ПРОГРАММИРОВАНИЕ В INTERNET

ИДЕНТИФИКАЦИЯ. АУТЕНТИФИКАЦИЯ. АВТОРИЗАЦИЯ

Этапы проверки пользователя

Идентификация =

процесс распознавания пользователя по его идентификатору.

Чаще всего для идентификации используются: email, имя пользователя, телефон...

https://vk.com/id186301730

То есть, пользователь говорит, кто он

Аутентификация =

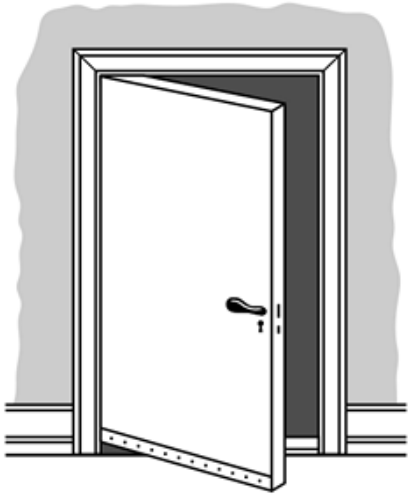
процедура проверки подлинности, то есть доказательство того, что пользователь именно тот, за кого себя выдает.

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

- Пароль – что-то, известное только пользователю (слово, PIN-код, код для замка, графический ключ)
- Устройство – что-то, имеющееся только у пользователя (токен, пластиковая карта, ключ от замка)
- Биометрика – что-то, присущее только пользователю (отпечаток пальца, сетчатка глаза, сканер лица)

Авторизация =

предоставление определённых прав для доступа к некоторым ресурсам.

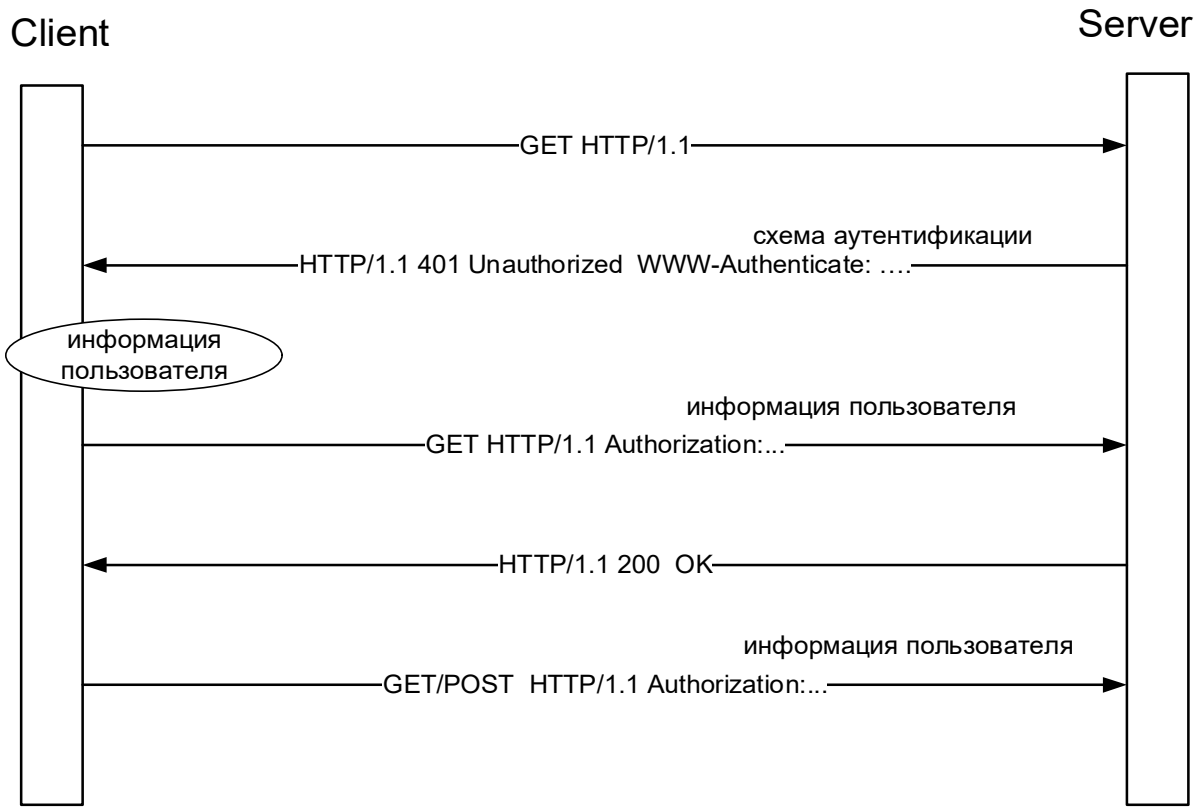


Многофакторная аутентификация =

метод, при котором пользователю для доступа к некоторым ресурсам необходимо несколькими различными факторами доказать, что именно он осуществляет вход.

Среди видов многофакторной аутентификации наиболее распространена двухфакторная аутентификация – метод, при котором пользователю для получения доступа необходимо предоставить два разных типа аутентификационных данных, например, что-то известное только пользователю (пароль) и что-то присущее только пользователю (отпечаток пальца).

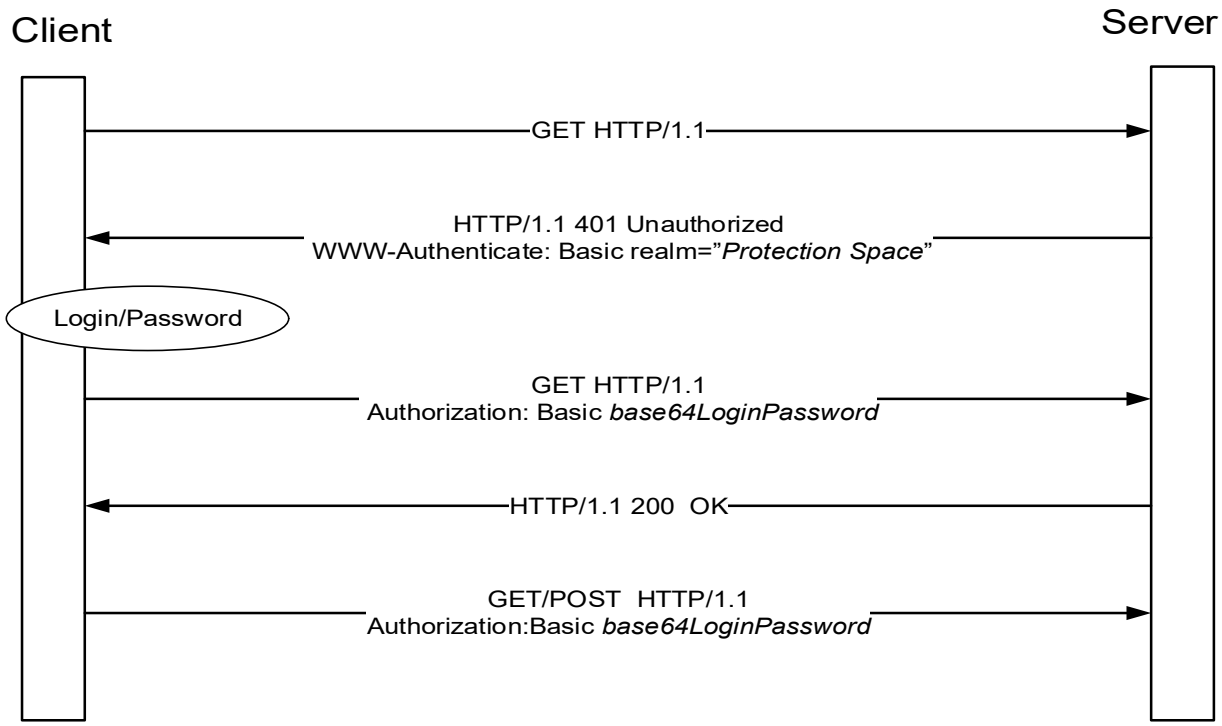
HTTP-аутентификация (RFC 7235)



Вначале сервер отвечает клиенту со статусом 401 (Unauthorized) и предоставляет информацию о порядке аутентификации через заголовок WWW-Authenticate, содержащий хотя бы один метод аутентификации. Клиент может аутентифицироваться, включив в следующий запрос заголовок Authorization с требуемыми данными.

Если (прокси) сервер получает корректные учетные данные, но они не подходят для доступа к запрашиваемому ресурсу, сервер должен отправить ответ со статус кодом 403 Forbidden.

