Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение   
высшего образования   
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Учебно-методическое пособие

по теме: **«https: как работает, для чего нужна цепочка шифрования»**

по дисциплине

«Криптографические методы защиты информации»

Екатеринбург 2020

**Оглавление**

[**Теория** 3](#_Toc40356452)

[**1.Введение** 3](#_Toc40356453)

[**2. Принцип работы HTTPS** 3](#_Toc40356454)

[**3. SSL и Сертификаты** 5](#_Toc40356455)

[**4. Для чего нужна цепочка сертификатов** 9](#_Toc40356456)

[**Лабораторный практикум** 11](#_Toc40356458)

[**Список литературы** 18](#_Toc40356459)

# **Теория**

## **Введение**

HTTPS (HyperText Transfer Protocol Secure) — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов TLS или устаревшего в 2015 году SSL. В отличие от HTTP с TCP-портом 80, для HTTPS по умолчанию используется TCP-порт 443. Протокол был разработан компанией Netscape Communications для браузера Netscape Navigator в 1994 году. На данный момент самым распространённым видом шифрования трафика http является SSL.

1. **Принцип работы протокола HTTPS**

HTTPS не является отдельным протоколом. Это обычный HTTP, работающий через шифрованные транспортные механизмы SSL и TLS. Он обеспечивает защиту от атак, основанных на прослушивании сетевого соединения — от снифферских атак и атак типа man-in-the-middle, при условии, что будут использоваться шифрующие средства и сертификат сервера проверен и ему доверяют.

По умолчанию HTTPS URL использует 443 TCP-порт. Чтобы подготовить веб-сервер для обработки https-соединений, администратор должен получить и установить в систему сертификат открытого и закрытого ключа для этого веб-сервера. В TLS используется как асимметричная схема шифрования (для выработки общего секретного ключа), так и симметричная (для обмена данными, зашифрованными общим ключом). Сертификат открытого ключа подтверждает принадлежность данного открытого ключа владельцу сайта. Сертификат открытого ключа и сам открытый ключ посылаются клиенту при установлении соединения; закрытый ключ используется для расшифровки сообщений от клиента.

Существует возможность создать такой сертификат, не обращаясь в ЦС. Подписываются такие сертификаты этим же сертификатом и называются само подписанными (self-signed). Без проверки сертификата каким-то другим способом (например, звонок владельцу и проверка контрольной суммы сертификата) такое использование HTTPS подвержено атаке man-in-the-middle.

Эта система также может использоваться для аутентификации клиента, чтобы обеспечить доступ к серверу только авторизованным пользователям. Для этого администратор обычно создаёт сертификаты для каждого пользователя и загружает их в браузер каждого пользователя. Также будут приниматься все сертификаты, подписанные организациями, которым доверяет сервер. Такой сертификат обычно содержит имя и адрес электронной почты авторизованного пользователя, которые проверяются при каждом соединении, чтобы проверить личность пользователя без ввода пароля.

В HTTPS для шифрования используется длина ключа 40, 56, 128 или 256 бит. Некоторые старые версии браузеров используют длину ключа 40 бит (пример тому — IE версий до 4.0), что связано с экспортными ограничениями в США. Длина ключа 40 бит не является сколько-нибудь надёжной. Многие современные сайты требуют использования новых версий браузеров, поддерживающих шифрование с длиной ключа 128 бит, с целью обеспечить достаточный уровень безопасности. Такое шифрование значительно затрудняет злоумышленнику поиск паролей и другой личной информации.

Традиционно на одном IP-адресе может работать только один HTTPS-сайт. Для работы нескольких HTTPS-сайтов с различными сертификатами применяется расширение TLS под названием Server Name Indication (SNI).

