui->setupUi(this);
    status_Label = new QLabel(this);
    statusBar()->addWidget(status_Label);
    status_Label->setText("Соединение␣с␣сервером␣отсутствует");
    ui->table->horizontalHeader()->setSectionResizeMode(1,
   QHeaderView::Stretch);
    m_pTcpSocket = new QTcpSocket(this);
    host = "127.0.0.1";
    port = 2323;
    connectToServ();
    connect(m_pTcpSocket, SIGNAL(connected()),
   SLOT(slotConnected()));
    connect(m_pTcpSocket, SIGNAL(disconnected()),
   SLOT(slotDisconnected()));
    connect(m_pTcpSocket, SIGNAL(readyRead()),
   SLOT(slotReadyRead()));
    connect(m_pTcpSocket,
   SIGNAL(error(QAbstractSocket::SocketError)),
            this,
   SLOT(slotError(QAbstractSocket::SocketError))
            );
    ui->menudf->addAction("Адрес␣сервера", this,
   SLOT(on_action_clicked()));
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::connectToServ()
{
    m_pTcpSocket->connectToHost(host, port);
}
void MainWindow::slotReadyRead()
{
    readyReadData = true;
}
QByteArray MainWindow::readData()
{
    QDataStream in(m_pTcpSocket);
    in.setVersion(QDataStream::Qt_4_2);
    QByteArray data;
    for (;;) {
        if (!m_nNextBlockSize) {
```

```
            if (m_pTcpSocket->bytesAvailable() < sizeof(quint16)) {
                break;
            }
            in >> m_nNextBlockSize;
        }
        if (m_pTcpSocket->bytesAvailable() < m_nNextBlockSize) {
            break;
        }
        m_nNextBlockSize = 0;
        in >> data;
    }
    readyReadData = false;
    return data;
}
void MainWindow::slotError(QAbstractSocket::SocketError err)
{
    QString strError =
        "Error: " + (err == QAbstractSocket::HostNotFoundError ?
                     "The host was not found." :
                     err == QAbstractSocket::RemoteHostClosedError
  ?
                     "The remote host is closed." :
                     err ==
  QAbstractSocket::ConnectionRefusedError ?
                     "The connection was refused." :
                     QString(m_pTcpSocket->errorString())
                    );
}
int MainWindow::sendToServer(QByteArray & data)
{
    QByteArray  arrBlock;
    QDataStream out(&arrBlock, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_2);
    out << quint16(0) << data;
    out.device()->seek(0);
    out << quint16(arrBlock.size() - sizeof(quint16));
    if(m_pTcpSocket->state() != QAbstractSocket::ConnectedState)
    {
        connectToServ();
        if(!m_pTcpSocket->waitForConnected(1000))
        {
            QMessageBox::critical(0, "Ошибка", "Нет соединения с
  сервером", "OK", QString(), 0, 1);
            return 1;
        }
    }
    m_pTcpSocket->write(arrBlock);
    QProgressDialog* dialogWait = new QProgressDialog("Ожидание
  ответа сервера...", "&Cancel", 0, 0);
    dialogWait->setMinimumDuration(0);
```

```
    dialogWait->setWindowTitle("Пожалуйста, подождите");
    dialogWait->setModal(true);
    while(1)
    {
        dialogWait->setValue(0) ;
        qApp->processEvents();
        if ((dialogWait->wasCanceled()) || (readyReadData == true))
             break;
    }
    dialogWait->setValue(0) ;
    delete dialogWait;
    return 0;
}
void MainWindow::slotConnected()
{
    status_Label->setText("Соединие с сервером установлено");
}
void MainWindow::slotDisconnected()
{
    status_Label->setText("Соединение с сервером отсутствует");
}
int MainWindow::readFile(QString &name, QByteArray &data)
{
    QFile file(name);
    if (!file.open(QIODevice::ReadOnly))
    {
        QMessageBox::warning(0, "Ошибка", "Ошибка чтения файла",
   "OK", QString(), 0, 1);
        return 1;
    }
    data += file.readAll();

