Аутентификация для веб-приложений

Теория, протоколы, стандарты.

Базовая терминология

- **Идентификация** — это процедура распознавания субъекта по его идентификатору

- *Идентификация выполняется при попытке войти в какую-либо систему (например, в операционную систему или в социальную сеть).*

Давайте перейдём к примерам, заодно разберемся, что такое **идентификатор**.

Базовая терминология

Когда нам звонят с неизвестного номера, что мы делаем? Правильно, спрашиваем “Кто это”, т.е. узнаём имя. Имя в данном случае и есть **идентификатор**, а ответ вашего собеседника — это будет **идентификация**.

Идентификатором может быть:

- номер телефона
- номер паспорта
- e-mail
- номер страницы в социальной сети и т.д.

Базовая терминология

- **Аутентификация** — процедура проверки подлинности, доказательство что субъект именно тот, за кого себя выдает.

Чтобы определить чью-то подлинность, можно воспользоваться тремя факторами:

- **Пароль** – то, что мы знаем (слово, PIN-код, код для замка, графический ключ)
- **Устройство** – то, что мы имеем (пластиковая карта, ключ от замка, USB-ключ)
- **Биометрика** – то, что является частью нас (отпечаток пальца, портрет, сетчатка глаза)

Базовая терминология

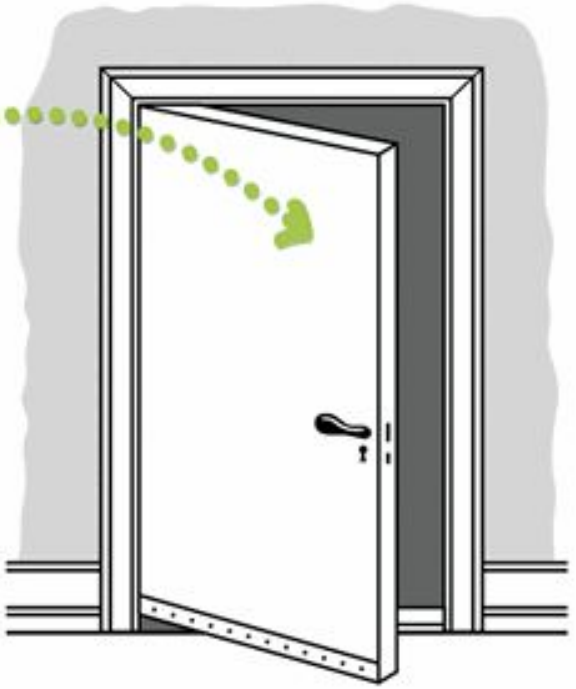
Когда определили ID, проверили подлинность, уже можно предоставить и доступ, то есть, выполнить авторизацию.

Авторизация – это предоставление доступа к какому-либо ресурсу.

Разберемся на примерах, что же это за загадочная авторизация:

- Открытие двери после проворачивания ключа в замке
- Доступ к электронной почте после ввода пароля
- Разблокировка смартфона после сканирования отпечатка пальца
- Выдача средств в банке после проверки паспорта и данных о

id1863017730



Идентификация

Определение

Кто там?

Аутентификация

Проверка

Чем докажешь? =)

Авторизация

Доступ

Открываю!

Примеры механизмов аутентификации.

Сказка «Волк и семеро козлят» является идеальным примером для демонстрации.

Здесь козлята выступают в роли системы безопасности, идентифицируя каждого, кто подходит к двери. В качестве данных для аутентификации выступает биометрия – тонкий голосок мамы-козы. И если в первый раз волк не смог пройти аутентификацию (его выдал грубый голос), то со второй попытки (после того как ему перековали горло, и он запел тонким голоском) он аутентифицировался как мама-коза и козлята «авторизовали» его в свою избу.

Несмотря на то, что сказка закончилась благополучно, доступ к козлятам был получен неправомерно. Волку удалось обмануть процессы идентификации и аутентификации и тем самым пройти авторизацию.

Многофакторная аутентификация

Многофакторная аутентификация представляет собой метод, при котором пользователю для доступа к учетной записи необходимо двумя различными факторами доказать, что именно он владелец учетной записи или что именно он осуществляет вход.

Среди видов многофакторной аутентификации наиболее распространена двухфакторная аутентификация (2FA) – метод, при котором пользователю для получения доступа необходимо предоставить два разных типа аутентификационных данных (см. слайд 4).

Многофакторная аутентификация

Доступ к ресурсам через ввод логина и пароля, является однофакторной аутентификацией, поскольку для входа используется только один тип аутентификационных данных — известный пользователю пароль.

Важно не путать «факторы» аутентификации и этапность.

Двуфакторная аутентификация, это когда нужен, например, пароль + отпечаток пальца.

Что же такое тогда многоэтапная аутентификация?

Однофакторная двухэтапная аутентификация

Многоэтапная проверка, добавляет дополнительный уровень безопасности к уже существующей модели логин/пароль. Теперь, для входа в систему, каждый пользователь должен будет ввести уникальный код, который будет сгенерирован для него специальным приложением или отправлен, например, через SMS.

Выбор технологии аутентификации

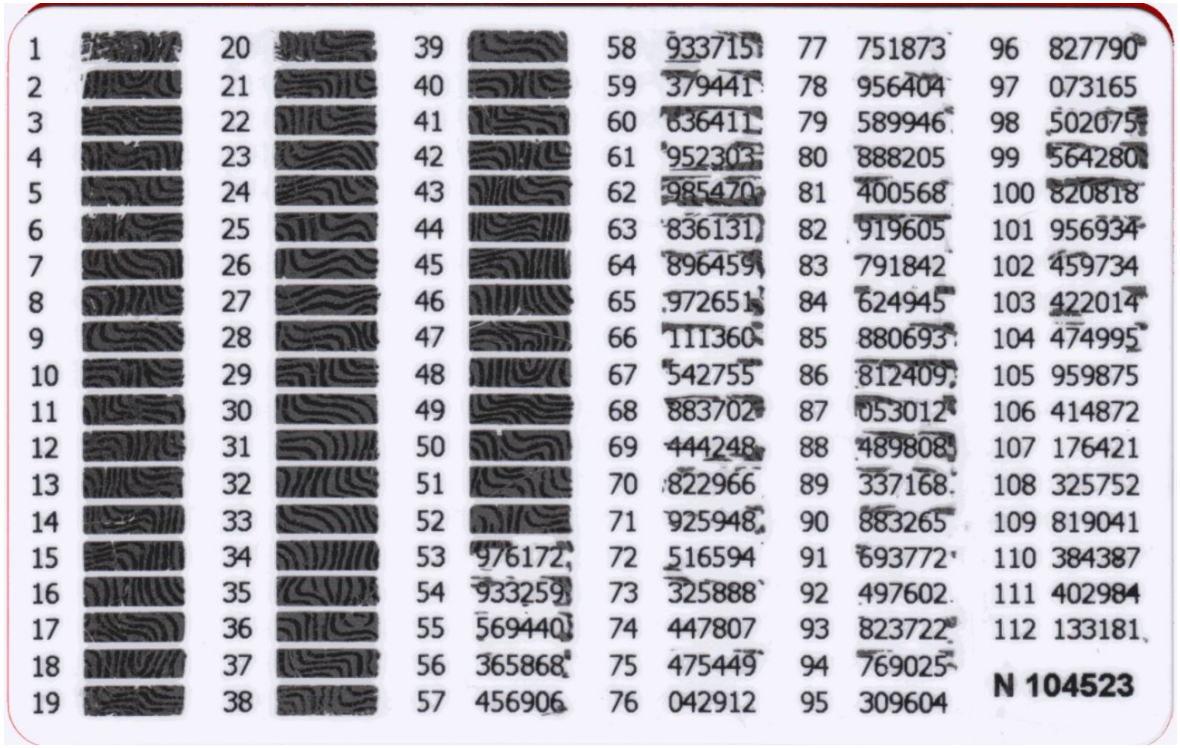
Выбирая для системы тот или иной фактор или способ аутентификации, необходимо, прежде всего, отталкиваться от требуемой степени защищенности, стоимости построения системы, обеспечения мобильности субъекта.

Выбор технологии аутентификации

- Требуется выполнить аутентификацию для доступа к системе, где кража, взлом, разглашение конфиденциальных сведений **не будут иметь значительных последствий**
- Рекомендуется минимальное требование — использование многоразовых паролей
- Пример: Регистрация на портале кулинарных рецептов.

Выбор технологии аутентификации

- Требуется выполнить аутентификацию для доступа к системе, где кража, взлом, разглашение конфиденциальных сведений причинят **небольшой ущерб**
- Рекомендуется минимальное требование — использование [одноразовых паролей]
- Пример: Проведение банковских операций.