**Идентификация в HTTPS**

* Идентификация сервера

HTTP/TLS запросы генерируются путём разыменования URI, вследствие чего имя хоста становится известно клиенту. В начале общения, сервер посылает клиенту свой сертификат, чтобы клиент идентифицировал его. Это позволяет предотвратить атаку человек посередине. В сертификате указывается URI сервера. Согласование имени хоста и данных, указанных в сертификате, происходит в соответствии с протоколом RFC2459

Если имя сервера не совпадает с указанным в сертификате, то пользовательские программы, например браузеры, сообщают об этом пользователю. В основном, браузеры предоставляют пользователю выбор: продолжить незащищённое соединение или прервать его

### Идентификация клиента

Обычно, сервер не располагает достаточной информацией о клиенте, для его идентификации. Однако, для обеспечения повышенной защищённости соединения используется так называемая two-way authentication. При этом сервер, после подтверждения его сертификата клиентом, также запрашивает сертификат. Таким образом, схема подтверждения клиента аналогична идентификации сервера

**Уязвимости HTTPS**

* Совместное использование HTTP и HTTPS

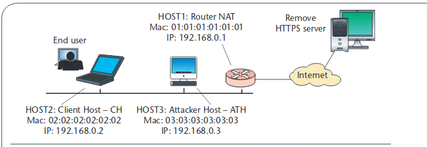
Когда сайты используют смешанную функциональность HTTP и HTTPS, это потенциально приводит к информационной угрозе для пользователя. Например, если основные страницы некоторого сайта загружаются, используя HTTPS, а CSS и JavaScript загружаются по HTTP, то злоумышленник в момент загрузки последних может подгрузить свой код и получить данные HTML-страницы. Многие сайты, несмотря на такие уязвимости, загружают контент через сторонние сервисы, которые не поддерживают HTTPS. Механизм HSTS позволяет предотвратить подобные уязвимости, заставляя принудительно использовать HTTPS соединение даже там, где по умолчанию используется HTTP.

### Атаки с использованием анализа трафика

В HTTPS были обнаружены уязвимости, связанные с анализом трафика. Атака с анализом трафика — это тип атаки, при которой выводятся свойства защищённых данных канала путём измерения размера трафика и времени передачи сообщений в нём. Анализ трафика возможен, поскольку шифрование SSL/TLS изменяет содержимое трафика, но оказывает минимальное влияние на размер и время прохождения трафика.

В мае 2010 года исследователи из Microsoft Research и Университета Индианы обнаружили, что подробные конфиденциальные пользовательские данные могут быть получены из второстепенных данных, таких например, как размеры пакетов. Анализатор трафика смог заполучить историю болезней, данные об использовавшихся медикаментах и проведённых операциях пользователя, данные о семейном доходе и пр. Всё это было произведено несмотря на использование HTTPS в нескольких современных веб-приложениях в сфере здравоохранения, налогообложения и других

### Человек посередине.

[](https://commons.wikimedia.org/wiki/File:Mim8989.png?uselang=ru)

**Рис.1 Стандартная конфигурация сети. Пользователь на хосте клиента (CH) хочет осуществить безопасную транзакцию, но уязвим для атаки «человек в середине».**

При атаке «человек посередине» используется то, что сервер HTTPS отправляет сертификат с открытым ключом в браузер. Если этот сертификат не заслуживает доверия, то канал передачи будет уязвимым для атаки злоумышленника. Такая атака заменяет оригинальный сертификат, удостоверяющий HTTPS-сервер, модифицированным сертификатом. Атака проходит успешно, если пользователь пренебрегает двойной проверкой сертификата, когда браузер отправляет предупреждение. Это особенно распространено среди пользователей, которые часто сталкиваются с самозаверенными сертификатами при доступе к сайтам внутри сети частных организаций

На рис. 1 представлена ситуация, когда злоумышленник является шлюзом между клиентом, осуществляющим безопасную транзакцию, и сервером. Таким образом через злоумышленника проходит весь трафик клиента и он может перенаправить его по своему усмотрению. Здесь делаются следующие шаги:

1. Злоумышленник встраивается между клиентом и сервером
2. Пересылает все сообщения от клиента серверу без изменений
3. Перехват сообщений от сервера, посланных по шлюзу по умолчанию
4. Создание самозаверенного сертификата и подмена им сертификата сервера
5. Отправление ложного сертификата клиенту
6. Когда клиент подтвердит сертификат, будут установлены защищённые соединения: между злоумышленником и сервером и другое — между злоумышленником и клиентом
7. **SSL и Сертификаты**

SSL (Secure Sockets Layer) — криптографический протокол, который подразумевает более безопасную связь. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Протокол широко использовался для обмена мгновенными сообщениями и передачи голоса через IP (англ. Voice over IP — VoIP) в таких приложениях, как электронная почта, интернет-факс и др.

