Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.

Специальность: 6-05-0611-05 «Вычислительные машины, системы и сети».

Специализация: 400201-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Б.В. Никульшин

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

ЗАДАНИЕ

по курсовому проекту студента

Курыловича Глеба Олеговича

**1** Тема проекта: «Разработка и реализация протокола аутентификации клиента на сервере с использованием функционала ssl»

**2** Срок сдачи студентом законченного проекта: 10 мая 2025 г.

**3** Исходные данные к проекту: нет.

**4** Содержание пояснительной записки (перечень подлежащих разработке

вопросов):

Титульный лист.

Реферат.

Введение.

1. Обзор литературы.

2. Функциональное проектирование

3. Системное проектирование

4. Результаты работы

Заключение.

Список использованных источников.

Приложения.

**5** Перечень графического материала (с точным указанием обязательных

чертежей):

**5.1** Схема алгоритма.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  курсового проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Выбор темы курсового проекта | 5 | 17.02 – 01.03 |  |
| Начальный этап ПЗ | 30 | 01.03 – 01.04 |  |
| Основная часть кода | 50 | 01.04 – 01.05 |  |
| Оформление пояснительной записки и графического материала | 15 | 01.05 – 10.05 | с выполнением  чертежа |
| Защита курсового проекта |  | 28.05 – 10.06 |  |

Дата выдачи задания: хх.хх.2025 г.

Руководитель Д.А. Жалейко

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_\_

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе на тему

Разработка и реализация протокола аутентификации клиента на сервере с использованием функционала ssl

БГУИР КР 1-40 02 01 211 ПЗ

Студент И.О.Фамилия

Руководитель Д.А. Жалейко

МИНСК 2025

**СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ…………………………………………………………………... 4](#_heading=h.gjdgxs)

[ВВЕДЕНИЕ……………………………………………………………………….](#_heading=h.1fob9te) 5

[1 ОБЗОР ЛИТЕРАТУРЫ………………………………………………………… 6](#_heading=h.3znysh7)

[1.1 Обзор методов и алгоритмов решения поставленной задачи……………6](#_heading=h.2et92p0)

[1.2 Обзор информационных технологий решения задачи……………………8](#_heading=h.tyjcwt)

[1.3 Постановка задачи…………………………………………………………..9](#_heading=h.3dy6vkm)

[2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ…………………………….....10](#_heading=h.1t3h5sf)

[2.1 Описание функций CRUD-операций……………………………………..11](#_heading=h.4d34og8)

[2.2 Описание функции добавления в корзину……………………………….1](#_heading=h.2s8eyo1)2

[2.3 Описание функции добавления в отчёт…………………………………..1](#_heading=h.17dp8vu)2

[3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ……………………………... 13](#_heading=h.3rdcrjn)

[3.1 Описание программных модулей…………………………………………13](#_heading=h.26in1rg)

[3.2 Примеры кода………………………………………………………………24](#_heading=h.lnxbz9)

[3.3 Разработка алгоритмов….…………………………………………………42](#_heading=h.lnxbz9)

[4 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ…………………………………… 44](#_heading=h.44sinio)

[ЗАКЛЮЧЕНИЕ…………………………………………………………………. 59](#_heading=h.2jxsxqh)

[ЛИТЕРАТУРА…………………………………………………………………... 60](#_heading=h.z337ya)

[ПРИЛОЖЕНИЕ А………………………………………………………………. 61](#_heading=h.3j2qqm3)

[ПРИЛОЖЕНИЕ Б………………………………………………………………..62](#_heading=h.1y810tw)

[ПРИЛОЖЕНИЕ В……………………………………………………………….. 63](#_heading=h.4i7ojhp)

[ПРИЛОЖЕНИЕ Г……………………………………………………………….. 64](#_heading=h.2xcytpi)

**ВВЕДЕНИЕ**

В современном мире существует огромное количество рисков, которые возникают при передаче конфиденциальной информации во время работы с веб-сайтами. Именно поэтому для повышения надёжности и снижения вероятностей утери личных данных был создан SSL-протокол. Его основное применение – шифрование и защита данных во время их передачи по интернету. Протокол использует ассиметричные алгоритмы шифрования, тем самым делая передачу данных от сервера к клиенту более безопасной.