Выбор технологии аутентификации

- Требуется выполнить аутентификацию для доступа к системе, причём кража, взлом, разглашение конфиденциальных сведений причинят **значительный ущерб**
- Рекомендуется минимальное требование — использование многофакторной аутентификации
- Пример: Проведение межбанковских операций на бирже.

Security

✓ Two-Factor Authentication (2FA) ✓ Identity Verification ✓ Anti-Phishing Code ✗ Withdrawal Whitelist

Two-Factor Authentication (2FA)

Security Key

Protect your account with a security key (e.g. Yubikey).

✗ Unset

Enable

Google Authenticator (Recommended)

Protect your account and transactions.
Having trouble?

✗ Unset

Enable

Phone Number Verification

Protect your account and transactions.

✓ [redacted]

Change Remove

Email Address Verification

Protect your account and transactions.

✓ [redacted]

Change Remove

Ближе к практике

Аналогичные термины применяются и в компьютерных системах, где традиционно под *идентификацией* понимают получение вашей учетной записи (identity) по username или email; под *аутентификацией* — проверку, что вы знаете пароль от этой учетной записи, а под *авторизацией* — проверку вашей роли в системе и решение о предоставлении доступа к запрошенной странице или ресурсу.

Аутентификация по паролю

Этот метод основывается на том, что пользователь должен предоставить username и password для успешной идентификации и аутентификации в системе. Пара username/password задается пользователем при его регистрации в системе, при этом в качестве username может выступать адрес электронной почты пользователя.

Применительно к веб-приложениям, существует несколько стандартных протоколов для аутентификации по паролю, которые мы сейчас с вами и рассмотрим.

HTTP authentication

- Этот протокол, описанный в стандартах HTTP 1.0/1.1, существует очень давно и до сих пор активно применяется в корпоративной среде. Применительно к веб-сайтам работает следующим образом:

1. Сервер, при обращении неавторизованного клиента к защищенному ресурсу, отсылает HTTP статус “401 Unauthorized” и добавляет заголовок “WWW-Authenticate” с указанием схемы и параметров аутентификации.

HTTP authentication

2. Браузер, при получении такого ответа, автоматически показывает диалог ввода username и password. Пользователь вводит детали своей учетной записи.

3. Во всех последующих запросах к этому веб-сайту браузер автоматически добавляет HTTP заголовок “Authorization”, в котором передаются данные пользователя для аутентификации сервером.

4. Сервер аутентифицирует пользователя по данным из этого заголовка. Решение о предоставлении доступа (авторизация) производится отдельно на основании роли пользователя, ACL или других данных учетной записи.

HTTP authentication schemas

Весь процесс стандартизирован и хорошо поддерживается всеми браузерами и веб-серверами. Существует несколько схем аутентификации, отличающихся по уровню безопасности:

Basic — наиболее простая схема, при которой username и password пользователя передаются в заголовке Authorization в незашифрованном виде (base64-encoded). Однако при использовании HTTPS (HTTP over SSL) протокола, является относительно безопасной.

HTTP authentication schemas

HTTP authentication schemas

Digest — challenge-response-схема, при которой сервер посылает уникальное значение nonce, а браузер передает MD5 хэш пароля пользователя, вычисленный с использованием указанного nonce. Более безопасная альтернативв Basic схемы при незащищенных соединениях.

NTLM (известная как Windows authentication) — также основана на challenge-response подходе, при котором пароль не передается в чистом виде. Эта схема не является стандартом HTTP, но поддерживается большинством браузеров и веб-серверов. Преимущественно используется для аутентификации пользователей Windows Active Directory в веб-приложениях.

HTTP authentication schemas

Negotiate — еще одна схема из семейства Windows authentication, которая позволяет клиенту выбрать между NTLM и Kerberos аутентификацией. **Kerberos** — более безопасный протокол, основанный на принципе Single Sign-On*. Однако он может функционировать, только если и клиент, и сервер находятся и являются частью одного и того же домена Windows.

Стоит отметить, что при использовании HTTP-аутентификации у пользователя нет стандартной возможности выйти из веб-приложения, кроме как закрыть все окна браузера.

Single Sign-On

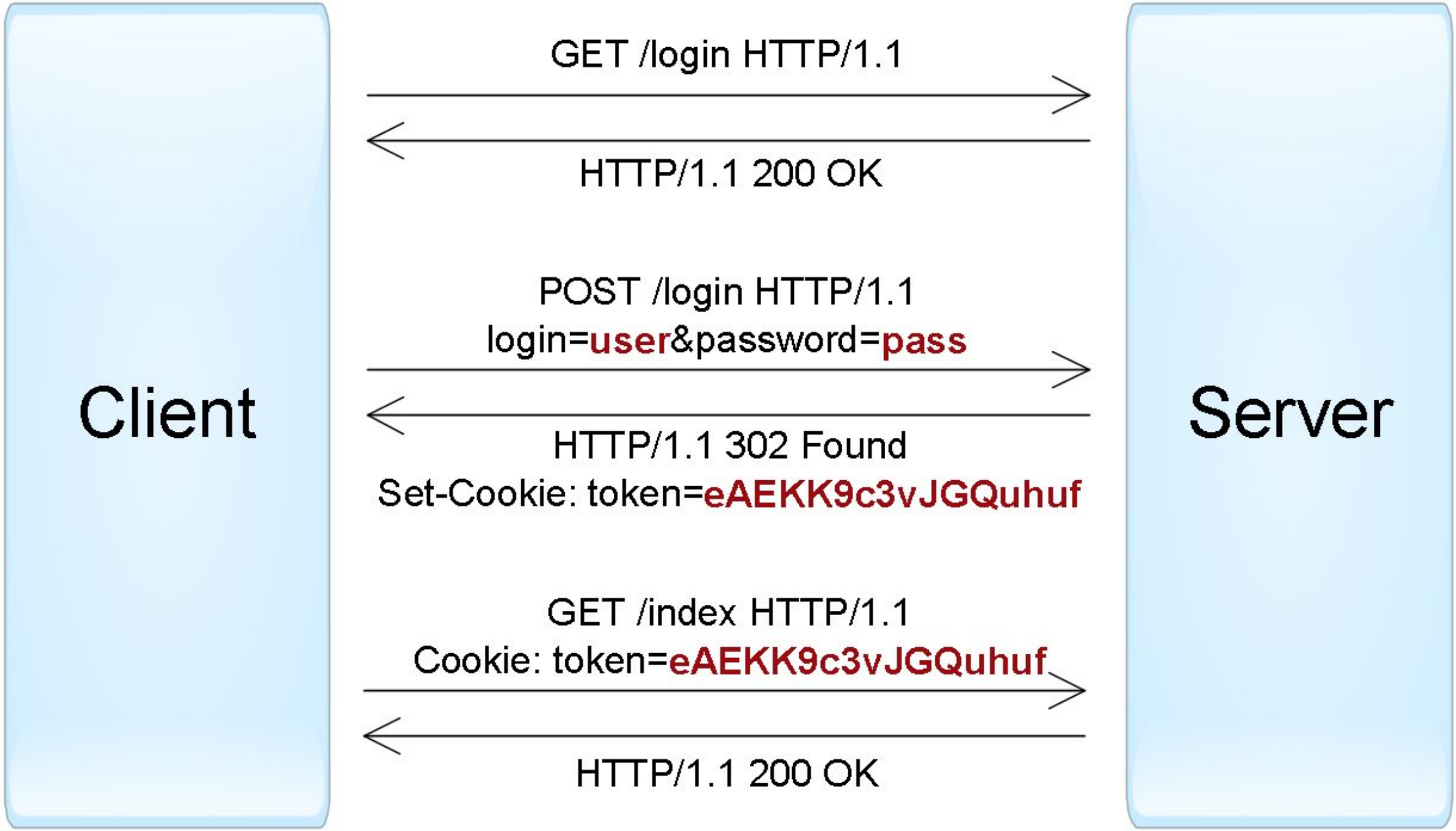
Технология единого входа — технология, при использовании которой пользователь переходит из одного раздела портала в другой, либо из одной системы в другую, не связанную с первой системой, без повторной аутентификации.

Делится на два вида:

- Корпоративный (Enterprise) SSO, подразумевающий установку агента на рабочие станции пользователей
- Web SSO, отдельный провайдер предоставляющий единую точку входа во все разделы портала(ов).

Forms authentication

Для этого протокола нет определенного стандарта, поэтому все его реализации специфичны для конкретных систем, а точнее, для модулей аутентификации фреймворков разработки.
Работает это по следующему принципу: в веб-приложение включается HTML-форма, в которую пользователь должен ввести свои username/password и отправить их на сервер через HTTP POST для аутентификации. В случае успеха веб-приложение создает session token, который обычно помещается в browser cookies. При последующих веб-запросах *session token* автоматически передается на сервер и позволяет приложению получить информацию о текущем пользователе для авторизации запроса.