Протокол SSL обеспечивает защищённый обмен данными за счёт двух следующих элементов:

* Аутентификация
* Шифрование

SSL использует асимметричную криптографию для аутентификации ключей обмена, симметричный шифр для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений.

Протокол SSL предоставляет «безопасный канал», который имеет три основных свойства:

* Канал является частным. Шифрование используется для всех сообщений после простого диалога, который служит для определения секретного ключа.
* Канал аутентифицирован. Серверная сторона диалога всегда аутентифицируется, а клиентская делает это опционально.
* Канал надёжен. Транспортировка сообщений включает в себя проверку целостности.

Преимуществом SSL является то, что он независим от прикладного протокола. Протоколы приложений (HTTP, FTP, TELNET и т. д.) могут работать поверх протокола SSL совершенно прозрачно, то есть SSL может согласовывать алгоритм шифрования и ключ сессии, а также аутентифицировать сервер до того, как приложение примет или передаст первый байт сообщения.

**Принцип работы**

Протокол SSL размещается между двумя протоколами: протоколом, который использует программа-клиент (HTTP, FTP, LDAP, TELNET и т.д.) и транспортным протоколом TCP/IP. SSL защищает данные, выступая в роли фильтра для обеих сторон и передаёт их далее на транспортный уровень[[4]](https://ru.wikipedia.org/wiki/SSL#cite_note-4)[[5]](https://ru.wikipedia.org/wiki/SSL#cite_note-5).

Работу протокола можно разделить на два уровня:

1. Слой протокола подтверждения подключения (Handshake Protocol Layer)
2. Слой протокола записи

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

1. Протокол подтверждения подключения (Handshake Protocol)
2. Протокол изменения параметров шифра (Cipher Spec Protocol)
3. Предупредительный протокол (Alert Protocol)

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

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

**Протокол подтверждения подключения** производит цепочку обмена данными, что в свою очередь начинает аутентификацию сторон и согласовывает шифрование, хеширование и сжатие. Следующий этап — аутентификация участников, которая осуществляется также протоколом подтверждения подключения.

**Протокол изменения параметров шифра** используется для изменения данных ключа (keyingmaterial) — информации, которая используется для создания ключей шифрования. Протокол состоит всего из одного сообщения, в котором сервер говорит, что отправитель хочет изменить набор ключей.

**Предупредительный протокол** содержит сообщение, которое показывает сторонам изменение статуса или сообщает о возможной ошибке. Обычно предупреждение отсылается тогда, когда подключение закрыто и получено неправильное сообщение, сообщение невозможно расшифровать или пользователь отменяет операцию.

**Цифровые сертификаты**

SSL-сертификат, это индивидуальная цифровая подпись вашего домена. Он может быть:

1. Самоподписанным. Это значит, что вы сами выдали себе сертификат, и сами его подписали.
2. Подписанный недоверенным центром сертификации. Это значит, что сертификат сайта проверен, но сам «проверяющий» доверия не удостоен.
3. Подписанный доверенным ЦС. Это значит, что данные сертификата проверены компанией, которая имеет на это право, они как минимум существуют.  
     
   Разберем их более подробно.  
     
   Самоподписанный сертификат не гарантирует ничего. Любой человек может взять и выдать себе такой сертификат. Все браузеры выдают клиенту предпреждение о том, что сертификат не надежен.  
   Подписанный не доверенным ЦС сертификат тоже не подтверждает ничего, т.к. существуют ЦС, продающие сертификаты всем желающим и без проверок. Большинство браузеров реагирует на такие сертификаты аналогично самоподписанным.  
     
