Лабораторная работа №6
«Сертификаты X.509 и инфраструктура открытых ключей»

Сертификаты – это структуры данных, применяемые для представления и проверки пользователя. Сертификаты X.509 необходимы для функционирования протокола TLS и основанных на нем, например HTTPS, где они используются для доказательства подлинности веб-сайтов. Сертификаты используются также в стандартах безопасного обмена сообщениями, например S/MIME, в VPN-решениях, например OpenVPN, в смарт-картах, при подписании программ и т. д. Факультативно их можно использовать также в протоколе IPsec.

Что такое сертификат X.509?
Сертификат – это структура данных, применяемая для представления и проверки пользователя. Сертификат можно сохранить в файле или передать по сети, он также может быть частью безопасного сетевого протокола, например TLS. Сертификат X.509 связывает идентичность объекта с открытым ключом посредством цифровой подписи.
Приведем пример текстового представления сертификата X.509:

```
Certificate:
      Data:
            Version: 3 (0x2)
            Serial Number: ... шестнадцатеричные байты ...
            Signature Algorithm: sha256WithRSAEncryption
            Issuer: C = US, O = Let’s Encrypt, CN = R3
            Validity
                  Not Before: Mar 4 12:43:52 2022 GMT
                  Not After : Jun 2 12:43:51 2022 GMT
            Subject: CN = www.openssl.org
            Subject Public Key Info:
                  Public Key Algorithm: rsaEncryption
                        RSA Public-Key: (2048 битов)
                        Modulus: ... шестнадцатеричные байты ...
                        Exponent: 65537 (0x10001)
            X509v3 extensions:
                  X509v3 Key Usage: critical
                        Digital Signature, Key Encipherment
                  X509v3 Extended Key Usage:
                        TLS Web Server Authentication,
                        TLS Web Client Authentication
                  X509v3 Basic Constraints: critical
                        CA:FALSE
                  X509v3 Subject Key Identifier:
                        ... шестнадцатеричные байты ..
                  X509v3 Authority Key Identifier:
                        ... шестнадцатеричные байты ...
                  ... дополнительные расширения X509v3 …
```

Как видим, сертификат X.509 состоит из следующих полей:

- **X.509 version** (версия X.509): современные сертификаты X.509 имеют версию 3;
- **Serial number** (серийный номер): уникальный номер сертификата среди всех сертификатов, подписанных одним издателем;
- **Signature algorithm** (алгоритм подписи): криптографическая функция хеширования и алгоритм цифровой подписи, использованные для подписания сертификата;
- **Issuer** (издатель): юридическое лицо, подписавшее сертификат, или, в терминологии X.509, выпустившее (издавшее) его. Издатель представлен в формате различимого имени (Distinguished Name – DN), который будет описан ниже;
- **Validity** (действительность): два поля временных меток, Not Before (не раньше) и Not After (не позже), определяющие срок действия сертификата;

- **Subject** (субъект): юридическое или физическое лицо, идентифицируемое сертификатом. Это может быть человек, сайт, организация или что-то еще. Поле Subject также представлено в формате различимого имени;
- **открытый ключ сертификата**: открытый ключ важен для проверки идентичности, представленной сертификатом. Поддерживаются разные типы ключей, в т. ч. RSA, DSA, ECDSA и EdDSA. Важно помнить, что сертификат X.509 не содержит закрытого ключа, только открытый;
- **X509v3 extensions** (расширения X509v3): содержат дополнительную информацию;
- **подпись сертификата**: предоставляется издателем сертификата.