Анализ существующих решений показал, что большинство приложений, которые работают с данными пользователя и используют клиент-серверную архитектуру используют SSL-протокол. Примерами таких приложений могут являться веб-браузеры, например, Google Chrome, Mozilla Firefox, или электронные почты.

Разрабатываемый проект является примером реализации SSL-протокола, который после можно использовать для безопасной аутентификации пользователя на сервере при написании клиент-серверного приложения.

**1 ОБЗОР ЛИТЕРАТУРЫ**

**1.1 Разработка и реализация протокола аутентификации клиента на сервере с использованием функционала ssl как проект**

Реализация проекта включает в себя несколько основных аспектов, которые необходимо учесть при создании: написание клиент-серверной части, установка защищённого соединения между ними, аутентификация клиента, используя протоколы безопасности. Основные функции:

- Создание SSL контекста

- Загрузка и проверка сертификатов

- Установка защищённого соединения

Реализация данного проекта для проверки аутентификации – создание безопасного механизма аутентификации клиента на сервере с использованием SSL/TLS, реализация двусторонней аутентификации, проверка подлинности клиента, используя цифровые сертификаты. Проект требует понимания принципов криптографии и работы протоколов для защиты данных. Одним из основополагающих аспектов при разработке является выбор подходящей библиотеки для использования всех возможностей SSL протокола.

Основная цель проекта - продемонстрировать возможности защиты данных с использованием SSL/TLS, показав на практике, как реализуется безопасная аутентификация в клиент-серверных приложениях. Результатом работы становится готовое решение, которое можно интегрировать в реальные системы для обеспечения конфиденциальности и целостности передаваемых данных.

**1.2 Анализ существующих аналогов**

Для разработки приложения, которое будет способно аутентифицировать клиента на сервере, необходимо учесть особенности и технологии, которые реализованы в аналогичных приложениях. Такой подход помогает не только оценить актуальность используемых методов, но и понять, какие аспекты можно заимствовать для повышения эффективности разрабатываемого протокола. Анализ существующих аналогов позволяет выявить их сильные и слабые стороны, определить, насколько они соответствуют поставленным требованиям, и найти области для потенциального улучшения. Это дает возможность заранее предусмотреть возможные трудности при реализации проекта и минимизировать риски, связанные с безопасностью и производительностью.

**1.2.1 Apache HTTP Sever**

Apache HTTP Server (или просто Apache) является одним из наиболее известных веб-серверов с открытым исходным кодом, который играет ключевую роль в современной инфраструктуре интернета. Он используется для обслуживания веб-приложений, предоставления доступа к данным и создания защищенных соединений между клиентами и серверами. Благодаря своей гибкости и масштабируемости Apache широко применяется в различных областях: от небольших сайтов до крупных корпоративных систем.

Работа Apache построена на процессной модели. В рамках процессной модели каждое соединение (обработка запроса) с сервером помещается в отдельный поток и проходит определенные этапы.

Для использования протокола SSL, необходимо использовать модуль mod\_ssl. Этот модуль позволяет серверу шифровать данные, передаваемые между клиентом и сервером, что защищает их от перехвата злоумышленниками. После подключения модуля в конфигурации сервера необходимо указать путь к SSL-сертификатам и закрытым ключам. Сертификаты используются для подтверждения подлинности сервера, а закрытый ключ обеспечивает возможность дешифрования данных.

**1.2.2. Nginx**

Nginx - это высокопроизводительный веб-сервер и обратный прокси-сервер, который также может выполнять функции балансировщика нагрузки, почтового прокси и HTTP-кэш-сервера. Разработанный с акцентом на скорость и масштабируемость, Nginx стал одним из самых популярных серверов в мире благодаря своей способности справляться с высокими нагрузками и эффективной обработке большого количества запросов.

Nginx реализует протокол SSL/TLS, предоставляя возможность создания защищённых соединений между клиентами и сервером. Это обеспечивает шифрование данных, а также аутентификацию, что делает взаимодействие безопасным. Шифрование данных основано на протоколе TLS, который защищает информацию от перехвата.