   Сертификат, подписанный доверенным источником (как пример — Thawte или VerySign) подтверждает, что:

* Данный сайт действительно принадлежит компании, за которую себя выдает, а не Васе-фишеру из соседнего подъезда.
* Компания, которую представляет сайт — действительно существует в жизни, а не в мыслях Васи из соседнего подъезда.
* Данные этой компании проверены и зарегистрированы центром сертификации.

На доверенные сертификаты браузеры ошибку не выдают.  
  
Но это технически. А теперь о том, что показывает доверенный сертификат посетителю вашего сайта.

* Это действительно тот сайт, на который мы шли, а не дефейс или фишинг.
* Сайт создан в серьез и надолго. В общем случае, желающие «поиграть недельку» не готовы выложить деньги за сертификат.
* Сайт принадлежит компании, либо зарегистрированному физ. лицу, а не неведомому анониму. Плюсы понятны — желающие обмануть или украсть редко стремятся удостоверить свою личность.
* Компанию волнует защищенность информации и подтверждение своей подлинности.
* Если что-то случится, эту компанию можно найти через сертификатора.  
    
  Многих пользователей (особенно зарубежных — наши пока к этому не привыкли) самоподписанный сертификат (или отсутствие SSL в вещах, касающихся услуг\финансов\privacy) может если и не отпугнуть, то поставить жирный минус в вашу пользу.  
    
  Мой личный вывод: на всех сайтах, связанных с онлайн-коммерцией, платежами, личной информацией SSL должен быть.

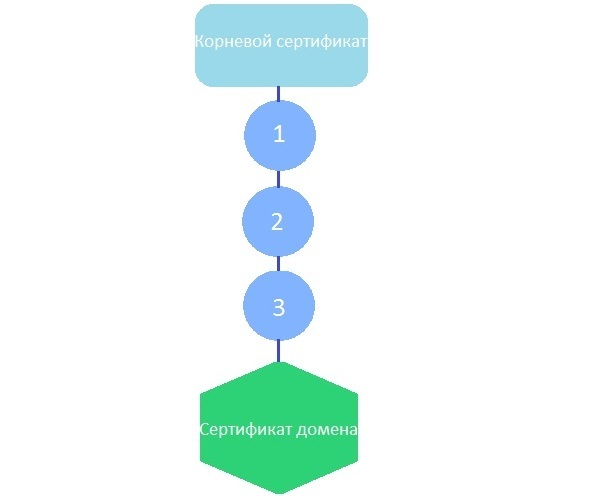
**Типы сертификатов:**  
**Esential SSL** — самый не дорогой и быстро оформляемый сертификат. Доступен как для юридических, так и для физических лиц. Проверяется только право владения доменным именем, личные данные или регистрация компании не проверяются. Выдается на 1 домен.  
  
**Instant SSL** — доступен и для физ. лиц, и для юр. лиц. Проверяется право владения доменом, регистрационные данные компании либо личность физ. лица. Выдается на 1 домен.  
  
**SGC SSL-сертификат.** — Аналогично Instant SSL, но с поддержкой 40-битных расширений (актуально для старых ОС и браузеров). Выдается на 1 домен, либо wildcard (см. ниже).  
  
**Обычный Wildcard.**— тоже самое, что и обычный сертификат, но выдается не на 1 домен, а на все поддомены корневого домена. Т.е. не только на domain.com, a и на [www.domain.com](http://www.domain.com/), bill.domain.com и т.д. Стоит на порядок дороже.  
  
**EV (Extended Validation) сертификат.** — сертификат расширенной проверки, доступен только юридическим лицам. Проверяется владение доменом, компания, нотариально заверенные переводы документов на английский язык, требует подтверждения данных третьей стороной. Позволяет установить на сайте картинку-подтверждение владением и отображается в браузерах как гарантированно доверенный (зеленым цветом), против желтого у обычных сертификатов. Стоит в 2-3 раза дороже обычного, регистрация занимает продолжительное время.

1. **Для чего нужна цепочка шифрования**

Установка SSL-сертификата является важным этапом при переходе на протокол безопасного соединения HTTPS — он обеспечивает сохранность персональных данных. Такой сайт вызывает больше доверия у пользователей. Также защищенность веб-ресурса может влиять на его позиции в поисковой выдаче.  
  