Как открытые и закрытые ключи, сертификаты X.509 кодируются в абстрактной синтаксической нотации версии 1 (**ASN.1**) и могут быть представлены в одном из двух форматов: **Distinguished Encoding Rules (DER)** или **Privacy-Enhanced Mail (PEM)**.
Поля Subject и Issuer сертификатов X.509 представлены в формате различимого имени, DN. Различимое имя выглядит следующим образом: `C = US, O = Let's Encrypt, CN = R3`. Как видно, DN состоит из пар ключ-значение. У ключей имеется точно определенная семантика; например, `C` означает `Country` (страна), а `O – Organization` (организация). Самым важным является ключ `CN`, или `Common Name` (стандартное имя). Это основное имя лица, идентифицируемого сертификатом. Это может быть имя сайта, например `www.openssl.org`; имя человека, например `John Doe`; или если сертификат выпущен для технических целей, то имя самого сертификата, например `Technical certificate 123` или просто R3.
Каждому сертификату X.509 соответствует закрытый ключ. Он не включен в состав сертификата, но является парным открытому ключу, включенному в сертификат, т. е. цифровая подпись, порожденная этим закрытым ключом, может быть проверена открытым ключом в сертификате. Этот закрытый ключ часто называют закрытым ключом сертификата. Владельцу сертификата необходим закрытый ключ, чтобы доказать, что он действительно владеет сертификатом, а не просто скопировал его. Такое доказательство владелец может предъявить, подписав какие-то данные своим закрытым ключом и попросив другую сторону проверить подпись с помощью открытого ключа, извлеченного из сертификата.
Сертификат X.509 обычно не является секретной информацией, его можно свободно распространять, как и открытый ключ. С другой стороны, закрытый ключ сертификата – секрет, который должен быть известен только его владельцу, поскольку всякий обладающий закрытым ключом может объявить себя владельцем сертификата и криптографически строго доказать это. Кража закрытого ключа может привести к краже личности. Например, если кто-то украдет закрытый ключ сертификата сайта `www.openssl.org` и при этом еще и организует атаку **отравления DNS** или **атаку с человеком посередине**, то сможет создать ложный сайт `www.openssl.org`, который пройдет проверку сертификата. Существуют способы смягчить последствия описанной кражи личности, например **списки отозванных сертификатов** (Certificate Revocation List – **CRL**) и **протокол онлайновой проверки состояния сертификата** (Online Certificate Status Protocol – **OCSP**), но у противника все равно может остаться достаточно времени для атаки, прежде чем скомпрометированный сертификат будет отозван. Другой пример: если закрытый ключ сертификата использован для подписания сообщений электронной почты, то всякий укравший ключ сможет подписывать почту от имени владельца сертификата.
Когда какие-то данные подписаны закрытым ключом сертификата, часто говорят, что они подписаны сертификатом. Сами сертификаты тоже могут быть подписаны. На самом деле любой сертификат X.509 подписан каким-то сертификатом, т. е. его закрытым ключом. Подписавший сертификат указан в поле `Issuer` в формате DN. Сертификат может быть подписан и своим собственным закрытым ключом и тогда называется самоподписанным. У самоподписанных сертификатов DN-адреса в полях `Subject` и `Issuer` одинаковы. Бывает и так, что один сертификат подписан другим, тот – третьим и т. д. В таком случае имеется цепочка сертификатов.

Цепочки сертификатов

Цепочкой подписания сертификатов, или цепочкой проверки сертификатов, или просто цепочкой сертификатов, или даже цепочкой доверия, называется упорядоченная последовательность сертификатов, в которой каждый сертификат подписан следующим в цепочке. Кроме, конечно, последнего. Последний сертификат является самоподписанным.

Зачем нужны цепочки сертификатов? Чтобы проверить их действительность. При проверке идентичности с использованием сертификата X.509 мы должны удостовериться в истинности двух утверждений:

1. **лицо, предъявившее сертификат для идентификации, является его владельцем.** Это утверждение доказывается с помощью закрытого ключа сертификата;
2. **предъявленный сертификат действителен.** Это утверждение доказывается с помощью цепочки сертификатов.

Тут все обстоит так же, как при идентификации человека по паспорту. Вы можете идентифицировать себя, предъявив паспорт, но этот паспорт должен быть действителен.

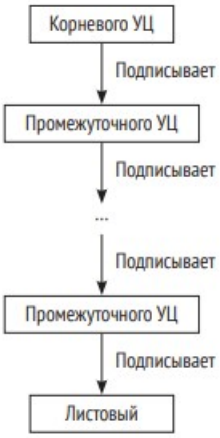
Как устроена проверка сертификата? Чтобы понять это, мы должны разобраться в том, как сертификаты создаются, или, в терминологии X.509, *выпускаются*.