**1.3 Постановка задачи**

Целью данной курсовой работы является разработка простейшего приложения, которое реализует протокол аутентификации клиента на сервере с использованием функционала SSL. Основная цель реализовать программу, которая сможет обеспечить безопасное и надёжное взаимодействие между участниками системы. Аутентификация клиента играет ключевую роль в современном цифровом мире, где защита конфиденциальности данных и предотвращение несанкционированного доступа становятся приоритетными задачами. Использование SSL позволяет реализовать механизм шифрования, который гарантирует защиту информации, передаваемой по сети, и минимизирует риски её компрометации.

Проект предполагает исследование существующих решений и технологий, применяемых в области аутентификации на основе SSL, с целью их анализа и адаптации под условия работы в Си. Специфика работы будет включать изучение и применение библиотек, таких как OpenSSL, для создания защищённого канала связи.

В ходе проекта планируется детально изучить функционал SSL и его возможности для реализации аутентификации клиента без использования традиционных списков. Особое внимание будет уделено проверке эффективности и надёжности предлагаемого протокола, а также его соответствию современным требованиям к защите данных. Ожидаемым результатом является создание рабочего решения, способного обеспечить безопасное взаимодействие клиента и сервера и стать основой для внедрения в реальных проектах. Такой подход позволит укрепить защиту информационных систем и повысить доверие пользователей к цифровым сервисам.

**2. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Системное проектирование направлено на реализацию приложения, которое сумеет реализовать SSL протокол. Программа содержит простейший клиент и сервер, на котором должна будет происходить аутентификация, а также сама программа аутентификации. Основными модулями будут являются клиентская и серверная часть, модули установки SSL-соединения и модуль для обмена данными, чтобы проверить работоспособность аутентификации.

**2.1 Модуль установки SSL-соединения**

Этот модуль отвечает за настройку и управление безопасным SSL-соединением между клиентом и сервером. Основными задачами являются: Инициализация SSL-библиотеки: необходимо обеспечить правильную инициализацию библиотеки SSL, такой как OpenSSL, чтобы воспользоваться её функционалом. Создание и настройка SSL-контекста: включает загрузку сертификатов, закрытых ключей и настройку параметров шифрования (например, выбор алгоритмов шифрования). Установление защищенного соединения: модуль создает SSL-соединение между клиентом и сервером, обеспечивая шифрование данных.

**2.2 Клиент**

Модуль клиента отвечает за взаимодействие с сервером. Основные задачи включают: Создание SSL-соединения: клиент должен установить защищенное соединение с сервером. Генерация запросов: клиент формирует запросы, отправляет их серверу и следит за корректным оформлением данных. Получение и обработка ответов: модуль анализирует ответы от сервера и выполняет дальнейшие действия на основе полученной информации.

**2.3 Сервер**

Этот модуль обеспечивает работу серверной части программы. Основные задачи включают: Настройка серверного сокета: настраивается TCP/SSL-сокет для приема подключений. Ожидание подключений: сервер ожидает соединений от клиентов и обрабатывает запросы.

**2.4 Модуль для обмена данными**

Модуль обмена данными обеспечивает передачу и прием сообщений в зашифрованном виде через SSL-канал. Основные задачи включают: Шифрованная передача данных: все данные перед отправкой шифруются для сохранения их конфиденциальности. Реализация механизмов передачи: отправка сообщений на сервер и получение ответов от сервера. Обеспечение целостности данных: используется проверка целостности, чтобы предотвратить изменения данных в процессе их передачи.

**2.5 Модуль аутентификации клиента**

Этот модуль обеспечивает проверку подлинности клиента для защиты от нежелательных подключений. Основными задачами являются: Чтение сертификатов клиента: модуль загружает сертификаты, предоставленные клиентом, и анализирует их содержание. Валидация сертификатов: проверяется их подлинность с использованием центра сертификации (CA), чтобы убедиться, что сертификат выдан доверенным источником. Обработка ошибок аутентификации: в случае некорректных сертификатов или ошибок в процессе проверки система должна выдать пользователю соответствующее уведомление.

**3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

**3.1 Описание программных модулей**

Программа содержит два файла: client.c и server.c

**3.1.1 client.c**

Данный программынй модуль реализует поведение клиента, который должен войти на сервер.

