Балансировка нагрузки при управлении ключами

30.10.2024. Подготовил Фёдоров Илья. Группа Б01-107

Оглавление

1. Введение в управление ключами

Управление ключами является краеугольным камнем информационной безопасности, обеспечивая защиту конфиденциальности, целостности и доступности данных. В современных цифровых системах ключи выступают в роли средств аутентификации, шифрования и дешифрования информации, играя важную роль в защите данных от несанкционированного доступа.

1.1 Что такое ключ?

Ключ - достаточно широкое понятие в теории информации. Ключами могут являться:

- Ключ товара интернет магазина, что эквивалентно id товара в базе данных
- Сессионный ключ, созданный в процессе TLS рукопожатия
- API key - уникальный идентификатор, который используется для аутентификации пользователя при взаимодействии с сайтом посредством API
- Сессионный идентификатор, чаще всего помещённый в cookie браузера, позволяющий связать пользователя с данными сессии

1.2 Что такое управление ключами?

Управление ключами — это процесс создания, распространения, хранения, обмена и уничтожения криптографических ключей в рамках системы безопасности. Этот процесс охватывает весь жизненный цикл ключа, гарантирующий, что каждое звено в цепочке передачи данных остается защищенным и надежным. Эффективное управление ключами позволяет организациям уверенно шифровать данные,

сохранять их конфиденциальность при передаче и хранении, а также предотвращать злоупотребление доступом.

1.3 Почему управление ключами критически важно для информационной безопасности?

В условиях увеличения объемов данных и использования облачных технологий, надежная защита информации требует строгого контроля над ключами, которые используются для ее шифрования. Без надлежащего управления ключами:

- Угрозы безопасности возрастают, так как злоумышленники могут получить доступ к данным через утерянные или угнанные ключи.
- Риски потери данных возрастают при утере шифровальных ключей, что делает данные навсегда недоступными.
- Комплаенс с нормативными требованиями нарушается, если ключи не защищены должным образом, что может привести к штрафам и ущербу репутации.

1.4 Основные принципы и методы управления ключами

Эффективное управление ключами основывается на следующих принципах:

- Секретность: Ключи должны быть защищены, их секретность сохраняется в течение всего жизненного цикла.
- Доступность: Ключи должны быть доступны для авторизованных пользователей и процессов, когда это необходимо.
- Управление жизненным циклом: Включает в себя генерацию, хранение, распределение, ротацию и уничтожение ключей.
- Сегрегация обязанностей: Разделение ответственности между различными сотрудниками или системами для предотвращения злоупотреблений.
- Мониторинг и аудит: Постоянное слежение за использованием ключей и проведение проверок для выявления и пресечения потенциальных угроз.