    file.close();
    return 0;
}
int MainWindow::writeFile(QString &nameWindows, QByteArray &data,
   QString &mask)
{
    QString name = QFileDialog::getSaveFileName(this,
            tr(nameWindows.toUtf8()), "", tr(mask.toUtf8()));
    if (name == "")
        return 1;
    QFile file(name);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Truncate))
    {
        QMessageBox::warning(0, "Ошибка", "Ошибка записи файла",
   "OK", QString(), 0, 1);
        return 1;
    }
    QDataStream outf(&file);
```

```
    for(int i = 0; i < data.size(); i++)
        outf << (unsigned char)data[i];
    if(outf.status() != QDataStream::Ok)
    {
        QMessageBox::warning(0, "Ошибка", "Ошибка␣записи␣файла",
   "OK", QString(), 0, 1);
        return 1;
    }
    file.close();
    return 0;
}
void MainWindow::nameToArray(QByteArray &arr, QMap<QByteArray,
   QByteArray> &m)
{
    X509V3Cert x509;
    arr += x509.X509NameToDer(m);
}
void MainWindow::on_getCert_Button_clicked()
{
    if ((ui->C_lineEdit->text() == "") || (ui->ST_lineEdit->text()
   == "") || (ui->L_lineEdit->text() == "")
            || (ui->O_lineEdit->text() == "") ||
   (ui->OU_lineEdit->text() == "") || (ui->CN_lineEdit->text() ==
   ""))
    {
        QMessageBox::warning(0, "Ошибка", "Заполните␣все␣поля",
   "OK", QString(), 0, 1);
        return;
    }
    QByteArray data;
    data.push_back(1);
    data.push_back(ui->period_spinBox->value());
    QMap<QByteArray, QByteArray> name;
    name["C"] = ui->C_lineEdit->text().toUtf8();
    name["ST"] = ui->ST_lineEdit->text().toUtf8();
    name["L"] = ui->L_lineEdit->text().toUtf8();
    name["O"] = ui->O_lineEdit->text().toUtf8();
    name["OU"] = ui->OU_lineEdit->text().toUtf8();
    name["CN"] = ui->CN_lineEdit->text().toUtf8();
    nameToArray(data, name);
    if (sendToServer(data) != 0)
        return;
    data = readData();
    if (data == "err")
    {
        QMessageBox::critical(0, "Ошибка", "Ошибка␣при␣передаче␣
   данных", "OK", QString(), 0, 1);
        return;
    }
    if (data == "success")
```

```
    {
        QMessageBox::information(0, "␣", "Запрос␣на␣сертификат␣
   отправлен", "OK", QString(), 0, 1);
        return;
    }
}

void MainWindow::on_cert_Button_clicked()
{
    ui—>cert_lineEdit—>setText(QFileDialog::getOpenFileName(this,
        tr("Выберите␣файл␣цифрового␣сертификата"), "",
   tr("Certificate␣Files␣(*.crt)")));

    if (ui—>cert_lineEdit—>text() == "") return;
    QByteArray data;
    if(readFile(ui—>cert_lineEdit—>text(), data) != 0)
        return;
    X509V3Cert certClient;
    if (certClient.openCert(data) != 0)
    {
        QMessageBox::warning(0, "Ошибка", "Выберите␣файл␣
   сертификата␣для␣проверки", "OK", QString(), 0, 1);
        return;
    }
    ui—>table—>setRowCount(9);
    for(int row = 0; row < ui—>table—>rowCount(); row++)
      for(int column = 0; column < ui—>table—>columnCount();
   column++)
        ui—>table—>setItem(row, column, new QTableWidgetItem());
    ui—>table—>item(0, 0)—>setText("Версия");
    ui—>table—>item(0, 1)—>setText("V2");
    ui—>table—>item(1, 0)—>setText("Серийный␣номер");
    ui—>table—>item(1,
   1)—>setText(QString::number(certClient.SerialNum, 16));
    ui—>table—>item(2, 0)—>setText("Алгоритм␣подписи");
    ui—>table—>item(2, 1)—>setText(certClient.SignatureAlgorithm);
    ui—>table—>item(3, 0)—>setText("Хэшалгоритм—␣подписи");
    ui—>table—>item(3, 1)—>setText(certClient.HashAlgorithm);
    ui—>table—>item(4, 0)—>setText("Издатель");
    QString tmp;
    QMapIterator<QByteArray, QByteArray> iter(certClient.IssuerDN);
    while (1)
    {
        iter.next();
        tmp += iter.key() + "=";
        tmp += iter.value();
        if (iter.hasNext())
            tmp += ",␣";
        else break;
    }
```