Как правило для шифрования информации, пересылаемой между браузером посетителя и сервером, достаточно одного сертификата. Однако некоторые ресурсы требуют большей надежности и многоуровневой защиты. Например, банковский ресурс предполагает проведение крупных финансовых операций. В этом случае необходим SSL-сертификат, который содержит не только сертификат для домена, но цепочку сертификатов (CA Bundle).  
  
Цепочка сертификатов SSL включает в себя сертификаты поручителей, подтверждающие валидность документа в целом. Структура CA Bundle имеет такой вид:

* Корневой сертификат (Root).
* Сертификаты посредников (Intermediate).

Каждый сертификат цепочки имеет электронную цифровую подпись, связывающую его с сертификатом на ступень ниже. Root CA является верхним звеном иерархии сертификатов. Уточнение CA (Certificate Authority) означает, что сертификаты выдает удостоверяющий центр, который подтверждает этим документом подлинность ключей шифрования.



**Как настроить цепочку сертификатов**

Структура звеньев цепочки зависит от типа сертификата. Данную последовательность как правило можно получить вместе с доменным сертификатом на электронную почту или скачать на сайте SSL-провайдера, выпустившего сертификат. В этом случае поручителем выступает центр сертификации. Следующий этап — настроить цепочку SSL. Это можно сделать двумя способами.

### **Создать текстовый документ**

Для этого необходимо поместить цепочку сертификатов в виде списка в текстовый документ. Как выглядит CA Bundle можно увидеть в примере ниже — сертификаты с расширением .crt размещены в заданной последовательности:

* CARoot.crt
* Intermediate1.crt
* Intermediate2.crt
* Intermediate3.crt
* domain.crt

Файл CARoot.crt здесь — корневой сертификат, связка файлов Intermediate выполняет функцию посредников, domain.crt является сертификатом домена. Поручителей в цепочке может быть много. Основная задача — следить, чтобы все звенья цепочки были связаны друг с другом цифровой подписью.  
  
Текстовый файл с цепочкой сертификатов следует сохранить как domain.ca-boundle.

### **Использовать командную строку**

В данной строке необходимо перечислить сертификаты посредников по порядку и завершить последовательность указанием имени файла domain.ca-bundle.

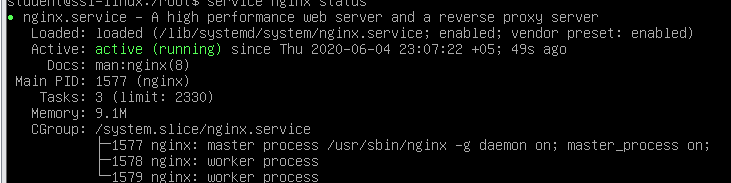
# **Лабораторный практикум**

В комплекте с лабораторной работы присутствуют готовые виртуальный машины Debian и Windows 7 для выполнения лабораторной работы. Лабораторная работа разделена на две части, первая часть подразумевает настройку сертификатов в Linux, вторая часть в Windows

**Организация HTTPS в NGINX путем создания самоподписанного SSL сертификата**

* 1. Ознакомьтесь с теоретическим минимумом с целью ознакомления о цепочках сертификата, а также структуру работы SSL сертификата. Это нам в дальнейшем потребуется при генерировании самоподписанного сертификата
  2. Первым шагом нам необходимо подготовить nginx сервер для дальнейшего создания и тестирования. Для этого запустите виртуальную машину SSL\_Linux (в приложении к лабораторной работе)
  3. Далее нужно авторизоваться под логином **student** и паролем 12345678(root не будем использовать в целях безопасности.
  4. Далее необходимо обновить дистрибутивы. Введите команду **sudo apt-get update**, затем **apt-get upgrade**.
  5. Далее необходимо установить веб-сервер Ngin. Для этого введите команду **sudo apt-get install nginx**.
  6. Далее необходимо проверить работу нашего веб-сервера. Введите команду **service nginx status**

**Вывод должен быть такой:**



**В таком случае установка выполнено корректно**

* 1. Далее перейдем непосредственно к настройке HTTPS. Первым шагом нам будет необходимо сгенерировать SSL сертификат.