Client
GET /login HTTP/1.1
HTTP/1.1 200 OK
POST /login HTTP/1.1
login=user&password=pass
HTTP/1.1 302 Found
Set-Cookie: token=eAEKK9c3vJGQuhuf
GET /index HTTP/1.1
Cookie: token=eAEKK9c3vJGQuhuf
HTTP/1.1 200 OK
Server

Forms authentication

Приложение может создать session token двумя способами:

1. Как **идентификатор** аутентифицированной **сессии** пользователя, которая хранится в памяти сервера или в базе данных. Сессия должна содержать всю необходимую информацию о пользователе для возможности авторизации его запросов.

Т.е. у пользователя нет никакой информации о том что содержится в его сессии, есть лишь идентификатор.

Forms authentication

2. Как зашифрованный и/или подписанный **объект**, содержащий данные о пользователе, а также период действия. Этот подход позволяет реализовать stateless-архитектуру сервера, однако требует механизма обновления сессионного токена по истечении срока действия. Несколько стандартных форматов таких токенов мы рассмотрим далее, после протоколов аутентификации, но, важно понимать что веб сервер может применять не стандартизированный подход для шифрования такого объекта.

Аутентификация по сертификатам

Аутентификации по паролю считается не очень надежным способом, так как пароль часто можно подобрать, а пользователи склонны использовать простые и одинаковые пароли в разных системах, либо записывать их на клочках бумаги. Если злоумышленник смог выяснить пароль, то пользователь зачастую об этом не узнает. Кроме того, разработчики приложений могут допустить ряд концептуальных ошибок, упрощающих взлом учетных записей. Поэтому, вместо простого пароля приходит на помощь сертификат с длинным ключом.

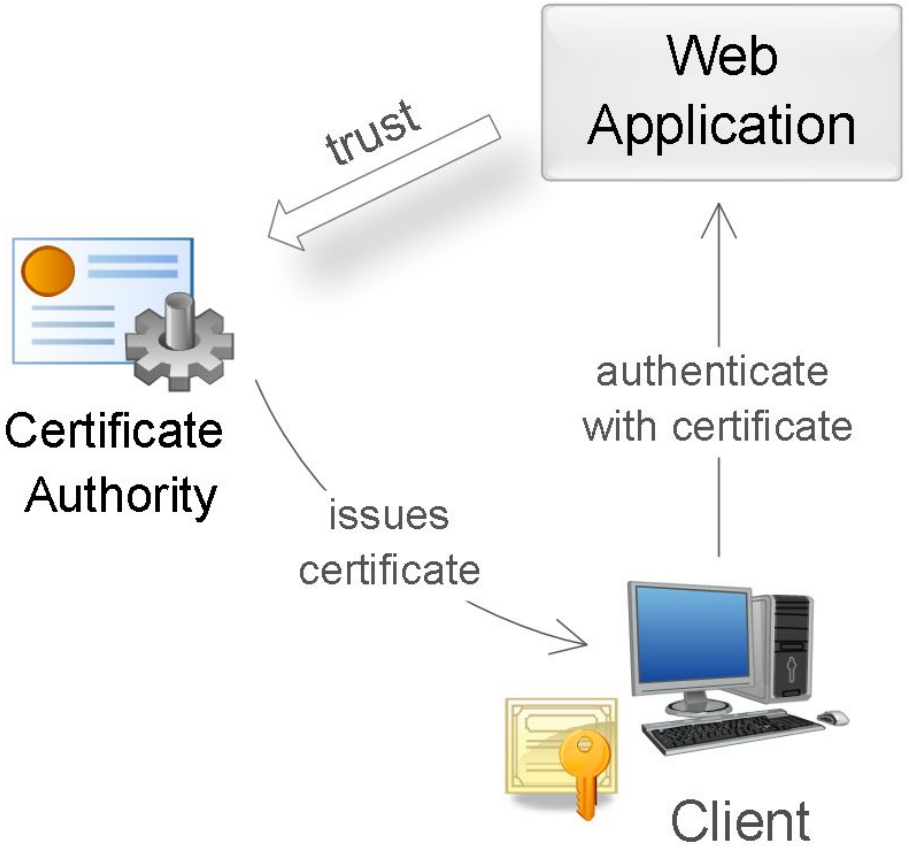
Аутентификация по сертификатам

Сертификат представляет собой набор атрибутов, идентифицирующих владельца, подписанный *certificate authority* (CA). CA выступает в роли посредника, который гарантирует подлинность сертификатов (по аналогии с ФМС, выпускающей паспорта). Также сертификат криптографически связан с закрытым ключом, который хранится у владельца сертификата и позволяет однозначно подтвердить факт владения сертификатом.

На стороне клиента сертификат вместе с закрытым ключом могут храниться в операционной системе, в браузере, в файле, на отдельном физическом устройстве (smart card, USB token). Обычно закрытый ключ дополнительно защищен паролем или PIN-кодом.

Аутентификация по сертификатам

В веб-приложениях традиционно используют сертификаты стандарта X.509. Аутентификация с помощью X.509-сертификата происходит в момент соединения с сервером и является частью протокола SSL/TLS. Этот механизм также хорошо поддерживается браузерами, которые позволяют пользователю выбрать и применить сертификат, если веб-сайт допускает такой способ аутентификации.



Аутентификация по сертификатам

Во время аутентификации сервер выполняет проверку сертификата на основании следующих правил:

1. Сертификат должен быть подписан доверенным certification authority (проверка цепочки сертификатов).
2. Сертификат должен быть действительным на текущую дату (проверка срока действия).
3. Сертификат не должен быть отозван соответствующим CA (проверка списков исключения).

Аутентификация по сертификатам

Использование сертификатов для аутентификации — куда более надежный способ, чем аутентификация посредством паролей. Это достигается созданием в процессе аутентификации цифровой подписи, наличие которой доказывает факт применения закрытого ключа в конкретной ситуации (non-repudiation). Однако трудности с распространением и поддержкой сертификатов делает такой способ аутентификации малодоступным в широких кругах.

Аутентификация по одноразовым паролям

Аутентификация по одноразовым паролям обычно применяется дополнительно к аутентификации по паролям для реализации *two-factor authentication* (2FA). В этой концепции пользователю необходимо предоставить данные двух типов для входа в систему: что-то, что он знает (например, пароль), и что-то, чем он владеет (например, устройство для генерации одноразовых паролей). Наличие двух факторов позволяет в значительной степени увеличить уровень безопасности, что м. б. востребовано для определенных видов веб-приложений.

Аутентификация по одноразовым паролям

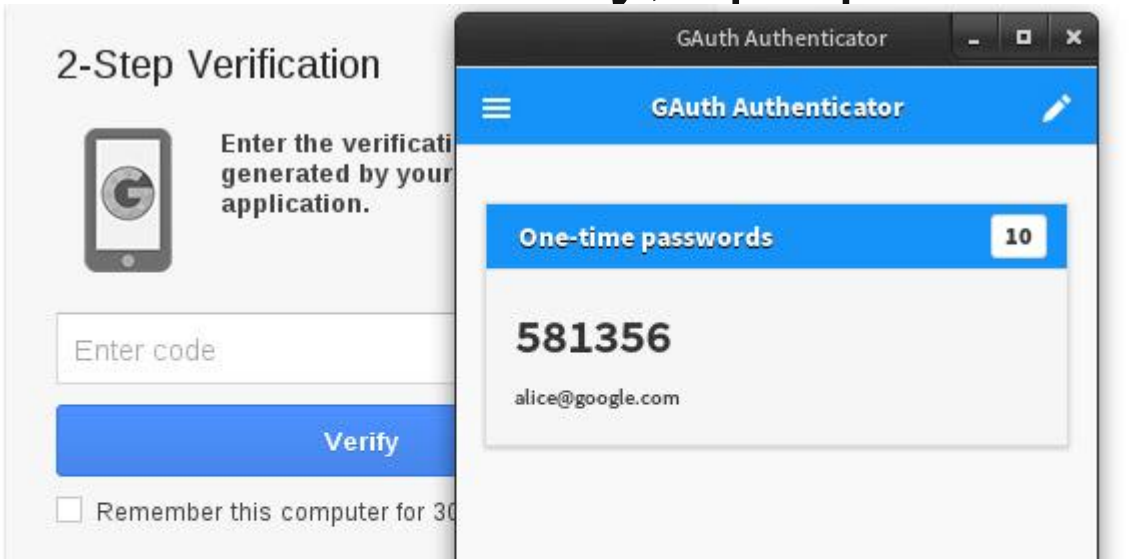
Другой популярный сценарий использования одноразовых паролей — дополнительная аутентификация пользователя во время выполнения важных действий: перевод денег, изменение настроек и т. п.
Существуют разные источники для создания одноразовых паролей. Наиболее популярные:

- Аппаратные либо программные токены.
- Случайно генерируемые коды (SMS)
- Scratch card (см. Предыдущие слайды)