Сертификаты большинства сайтов в интернете выпущены так называемыми **удостоверяющими центрами** (**УЦ**). УЦ обычно взимает плату за выпуск сертификата. Некоторые УЦ, например Let’s Encrypt, выпускают сертификаты бесплатно. CA представляют себя как **центры доверия** (trusted third party). Идея в том, что вы доверяете УЦ выпуск сертификатов и считаете выпущенные ими сертификаты действительными. Вы уверены, что УЦ проверяет личности заказавших сертификат и выпускает сертификат только для тех, чья личность совпадает с записанной в поле Subject сертификата.

Для проверки сертификата необходимо построить цепочку, оканчивающуюся надежным сертификатом. Рассмотрим широко распространенный случай проверки сертификата веб-сайта. Первым сертификатом в цепочке будет сертификат, идентифицирующий сайт. Этот сертификат не подписывает и не выпускает никаких других сертификатов. Сертификаты, которые не подписывают других сертификатов, называются **листовыми**. Сертификат сайта подписан вторым сертификатом в цепочке. В большинстве случаев этот второй сертификат не является самоподписанным, а потому на нем цепочка не обрывается. Сертификаты, которые подписывают другие сертификаты, но не являются самоподписанными, называются **сертификатами промежуточных УЦ**. В типичной цепочке, начинающей сертификатом сайта, будет один или два сертификата промежуточных УЦ. После всех промежуточных сертификатов идет последний сертификат в цепочке, который является самоподписанным. Самоподписанные сертификаты, которые подписывают другие сертификаты, называются **корневыми**.

Цепочка сертификатов выглядит следующим образом:

В интернете также часто встречается вертикально расположенная цепочка сертификатов:



Отметим, что горизонтальное представление выглядит как связанный список, в котором сертификат, подлежащий проверке, находится в начале, а надежный сертификат – в конце. Именно так представлены цепочки сертификатов в OpenSSL.
В наших описаниях и рисунках мы упоминали промежуточные УЦ. Но зачем они нужны? Почему не подписывать все листовые сертификаты корневыми? Тому есть несколько причин:

- если промежуточный сертификат скомпрометирован, то его можно отозвать, не отзывая корневой сертификат и прочие промежуточные сертификаты, подписанные тем же корневым. Поэтому область воздействия скомпрометированного сертификата сужается;
- многие УЦ поддерживают автоматический выпуск сертификатов в онлайновом режиме. Это означает, что закрытый ключ подписывающего сертификата должен находиться на сервере, который прямо или косвенно доступен из интернета. Это создает риски. Лучше подвергнуть риску промежуточный сертификат, чем корневой. Поскольку корневой сертификат подписывает другие не слишком часто, его закрытый ключ можно хранить в безопасном месте, не подключенном к сети, например на флеш-накопителе в стальном сейфе, в запертом и охраняемом помещении;
- писки отозванных сертификатов крупных УЦ могут быть очень велики. Имея несколько промежуточных сертификатов, УЦ может разделить свой пул выпущенных сертификатов между промежуточными УЦ и публиковать меньшие по размеру CRL, по одному на каждый промежуточный УЦ.

Итак, цепочки сертификатов нужны для проверки сертификатов и могут содержать листовые (концевые), промежуточные и корневые сертификаты. Но кто проверяет эти цепочки и как проверяющая сторона получает все сертификаты? Всякий, кто может получить необходимые сертификаты и построить цепочку, может проверить сертификат. Вернемся к проверке сертификата сайта. В этом случае проверкой сертификата занимается браузер или другой HTTPS-клиент. Браузер получает сертификат сервера во время процедуры квитирования TLS. А как браузер получает сертификат корневого УЦ? Из хранилища сертификатов в браузере или в ОС. В большинстве операционных систем имеется общесистемное хранилище сертификатов, в котором они разбиты на категории: сертификаты надежных УЦ, сертификаты ненадежных УЦ или сертификаты клиентов с закрытыми ключами. Эти хранилища регулярно обновляются с помощью механизмов обновления ОС. В некоторых браузерах имеются собственные хранилища сертификатов, которые можно использовать вместе или вместо хранилища ОС. Такие хранилища обновляются самими браузерами и необязательно в момент выпуска новой версии, а периодически.
Кто решает, каким сертификатам можно доверять? Лицо, отвечающее за обслуживание хранилища сертификатов, решает, как его инициализировать и какие изменения (добавления или удаления) включать в обновления. В случае хранилища ОС обслуживание осуществляет производитель ОС, а в случае хранилища браузера – производитель браузера. Пользователь