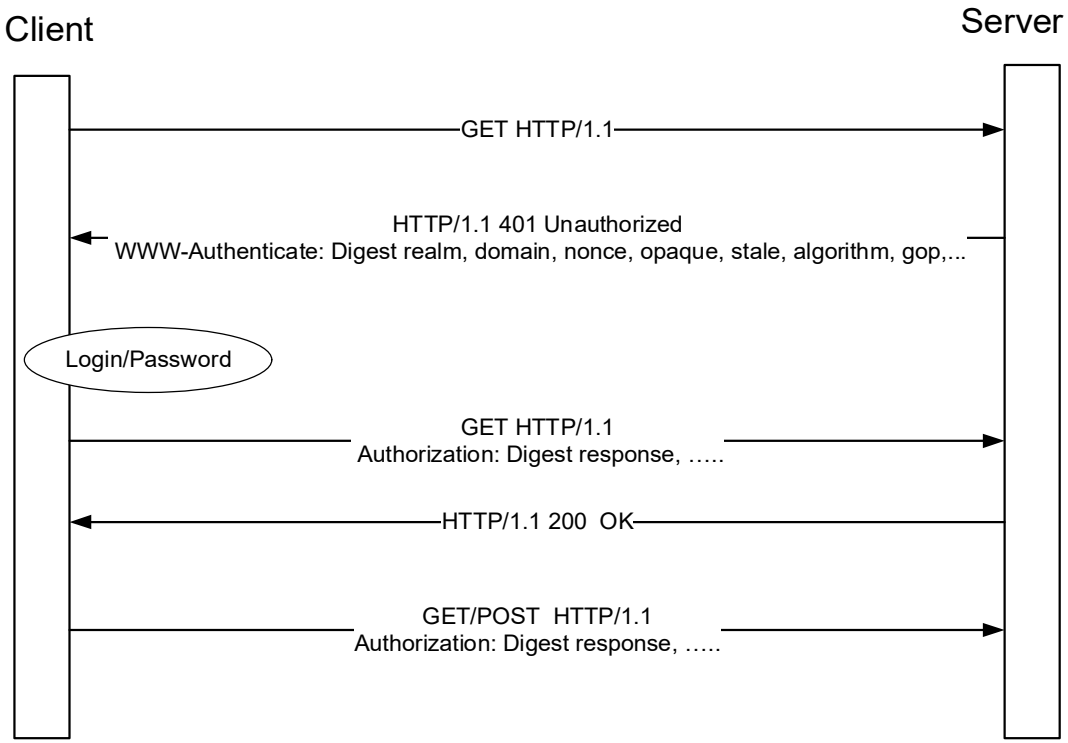
Basic-аутентификация (RFC 7617)



Basic-аутентификация использует кодирование в base64 для генерации строки, которая содержит информацию об имени пользователя и пароле:

base64("username:password")

Digest-аутентификация (RFC 7616)



Это аутентификация, при которой пароль пользователя передается в хешированном виде. Пароль хешируется всегда с добавлением произвольной строки символов, которая генерируется на каждое соединение заново.

Сервер дает клиенту одноразовый номер использования (nonce), который он комбинирует с именем пользователя, realm, паролем и запросом URI. Клиент хэширует все эти поля с помощью метода хэширования MD5 (по умолчанию) для получения ключа hash.

WWW-Authenticate

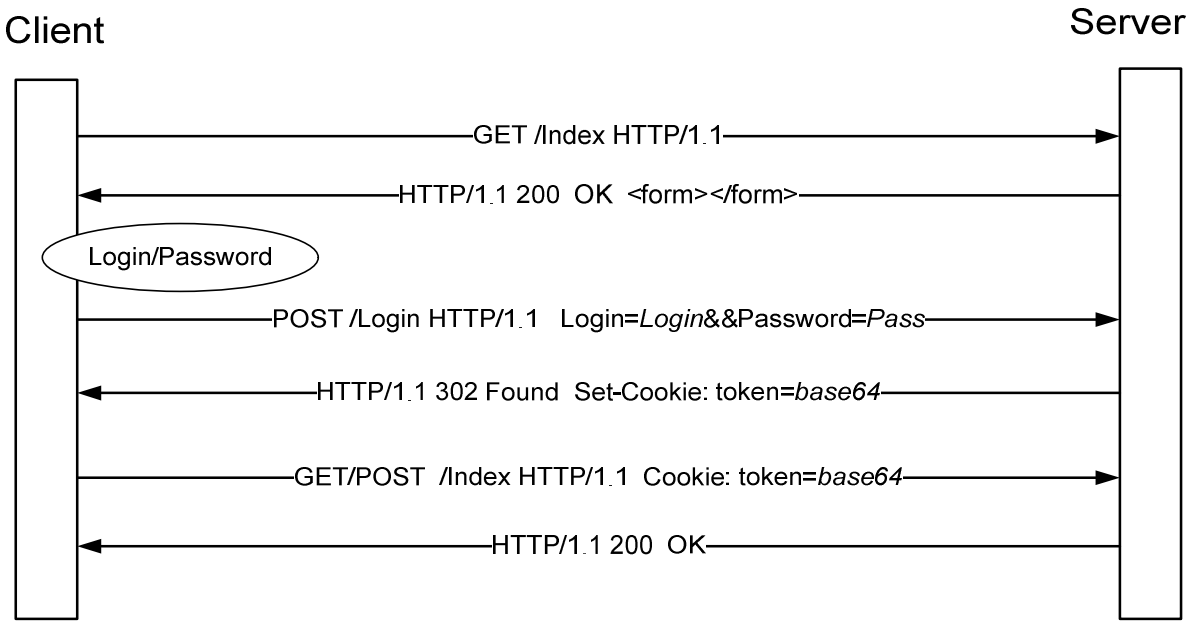
- realm
- domain
- nonce
- opaque
- stale
- algorithm
- qop
- charset
- userhash

Authorization

- response
- username
- username*
- realm
- uri
- qop
- cnonce
- nc
- userhash

Параметры, выделенные желтым цветом, могут использоваться не только при Digest-аутентификации, но и при Basic-аутентификации

Forms-аутентификация



Для этой аутентификации нет определенного стандарта.

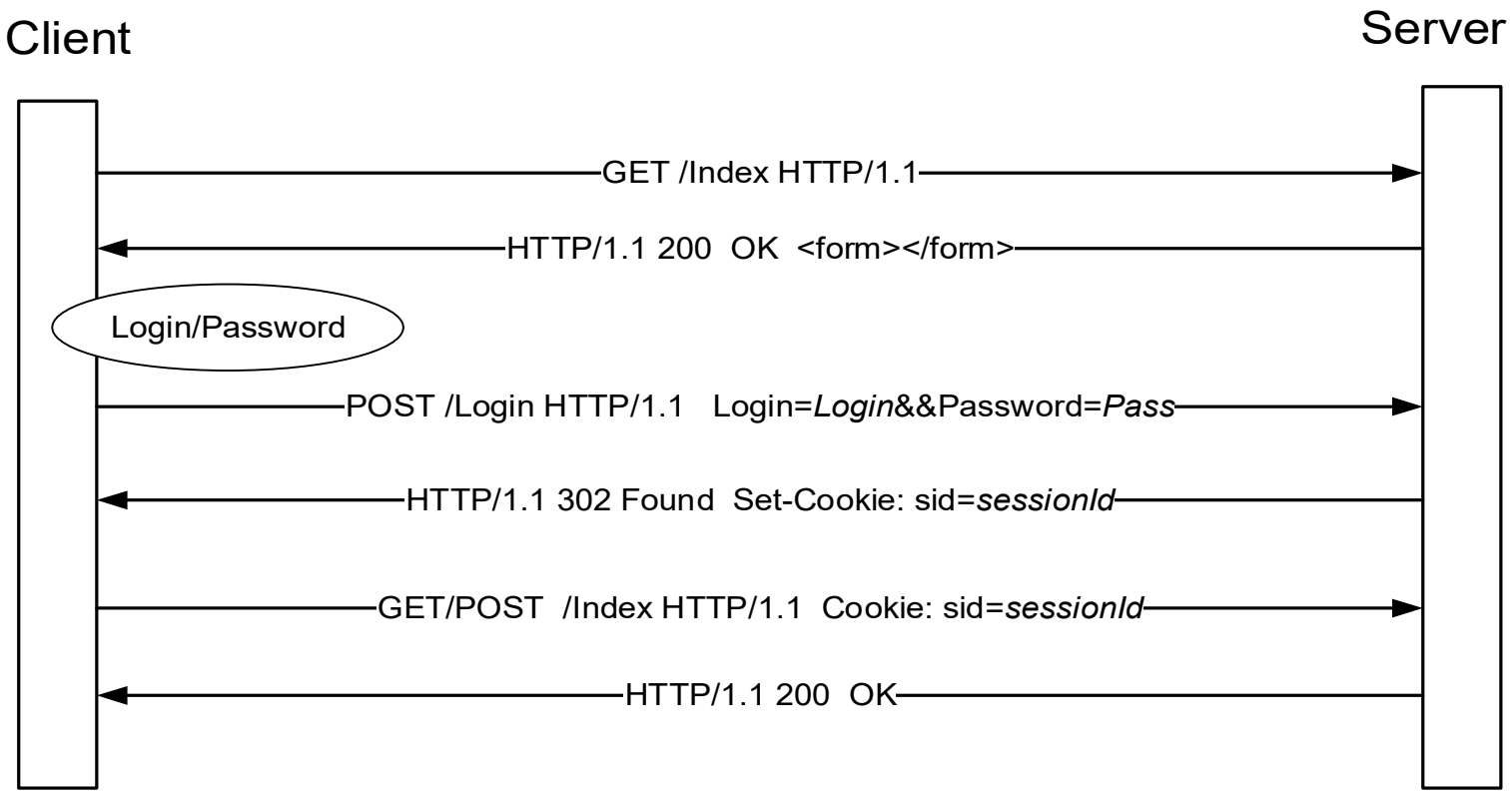
В веб-приложение включается HTML-форма, в которую пользователь должен ввести свои username/password и отправить их на сервер для аутентификации. В случае успеха веб-приложение создает session token, который обычно помещается в cookies. При последующих запросах session token автоматически передается на сервер и позволяет приложению получить информацию о текущем пользователе для авторизации запроса.