Аутентификация по одноразовым паролям

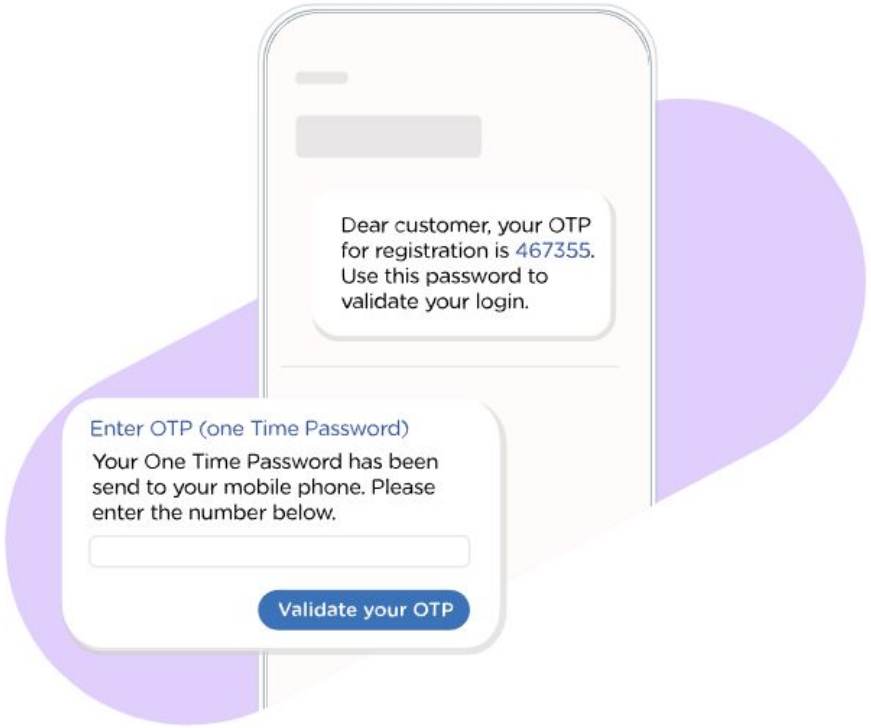
Аппаратные или программные токены, которые могут генерировать одноразовые пароли на основании секретного ключа, введенного в них, и текущего времени. Секретные ключи пользователей, являющиеся фактором владения, также хранятся на сервере, что позволяет выполнить проверку введенных одноразовых паролей. Пример аппаратной реализаций токенов — YubiKey; программной — пр le Au

Токен Security Key NFC by Yubico, ц...

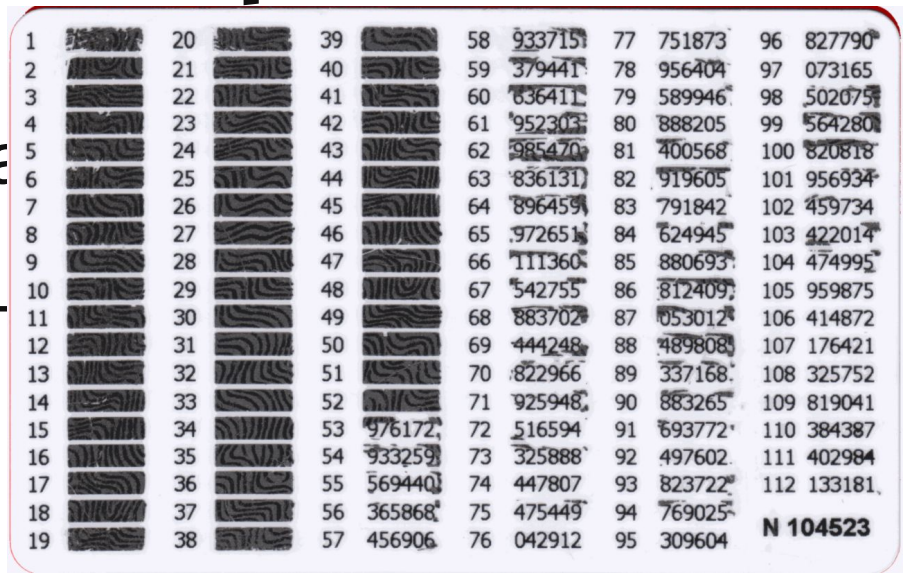


Аутентификация по одноразовым паролям

Случайно генерируемые коды, передаваемые пользователю через SMS или другой канал связи. В этой ситуации фактор владения — телефон пользователя (точнее — SIM-карта, привязанная к определенно



Аутентификация по одноразовым паролям



Распечатка или scratch card со списком зара
сформированных одноразовых паролей.
Для каждого нового входа в систему требуе
ввести новый одноразовый пароль.

В веб-приложениях такой механизм аутентификации часто реализуется посредством расширения forms authentication: после первичной аутентификации по паролю, создается сессия пользователя, однако в контексте этой сессии пользователь не имеет доступа к приложению до тех пор, пока он не выполнит дополнительную аутентификацию по одноразовому паролю.

Аутентификация по ключам доступа

Этот способ чаще всего используется для аутентификации устройств, сервисов или других приложений при обращении к веб-сервисам. Здесь в качестве секрета применяются ключи доступа (*access key, API key*) — длинные уникальные строки, содержащие произвольный набор символов, по сути заменяющие собой комбинацию username/password.

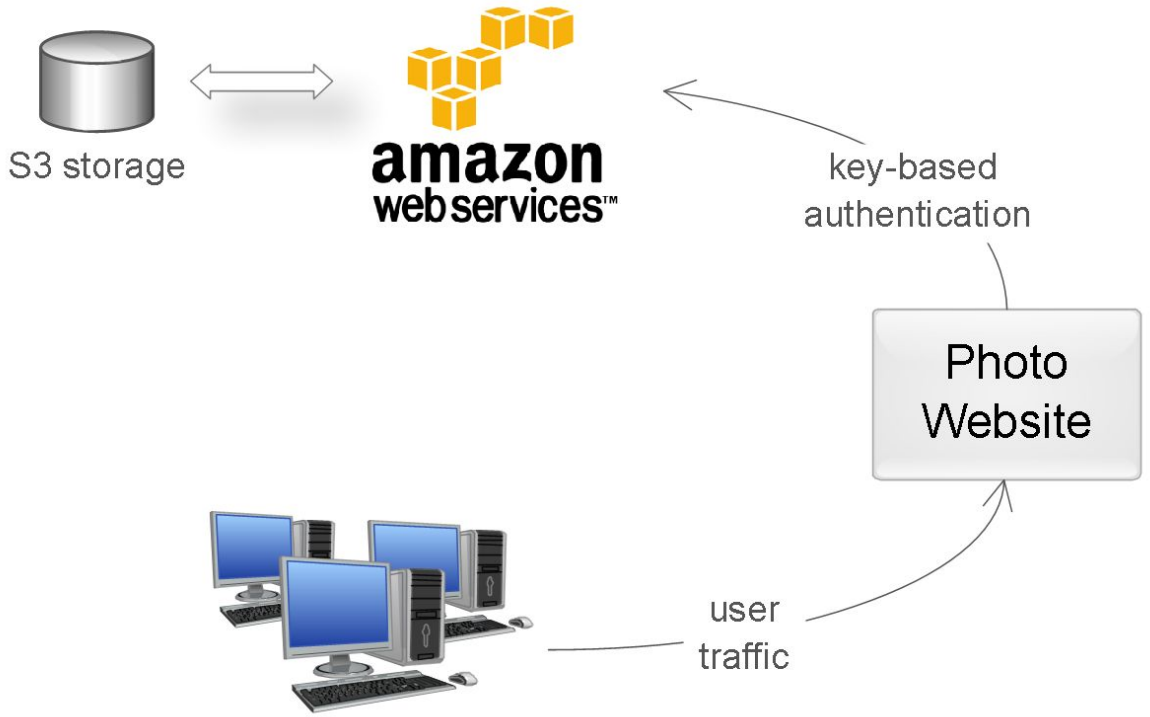
Чаще всего используется при взаимодействии **backend-to-backend.**

Аутентификация по ключам доступа

В большинстве случаев, сервер генерирует ключи доступа по запросу пользователей, которые далее сохраняют эти ключи в клиентских приложениях. При создании ключа также возможно ограничить срок действия и уровень доступа, который получит клиентское приложение при аутентификации с помощью этого ключа.

Аутентификация по ключам доступа

Хороший пример применения аутентификации по ключу — облако Amazon Web Services. Предположим, у пользователя есть веб-приложение, позволяющее загружать и просматривать фотографии, и он хочет использовать сервис Amazon S3 для хранения файлов. В таком случае, пользователь через консоль AWS может создать ключ, имеющий ограниченный доступ к облаку: только чтение/запись его файлов в Amazon S3. Этот ключ в результате можно применить для аутентификации веб-приложения в облаке AWS.