TLS / SSL работает с использованием комбинации открытого сертификата и закрытого ключа. Ключ SSL хранится в секрете на сервере. Он используется для шифрования контента, отправляемого клиентам. Сертификат SSL является общедоступным для всех, кто запрашивает контент. Его можно использовать для расшифровки содержимого, подписанного соответствующим ключом SSL.

Создадим самоподписанный ключ и пару сертификатов с OpenSSL в одной команде:

**sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt**

* 1. Вам будет задан ряд вопросов. Прежде чем перейти к этому, давайте посмотрим, что происходит в команде, которую мы выдаем:
* \* openssl \*: это основной инструмент командной строки для создания и управления сертификатами, ключами и другими файлами OpenSSL.
* \* req \*: эта подкоманда указывает, что мы хотим использовать управление запросом подписи сертификата X.509 (CSR). «X.509» - это стандарт инфраструктуры открытых ключей, которого придерживаются SSL и TLS для управления ключами и сертификатами. Мы хотим создать новый сертификат X.509, поэтому мы используем эту подкоманду.
* \* -x509 \*: Это дополнительно изменяет предыдущую подкоманду, сообщая утилите, что мы хотим сделать самозаверяющий сертификат вместо генерации запроса на подпись сертификата, как это обычно происходит.
* \* -nodes \*: Это говорит OpenSSL пропустить опцию для защиты нашего сертификата парольной фразой. Нам нужно, чтобы Nginx мог читать файл без вмешательства пользователя при запуске сервера. Парольная фраза предотвратит это, потому что мы должны будем вводить ее после каждого перезапуска.
* \* -days 365 \*: эта опция устанавливает период времени, в течение которого сертификат будет считаться действительным. Мы установили это на один год здесь.
* \* -newkey rsa: 2048 \*: указывает, что мы хотим создать новый сертификат и новый ключ одновременно. Мы не создали ключ, необходимый для подписи сертификата на предыдущем шаге, поэтому нам нужно создать его вместе с сертификатом. Часть + **rsa: 2048** + говорит ему сделать ключ RSA длиной 2048 бит.
* \* -keyout \*: эта строка сообщает OpenSSL, куда поместить созданный нами файл закрытого ключа.
* \* -out \*: Это сообщает OpenSSL, где разместить сертификат, который мы создаем.

Заполните соответствующие подсказки. \* Самая важная строка - это та, которая запрашивает `+ Common Name (например, FQDN сервера или ВАШЕ имя) + `. Вам необходимо ввести доменное имя, связанное с вашим сервером, или, более вероятно, публичный IP-адрес вашего сервера. \*

* 1. Далее, нам также следует создать сильную группу Диффи-Хеллмана, которая используется для согласования [Perfect Forward Secrecy](https://en.wikipedia.org/wiki/Forward_secrecy) с клиентами. Введите команду **sudo openssl dhparam -out /etc/nginx/dhparam.pem 4096.** Это займет некоторое время.
  2. Мы создали файлы ключей и сертификатов в каталоге /etc/ssl/. Далее нам потребуется изменить конфигурацию Nginx для использования SSL сертификата
  3. Мы внесем несколько изменений в нашу конфигурацию.Мы создадим фрагмент конфигурации, содержащий наш ключ SSL и расположение файла сертификата. Мы создадим фрагмент конфигурации, содержащий строгие настройки SSL, которые могут быть использованы с любыми сертификатами в будущем. Мы настроим наши серверные блоки Nginx для обработки запросов SSL и будем использовать два приведенных выше фрагмента.

Во первых создадим новый фрагмент конфигурации Nginx в каталоге /etc/nginx/snippets. Введите команду **sudo nano /etc/nginx/snippets/self-signed.conf**

В этом файле нам нужно установить директиву + ssl\_certificate для вашего файла сертификата, а ключ` + ssl\_certificate\_key` - для соответствующего ключа. В нашем случае это будет выглядеть так:

**ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;**

**ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;**

Далее закройте файл

* 1. Далее мы создадим еще один фрагмент, который определит некоторые настройки SSL. Это обеспечит Nginx надежным набором шифров SSL и включит некоторые расширенные функции, которые помогут обеспечить безопасность нашего сервера.