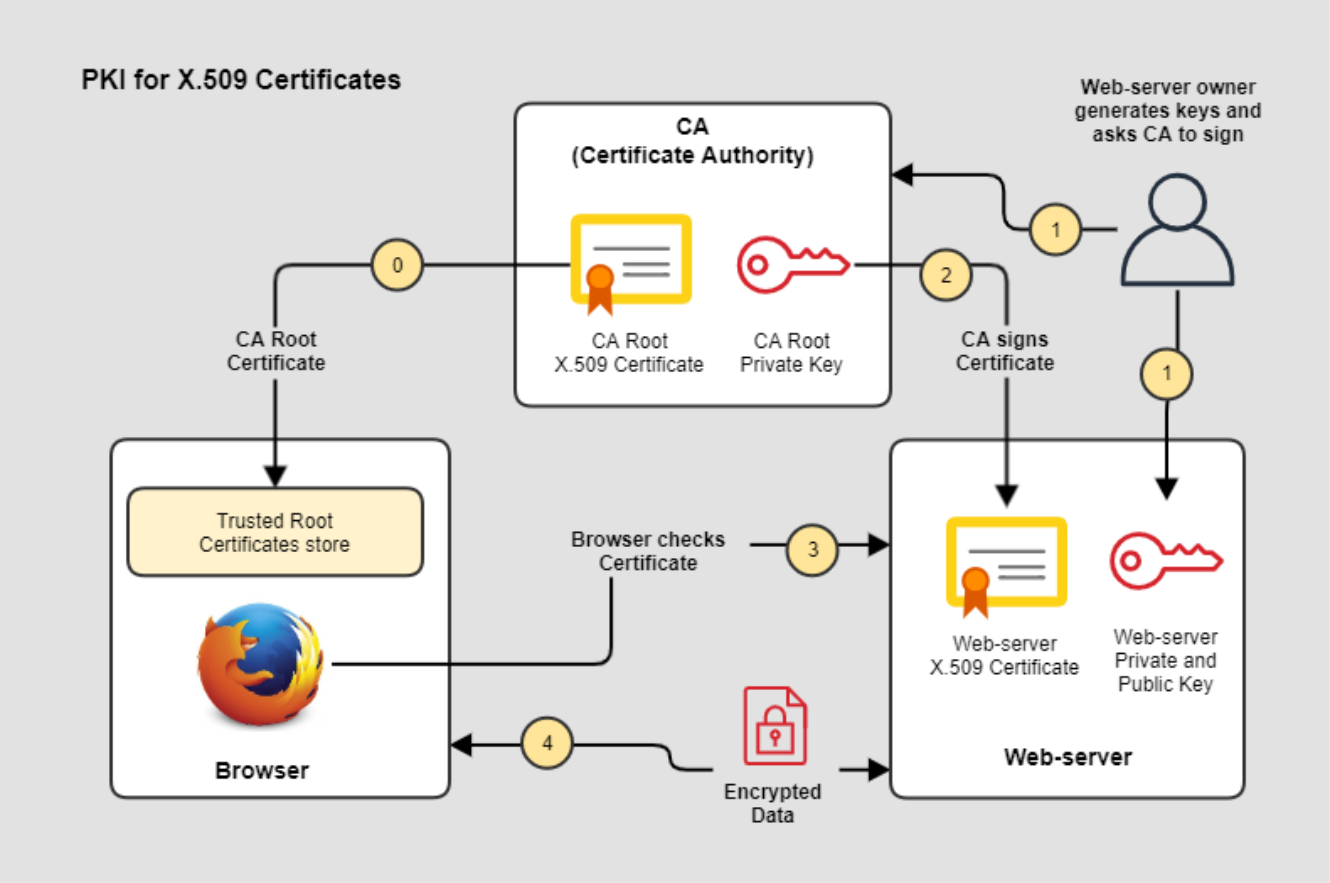
Методы управления ключами могут включать использование аппаратных модулей безопасности (HSM), программных решений для управления ключами, таких как PKI (Инфраструктура открытых ключей) и передовые практики хранения ключей в облаке.

2 Открытые стандарты и протоколы для управления ключами

PKI (Инфраструктура открытых ключей): PKI представляет собой систему, которая использует криптографию открытых ключей для обеспечения безопасности данных и аутентификации пользователей. Она поддерживает создание, управление, распространение, использование, хранение и отзыв цифровых сертификатов. Основные компоненты PKI включают сертификаты X.509, центры сертификации (CA) и центры регистрации (RA). Разберём эту систему поподробнее

PKI - Public Key Infrastructure

На схеме упращенный принцип действия PKI для организации HTTPS в сети Интернет:



PKI построена на принципе публичного и приватного ключа. Для нас важно понимать, что:

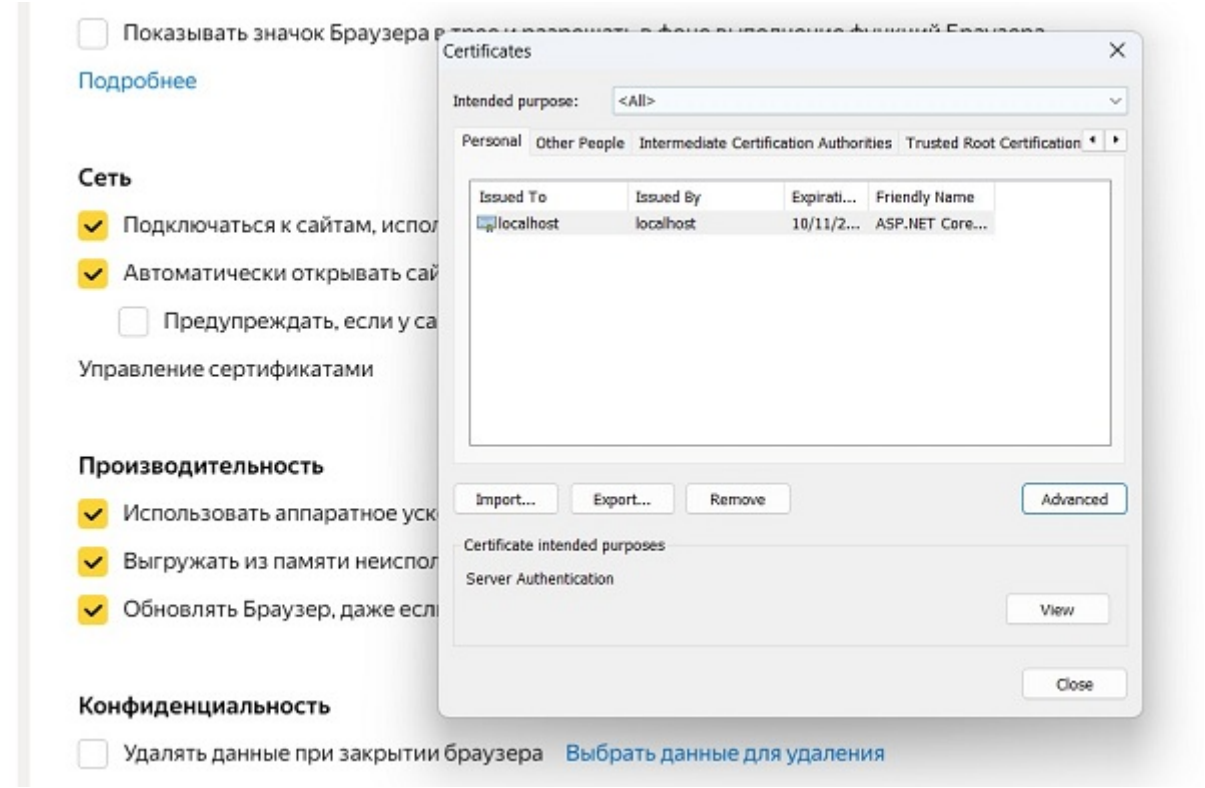
- Публичный ключ можно и нужно распространять. Приватный ключ должен храниться в секрете
- С помощью публичного ключа шифруют данные, которые впоследствии можно будет расшифровать только с помощью приватного ключа
- Публичный и закрытый ключ криптографически связаны