ОС и браузера также может добавлять и удалять сертификаты. Однако не так много пользователей обладают необходимыми для этого знаниями, да и потребность или желание изменить содержимое хранилища сертификатов возникает нечасто. Поэтому в большинстве случаев хранилище сертификатов содержит только те сертификаты, которые решил включить производитель ОС или браузера.
Содержимое хранилищ сертификатов в разных ОС и браузерах полностью или почти полностью совпадает. В них помещаются сертификаты хорошо известных корневых УЦ. Если вы заходите на сайт, имеющий сертификат, выпущенный одним из этих УЦ, то браузер сможет проверить сертификат сайта и покажет вам страницу.
Как строится цепочка сертификатов при проверке сертификата? Очевидно, что первым сертификатом в цепочке является проверяемый, например сертификат сайта. Но как решить, какой сертификат помещать в цепочку следующим? Найти следующий сертификат можно двумя способами. Во-первых, использовать поле `Issuer` текущего сертификата и поискать сертификат с таким же DN в поле `Subject`. Поле `Issuer` всегда заполнено в любом сертификате, но иногда можно найти несколько сертификатов с одним и тем же субъектом. Поэтому поиск по `Issuer` неоднозначен. Второй способ – воспользоваться расширением X509v3 *«Authority Key Identifier»* (идентификатор ключа УЦ), которое содержит криптографический хеш открытого ключа сертификата издателя. Однако расширения X509v3 факультативны, поэтому в конкретном сертификате расширение может отсутствовать. Кроме того, при возобновлении сертификата ключ иногда остается тем же, а срок действия изменяется. Поэтому второй метод тоже неоднозначен. Следовательно, иногда для построения правильной цепочки сертификатов необходимо анализировать несколько путей подписания. Если хотя бы один путь позволяет построить цепочку до надежного сертификата и все ограничения, в частности на срок действия и другие, налагаемые расширениями X509v3, удовлетворены, то можно считать, что сертификат успешно проверен. OpenSSL поддерживает такой анализ нескольких путей подписания при построении цепочки сертификатов.
В большинстве случаев цепочка сертификатов заканчивается самоподписанным сертификатом корневого УЦ. Но, строго говоря, это необязательно. OpenSSL поддерживает цепочки, завершающиеся несамоподписанными сертификатами. Однако использование таких цепочек считается дурным тоном.

Как выпускаются сертификаты X.509
Процедура генерирования сертификата X.509 состоит из нескольких шагов:

1. заявитель (будущий владелец сертификата) генерирует закрытый и открытый ключи сертификата;
2. заявитель генерирует **запрос на подписание сертификата** (Certificate Signing Request – **CSR**). CSR содержит наименование субъекта, открытый ключ будущего сертификата, запрошенные заявителем расши#рения X509v3 и подпись CSR. CSR подписывается закрытым ключом сертификата;
3. заявитель отправляет CSR удостоверяющему центру для подписания;
4. УЦ проверяет личность заявителя;
5. УЦ изготавливает сертификат на основе информации, взятой из CSR. УЦ также добавляет в сертификат другую информацию, например наименование издателя, поля срока действия и расширения X509v3. Наконец, УЦ подписывает сертификат;
6. УЦ отправляет сертификат заявителю, который становится его владельцем, или держателем.

Отметим, что ни на каком этапе ни одна сторона не раскрывает другой свой закрытый ключ.
Для большинства выпускаемых сертификатов личность проверяется автоматически в онлайновом режиме, без каких-либо контактов между владельцами сайта и сотрудниками

УЦ. Что же тогда проверяется? Что заявитель действительно управляет сайтом и его доменным именем. Проверки автоматические и могут выглядеть следующим образом:

- разместить некий файл по конкретному URL-адресу, принадлежащему сайту;
- добавить DNS-запись в домен сайта;
- создать адрес электронной почты в домене сайта и получить на этот адрес пароль.