**3.1.2. server.c**

Данный программный модуль реализует логику сервера, который должен пропустить клиента при наличии у него подходящего SSL сертификата

**3.2 Примеры кода**

**3.2.1 Функция createContext**

1 Выбор метода SSL/TLS

SSL\_CTX \*create\_context() {

const SSL\_METHOD \*method;

SSL\_CTX \*ctx;

method = TLS\_server\_method();

2 Создание SSL-контекста

ctx = SSL\_CTX\_new(method);

if (!ctx) {

perror("Unable to create SSL context");

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

return ctx;

}

Функция создаёт SSL/TLS-контекст для установления защищённых соединений с использованием OpenSSL. Она задаёт метод соединения, инициализирует SSL-контекст с этим методом и проверяет успешность создания. Если контекст не создаётся, выводится сообщение об ошибке, и программа завершает работу. Если всё проходит успешно, возвращается созданный контекст, который затем используется для настройки соединений.

**3.2.2 Функция configureContext**

1 Загрузка сертификата и ключа сервера

void configure\_context(SSL\_CTX \*ctx) {

if (SSL\_CTX\_use\_certificate\_file(ctx, SERVER\_CERT, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

if (SSL\_CTX\_use\_PrivateKey\_file(ctx, SERVER\_KEY, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

2 Проверка соответствия ключа и сертификата

if (!SSL\_CTX\_check\_private\_key(ctx)) {

fprintf(stderr, "Private key does not match the certificate public key\n");

exit(EXIT\_FAILURE);

}

3 Настройка CA-сертификата

if (SSL\_CTX\_load\_verify\_locations(ctx, CA\_CERT, NULL) != 1) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

4 Настройка проверки клиентских сертификатов

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER | SSL\_VERIFY\_FAIL\_IF\_NO\_PEER\_CERT, NULL);

SSL\_CTX\_set\_verify\_depth(ctx, 4);

}

Функция configure\_context() настраивает SSL/TLS-контекст. Она загружает сертификат и приватный ключ сервера, проверяет их соответствие, подключает CA-сертификат для верификации клиента, задаёт правила проверки клиентских сертификатов и глубину проверки. В случае ошибок выводятся сообщения, а программа завершает работу. Всё это обеспечивает безопасность соединения.

**3.2.3 Функция verifyCallback**

1. Функция проверяет результат предварительной верификации (preverify\_ok). Если проверка провалена, выводит ошибку и возвращает 0.

int verify\_callback(int preverify\_ok, X509\_STORE\_CTX \*ctx) {

if (!preverify\_ok) {

fprintf(stderr, "Certificate verification failed\n");

return 0;

}

2 Получение текущего проверяемого сертификата и определение глубины проверки

X509 \*cert = X509\_STORE\_CTX\_get\_current\_cert(ctx);

int depth = X509\_STORE\_CTX\_get\_error\_depth(ctx);

printf("Certificate verification depth: %d\n", depth);

3 Получение субъекта сертификата

if (depth == 0) { // Это сертификат конечного объекта (сервера)

char \*subject = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

printf("Server certificate subject: %s\n", subject);

OPENSSL\_free(subject);

}

return 1;

}

Функция verify\_callback используется для проверки сертификатов в процессе установления SSL/TLS-соединения. Она проверяет результат предварительной верификации, выводит сообщение об ошибке, если верификация провалилась, и проводит дополнительную проверку сертификата, включая его уровень и данные субъекта. На глубине 0 (сертификат сервера) отображается информация о субъекте сертификата. Функция возвращает 1.

**3.3 Разработка алгоритмов**

**Схема алгоритма** create\_context представлена в приложении А, схема алгоритма configure\_context представлена в приложении Б.

**3.3.1 Схема алгоритма create\_context**

Шаг 1. Начало.

Шаг 2. Объявить переменные method – указатель на структуру SSL\_METHOD, ctx – указатель на структуру SSL\_CTX.

Шаг 3. Вызвать функцию TLS\_server\_method() для определения метода и режима работы и присвоить перменной method.

Шаг 4. Вызвать функцию SSL\_CTX\_new(method) для создания контекста и присвоить переменной ctx.

Шаг 5. Если перемнная ctx не пустая, перейти к шагу 8, иначе перейти к шагу 6.

Шаг 6. Вывести сообщение об ошибке.

Шаг 7. Завершить приложение с ошибкой.

Шаг 8. Вернуть переменную ctx из функции.

Шаг 9. Конец.