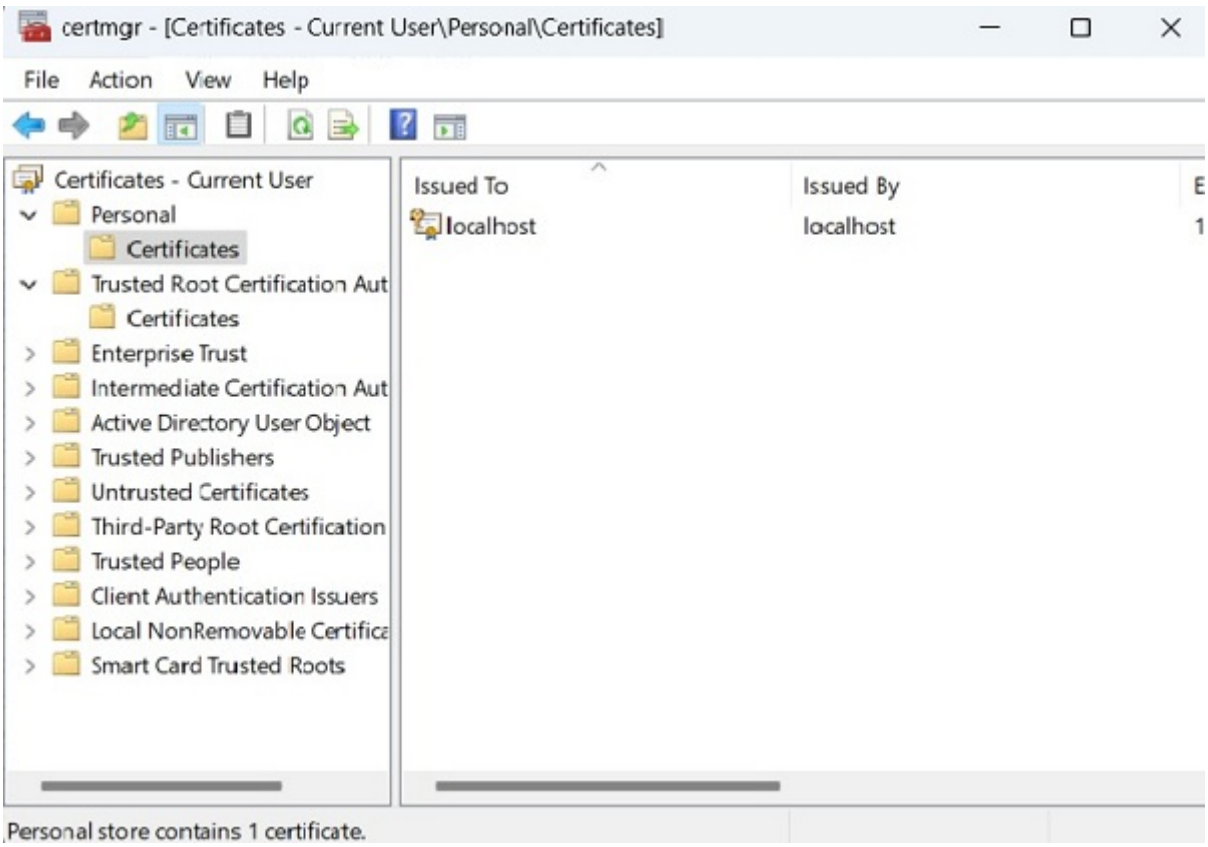
CA(Certificate Authority)

CA - удостоверяющий центр, организация, выдающая цифровые сертификаты. С CA можно взаимодействовать как в ручном, так и в автоматическом режиме. Для автоматического взаимодействия существует несколько протоколов:

- Automatic Certificate Management Environment (ACME)
- Simple Certificate Enrollment Protocol (SCEP)
- Enrollment over Secure Transport (EST)

У каждой операционной системы и каждого браузера хранится список таких серверов. Если у вас рабочий ноутбук, системный администратор может добавить сервер, для выдачи собственных сертификатов. Посмотреть список сертификатов в Яндекс браузере можно по адресу Настройки>Системные>Управление SSL сертификатами. В Windows - с помощью команды **certmgr.msc** в поисковой строке.





X.509 certificate

Что необходимо знать про сертификаты:

- Сертификаты, которые относятся к удостоверяющим центрам, называются корневыми (root certificates) или промежуточными (intermediate certificates) в зависимости от типа CA.
- Приватным ключом корневого CA можно подписывать как сертификаты промежуточных CA, так и сертификаты конечных точек (end-point). Обычно конечные сертификаты подписываются

приватными ключами промежуточных CA.

- Приватными ключами конечных точек нельзя подписывать другие сертификаты.
- Сертификаты выдаются в формате X.509:
    - Такой сертификат содержит структуру данных, которая состоит из идентификационных данных (Identity), дополнительных расширений (Extensions) и публичного ключа (Public Key) его владельца.
    - Сертификат подписывается корневым или промежуточным CA и его подпись связывает Identity владельца с его публичным ключом.