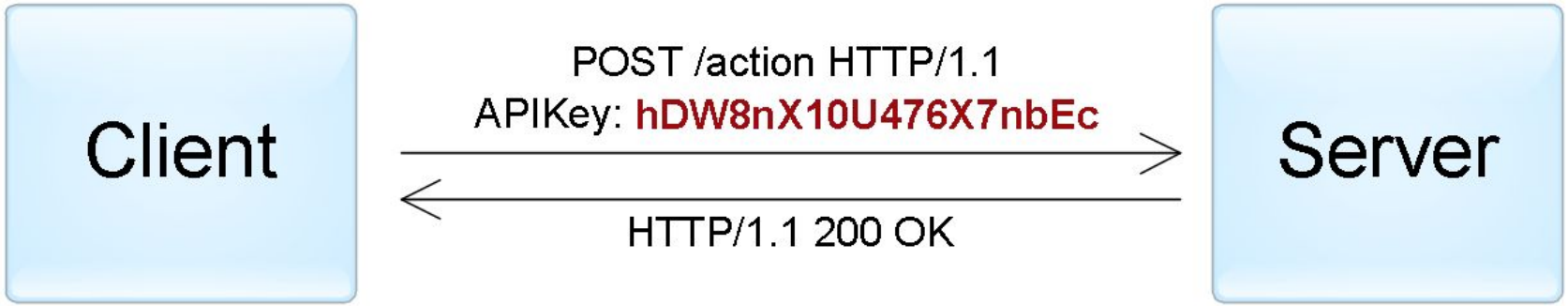


Аутентификация по ключам доступа

Использование ключей позволяет избежать передачи пароля пользователя сторонним приложениям (в примере выше пользователь сохранил в веб-приложении не свой пароль, а ключ доступа). Ключи обладают значительно большей энтропией по сравнению с паролями, поэтому их практически невозможно подобрать. Кроме того, если ключ был раскрыт, это не приводит к компрометации основной учетной записи пользователя — достаточно лишь аннулировать этот ключ и создать новый.

Аутентификация по ключам доступа

С технической точки зрения, здесь не существует единого протокола: ключи могут передаваться в разных частях HTTP-запроса: URL query, request body или HTTP header. Как и в случае аутентификации по паролю, наиболее оптимальный вариант — использование HTTP header. В некоторых случаях используют HTTP-схему Bearer для передачи токена в заголовке
(Authorization: ApiKey [token]). Чтобы избежать перехвата ключей, соединение с сервером должно быть обязательно защищено протоk



Аутентификация по ключам доступа

Кроме того, существуют более сложные схемы аутентификации по ключам для незащищенных соединений. В этом случае, ключ обычно состоит их двух частей: публичной и секретной. Публичная часть используется для идентификации клиента, а секретная часть позволяет сгенерировать подпись. Например, по аналогии с digest authentication схемой, сервер может послать клиенту уникальное значение nonce или timestamp, а клиент — возвратить хэш или HMAC этого значения, вычисленный с использованием секретной части ключа. Это позволяет избежать передачи всего ключа в оригинальном виде и защищает от replay attacks.

Аутентификация по токенам

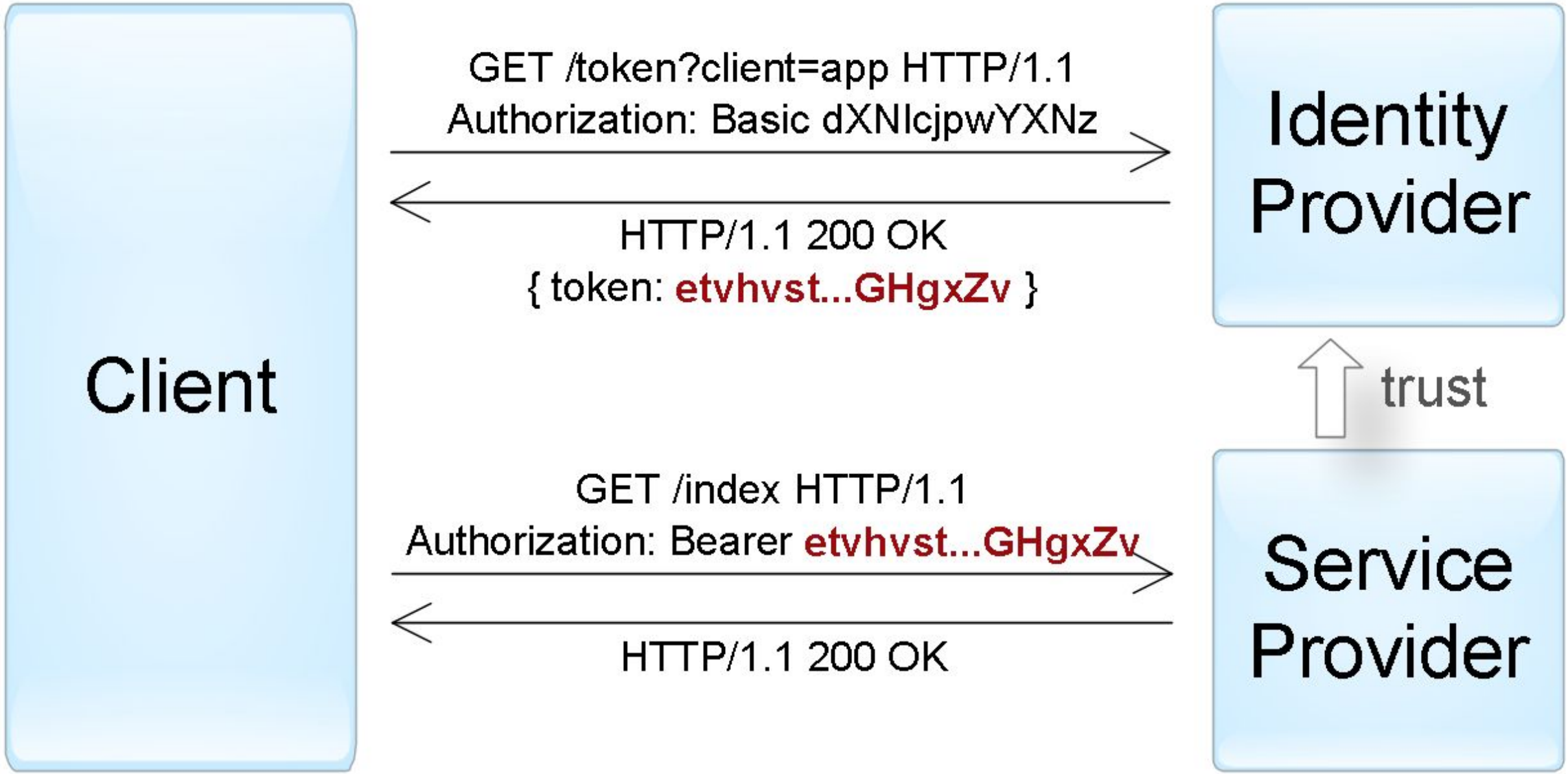
Такой способ аутентификации чаще всего применяется при построении распределенных систем *Single Sign-On* (SSO), где одно приложение (*service provider* или *relying party*) делегирует функцию аутентификации пользователей другому приложению (*identity provider* или *authentication service*). Типичный пример этого способа — вход в приложение через учетную запись в социальных сетях. Здесь социальные сети являются сервисами аутентификации, а приложение доверяет функцию аутентификации пользователей социальным сетям.

Аутентификация по токенам

Реализация этого способа заключается в том, что identity provider (IP) предоставляет достоверные сведения о пользователе в виде токена, а service provider (SP) приложение использует этот токен для идентификации, аутентификации и авторизации пользователя.
На общем уровне, весь процесс выглядит следующим образом:
1. Клиент аутентифицируется в IP одним из способов, специфичным для него (пароль, ключ доступа, сертификат, Kerberos, итд.).