**3.3.2 Схема алгоритма configure\_context**

Входные: ctx – указатель на перемнную типа SSL\_CTX с контекстом для клиента

Шаг 1. Начало

Шаг 2. Загрузить CA-сертификаты в контекст. Если прошло успешно, перейти к шагу 5, иначе перейти к Шагу 3.

Шаг 3. Вывести сообщение об ошибке.

Шаг 4. Завершить программу с ошибкой

Шаг 5. Проверить загрузку файла сертификата и связать его с SSL контекстом. Если загружен успешно, перейти к шагу 8, иначе перейти к шагу 6.

Шаг 6. Вывести сообщение об ошибке.

Шаг 7. Завершить программу с ошибкой.

Шаг 8. Загрузить приватный ключ сертификата и связать его с SSL контекстом. Если загрузка прошла успешно, перейти к шагу 11, иначе перейти к шагу 9.

Шаг 9. Вывести сообщение об ошибке.

Шаг 10. Завершить сообщение с ошибкой.

Шаг 11. Проверить соответствие загруженного ключа и сетификата. Если есть совпадение, перейти к шагу 14, иначе перейти к шагу 12.

Шаг 12. Вывести сообщение об ошибке.

Шаг 13. Завершить сообщение с ошибкой.

Шаг 13. Конец

**4 РЕЗУЛЬТАТЫ РАБОТЫ**

Готовность сервера к работе и консольный интерфейс программы представлен на рисунке 4.1

C:\Users\Gleb\Downloads\Telegram Desktop\area_20250526_111005.png

Рисунок 4.1 – Готовность сервера к работе

Подключение со стороны клиента при подключении к серверу представлено на рисунке 4.2

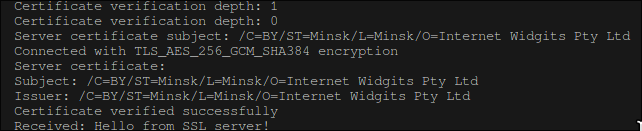


Рисунок 4.2 – Подключение клиента к серверу

Подключение клиента к серверу, вид со стороны сервера представлен на рисунке 4.3

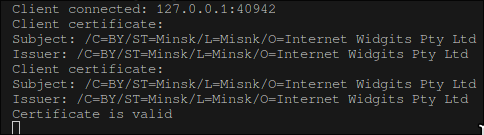


Рисунок 4.3 – подключение клиента к серверу, вид со стороны сервера

**ЗАКЛЮЧЕНИЕ**

В рамках работы проведена всесторонняя интеграция функционала, позволяющего обеспечить безопасность передачи данных через защищенные каналы. Одним из ключевых этапов стало создание SSL-контекста, настройка сертификатов и ключей, а также организация проверки их соответствия. Внедрение механизма верификации сторон позволило минимизировать риски несанкционированного доступа и атак на систему. Реализация логики проверки клиентских сертификатов, включая их структуру и иерархию, обеспечила строгий контроль над процессом взаимодействия клиента с сервером.

Полученный протокол доказал свою надежность, поддерживая высокую скорость обработки и стабильность работы системы в условиях реальных нагрузок. Проведенная работа показала, что внедрение SSL существенно повышает уровень безопасности, что особенно актуально в современных условиях цифровизации. Разработанный механизм соответствует актуальным стандартам шифрования, что делает его универсальным и пригодным для дальнейшего использования в более сложных системах. Данный проект демонстрирует успешное применение теоретических знаний на практике, а также подтверждает важность защищенных коммуникаций в развитии технологий.

Список использованных источников

[1] Луцик Ю. А. Объектно-ориентированное программирование на языке С++:учеб. пособие /Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008.–266 с.

[2] Ристих И. Bulletproof SSL and TLS

[3]. Основы алгоритмизации и программирования : лаборатор. практикум для студентов специальности 1-40 02 01 «Вычисл. машины, системы и сети» всех форм обучения. В 2 ч. Ч. 2 / сост. Ю. А. Луцик [и др.]. – Минск : БГУИР, 2010. – 36 с. : ил.