Certificate chain / certificate path validation

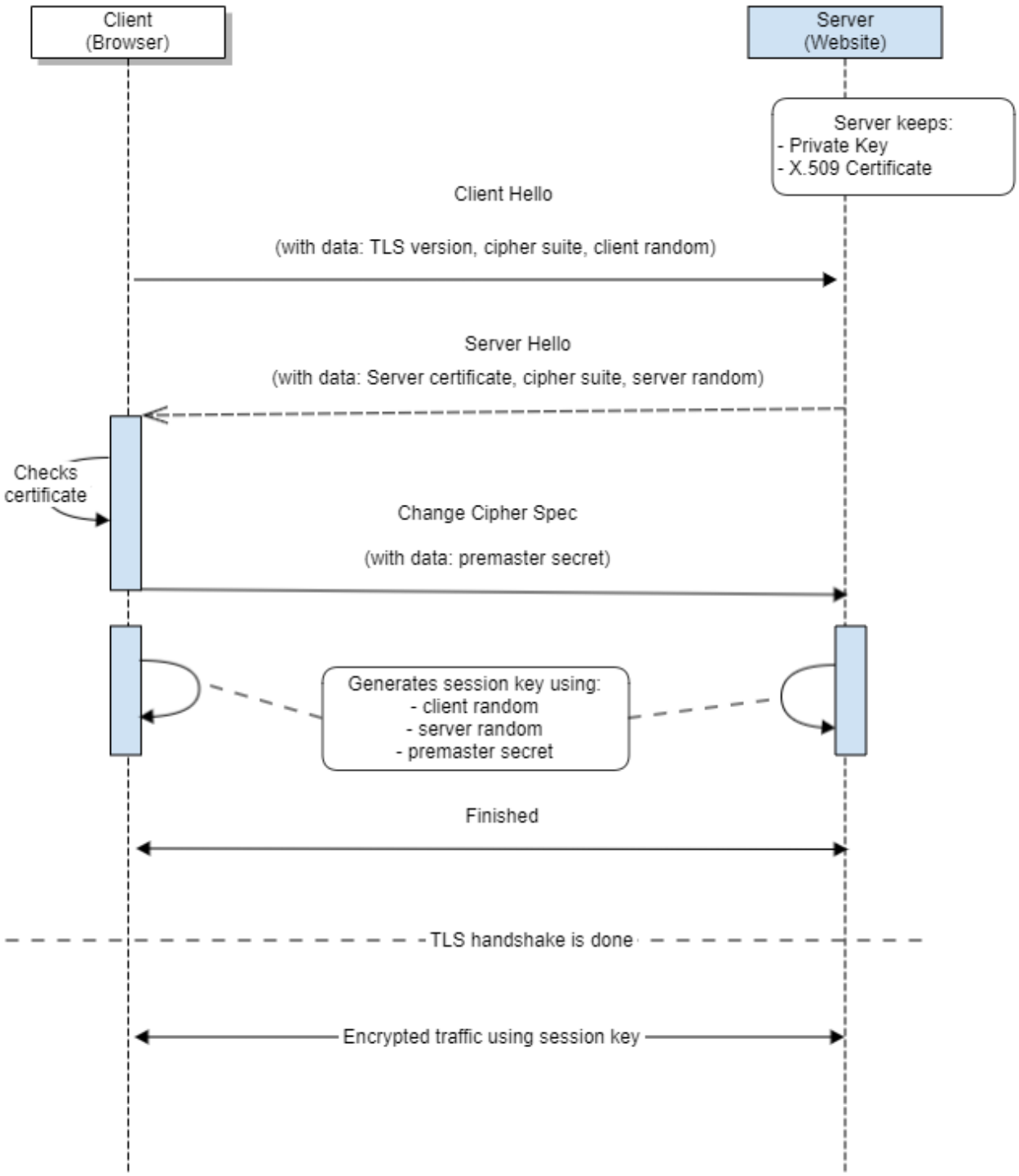
Корневой сертификата <--> Промежуточный сертификат <--> Конечный сертификат

Обычно конечный сертификат подписан приватным ключом промежуточного сертификата, промежуточный сертификат подписан приватным ключом корневого сертификата Корневой сертификат подписан собственным приватным ключом

Процесс проверки конечного сертификата, который получает браузер, называется Certificate path validation

TLS(Transport Layer Security) handshake

HTTPS использует протокол TLS, вот схема его действия



Важно иметь в виду, что после проверки сертифика сервера, браузер и сервер генерируют **сессионный ключ**, который будет использоваться для дальнейшего симметричного шифрования данных.