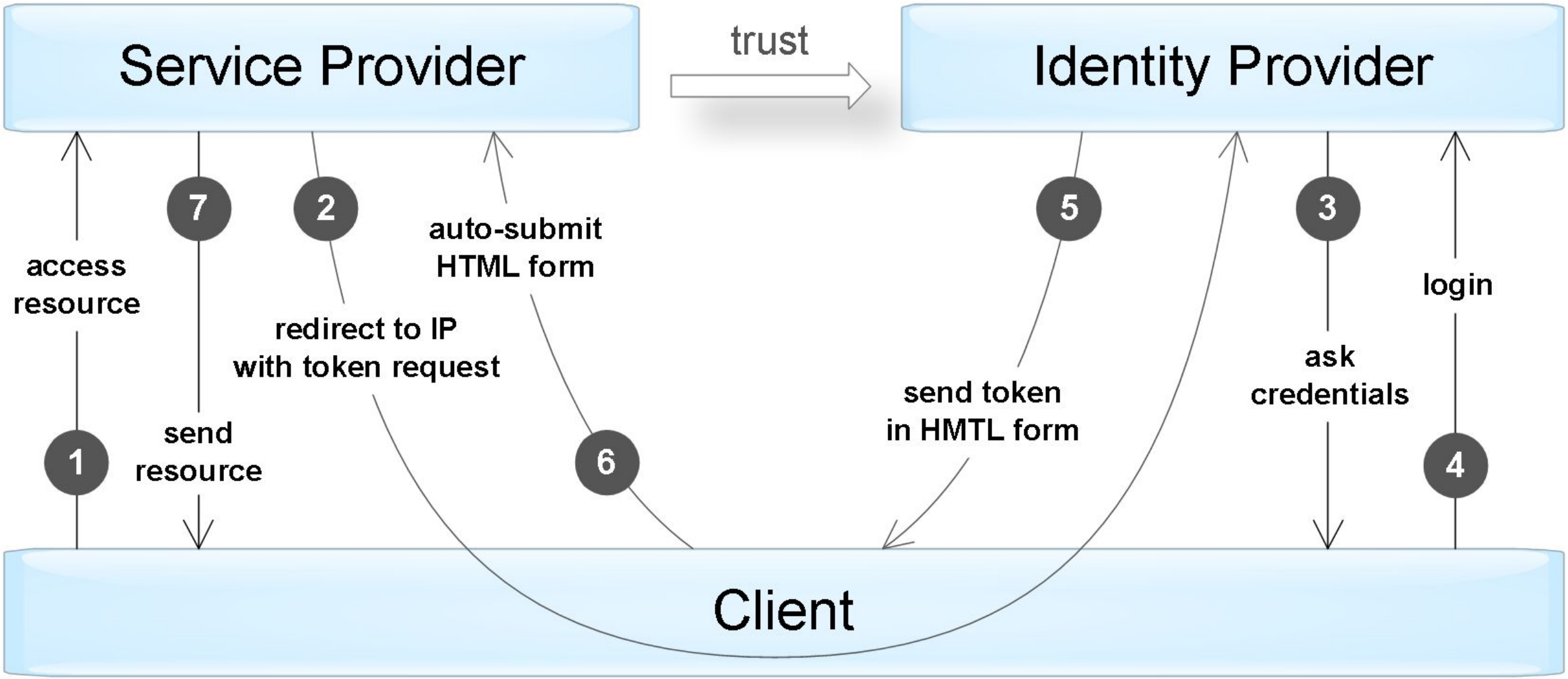
2. Клиент просит IP предоставить ему токен для конкретного SP-приложения. IP генерирует токен и отправляет его клиенту.

3. Клиент аутентифицируется в SP-приложении при помощи этого токена.


Client
GET /token?client=app HTTP/1.1
Authorization: Basic dXNlcjpwYXNz
HTTP/1.1 200 OK
{ token: etvhvst...GHgxZv }
Identity Provider
trust
GET /index HTTP/1.1
Authorization: Bearer etvhvst...GHgxZv
HTTP/1.1 200 OK
Service Provider

Аутентификация по токенам

Процесс, описанный на предыдущем слайде, отражает механизм аутентификации *активного* клиента, т. е. такого, который может выполнять запрограммированную последовательность действий (например, iOS/Android приложения). Браузер же — *пассивный* клиент в том смысле, что он только может отображать страницы, запрошенные пользователем. В этом случае аутентификация достигается посредством автоматического перенаправления браузера между веб-приложениями identity provider и service provider.


Service Provider
trust
Identity Provider
access resource
1
send resource
7
redirect to IP with token request
2
auto-submit HTML form
6
send token in HMTL form
5
ask credentials
3
login
4
Client

Аутентификация по токенам

Сам токен обычно представляет собой структуру данных, которая содержит информацию, кто сгенерировал токен, кто может быть получателем токена, срок действия, набор сведений о самом пользователе (claims). Кроме того, токен дополнительно подписывается для предотвращения несанкционированных изменений и гарантий подлинности.

Аутентификация по токенам

При аутентификации с помощью токена SP-приложение должно выполнить следующие проверки:

1. Токен был выдан доверенным IP приложением (проверка поля *issuer*).
2. Токен предназначается текущему SP-приложению (проверка поля *audience*).
3. Срок действия токена еще не истек (проверка поля *expiration date*).
4. Токен подлинный и не был изменен (проверка подписи).

В случае успешной проверки SP-приложение выполняет авторизацию запроса на основании данных о пользователе, содержащихся в токене.

Аутентификация по токенам

Существует несколько стандартов, в точности определяющих протокол взаимодействия между клиентами (активными и пассивными) и IP/SP-приложениями и **формат** поддерживаемых **токенов**. Среди наиболее популярных стандартов:

- OAuth 2.0
- OpenID Connect
- SAML
- WS-Federation

Форматы токенов

Simple Web Token (SWT) — наиболее простой формат, представляющий собой набор произвольных пар имя/значение в формате кодирования HTML form. Стандарт определяет несколько зарезервированных имен: Issuer, Audience, ExpiresOn и HMACSHA256. Токен подписывается с помощью симметричного ключа, таким образом оба IP- и SP-приложения должны иметь этот ключ для возможности создания/проверки токена.

Форматы токенов

Пример SWT токена (после декодирования).

Issuer=http://auth.myservice.com&
Audience=http://myservice.com&
ExpiresOn=1435937883&
UserName=John Smith&
UserRole=Admin&
HMACSHA256=KOUQRPSpy64rvT2KnYyQKtFFXUIggnesSpE7ADA4o9w

Форматы токенов

JSON Web Token (JWT) — содержит три блока, разделенных точками: заголовок, набор полей (claims) и подпись. Первые два блока представлены в JSON-формате и дополнительно закодированы в формат base64. Набор полей содержит произвольные пары имя/значения, притом стандарт JWT определяет несколько зарезервированных имен (iss, aud, exp и другие). Подпись может генерироваться при помощи и симметричных алгоритмов шифрования, и асимметричных. Кроме того, существует отдельный стандарт, отписывающий формат зашифрованного JWT-токена.