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.

2. Как зашифрованный и/или подписанный объект, содержащий данные о пользователе, а также период действия. Этот подход позволяет реализовать stateless-архитектуру сервера.

Forms-аутентификация на основе сессий



Реализация forms-аутентификации на основе сессий

```json
[
  {
    "id": 1,
    "login": "user1",
    "password": "password1",
    "age": 21
  },
  {
    "id": 2,
    "login": "user2",
    "password": "password2",
    "age": 20
  },
  {
    "id": 3,
    "login": "user3",
    "password": "password3",
    "age": 21
  }
]
```

```html
<html>

<head>
    <meta charset="utf-8">
    <title>cwp_24</title>
</head>

<body>
    <form action="/login" method="POST">
        <input type="text" placeholder="Enter Username" name="username" required>
        <input type="password" placeholder="Enter Password" name="password" required>

        <button type="submit">Login</button>
        <button type="reset" class="cancelbtn">Cancel</button>
    </form>
</body>

</html>
```

```
const express = require('express'),
    app = express(),
    bodyParser = require('body-parser'),
    session = require('express-session'),
    passport = require('passport'),
    localStrategy = require('passport-local').Strategy,
    users = require('./users');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(session({ secret: 'you secret key' }));
app.use(passport.initialize());
app.use(passport.session());

passport.serializeUser((user, done) => done(null, user));
passport.deserializeUser((user, done) => done(null, user));
```

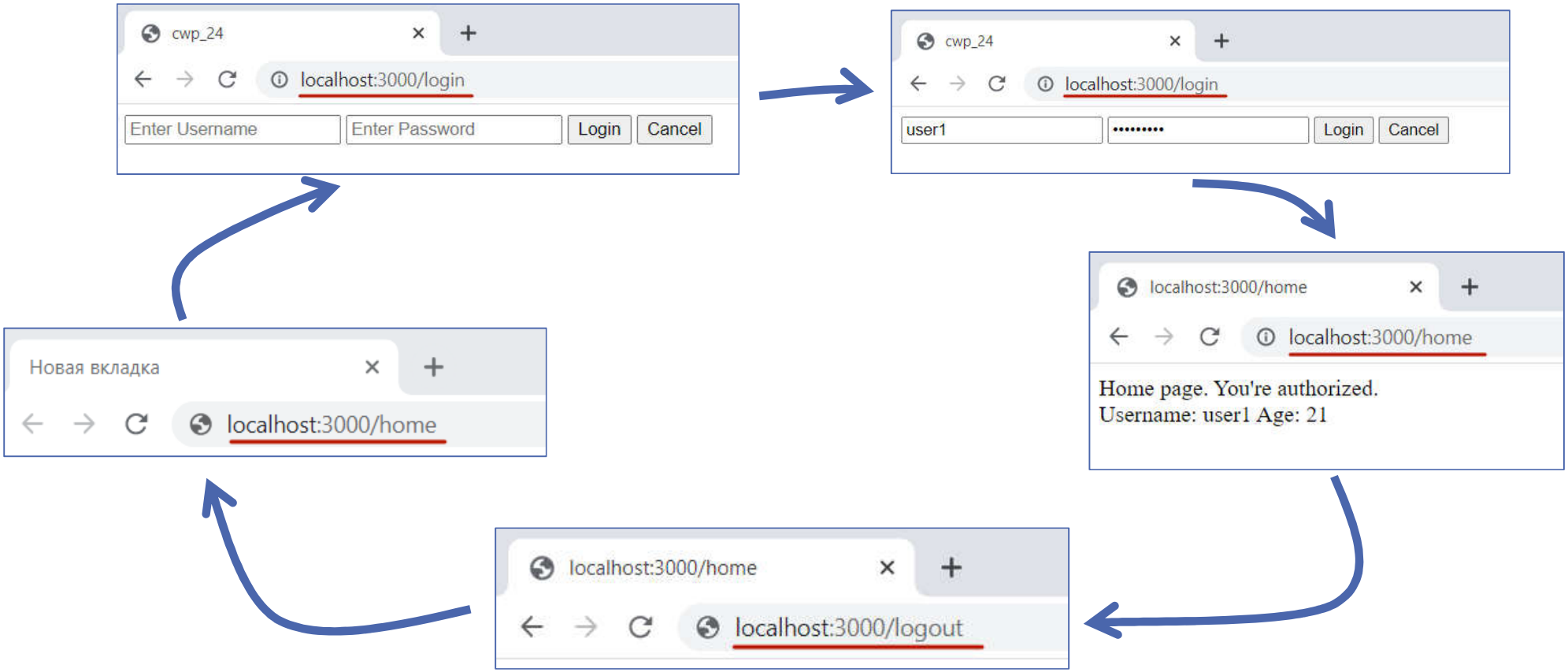
Для Node.js аутентификации по логину и паролю необходимо установить пакеты passport и passport-local. Для использования сессии необходимо использовать функцию промежуточной обработки passport.session().

```
passport.use(
    new localStrategy((username, password, done) => {
        for (let user of users) {
            if (username === user.login && password === user.password)
                return done(null, user);
        }
        return done(null, false, { message: 'Wrong login or password' });
    })
);

app.get('/login', (req, res) => {
    res.sendFile(__dirname + '/24_03.html');
});

app.post('/login', passport.authenticate('local', { successRedirect: '/home', failureRedirect: '/login' }));

app.get('/home',
    (req, res, next) => {
        if (req.user) next();
        else res.redirect('/login');
    }, (req, res) => {
        res.send(`Home page. You're authorized.<br /> Username: ${req.user.login} Age: ${req.user.age}`);
    }
);

app.get('/logout', (req, res) => {
    req.logout();
    res.redirect('/login');
});

app.listen(3000, function () { console.log('Start server, port: ', 3000); })
```