Создание локального самоподписанного сертификата с помощью утилиты MakeCert.exe

Итак, давайте создадим свой собственный сертификат

Генерирум командой в PS **New-SelfSignedCertificate -DnsName localhost:8080 -CertStoreLocation cert:\LocalMachine\My**

По-умолчанию генерируется самоподписанный сертификат со следующим параметрами:

- Криптографический алгоритм: RSA;
- Размер ключа: 2048 бит;
- Допустимые варианты использования ключа: Client Authentication и Server Authentication;
- Сертификат может использоваться для: Digital Signature, Key Encipherment ;
- Срок действия сертификата: 1 год.
- Криптопровадер: Microsoft Software Key Storage Provider

Экспортируем сертификат

1. Экспорт сертификата в файл:

$cert = Get-ChildItem Cert:\LocalMachine\My | Where-Object { $_.DnsNameList -contains "localhost" } Export-Certificate -Cert $cert -FilePath "C:\Users\Ваше_Имя_Пользователя\certificate.cer"

2. Экспорт приватного ключа в файл:

$password = ConvertTo-SecureString -String "ВашПароль" -Force -AsPlainText Export-PfxCertificate -Cert $cert -FilePath "C:\Users\Ваше_Имя_Пользователя\certificate.pfx" -Password $password

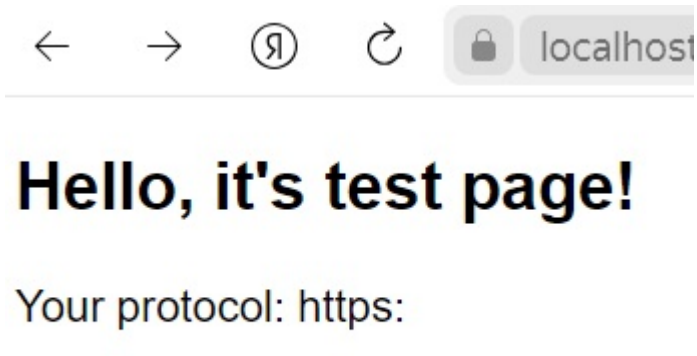
Добавляем сертификат в браузер

Настройки>Системные>Управление SSL сертификатами Выбираем наш .cer файл.

Создаём https сервер на express js

```
const express = require('express');
const path = require("path");
const https = require('https');
const fs = require('fs');

const app = express();

app.use(express.static(path.join(__dirname, "../dist")));

app.get("*", (req, res) => {
  res.sendFile(path.join(__dirname, "../dist", "index.html"));
});

const options = {
  pfx: fs.readFileSync(path.join(__dirname,'your path to .pfx file')),
  passphrase: "your Thumbprint",
};

https.createServer(options, app).listen(443, () => {
  console.log('HTTPS сервер запущен на порту 443');
});
```

Проверяем https соединение



Ура, нам удалось поднять сервер на https

Балансировка при управлении ключами, созданными в просессе TLS

К сожалению, я не обнаружил информации, как можно балансировать этими ключами

KMIP (Key Management Interoperability Protocol)

KMIP (Key Management Interoperability Protocol): KMIP — это стандартный протокол, разработанный для объединения управления ключами в различных системах и приложениях. KMIP обеспечивает единый формат для передачи данных между клиентами и серверами, что упрощает процесс интеграции и повышает совместимость различных решений по управлению ключами.

Управление сессиями

Добавляем сессии на наш сайт, теперь мы сможем идентифицировать пользователя с помощью cookie

```
const session = require("express-session");

app.use(
  session({
    secret: "your-secret-key",
    resave: false,
    saveUninitialized: true,
    cookie: { secure: true },
  })
);

app.get("/testSession", (req, res)=>{
  if (req.session.views) {
    req.session.views++;
    res.send(`Вы посетили эту страницу ${req.session.views} раз`);
  } else {
    req.session.views = 1;
    res.send('Добро пожаловать! Это ваш первый визит.');
  }
});
```

С официального сайта express.js, выясняем:

> Warning The default server-side session storage, MemoryStore, is purposely not designed for a production environment. It will leak memory under most conditions, does not scale past a single process, and is meant for debugging and developing.