Введите команду **sudo nano /etc/nginx/snippets/ssl-params.conf**

* 1. Скопируйте в файл следующее содержимое:

ssl\_protocols TLSv1.2;

ssl\_prefer\_server\_ciphers on;

ssl\_dhparam /etc/nginx/dhparam.pem;

ssl\_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384;

ssl\_ecdh\_curve secp384r1; # Requires nginx >= 1.1.0

ssl\_session\_timeout 10m;

ssl\_session\_cache shared:SSL:10m;

ssl\_session\_tickets off; # Requires nginx >= 1.5.9

ssl\_stapling on; # Requires nginx >= 1.3.7

ssl\_stapling\_verify on; # Requires nginx => 1.3.7

resolver 8.8.8.8 8.8.4.4 valid=300s;

resolver\_timeout 5s;

# Disable strict transport security for now. You can uncomment the following

# line if you understand the implications.

# add\_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";

add\_header X-Frame-Options DENY;

add\_header X-Content-Type-Options nosniff;

add\_header X-XSS-Protection "1; mode=block";

Поскольку мы используем самозаверяющий сертификат, сшивание SSL не будет использоваться. Nginx выдаст предупреждение, но продолжит работать правильно. Закройте файл.

* 1. Теперь, когда у нас есть фрагменты, мы можем настроить нашу конфигурацию Nginx для включения SSL.

В этом руководстве мы предполагаем, что вы используете пользовательский файл конфигурации блока сервера в каталоге  **/etc/nginx/sites-available**. Для этого примера мы будем использовать  **/etc/nginx/sites-available/example.com** . Замените ваше имя файла конфигурации при необходимости.

* 1. Откройте файл конфигурации, чтобы внести изменения. Введите команду **sudo nano /etc/nginx/sites-available/**
  2. В существующем файле конфигурации измените два оператора listen чтобы исползовать порт 443 и SSL, а затем включите два файла фрагмента, которые мы создали на предающих шагах

server {

listen ;

listen [::]:;

server\_name ;

include snippets/self-signed.conf;

include snippets/ssl-params.conf;

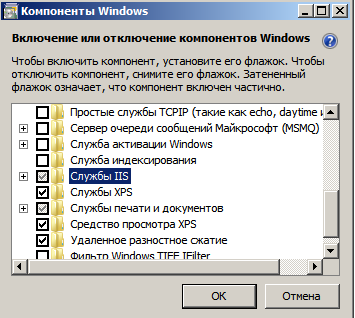
}

Конфигурация должна выглядеть так

* 1. Теперь, когда мы настроили работу Nginx сервера с сертификатами SSL. Мы должны убедиться, что в файлах конфигурации нет ошибок. Введите команду **sudo nginx -t**
  2. Перезагрузите сервер Nginx. Введите команду **sudo systemctl restart nginx**
  3. Далее протестируем SSL сертификат. Введите команду **ip a** чтобы увидеть ip адрес сервера.
  4. Далее откройте браузер и введите в адрессной строке **https://ваш ip/**
  5. Поскольку созданный нами сертификат не подписан одним из доверенных центров сертификации вашего браузера, вы, скорее всего, увидите страшное предупреждение.
  6. Это ожидается и нормально. Нас интересует только аспект шифрования нашего сертификата, а не проверка третьей стороной подлинности нашего хоста.
  7. Если все открылось и работает, значит все прошло успешно. На этом первая часть лабораторной работы завершена