Форматы токенов

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaXNzIjoiaHR0cHM6Ly9hdXRoLm15
c2VydmljZS5jb20iLCJhdWQiOiJodHRwczovL2F
wcGxpY2F0aW9uLm15c2VydmljZS5jb20iLCJpYX
QiOjE1MTYyMzkwMjJ9.TGlqTv-
kzCaRJwXH5yiH5scXfsImDjJP0VcBzcFn4JI
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iss": "https://auth.myservice.com",
  "aud": "https://application.myservice.com",
  "iat": 1516239022
}
```

Форматы токенов

Security Assertion Markup Language (SAML) — определяет токены (SAML assertions) в XML-формате, включающем информацию об эмитенте, о субъекте, необходимые условия для проверки токена, набор дополнительных утверждений (statements) о пользователе. Подпись SAML-токенов осуществляется при помощи ассиметричной криптографии. Кроме того, в отличие от предыдущих форматов, SAML-токены содержат механизм для подтверждения владения токеном, что позволяет предотвратить перехват токенов через man-in-the-middle-атаки при использовании незащищенных соединений.

Стандарт SAML

Стандарт Security Assertion Markup Language (SAML) описывает способы взаимодействия и протоколы между identity provider и service provider для обмена данными аутентификации и авторизации посредством токенов.

Этот основополагающий стандарт — достаточно сложный и поддерживает много различных сценариев интеграции систем.

Рассмотрим основные «строительные блоки» стандарта.

Стандарт SAML

Assertions — собственный формат SAML токенов в XML формате.

Protocols — набор поддерживаемых сообщений между участниками, среди которых — запрос на создание нового токена, получение существующих токенов, выход из системы (logout), управление идентификаторами пользователей, и другие.

Bindings — механизмы передачи сообщений через различные транспортные протоколы. Поддерживаются такие способы, как HTTP Redirect, HTTP POST, HTTP Artifact (ссылка на сообщения), SAML SOAP, SAML URI (адрес получения сообщения) и другие.

Стандарт SAML

Profiles — типичные сценарии использования стандарта, определяющие набор assertions, protocols и bindings необходимых для их реализации, что позволяет достичь лучшей совместимости. Web Browser SSO — один из примеров таких профилей.

Более подробно рассмотреть этот стандарт можно по ссылке

https://rtfm.co.ua/what-is-saml-obzor-struktura-i-trassirovka-zaprosov-na-primere-jenkins-i-okta-saml-sso/#SAML_profiles

Стандарты WS-Trust и WS-Federation

WS-Trust и **WS-Federation** входят в группу стандартов WS-* (Web Services – Security), описывающих SOAP/XML-веб сервисы. Эти стандарты разрабатываются группой компаний, куда входят Microsoft, IBM, VeriSign и другие. Наряду с SAML, эти стандарты достаточно сложные, используются преимущественно в корпоративных сценариях.

Стандарты WS-Trust и WS-Federation

Стандарт **WS-Trust** описывает интерфейс сервиса авторизации, именуемого Secure Token Service (STS). Этот сервис работает по протоколу SOAP и поддерживает создание, обновление и аннулирование токенов. При этом стандарт допускает использование токенов различного формата, однако на практике в основном используются SAML-токены.

Стандарты WS-Trust и WS-Federation

Стандарт **WS-Federation** касается механизмов взаимодействия сервисов между компаниями, в частности, протоколов обмена токенов. При этом WS-Federation расширяет функции и интерфейс сервиса STS, описанного в стандарте WS-Trust. Среди прочего, стандарт WS-Federation определяет:

- Формат и способы обмена метаданными о сервисах.
- Функцию единого выхода из всех систем (single sign-out).
- Сервис атрибутов, предоставляющий дополнительную информацию о пользователе.
- Сервис псевдонимов, позволяющий создавать альтернативные имена пользователей.
- Поддержку пассивных клиентов (браузеров) посредством перенаправления.

Стандарты OAuth и OpenID Connect

В отличие от SAML и WS-Federation, стандарт OAuth (Open Authorization) не описывает протокол аутентификации пользователя. Вместо этого он определяет механизм получения доступа одного приложения к другому от имени пользователя. Однако существуют схемы, позволяющие осуществить аутентификацию пользователя на базе этого стандарта.

Первая версия стандарта разрабатывалась в 2007 – 2010 гг., а текущая версия 2.0 опубликована в 2012 г. Версия 2.0 значительно расширяет и в то же время упрощает стандарт, но обратно несовместима с версией 1.0. Сейчас OAuth 2.0 очень популярен и используется повсеместно для предоставления делегированного доступа и третье-сторонней аутентификации пользователей.

Стандарты OAuth и OpenID Connect

Чтобы лучше понять сам стандарт, рассмотрим пример веб-приложения, которое помогает пользователям планировать путешествия. Как часть функциональности оно умеет анализировать почту пользователей на наличие писем с подтверждениями бронирований и автоматически включать их в планируемый маршрут. Возникает вопрос, как это веб-приложение может безопасно получить доступ к почте пользователей, например, к Gmail?

> *Попросить пользователя указать данные своей учетной записи?* — очень плохой вариант.
> *Попросить пользователя создать ключ доступа?* — возможно, но для юзера это может быть сложно.

Стандарты OAuth и OpenID Connect

Как раз эту проблему и позволяет решить стандарт OAuth: он описывает, как приложение путешествий (client) может получить доступ к почте пользователя (resource server) с разрешения пользователя (resource owner). В общем виде весь процесс состоит из нескольких шагов:

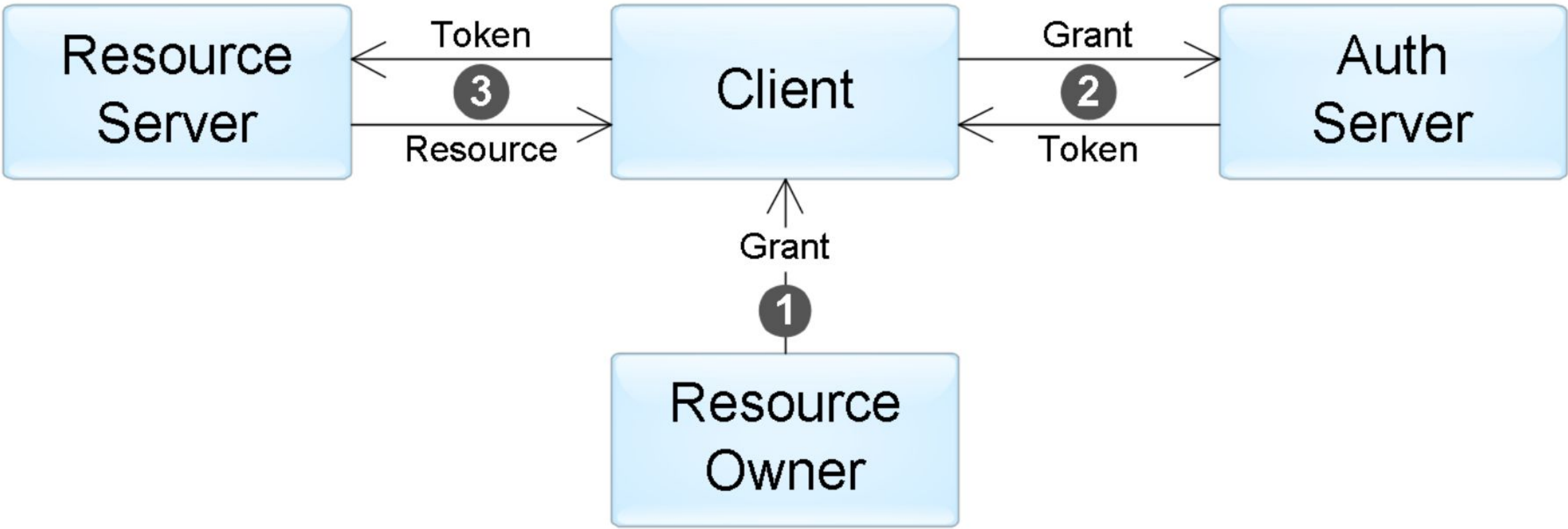
1. Пользователь (resource owner) дает разрешение приложению (client) на доступ к определенному ресурсу в виде гранта.

(что такое грант после описания шагов)

Стандарты OAuth и OpenID Connect

2. Приложение обращается к серверу авторизации и получает токен доступа к ресурсу в обмен на свой грант. В нашем примере сервер авторизации — Google. При вызове приложение дополнительно аутентифицируется при помощи ключа доступа, выданным ему при предварительной регистрации.

3. Приложение использует этот токен для получения требуемых данных от сервера ресурсов (в нашем случае — сервис Gmail).


Resource Server
Token
3
Resource
Client
Grant
2
Token
Auth Server
Grant
1
Resource Owner

Стандарты OAuth и OpenID Connect

Стандарт описывает четыре вида грантов, которые определяют возможные сценарии применения:

Authorization Code — этот грант пользователь может получить от сервера авторизации после успешной аутентификации и подтверждения согласия на предоставление доступа. Такой способ наиболее часто используется в веб-приложениях. Процесс получения гранта очень похож на механизм аутентификации пассивных клиентов в SAML и WS-Federation.

Стандарты OAuth и OpenID Connect

Implicit — применяется, когда у приложения нет возможности безопасно получить токен от сервера авторизации (например, JavaScript-приложение в браузере). В этом случае грант представляет собой токен, полученный от сервера авторизации, без обмена кода авторизации на сам токен.

(п.с. Не самый безопасный вариант, рекомендуется использовать вместе с PKCE flow)

Стандарты OAuth и OpenID Connect

Resource Owner Password Credentials — грант представляет собой пару username/password пользователя. Может применяться, если приложение является «интерфейсом» для сервера ресурсов (например, приложение — мобильный клиент для Gmail).

По умолчанию выключен во большинстве Authorization Service’ах.

Стандарты OAuth и OpenID Connect

Client Credentials — в этом случае нет никакого пользователя, есть приложение которое собирается получает доступ к своим ресурсам при помощи своих ключей доступа (т.е. вариант взаимодействия Backend-To-Backend от лица всей организации (группы юзеров))

Стандарты OAuth и OpenID Connect

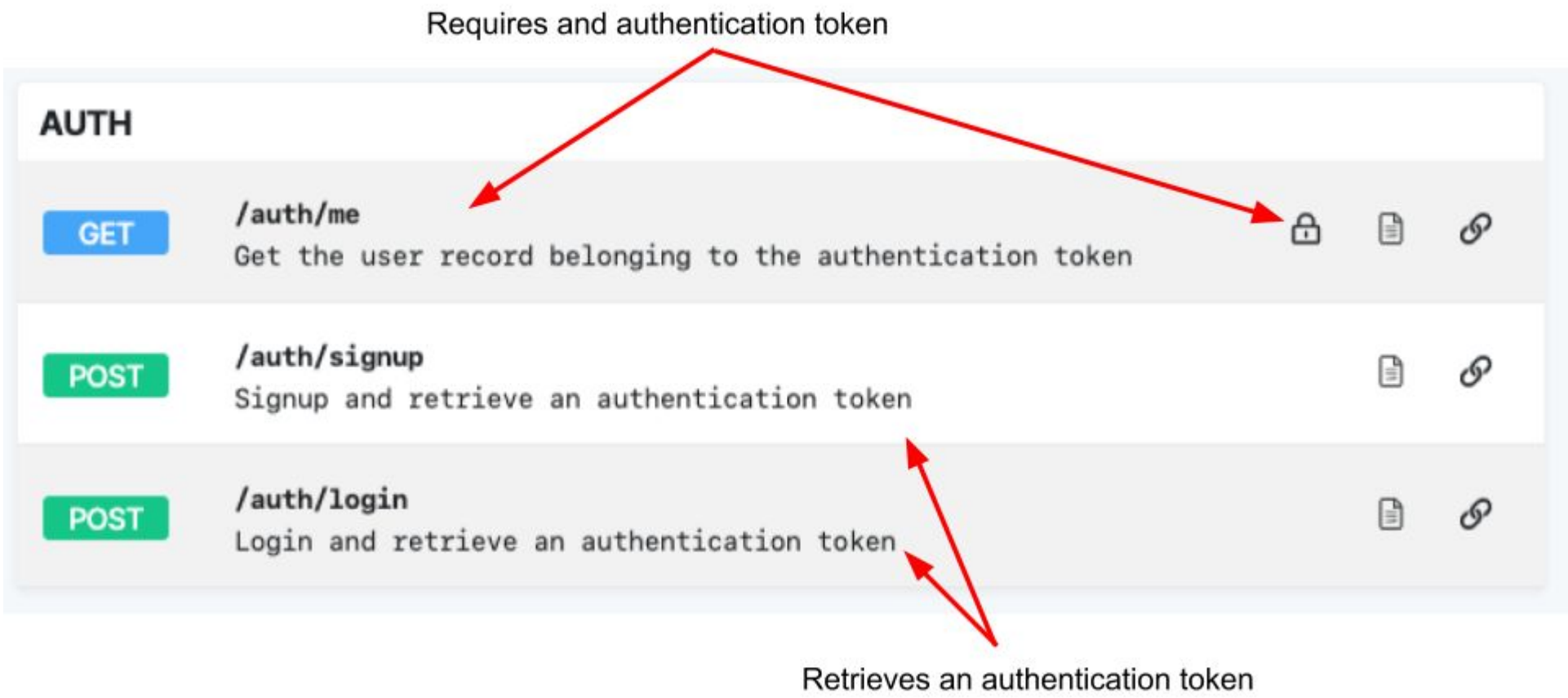
Стандарт не определяет формат токена, который получает приложение: в сценариях, адресуемых стандартом, приложению нет необходимости анализировать токен, т. к. он лишь используется для получения доступа к ресурсам. Поэтому ни токен, ни грант сами по себе не могут быть использованы для аутентификации пользователя.

Однако если приложению необходимо получить достоверную информацию о пользователе, существуют несколько способов это сделать

Стандарты OAuth и OpenID Connect

Зачастую API сервера ресурсов включает операцию, предоставляющую информацию о самом пользователе (например, /me в Facebook API). Приложение может выполнять эту операцию каждый раз после получения токена для идентификации клиента. Такой метод иногда называют *псевдо-аутентификацией*.

Стандарты OAuth и OpenID Connect



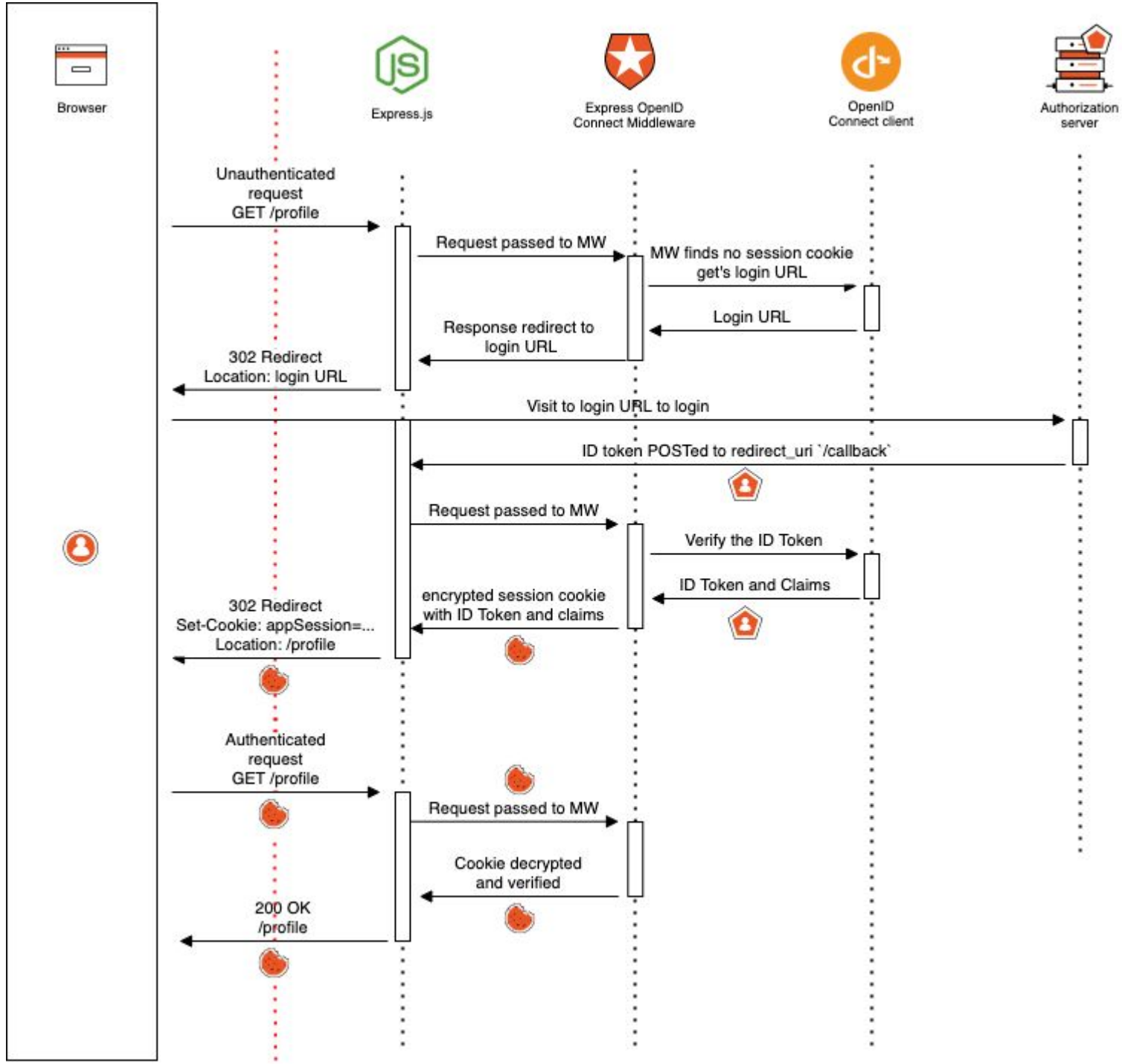
Стандарты OAuth и OpenID Connect

Использовать стандарт **OpenID Connect**, разработанный как слой учетных данных поверх OAuth (опубликован в 2014 г.). В соответствии с этим стандартом, сервер авторизации предоставляет дополнительный identity token на шаге № 2. Этот токен в формате JWT будет содержать набор определенных полей (claims) с информацией о пользователе.

Sample ID Token

This example shows the contents of an ID token. Notice that the audience value (located in the `aud` claim) is set to the application's identifier, which means that only this specific application should consume the token.