То есть базовая реализация express-session не подразумевает оптимизаций и балансировки

Оптимизация и балансировка сессий в Express

Выбор подходящего хранилища для сессий

- Redis: Быстрое и надежное хранилище данных в памяти. После дополнительной настройки Redis предоставляет фичи:

1. Оптимизация производительности:
    - In-memory Storage: Все данные хранятся в оперативной памяти, что обеспечивает чрезвычайно быструю операцию чтения и записи.
    - Persistence Options: Redis поддерживает различные режимы сохранения данных на диск, к примеру, RDB (снимки состояния базы данных) и AOF (журналируемые операции), которые помогают восстановить данные после перезагрузки или сбоя.
    - Data Structures: Поддержка различных структур данных, таких как строки, списки, множества, отсортированные множества, хэш-таблицы и т.д., позволяет более эффективно обрабатывать данные.
2. Балансировка нагрузки:
    - Replication: Redis поддерживает мастер-слейв репликацию, где данные с главного сервера могут быть автоматически синхронизированы с подчинёнными серверами для обеспечения отказоустойчивости и балансировки нагрузки на чтение.
    - Sharding (Partitioning): Это метод распределения данных между несколькими узлами для увеличения объема данных, который может поддерживаться, а также для распределения нагрузки.
    - Redis Sentinel: Используется для управления экземплярами Redis. Sentinel может автоматически переключать трафик на резервные сервера в случае сбоя основного.
    - Redis Cluster: Обеспечивает горизонтальную масштабируемость и позволяет автоматически разбивать данные между несколькими узлами без необходимости внешнего координирующего сервиса.

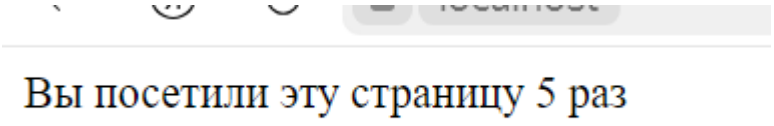
- Mongo DB
  Облачная no SQL база данных где даже настраивать ничего не надо, все оптимизации сделаны за вас. Вы только платите определённую сумму, в зависимости от подписки

Для своего сервера я выберу Redis, добавим на наш сервер с помощью следующего кода

```
const RedisStore = require("connect-redis").default;
const { createClient } = require("redis");
```

```
(async function () {
  const client = createClient();

  client.on("error", (err) => console.log("Redis Client Error", err));

  await client.connect();

  app.use(
    session({
      store: new RedisStore({
        client: client,
      }),
      secret: "your-secret-key",
      resave: false,
      saveUninitialized: false,
    })
  );
})();
```

Проверяем работоспособность



Вы посетили эту страницу 5 раз

Также существуют уже готовые облачные решения и решения с аудитом. Речь о которых пойдёт в следедующей главе

3. Современные инструменты и решения для эффективного управления ключами

HashiCorp Vault

HashiCorp Vault: HashiCorp Vault — это мощное и гибкое решение для управления секретами и ключами, которое обеспечивает безопасное хранение, доступ и контроль. Vault поддерживает такие функции, как временные токены, динамическое создание ключей, централизованное управление доступом и аудит. Это делает его идеальным выбором для организаций, стремящихся к обеспечению максимальной безопасности своих данных.

HashiCorp Vault изначально не предназначен для балансировки нагрузок, однако он очень хорошо подходит для безопасного управления секретами, включая API keys

На балансировку нагрузки в первую очередь влияет выбор Storage Backend - место, где хранить секреты. HashiCorp Vault предлагает несколько вариантов на выбор:

- HashiCorp Consul
- Integrated Storage
- Database (PostgreSQL, MySQL, MongoDB)

При нагрузочном тестировании оказалось, что Integrated Storage оказался медленнее, чем PostgreSQL

- 21 тысяч Integrated Storage
- 24.8 тысяч PostgreSQL

Для конкретики приведу виды оптимизаций, использующиеся в PostgreSQL:

1. Оптимизация запросов:
    - Планировщик запросов: Генерирует и оценивает разные планы выполнения для запросов. Планировщик использует статистику базы данных для выбора оптимального плана.
    - Индексы: Поддержка разных типов индексов (B-tree, Hash, GiST, GIN, и др.) для ускорения доступа к данным.
    - Параллельное выполнение: PostgreSQL может параллельно выполнять некоторые части запросов, улучшая производительность для больших объемов данных.
    - Кеш данных и планов запросов: Включает буферное кеширование и хранение ранее выполненных планов запросов для ускорения повторных обращений.
2. Балансировка нагрузки:
    - Репликация: Поддерживает стриминговую репликацию для создания резервных копий, балансировки нагрузки на чтение и обеспечения высокой доступности.
    - Шардинг: В то время как встроенной поддержки шардинга в PostgreSQL нет, его можно реализовать с помощью расширений и сторонних инструментов (например, Citus).
    - Кластеризация: Позволяет распределять нагрузку между несколькими серверами и узлами.
3. Автоматизация и управление ресурсами:
    - Автоматический вакуум: Очистка и реорганизация неиспользуемых данных в таблицах для поддержания производительности.
    - Настройка параметров конфигурации: PostgreSQL предоставляет множество параметров, которые можно настроить для улучшения производительности, таких как sharedbuffers, workmem, maintenanceworkmem и др.
4. Продвинутые механизмы хранения:
    - Таблицы BRIN и партицирование: Для эффективной работы с большими таблицами и улучшения производительности запросов.
    - Поддержка JSON и других форматов: Возможность работы с полуструктурированными данными и индексами для ускорения поиска по JSON.
5. Инструменты мониторинга:

- pgstatstatements: Расширение для отслеживания производительности запросов и выявления "тяжелых" запросов, вызывающих нагрузку на систему.
- EXPLAIN и EXPLAIN ANALYZE: Команды для анализа плана выполнения запросов и поиска узких мест.

AWS KMS

AWS KMS (Key Management Service): AWS KMS — это управляемый сервис AWS, обеспечивающий создание и контроль ключей шифрования, используемых для защиты данных в различных сервисах AWS. AWS KMS поддерживает функции генерации ключей, управления доступом, аудита и мониторинга, что облегчает управление ключами и повышает безопасность данных в облачной инфраструктуре.

Azure Key Vault

Azure Key Vault: Azure Key Vault — это облачное решение от Microsoft, предназначенное для управления ключами, секретами и сертификатами. Key Vault упрощает управление ключами шифрования и предоставляет механизмы для их безопасного хранения, а также встроенные функции управления доступом и аудита.

3.1 Влияние облачных технологий на управление ключами и балансировку нагрузки

Управление ключами в облаке: Облачные технологии значительно упростили процесс создания, хранения и управления криптографическими ключами. Поставщики облачных услуг, такие как AWS, Microsoft Azure и Google Cloud, предлагают интегрированные решения для управления ключами, которые обеспечивают высокий уровень безопасности и удобства. Эти решения позволяют организациям централизовать управление ключами и автоматизировать процессы, связанные с их использованием.

4. Заключение

В ходе эссэ было рассмотрено множесто вариантов в ключей (в основном в web сегменте), такие как:

- ключ TLS при https
- сессионный идентификатор
- API keys

Мы узнали, какие существуют методы балансировки и управления ключами. Исходя из всего вышесказанного, можно сделать вывод, что основным критерием балансировки является выбор хранилища ключей будь то Redis или Mongo DB при управлении сессионными идентификаторами или HashiCorp Consul для API keys

Ссылки на источники

- Сам себе PKI: Теория на примере Let’s Encrypt. https://habr.com/ru/articles/671728/
- Инфраструктура открытых ключей https://ru.wikipedia.org/wiki/Инфраструктура_открытых_ключей
- Автостопом по HashiCorp Vault https://habr.com/ru/companies/jetinfosystems/articles/762194/
- HashiCorp Vault как центр сертификации (CA) / Vault PKI https://itdraft.ru/2020/12/02/hashicorp-vault-kak-czentr-sertifikaczii-ca-vault-pki/

- Как создать самоподписанный SSL сертификат в Windows? https://winitpro.ru/index.php/2015/12/28/kak-sozdat-samopodpisannyj-sertifikat-v-windows/
- express-session - https://expressjs.com/en/resources/middleware/session.html