**Организация HTTPS в IIS (Windows)**

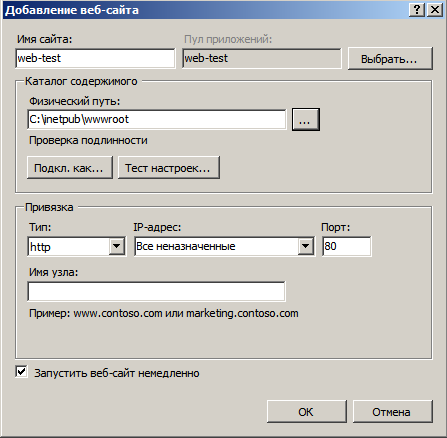
* 1. Запустите виртуальную машину SSL\_Windows
  2. Открываем Панель управления -> Программы -> Включение или отключение компонентов Windows.
  3. Находим в списке раздел Службы IIS. Раскрываем его и выбираем нужные компоненты



* 1. После установки компонентов необходимо **обязательно** перезагрузить виртуальную машину
  2. Далее создадим тестовый сайт для тестирования в нем сертификата SSL.

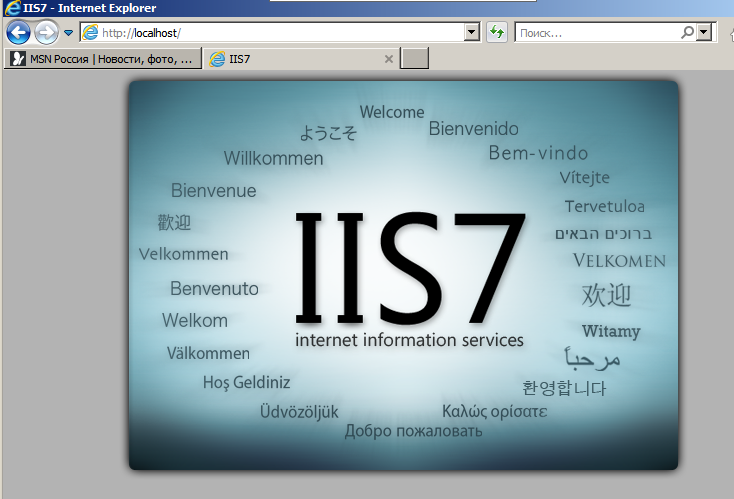
Открываем Панель управления -> Система и безопасность -> Администрирование -> Управление компьютером (можно быстрее: меню Пуск -> правый клик на Компьютер -> в меню выбрать пункт Управление). В открывшемся окне раскрываем группу “Службы и приложения” и открываем “Диспетчер служб IIS”. В окне Подключения выбираем папку Сайты, затем в правом окне Действия нажимаем на ссылку “Добавить веб-сайт”

* 1. Заполните нужные поля



Нажмите ОК. На этом этапе настройка сервера завершена

* 1. Откройте браузер и убедитесь, что сервер работает



* 1. Повторно откройте IIS менеджер и перейдите в корень настроек (имя виртуальной машины.
  2. Далее найдите и откройте настройку **Сертификаты сервера (**кликните два раза)
  3. Справа в меню **Действия** выберите **Создать самозаверенный сертификат**
  4. Введите любое имя сертификата и нажмите ОК.
  5. Далее найдите в IIS Manager свой сайт и нажмите правую кнопку мыши и нажмите кнопку **Изменить привязки**
  6. Далее добавьте новую HTTPS привязку и укажите в списке наш полученный сертификат.
  7. Далее откройте командную строку и введите команду **iisreset /restart**
  8. Повторно перезагрузите компьютер
  9. Далее нам нужно проверить, что мы правильно создали наш сертификат. Откройте браузер Explorer на виртуальной машине и введите [**https://localhost**](https://localhost)
  10. Если все верно настроено, то вы увидите



* 1. Нажмите Продолжить открытие этого веб-сайта. Также как и ситуация в nginx, самоподписанной сертификат не был подтвержден центрами сертификатами и поэтому мы видим данное предупреждение, но нам была важна суть установки SSL сертификата. И это мы успешно сделали. На этом лабораторная работа завершена

# **Список литературы**

* 1. <https://ru.wikipedia.org/wiki/SSL>
  2. <https://1cloud.ru/blog/cepochka-ssl-sertifikatov>
  3. <https://ru.wikipedia.org/wiki/Nginx>
  4. <https://ru.wikipedia.org/wiki/HTTPS>
  5. <https://te-st.ru/2014/12/03/what-is-ssl/>