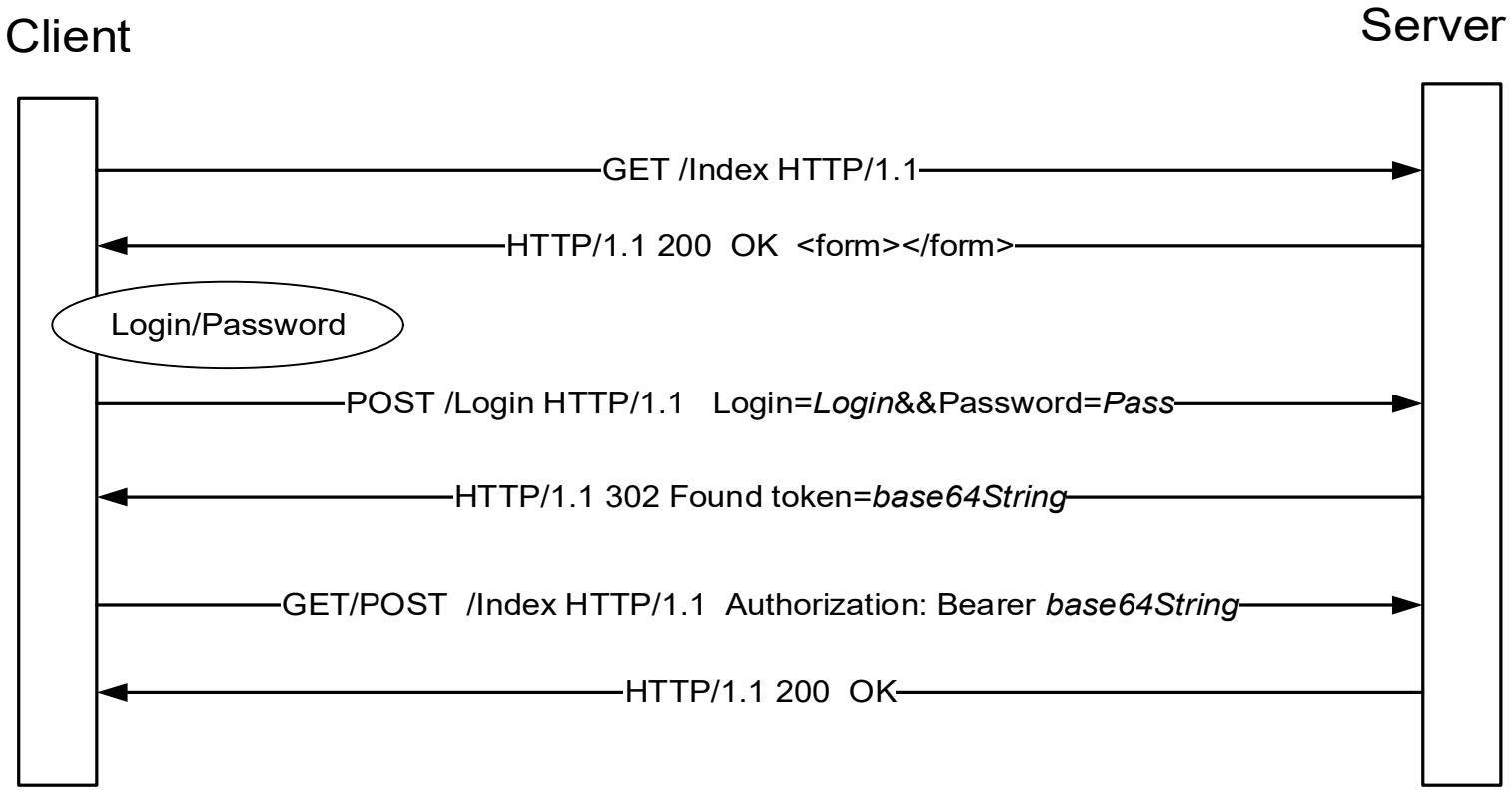
Для записи/получения данных пользователя в/из сессии необходимо использовать функции промежуточной обработки serializeUser() и deserializeUser().

Метод logout() удаляет свойство req.user и очищает сеанс (если есть).

Проверка работоспособности



Forms-аутентификация на основе токенов



Токен =

битовая последовательность, построенная по определенному принципу, которая позволяет точно идентифицировать объект и определить уровень его привилегий.

Токен обычно содержит в себе следующую информацию Issuer (кто выдал), Audience (кому выдан), Expires On (время жизни), Claim (сведение о пользователе), подпись (для защиты от изменений и для гарантий подлинности).

Применяется, как правило, для реализации Single Sign-On (технология единого входа) в распределенных системах.

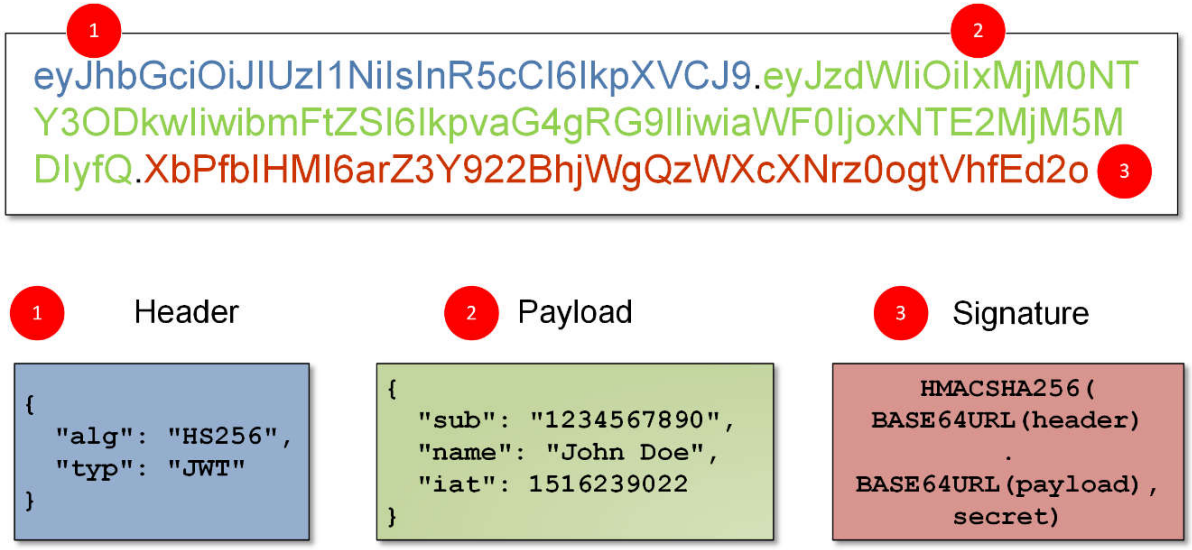
Форматы токенов

SWT (Simple Web Token) – 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.

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

Форматы токенов

JWT (JSON Web Token, RFC 7519) – содержит три блока, разделенных точками: заголовок, полезная нагрузка (payload) и подпись. Payload содержит произвольные пары имя/значения, притом стандарт JWT определяет несколько зарезервированных имен (iss, sub, aud, exp, iat и другие). Подпись может генерироваться при помощи и **симметричных** алгоритмов шифрования, и **асимметричных**.



Форматы токенов

SAML (Security Assertion Markup Language) – определяет токены в XML-формате. Подпись SAML-токенов осуществляется при помощи **ассиметричной** криптографии. Кроме того, в отличие от предыдущих форматов, SAML-токены содержат механизм для подтверждения владения токеном.

```
<assertion id="_4fe09cda-cad9-49dd-b493-93494e1ae4f9" issueinstant="2012-09-18T20:42:11.626Z"
    version="2.0" xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
<issuer>https://test05.accesscontrol.windows.net/</issuer>
<ds:signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:signedinfo>
        <ds:canonicalizationmethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:signaturemethod algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
        <ds:reference uri="#_4fe09cda-cad9-49dd-b493-93494e1ae4f9">
            <ds:transforms>
                <ds:transform algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
                <ds:transform algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:transforms>
            <ds:digestmethod algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
            <ds:digestvalue>8qmfRKuATFuo4M96xuci7HCLUGUeO3eBxHOi9/HaFNU=</ds:digestvalue>
        </ds:reference>
    </ds:signedinfo>
<ds:signaturevalue>UWcXJElfrP8hfdNi8ipzSjfxCYGYzoylkn5HdSa8IhphvyZBvbZl1OFEbMSygoo8xNgnywUNPuzZP8nV7Cw
    <keyinfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <X509Data>       <X509Certificate>MIIDCDCCAfCgAwIBAgIQRmI8p7P/aphMv5Kr9vQpqTANBgkqhkiG9w0BAQUF
        </X509Data>
    </keyinfo>
</ds:signature>
<subject>
    <NameID>abc1def2ghi3jkl4mno5pqr6stu7vwx8yza9bcd0efg=</NameID>
    <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer" />
</subject>
    <conditions notbefore="2012-09-18T20:42:11.610Z" notonorafter="2012-09-18T21:42:11.610Z">
        <AudienceRestriction>
            <Audience>http://localhost:63000/</Audience>
        </AudienceRestriction>
    </conditions>
    <attributestatement>
        <Attribute Name="http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprov
            <AttributeValue>uri:WindowsLiveID</AttributeValue>
        </Attribute>
    </attributestatement>
</assertion>
```

Реализация forms-аутентификации на основе токенов

```
const express = require('express'),
  app = express(),
  bodyParser = require('body-parser'),
  jwt = require('jsonwebtoken'),    // npm install jsonwebtoken
  users = require('./users');

const tokenKey = '1a2b-3c4d-5e6f-7g8h';

app.use(bodyParser.json())
app.use((req, res, next) => {
  if (req.headers.authorization) {
    jwt.verify(req.headers.authorization.split(' ')[1], tokenKey, (err, payload) => {
      if (err) next();
      else if (payload) {
        req.payload = payload;
        next();
      }
    })
  }
  next();
})
```

Данные от клиента приходят в заголовке
Authorization: Bearer <token>

Метод jwt.verify(token, secretOrPublicKey, [options, callback]) проверяет jwt. Возвращает декодированные полезные данные, если подпись действительна, а необязательные срок действия, аудитория или эмитент действительны. В противном случае выдаст ошибку.