Именно поэтому HTTPS-сертификаты обычно называются сертификатами **с подтвержденным доменом** (Domain-Validated – **DV**).

Что именно удостоверяет такой DV-сертификат сайта? То, что браузер пользователя подключился к сайту с доменным именем, указанным в сертификате, например `www.openssl.org`, – не больше, не меньше. DV-сертификат не гарантирует, что сайт представляет какую-то конкретную организацию, например проект OpenSSL. Важно понимать, что безопасно только подключение. Сам же сайт может быть вовсе не безопасен.

Почему же DV-сертификаты все-таки полезны? Потому что они защищают пользователей браузера от атак с отравлением DNS и с человеком посередине. Пользователь может с высокой вероятностью предполагать, что действительно находится на сайте, который указан в адресной строке браузера, а не на поддельном сайте, имитирующем подлинный.

Существуют ли сертификаты, подтверждающие, что сайт представляет конкретную организацию, а не только доменное имя? Да, они называются сертификатами **с расширенным подтверждением** (Extended Validation – **EV**). При выпуске EV-сертификата УЦ выполняет гораздо больше проверок запроса на подписание (CSR). УЦ проверяет, что CSR поступил из организации, владеющей доменом, что название организации в поле `Subject` запроса совпадает с истинным названием, что организация существует и надлежащим образом зарегистрирована в государственном реестре организаций, что адрес и телефон организации реальны, что человек, запросивший сертификат, действует от имени организации, и т. д. EV-сертификаты выпускаются только для юридических лиц.

Об EV-статусе выпущенного сертификата свидетельствует расширение X509v3. К сожалению, на данный момент не существует стандартизованного расширения X509v3 для этой цели; разные УЦ используют разные расширения. Поэтому браузер должен поддерживать несколько расширений, указывающих на EV-статус.

Существуют также сертификаты **с подтверждением юридического лица** (Organization Validation – **OV**) и **с подтверждением физического лица** (Individual Validation – **IV**). Они занимают промежуточное положение между DV и EV-сертификатами. При выпуске таких сертификатов УЦ проверяет подлинность юридического или физического лица, а выпущенный сертификат содержит название владельца в поле `Subject`. Отличие от EV-сертификатов состоит в том, что при обработке CSR выполняется меньше проверок.

Мы уже несколько раз упоминали расширения X509v3. Поговорим о них подробнее.

Что такое расширения X509v3?

Расширения X509v3 – это дополнительные поля, которые можно включить в сертификат X.509. Расширения могут налагать ограничения на использование сертификата или предоставлять дополнительную информацию о нем. Для примера рассмотрим расширения X509v3 в сертификате сайта www.openssl.org:

- **использование ключа и расширенное использование ключа**: ограничивают использование сертификата определенными целями. Учитывать эти ограничения или нет – дело программы, проверяющей сертификат. Если проверяющая программа не понимает или не придает значения некоторым ограничениям X509v3, то они игнорируются;
- **базовое ограничение CA:FALSE**: говорит, что этот сертификат нельзя использовать для выпуска других сертификатов, т. е. проверяющая программа не должна рассматривать его как сертификат промежуточного или корневого УЦ;
- **идентификатор ключа субъекта и идентификатор ключа УЦ**: информационные расширения, помогающие искать сертификат издателя по выпущенному сертификату;

- **доступ к информации об УЦ**: информационное расширение, помогающее находить ресурсы УЦ, например OCSP-сервер или сертификаты УЦ, полезные для проверки текущего сертификата;
- **альтернативное имя субъекта**: информационное расширение, в котором перечисляются доменные имена сайтов, которым разрешено использовать этот сертификат. Помимо www.openssl.org, могут быть включены другие имена, например mail.openssl.org или просто openssl.org;
- **подписанные временные метки (SCT) предсертификатов в журналах прозрачности сертификатов (CT)**. CT – это средство мониторинга, помогающее обнаруживать выпуск поддельных сертификатов X.509.