```
{
  "iss": "http://my-domain.auth0.com",
  "sub": "auth0|123456",
  "aud": "my_client_id",
  "exp": 1311281970,
  "iat": 1311280970,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "gender": "female",
  "birthdate": "0000-10-31",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```


Browser
Express.js
Express OpenID Connect Middleware
OpenID Connect client
Authorization server
Unauthenticated request GET /profile
Request passed to MW
MW finds no session cookie get's login URL
Login URL
Response redirect to login URL
302 Redirect Location: login URL
Visit to login URL to login
ID token POSTed to redirect_uri `/callback`
Request passed to MW
Verify the ID Token
ID Token and Claims
encrypted session cookie with ID Token and claims
302 Redirect Set-Cookie: appSession=... Location: /profile
Authenticated request GET /profile
Request passed to MW
Cookie decrypted and verified
200 OK /profile

Авторизация в NodeJS приложениях

В рамках нашего курса предлагаю использовать популярную middleware - PassportJS, для того чтобы разобраться с всеми проблемами аутентификации.

Passport отлично справляется с тем чтобы выделить среди других элементов веб-приложений именно аспекты аутентификации.

Это позволяет Passport легко настроить любое веб-приложение на базе Express, так же, как мы можем просто настроить другой связующий Express-софт, например, logging, body-parsing, cookie-parsing, session-handling и т.д.

Авторизация в NodeJS приложениях

Passport предлагает нам на выбор свыше 140 механизмов аутентификации. Вы можете проводить аутентификацию с помощью локального/удаленного экземпляра объекта базы данных или использовать единый вход с использованием OAuth, предоставляемый Facebook, Twitter, Google и т.д., для аутентификации в ваших аккаунтах социальных медиа.

Или вы можете выбрать из обширного списка провайдеров, которые поддерживают аутентификацию с помощью Passport и предоставляют для него модуль узла.