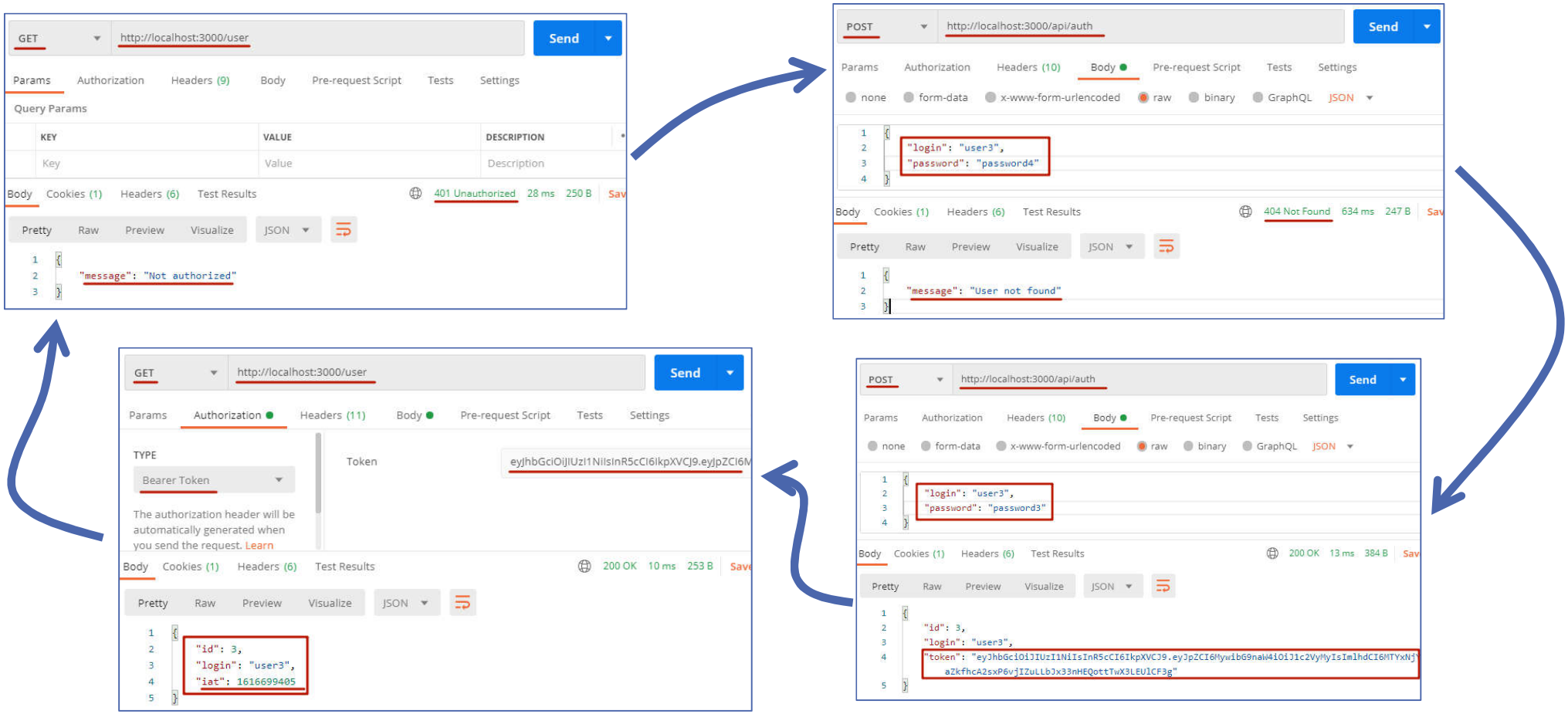
Реализация forms-аутентификации на основе токенов

```
app.post('/api/auth', (req, res) => {
  for (let user of users) {
    if (req.body.login === user.login && req.body.password === user.password) {
      return res.status(200).json({
        id: user.id,
        login: user.login,
        token: jwt.sign({ id: user.id, login: user.login }, tokenKey),
      });
    }
  }
  return res.status(404).json({ message: 'User not found' });
})

app.get('/user', (req, res) => {
  if (req.payload) return res.status(200).json(req.payload);
  else
    return res.status(401).json({ message: 'Not authorized' });
})

app.listen(3000, () => console.log('Start server, port: ', 3000))
```

Поскольку токены не шифруются крайне не рекомендуется хранить в них какую-либо sensitive data (пароли, данные платежных карт и др.)

С помощью метода jwt.sign(payload, secretOrPrivateKey, [options, callback]) можно подписать токен. В options можно настроить algorithm, expiresIn, audience, issuer и др.

Проверка работоспособности



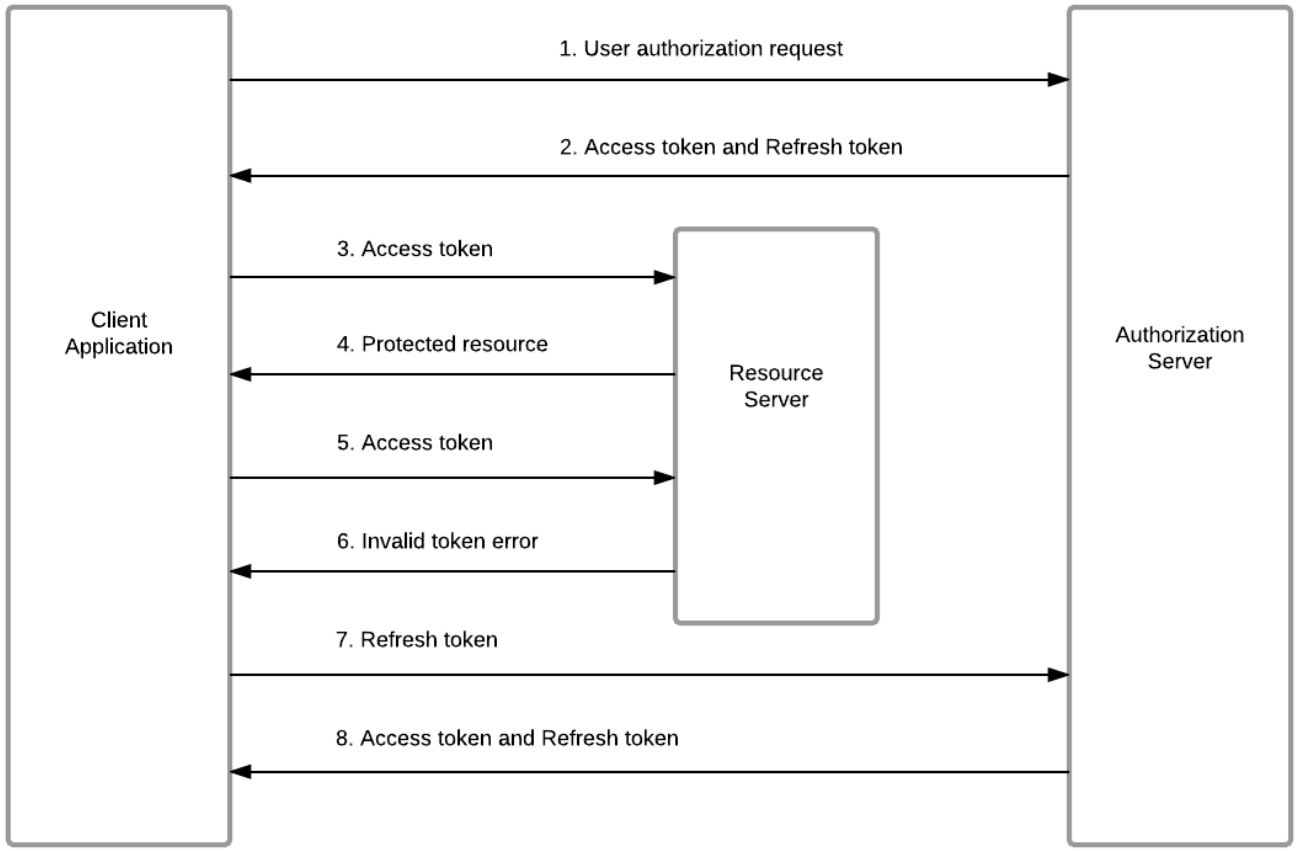
access-токен

- используются для получения доступа к защищенным ресурсам;
- короткоживущий;
- многоразовый;
- при краже access-токена им можно пользоваться ограниченное время.