Расширения X509v3 – необязательные части формата сертификата X.509, но благодаря им использовать сертификаты удобнее. Кроме того, некоторые программы, проверяющие сертификаты, например браузеры, могут ожидать присутствия определенных расширений и изменять результат проверки в зависимости от содержимого этих расширений или их отсутствия.
Сертификаты X.509 полезны, но не в отрыве от окружения. Сертификаты являются частью более крупной инфраструктуры открытых ключей.

Инфраструктура открытых ключей X.509
Инфраструктура открытых ключей (PKI) X.509 – это комбинация стандартов, алгоритмов, структур данных, программного и аппаратного обеспечения, организаций и процедур, необходимых для создания, хранения, передачи, использования, отзыва и других действий по управлению сертификатами X.509 и их ключами.
Звучит запутанно, но мы только что разобрали, как X.509 PKI работает во Всемирной паутине. УЦ выпускают сертификаты, которые используются сайтами и проверяются браузерами. Именно так миллионы пользователей веба ежедневно и автоматически проверяют подлинность сайтов. Некоторые сайты поддерживают аутентификацию клиентов с помощью клиентских сертификатов. В таких случаях не только сайт должен предъявить свой сертификат, но и браузер пользователя должен предъявить сертификат сайту, чтобы сайт мог сертифицировать, аутентифицировать и авторизовать пользователя.
X.509 PKI используется не только в вебе. Сертификаты X.509 применяются для передачи почтовых сообщений, для взаимодействия автоматизированных компьютерных систем, при подписании программ и т. д.
На этом теоретическая часть закончена. Перейдем к практической части и сгенерируем несколько сертификатов.

Генерирование самоподписанного сертификата
Чтобы сгенерировать сертификат, нужно сгенерировать пару ключей, запрос на подписание сертификата и, наконец, сам сертификат.
Мы будем использовать следующие подкоманды `openssl`: `genpkey`, `pkey`, `req` и `x509`. Они документированы на страницах руководства:

```
$ man openssl-genpkey
$ man openssl-pkey
$ man openssl-req
$ man openssl-x509
```

Ниже описана процедура генерирования самоподписанного сертификата.

1. Для начала сгенерируем пару ключей. На этот раз выберем тип ключа ED448:

```
$ openssl genpkey -algorithm ED448 -out root_keypair.pem
```

Команда ничего не напечатала, это значит, что ошибок не было.

2. Проверим созданный ключ:

```
$ openssl pkey -in root_keypair.pem -noout –text

ED448 Private-Key:
      priv:
            e2:62:21:f0:32:25:20:ca:84:f9:b8:4f:0a:9f:51:
            51:3b:68:d0:0d:3a:91:c9:68:38:b4:2f:d0:53:af:
            62:5a:06:9d:b0:f5:86:11:73:f5:be:39:9a:78:be:
            ec:a2:53:d8:91:ad:8b:e5:2e:e2:b3:a3
      pub:
            70:3c:22:d9:9f:f8:d6:76:e0:4f:46:e8:74:7b:5f:
            98:98:ee:90:49:af:07:ba:05:a4:3b:b3:2c:e3:20:
            1a:00:cf:11:5c:76:93:32:0a:91:14:98:fa:dd:83:
            7b:9c:00:46:c8:d3:df:67:23:ea:e1:80
```

3. Сгенерируем CSR-запрос.

```
$ openssl req \
      -new \
      -subj "/CN=Root CA" \
      -addext "basicConstraints=critical,CA:TRUE" \
      -key root_keypair.pem \
      -out root_csr.pem
```

Заметим, что мы добавили расширение CA:TRUE. Это необходимо для сертификатов УЦ.