[4]. Демидович, Е. М. Основы алгоритмизации и программирования. Язык Си / Е. М. Демидович. – СПб. : БХВ-Петербург, 2008. – 440 с.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Схема алгоритма функции create\_context

**ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема алгоритма функции configure\_context

**ПРИЛОЖЕНИЕ В**

*(обязательное)*

Ведомость документов

**Приложение Г**

*(обязательное)*

Код программы

**Файл Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#include <openssl/x509\_vfy.h>

#define PORT 4433

#define SERVER "127.0.0.1"

#define CA\_CERT "ca.crt" // Сертификат удостоверяющего центра для проверки сервера

#define CLIENT\_CERT "client.crt" // Клиентский сертификат

#define CLIENT\_KEY "client.key" // Клиентский приватный ключ

void init\_openssl() {

SSL\_load\_error\_strings();

OpenSSL\_add\_ssl\_algorithms();

}

void cleanup\_openssl() {

EVP\_cleanup();

}

SSL\_CTX \*create\_context() {

const SSL\_METHOD \*method;

SSL\_CTX \*ctx;

method = TLS\_client\_method();

ctx = SSL\_CTX\_new(method);

if (!ctx) {

perror("Unable to create SSL context");

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

return ctx;

}

void configure\_context(SSL\_CTX \*ctx) {

/\* Загружаем CA сертификат для проверки сервера \*/

if (SSL\_CTX\_load\_verify\_locations(ctx, CA\_CERT, NULL) != 1) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

/\* Устанавливаем уровень проверки \*/

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, NULL);

SSL\_CTX\_set\_verify\_depth(ctx, 4);

/\* Если используется клиентский сертификат \*/

if (SSL\_CTX\_use\_certificate\_file(ctx, CLIENT\_CERT, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

if (SSL\_CTX\_use\_PrivateKey\_file(ctx, CLIENT\_KEY, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

if (!SSL\_CTX\_check\_private\_key(ctx)) {

fprintf(stderr, "Private key does not match the certificate public key\n");

exit(EXIT\_FAILURE);

}

}

int verify\_callback(int preverify\_ok, X509\_STORE\_CTX \*ctx) {

if (!preverify\_ok) {

fprintf(stderr, "Certificate verification failed\n");

return 0;

}

/\* Дополнительная проверка сертификата \*/

X509 \*cert = X509\_STORE\_CTX\_get\_current\_cert(ctx);

int depth = X509\_STORE\_CTX\_get\_error\_depth(ctx);

printf("Certificate verification depth: %d\n", depth);

/\* Можно добавить проверку Subject, Issuer, срок действия и т.д. \*/

if (depth == 0) { // Это сертификат конечного объекта (сервера)

char \*subject = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

printf("Server certificate subject: %s\n", subject);

OPENSSL\_free(subject);

}

return 1;

}

int main(int argc, char \*\*argv) {

SSL\_CTX \*ctx;

int sock;

struct sockaddr\_in addr;

SSL \*ssl;

char buf[1024];

int bytes;

X509 \*cert;

init\_openssl();

ctx = create\_context();

configure\_context(ctx);

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, verify\_callback);

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Unable to create socket");

exit(EXIT\_FAILURE);

}

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

inet\_pton(AF\_INET, SERVER, &addr.sin\_addr);

if (connect(sock, (struct sockaddr\*)&addr, sizeof(addr)) < 0) {

perror("Unable to connect");

exit(EXIT\_FAILURE);

}

ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, sock);

if (SSL\_connect(ssl) <= 0) {

ERR\_print\_errors\_fp(stderr);

SSL\_shutdown(ssl);

SSL\_free(ssl);

close(sock);

SSL\_CTX\_free(ctx);

cleanup\_openssl();

exit(EXIT\_FAILURE);

}

printf("Connected with %s encryption\n", SSL\_get\_cipher(ssl));

/\* Проверяем сертификат сервера \*/

cert = SSL\_get\_peer\_certificate(ssl);

if (cert) {

char \*subject = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

char \*issuer = X509\_NAME\_oneline(X509\_get\_issuer\_name(cert), 0, 0);

printf("Server certificate:\nSubject: %s\nIssuer: %s\n", subject, issuer);

long verify\_result = SSL\_get\_verify\_result(ssl);

if (verify\_result != X509\_V\_OK) {

printf("Certificate verification error: %s\n", X509\_verify\_cert\_error\_string(verify\_result));