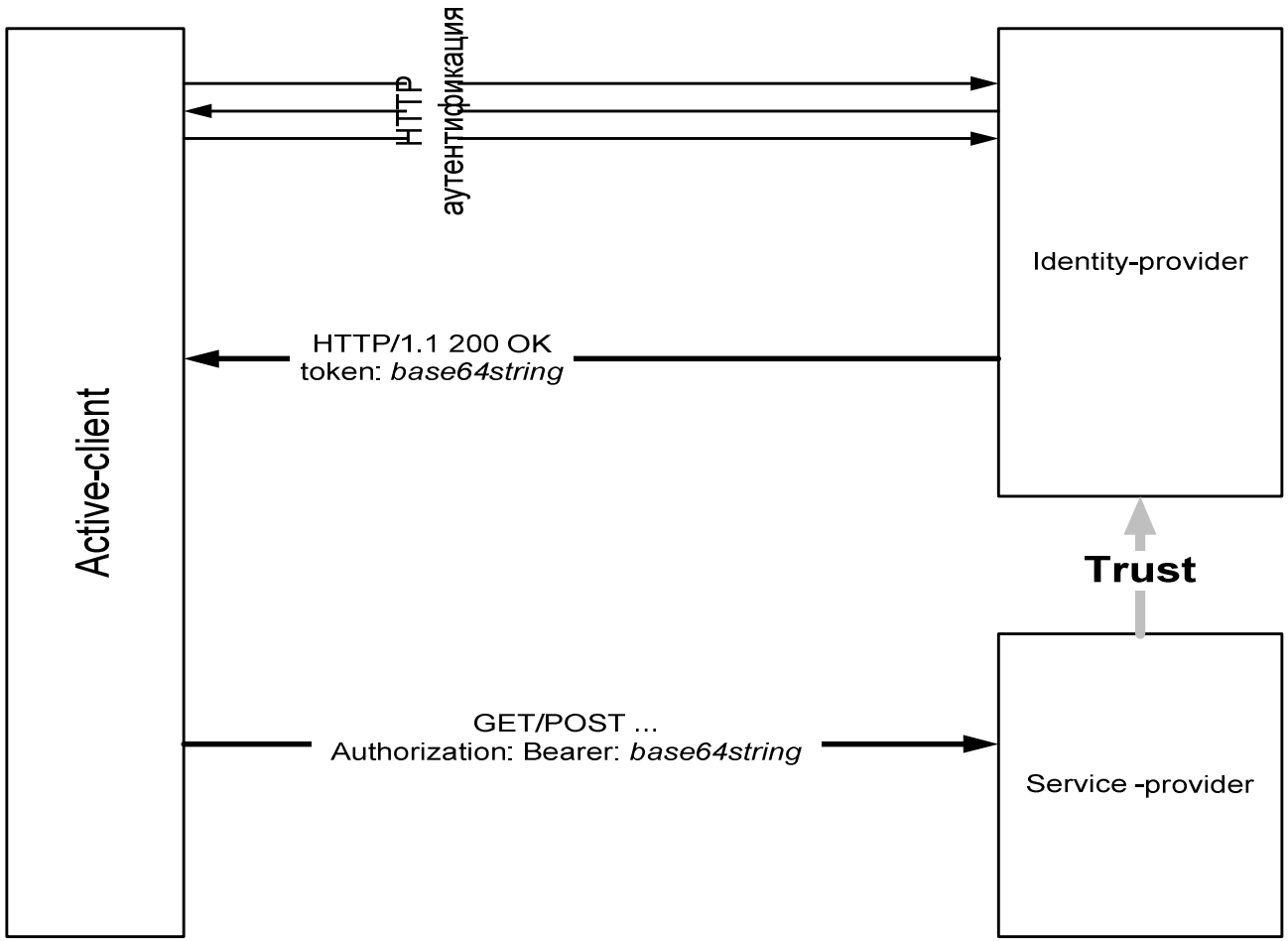
refresh-токены

- используется, когда access-токен истёк, для получения новой пары refresh-токен + access-токен; отправляется на специальный url;
- долгоживущий;
- одноразовый;
- при краже refresh-токен он становится бесполезным при ре-логине (при условии, что они отслеживаются в black или white списках).

Обновление access-токена

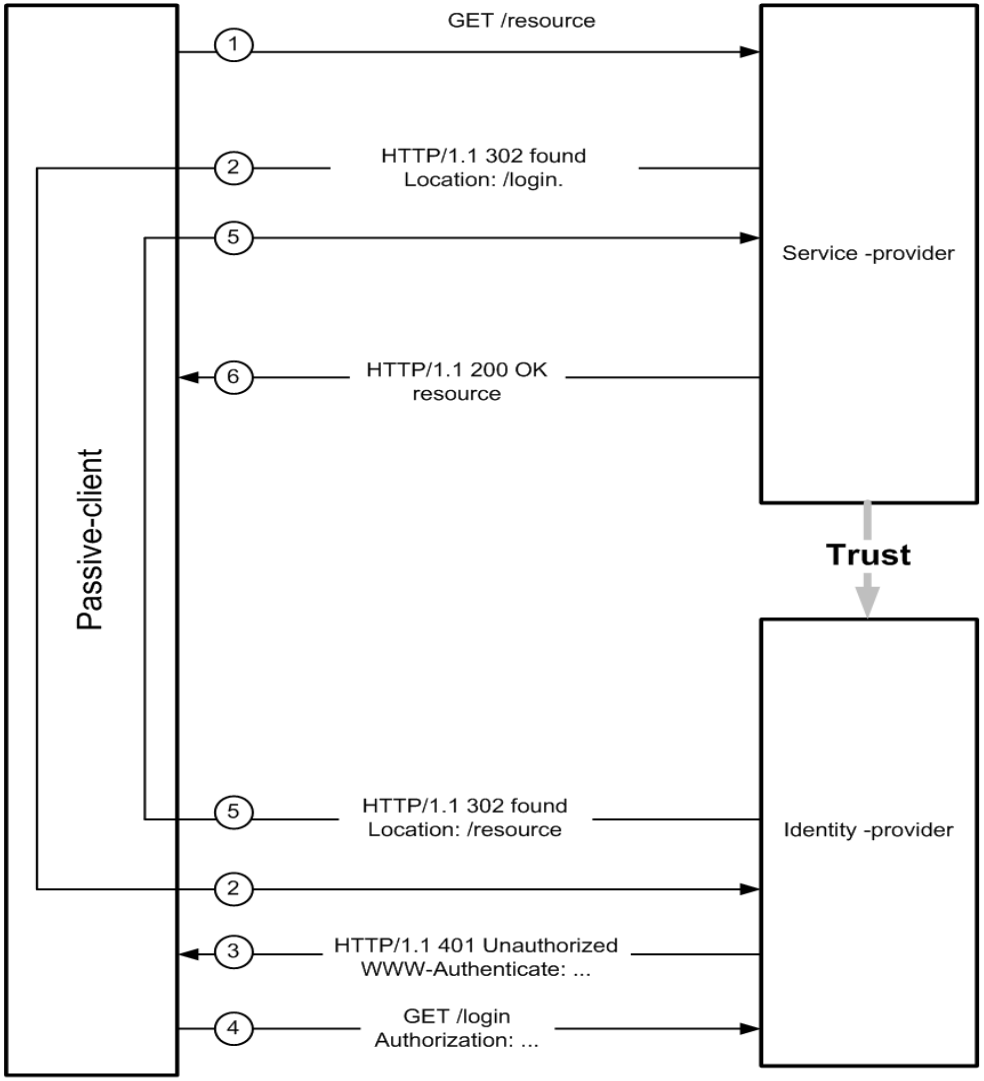


Токен-аутентификация с активным клиентом



Активный клиент – программный код, который может выполнять запрограммированную последовательность действий.