4. Ничего не напечатано, и создан файл, root_csr.pem, содержащий наш CSR-запрос. Посмотрим, что в нем:

```
$ openssl req -in root_csr.pem -noout –text

Certificate Request:
      Data:
      Version: 1 (0x0)
      Subject: CN = Root CA
      Subject Public Key Info:
            Public Key Algorithm: ED448
                  ED448 Public-Key:
                  pub:
                        ... шестнадцатеричные байты ...
      Attributes:
            Requested Extensions:
                  X509v3 Basic Constraints: critical
                        CA:TRUE
      Signature Algorithm: ED448
      Signature Value:
            ... шестнадцатеричные байты …
```

5. Теперь сгенерируем самоподписанный сертификат, используя CSR запрос и пару ключей. Срок действия нового сертификата будет составлять 3650 дней, т. е. около 10 лет:

```
$ openssl x509 \
      -req \
      -in root_csr.pem \
      -copy_extensions copyall \
      -key root_keypair.pem \
      -days 3650 \
      -out root_cert.pem
```

Обратите внимание на параметр `-copy_extensions copyall`. Он нужен, потому что по умолчанию команда openssl `x509` не копирует расширения X509v3 из CSR-запроса в сертификат.

6. Ничего не напечатано, и создан файл, root_cert.pem, содержащий самоподписанный сертификат. Посмотрим, что в нем:

```
$ openssl x509 -in root_cert.pem -noout –text
```

```
Certificate:
      Data:
            Version: 3 (0x2)
            Serial Number:
                  ... шестнадцатеричные байты ...
            Signature Algorithm: ED448
            Issuer: CN = Root CA
            Validity
                  Not Before: Mar 27 22:25:17 2022 GMT
                  Not After : Mar 24 22:25:17 2032 GMT
            Subject: CN = Root CA
            Subject Public Key Info:
                  Public Key Algorithm: ED448
                        ED448 Public-Key:
                        pub:
                              ... шестнадцатеричные байты ...
            X509v3 extensions:
                  X509v3 Basic Constraints: critical
                        CA:TRUE
                  X509v3 Subject Key Identifier:
                        ... шестнадцатеричные байты ...
      Signature Algorithm: ED448
      Signature Value:
            ... шестнадцатеричные байты ...
```

Обратите внимание, что поля `Issuer` и `Subject` одинаковы. Это признак самоподписанного сертификата.

Мы успешно сгенерировали самоподписанный сертификат, который будем использовать в качестве сертификата корневого УЦ. Теперь сгенерируем два несамоподписанных сертификата.

Генерирование несамоподписанного сертификата

При генерировании самоподписанного сертификата мы воспользовались общим подходом, а не длинными комбинированными командами. Поэтому процедура генерирования самоподписанного сертификата будет очень похожа. Мы сгенерируем два несамоподписанных сертификата. Один будет использоваться как сертификат промежуточного УЦ, а второй – как листовый сертификат.

1. Сначала сгенерируем пару ключей для сертификата промежуточного УЦ:

```
$ openssl genpkey \
      -algorithm ED448 \
      -out intermediate_keypair.pem
```

2. Затем сгенерируем CSR-запрос:

```
$ openssl req \
      -new \
      -subj "/CN=Intermediate CA" \
      -addext "basicConstraints=critical,CA:TRUE" \
      -key intermediate_keypair.pem \
      -out intermediate_csr.pem
```

Обратите внимание, что поле `Subject` для сертификата промежуточного УЦ отличается от предыдущего.

3. Теперь выпустим сертификат промежуточного УЦ и подпишем его закрытым ключом корневого сертификата:

```
$ openssl x509 \
      -req \
      -in intermediate_csr.pem \
      -copy_extensions copyall \
      -CA root_cert.pem \
```

```
      -CAkey root_keypair.pem \
      -days 3650 \
      -out intermediate_cert.pem
```

Здесь мы задали флаги `-CA` и `-CAkey` вместо `-key`. Флаг `-CA` нужен для копирования поля `Subject` из сертификата УЦ в поле `Issuer` выпущенного сертификата, а также для заимствования другой необходимой информации из сертификата выпускающего УЦ, например идентификатора ключа УЦ, если используется соответствующее расширение X509v3.