} else {

printf("Certificate verified successfully\n");

}

OPENSSL\_free(subject);

OPENSSL\_free(issuer);

X509\_free(cert);

} else {

printf("No server certificate provided\n");

}

bytes = SSL\_read(ssl, buf, sizeof(buf));

buf[bytes] = 0;

printf("Received: %s", buf);

SSL\_shutdown(ssl);

SSL\_free(ssl);

close(sock);

SSL\_CTX\_free(ctx);

cleanup\_openssl();

return 0;

}

**Файл Server.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#define PORT 4433

#define SERVER\_CERT "server.crt"

#define SERVER\_KEY "server.key"

#define CA\_CERT "ca.crt" // Сертификат удостоверяющего центра для проверки клиентов

void init\_openssl() {

SSL\_load\_error\_strings();

OpenSSL\_add\_ssl\_algorithms();

}

void cleanup\_openssl() {

EVP\_cleanup();

}

SSL\_CTX \*create\_context() {

const SSL\_METHOD \*method;

SSL\_CTX \*ctx;

method = TLS\_server\_method();

ctx = SSL\_CTX\_new(method);

if (!ctx) {

perror("Unable to create SSL context");

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

return ctx;

}

void configure\_context(SSL\_CTX \*ctx) {

if (SSL\_CTX\_use\_certificate\_file(ctx, SERVER\_CERT, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

if (SSL\_CTX\_use\_PrivateKey\_file(ctx, SERVER\_KEY, SSL\_FILETYPE\_PEM) <= 0) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

if (!SSL\_CTX\_check\_private\_key(ctx)) {

fprintf(stderr, "Private key does not match the certificate public key\n");

exit(EXIT\_FAILURE);

}

if (SSL\_CTX\_load\_verify\_locations(ctx, CA\_CERT, NULL) != 1) {

ERR\_print\_errors\_fp(stderr);

exit(EXIT\_FAILURE);

}

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER | SSL\_VERIFY\_FAIL\_IF\_NO\_PEER\_CERT, NULL);

SSL\_CTX\_set\_verify\_depth(ctx, 4);

}

int verify\_callback(int preverify\_ok, X509\_STORE\_CTX \*ctx) {

if (!preverify\_ok) {

fprintf(stderr, "Certificate verification failed\n");

return 0;

}

/\* Можно добавить дополнительную проверку здесь \*/

return 1;

}

int main(int argc, char \*\*argv) {

int sock, client;

struct sockaddr\_in addr;

SSL\_CTX \*ctx;

init\_openssl();

ctx = create\_context();

configure\_context(ctx);

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER | SSL\_VERIFY\_FAIL\_IF\_NO\_PEER\_CERT, verify\_callback);

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Unable to create socket");

exit(EXIT\_FAILURE);

}

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(sock, (struct sockaddr\*)&addr, sizeof(addr)) {

perror("Unable to bind");

exit(EXIT\_FAILURE);

}

if (listen(sock, 1) < 0) {

perror("Unable to listen");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

struct sockaddr\_in client\_addr;

socklen\_t len = sizeof(client\_addr);

SSL \*ssl;

const char reply[] = "Hello from SSL server!\n";

X509 \*cert;

client = accept(sock, (struct sockaddr\*)&client\_addr, &len);

if (client < 0) {

perror("Unable to accept");

continue;

}

printf("Client connected: %s:%d\n",

inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, client);

if (SSL\_accept(ssl) <= 0) {

ERR\_print\_errors\_fp(stderr);

SSL\_shutdown(ssl);

SSL\_free(ssl);

close(client);

continue;

}

cert = SSL\_get\_peer\_certificate(ssl);

if (cert) {

char \*subject = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

char \*issuer = X509\_NAME\_oneline(X509\_get\_issuer\_name(cert), 0, 0);

printf("Client certificate:\nSubject: %s\nIssuer: %s\n", subject, issuer);

OPENSSL\_free(subject);

OPENSSL\_free(issuer);

X509\_free(cert);

} else {

printf("No client certificate provided\n");

}

SSL\_write(ssl, reply, strlen(reply));

SSL\_shutdown(ssl);

SSL\_free(ssl);

close(client);

}

close(sock);

SSL\_CTX\_free(ctx);

cleanup\_openssl();

return 0;

}