```
    ui->table->item(4, 1)->setText(tmp);
    ui->table->item(5, 0)->setText("Действителен с");
    ui->table->item(5,
   1)->setText(certClient.NotBefore.toString("dd.MM.yyyy
   hh:mm:ss"));
    ui->table->item(6, 0)->setText("Действителен по");
    ui->table->item(6,
   1)->setText(certClient.NotAfter.toString("dd.MM.yyyy
   hh:mm:ss"));
    ui->table->item(7, 0)->setText("Субъект");
    tmp = "";
    QMapIterator<QByteArray, QByteArray>
   iter2(certClient.SubjectDN);
    while (1)
    {
        iter2.next();
        tmp += iter2.key() + "=";
        tmp += iter2.value();
        if (iter2.hasNext())
            tmp += ", ";
        else break;
    }
    ui->table->item(7, 1)->setText(tmp);
    ui->table->item(8, 0)->setText("Открытый ключ");
    tmp = "";
    for(int i = 0; i < certClient.PubKey.H.length(); i++)
        tmp += QString::number(certClient.PubKey.H[i], 16);
    tmp += '\n';
    tmp += QString::number(certClient.PubKey.t, 16);
    ui->table->item(8, 1)->setText(tmp);
    ui->checkCert_Button->setEnabled(true);
}

void MainWindow::on_checkCert_Button_clicked()
{
    if (ui->cert_lineEdit->text() == "")
    {
        QMessageBox::warning(0, "Ошибка", "Выберите файл
   сертификата для проверки", "OK", QString(), 0, 1);
        return;
    }
    QByteArray data;
    data.push_back(2);
    if(readFile(ui->cert_lineEdit->text(), data) != 0)
        return;

    if (sendToServer(data) != 0)
        return;
    data = readData();
    if (data.size() == 0)
```

```
        return;
    if (data == "err")
    {
        QMessageBox::warning(0, "Ошибка", "Целостность␣сертификата␣
   не␣может␣быть␣проверена", "OK", QString(), 0, 1);
        return;
    }
    if (data == "valid")
    {
        QMessageBox::information(0, "Сертификат␣действителен",
   "Сертификат␣действителен", "OK", QString(), 0, 1);
        return;
    }
    if (data == "expired")
    {
        QMessageBox::warning(0, "Сертификат␣недействителен", "Срок␣
   дейтсвия␣сертификата␣истек␣или␣еще␣не␣наступил", "OK",
   QString(), 0, 1);
        return;
    }
    if (data == "notValid")
    {
        QMessageBox::warning(0, "Сертификат␣недействителен",
   "Целостность␣сертификата␣нарушена", "OK", QString(), 0, 1);
        return;
    }
    if (data == "revoked")
    {
        QMessageBox::warning(0, "Сертификат␣недействителен",
   "Сертификат␣аннулирован␣удостоверяющим␣центром", "OK",
   QString(), 0, 1);
        return;
    }
}

void MainWindow::on_action_clicked()
{
    bool bOk;
    QString str = QInputDialog::getText( 0,"Настройки","IPадрес—␣и␣
   номер␣порта␣
   сервера:",QLineEdit::Normal,host.append(":%1").arg(port),&bOk);
    if (bOk) {host = str;}
}
```