Identity-provider (или Authorization) - сервер, генерирующий token.
Service-provider (или Resource) – сервер, предоставляющий сервис клиенту.
Service-provider и Identity-provider должны иметь общий секретный ключ для подписи/проверки токена.



Токен-аутентификация с пассивным клиентом

Пассивный клиент – такой клиент, который может только отображать страницы, запрошенные пользователем. Пассивным клиентов является браузер.

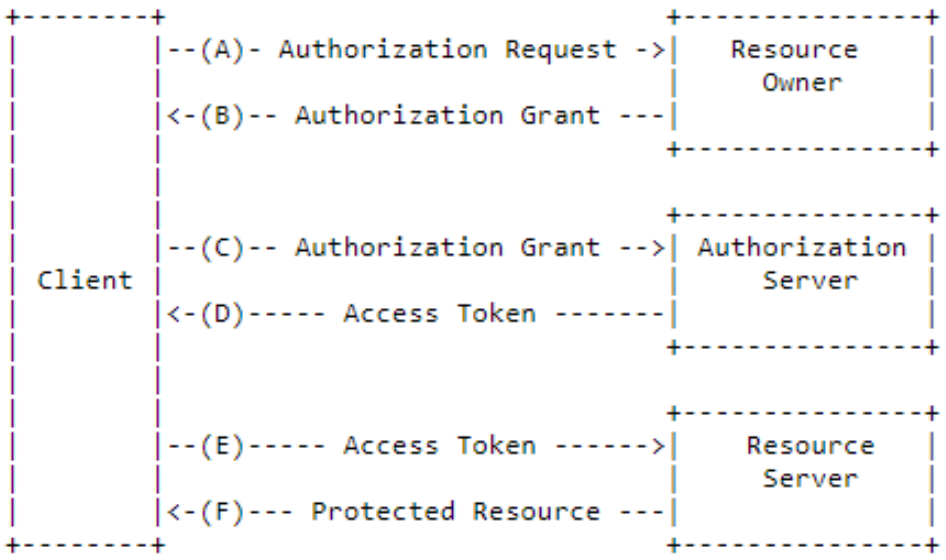
В таком случае аутентификация достигается посредством автоматического перенаправления между провайдерами.

OAuth =

это открытый протокол авторизации, который позволяет предоставить третьей стороне ограниченный доступ к защищённым ресурсам пользователя без необходимости передавать ей (третьей стороне) логин и пароль.

Задача OAuth — сделать так, чтобы пользователь имел возможность работать на сервисе одного приложения с защищенными данными второго приложения, вводя пароль к этим данным исключительно на втором приложении и оставаясь при этом на сайте первого приложения.

Протокол OAuth



- Resource Owner – пользователь, который заходит на Client и дает ему разрешение использовать свои закрытые данные из Authorization Server.
- Client – приложение или интернет сайт, которым пользуется Resource Owner и которое взаимодействует с Authorization Server и Resource Server для получения закрытых данных пользователя.
- Authorization Server – сервер, который проверяет логин/пароль пользователя.
- Resource Server – сервер, который хранит закрытую пользовательскую информацию, которую можно получить с помощью API.

Authorization Server и Resource Server могут быть совмещены в одну систему.

Authorization Grant

=

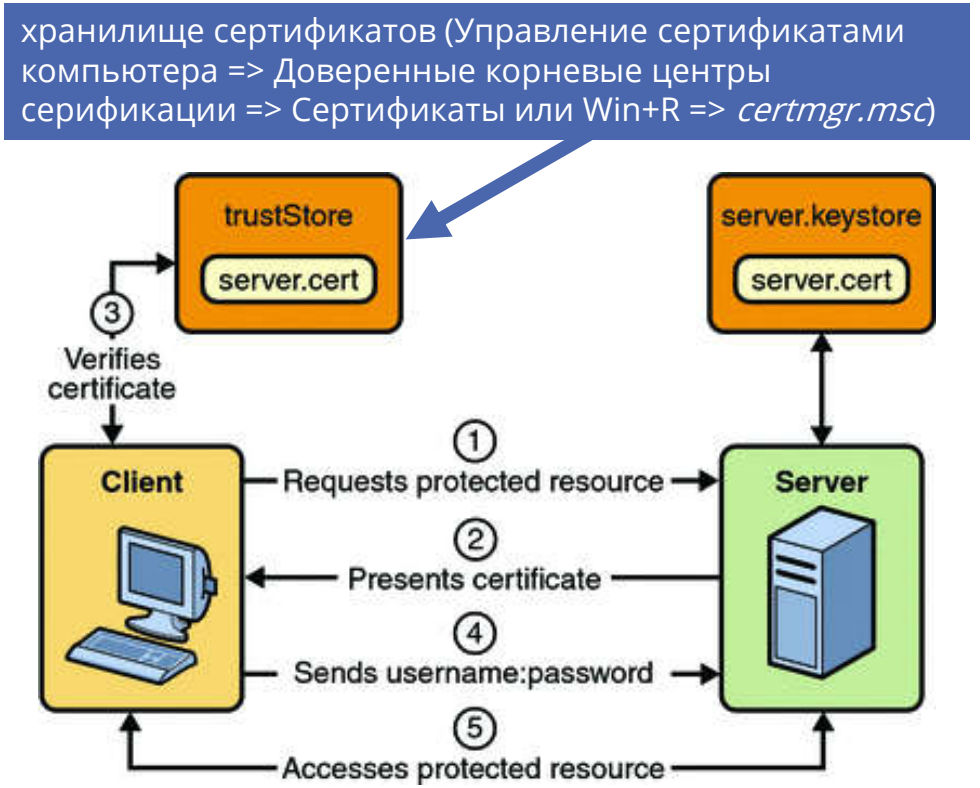
это учетные данные, представляющие авторизацию владельца ресурса (Resource Owner) (для доступа к его защищенным ресурсам), используемую клиентом (Client) для получения токена доступа.

В спецификации определяется четыре типа разрешения (grant): код авторизации (authorization code), неявный (implicit), пароль владельца ресурса (resource owner password) и учетные данные клиента (client credentials).

Аутентификация на основе сертификатов

В веб-приложениях традиционно используют сертификаты стандарта X.509.

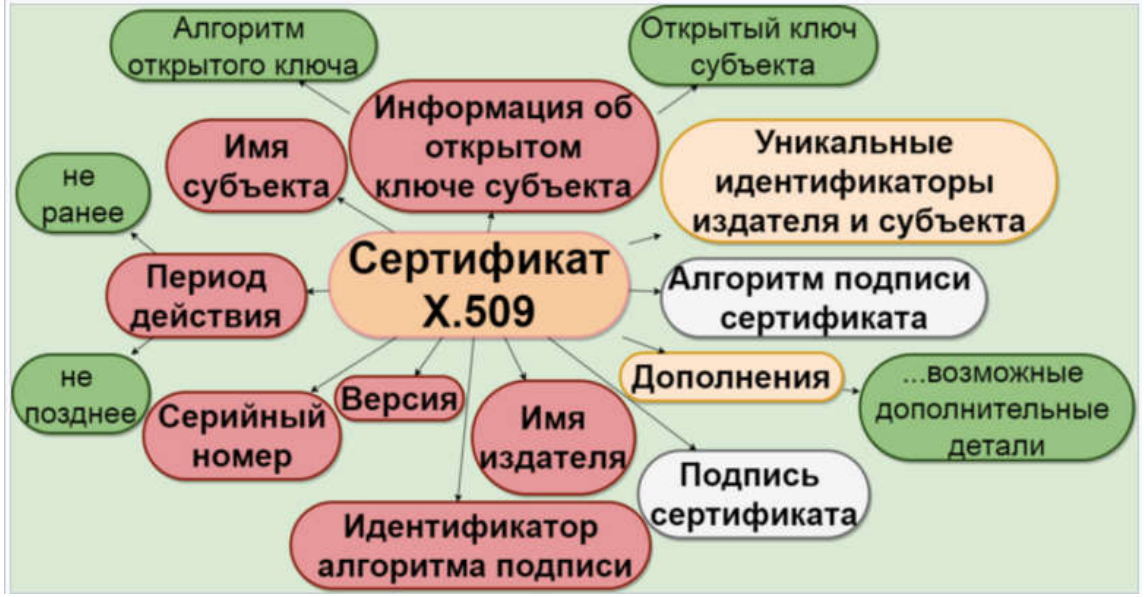
Аутентификация с помощью X.509-сертификата происходит в момент соединения с сервером и является частью протокола SSL/TLS.



X.509 =

стандарт хранения и транспортировки атрибутов безопасности. Сертификаты выдают центры сертификации (Certificate Authority, CA).