4. Посмотрим на выпущенный сертификат промежуточного УЦ:

```
$ openssl x509 -in intermediate_cert.pem -noout -text

Certificate:
      Data:
            Version: 1 (0x0)
            Serial Number:
                  ... шестнадцатеричные байты ...
            Signature Algorithm: ED448
            Issuer: CN = Root CA
            Validity
                  Not Before: Mar 27 23:11:35 2022 GMT
                  Not After: Mar 24 23:11:35 2032 GMT
            Subject: CN = Intermediate CA
            Subject Public Key Info:
                  Public Key Algorithm: ED448
                        ED448 Public-Key:
                        pub:
                        ... шестнадцатеричные байты ...
            X509v3 extensions:
                  X509v3 Basic Constraints: critical
                        CA:TRUE
                  X509v3 Subject Key Identifier:
                        ... шестнадцатеричные байты ...
                  X509v3 Authority Key Identifier:
                        ... шестнадцатеричные байты ...
      Signature Algorithm: ED448
      Signature Value:
            ... шестнадцатеричные байты ...
```

Обратите внимание, что поля `Issuer` и `Subject` различны. Это значит, что сертификат несамоподписанный.

5. Теперь выпустим листовый сертификат и подпишем его закрытым ключом сертификата промежуточного УЦ. Делается это так же, как при выпуске сертификата промежуточного УЦ.

```
$ openssl genpkey -algorithm ED448 -out leaf_keypair.pem
$ openssl req \
      -new \
      -subj "/CN=Leaf" \
      -addext "basicConstraints=critical,CA:FALSE" \
      -key leaf_keypair.pem \
      -out leaf_csr.pem
$ openssl x509 \
      -req \
      -in leaf_csr.pem \
      -copy_extensions copyall
      -CA intermediate_cert.pem \
      -CAkey intermediate_keypair.pem \
      -days 3650 \
      -out leaf_cert.pem
```

На этот раз мы задали `CA:FALSE`, а не `CA:TRUE`, потому что листовые сертификаты не должны использоваться для выпуска других сертификатов.

6. Посмотрим на сгенерированный листовый сертификат:

```
$ openssl x509 -in leaf_cert.pem -noout –text

Certificate:
      Data:
            Version: 1 (0x0)
            Serial Number:
                  ... шестнадцатеричные байты ...
            Signature Algorithm: ED448
            Issuer: CN = Intermediate CA
            Validity
                  Not Before: Mar 27 23:34:19 2022 GMT
                  Not After: Mar 24 23:34:19 2032 GMT
            Subject: CN = Leaf
            Subject Public Key Info:
                  Public Key Algorithm: ED448
                        ED448 Public-Key:
                        pub:
                              ... шестнадцатеричные байты ...
            X509v3 extensions:
                  X509v3 Basic Constraints: critical
                        CA:FALSE
                  X509v3 Subject Key Identifier:
                        ... шестнадцатеричные байты ...
                  X509v3 Authority Key Identifier:
                        ... шестнадцатеричные байты ...
      Signature Algorithm: ED448
      Signature Value:
            ... шестнадцатеричные байты ...
```

Теперь проверим листовый сертификат с помощью двух других.

Проверка сертификата в командной строке

Проверить сертификат позволяет команда `openssl verify`, документированная на странице руководства `openssl-verify`.

Проверим только что сгенерированный листовый сертификат. Будем считать, что наш сертификат корневого УЦ надежный. Сертификат промежуточного УЦ будем считать ненадежным, но он позволит построить цепочку сертификатов.

Вот как проверяется листовый сертификат в командной строке:

```
$ openssl verify \
      -verbose \
      -show_chain \
      -trusted root_cert.pem \
      -untrusted intermediate_cert.pem \
      leaf_cert.pem

leaf_cert.pem: OK
Chain:
depth=0: CN = Leaf (untrusted)
depth=1: CN = Intermediate CA (untrusted)
depth=2: CN = Root CA
```

Обратите внимание на флаги `-trusted` и `-untrusted`. Флаг `-trusted` задает файл, содержащий один или несколько надежных сертификатов. Чтобы проверить сертификат, `openssl verify` должна построить цепочку от проверяемого сертификата до надежного. Флаг `-untrusted` задает файл, содержащий один или несколько ненадежных сертификатов. Ненадежные сертификаты олезны в качестве промежуточных в цепочке. Оба флага можно использовать несколько раз для задания нескольких файлов.