X.509-сертификат содержит имя субъекта, которому выдан сертификат, имя издателя, серийный номер, срок действия, открытый ключ субъекта, алгоритм подписи сертификата, подпись сертификата и др.



Аутентификация на основе сертификатов

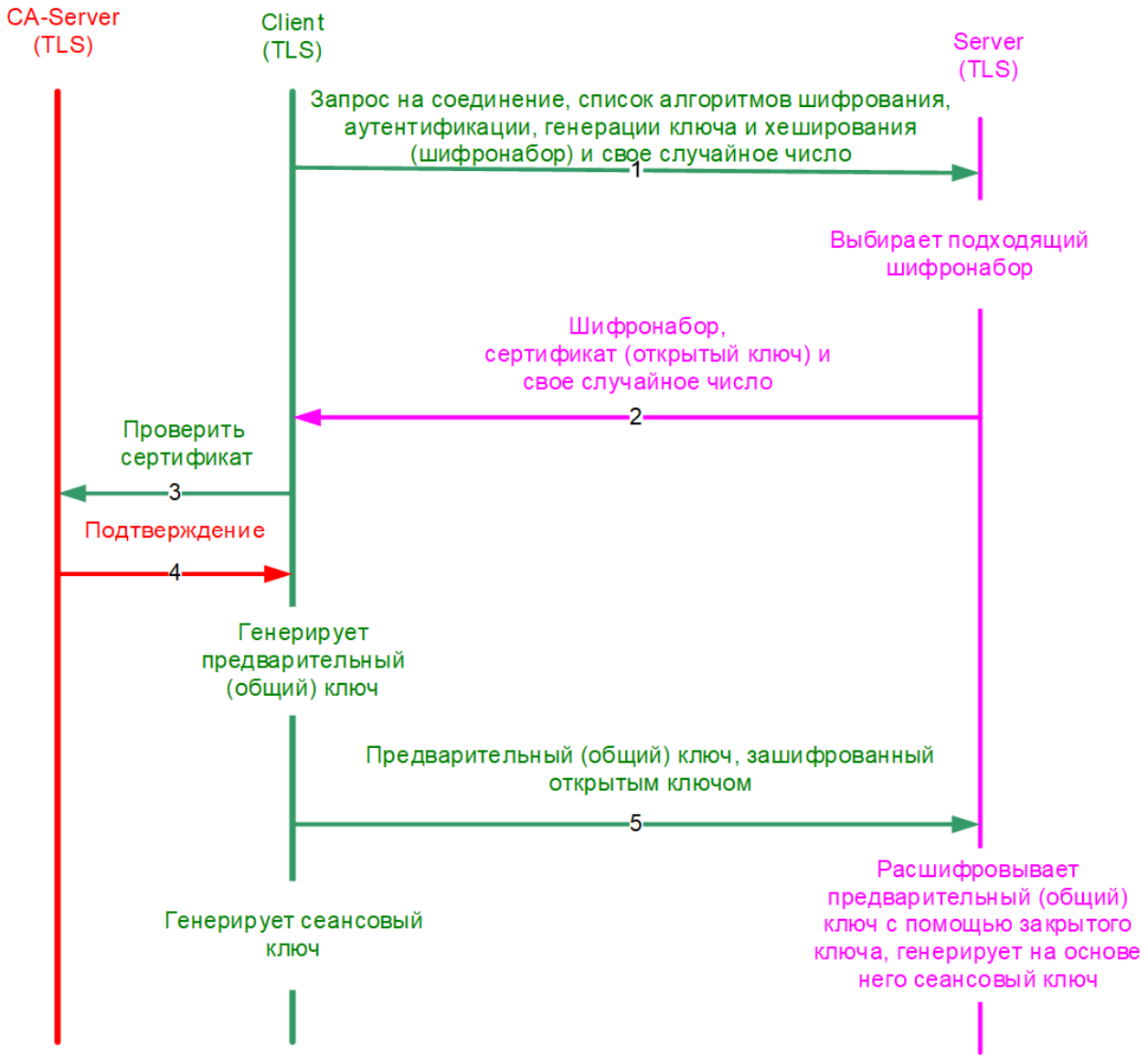
Во время аутентификации происходит проверка сертификата:

- проверка подписи (д.б. подписан CA);
- проверка срока действия (д.б. действительным);
- проверка списков исключения (не д.б. отозван CA);
- после успешной аутентификации веб-приложение может выполнить авторизацию запроса на основании данных сертификата.

TLS
(Transport Layer Security) =

криптографический протокол, обеспечивающий защищённую передачу данных между узлами в сети Интернет (RFC 2246, RFC 5246, RFC 6176, RFC 8446). Последняя версия 1.3. Работает поверх потокового надежного соединения (для ненадежной передачи есть DTSL).

Обеспечивает конфиденциальность (сокрытие информации), целостность (обнаружение подмены) и аутентификацию узлов (проверка подлинности источника сообщений).



TLS handshake (рукопожатие)

Шифронаборы

Algorithms Supported in TLS 1.0 - 1.2 Cipher Suites

Key exchange / agreement	Authentication	Block / Stream Ciphers	Message Authentication
RSA	RSA	RC4	Hash-based MD5
Diffie-Hellman	DSA	Triple DES	SHA Hash Function
ECDH	ECDSA	AES	
SRP		IDEA	
PSK		DES	
		Camellia	

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
ECDHE - Диффи-Хеллман на эллиптических кривых для генерации общего секрета;
ECDSA - аутентификация данных на этапе установления соединения на основе цифровой подписи на эллиптических кривых;
AES_128 – шифрование полезной нагрузки с помощью алгоритма AES с 128-битным ключом в режиме GCM;
SHA256 - для хеширования применяется алгоритм SHA с 256-битным ключом.

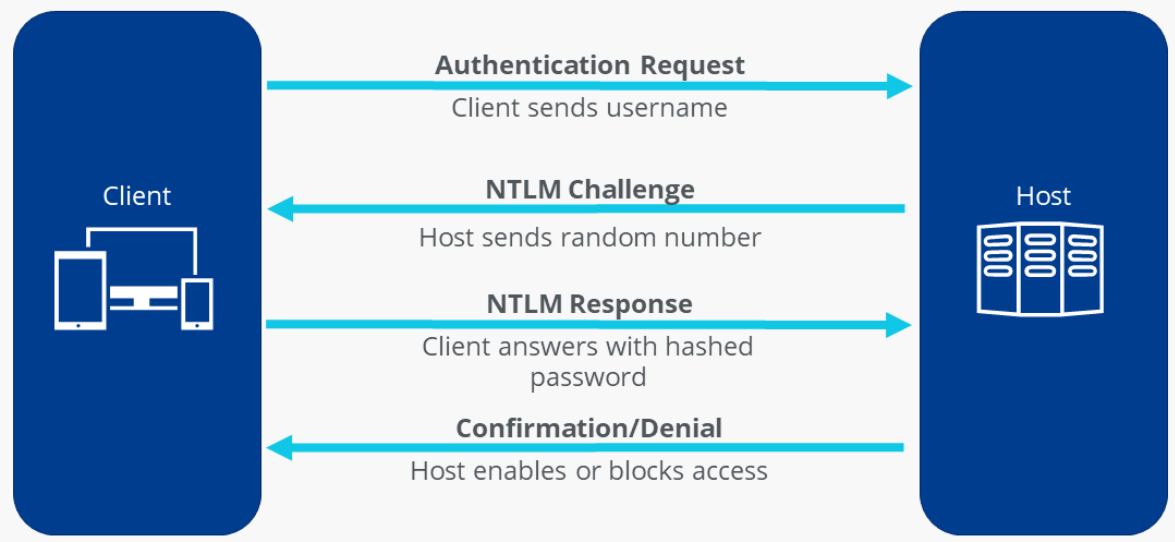
```
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
```

Шифронаборы (Cipher Suites, криптонаборы) состоят из:

- криптосистемы для получения общего секрета;
- криптосистемы для аутентификации сервера;
- шифр для защиты передаваемых данных;
- хеш-функция для кода аутентификации HMAC.

Windows-аутентификация

NTLM: 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.



Windows-аутентификация

Negotiate: клиенту можно выбрать между NTLM и Kerberos-аутентификацией. Kerberos – протокол, основанный на принципе Single Sign-On, клиент и сервер должны находятся в зоне intranet и являться частью домена Windows. Для работы Kerberos клиенту необходимо запросить билет у KDC (центр распределения ключей, хранит базу данных с информацией об учётных записях всех клиентов сети).