Authentication in Revas

7 maggio 2021

What is Authentication?

Authentication is the process of determining whether someone or something is, in fact, who or what it declares itself to be.

Revas uses token based authentication

Token-based authentication is a protocol which allows users to verify their identity, and in return receive a unique access token.
During the life of the token, users then access the website the token has been issued for, rather than having to re-enter credentials each time they go back to the same webpage.
Auth tokens work like a stamped ticket. The user retains access as long as the token remains valid.

What will we see

- How to protect tokens
- How to represent tokens
- How to obtain tokens

How to pass information securely?

Cryptography

Ancient practice and study of techniques for secure communication in the presence of third parties called adversaries.

Very used in military communications.

Common cryptographic algorithms

Symmetric encryption

The simplest and best-known encryption technique. **It uses one key for both encryption and decryption.** The plaintext is encrypted using a key, and the same key is used at the receiving end to decrypt the received ciphertext. The host in the communication process would have **received the key through external means.**

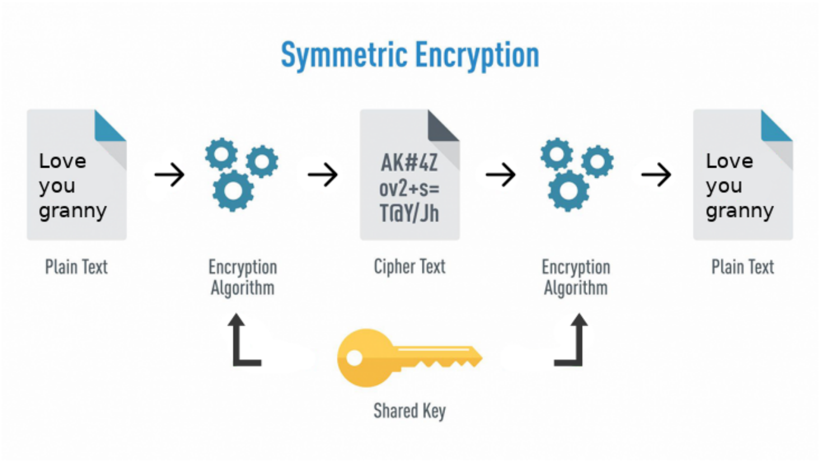
Adavantages

- Fast
- Easy

Disadvantages

- Hard to distribute key
- If key is leaked, you have to replace it and tell everyone the new key again
- Do not have proof of who sent the message

Symmetric encryption



Common cryptographic algorithms

Asymmetric encryption

More secure than symmetric encryption as it uses two keys for the process. The public key used for encryption is available to everyone but the private key is not disclosed. When a message is encrypted using a public key, it can only be decrypted using a private key. However, when a message is encrypted using a private key, it can be decrypted using a public key.

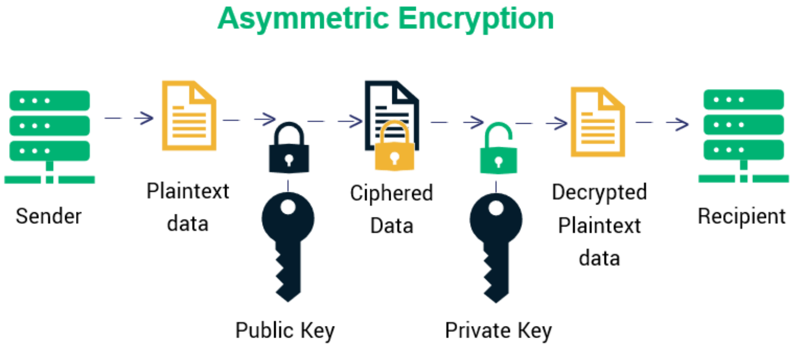
Adavantages:

- Public key can be safely shared
- Proof that the message was signed by whoever is in possession of the private key

Disadvantages:

- Slow

Asymmetric encryption



Revas Tokens

Token can be signed and/or encrypted, in Revas it’s sufficient to digitally signed them. Cryptograhy is used to digitally sign them.

Digital Signature

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity)

Signing algorithms

HS256 (HMAC with SHA-256): A symmetric algorithm
RS256 (RSA Signature with SHA-256): An asymmetric algorithm

Most secure signing algorithm

The most secure practice, is to use RS256 because:

- With RS256, you are sure that only the holder of the private key (Auth0) can sign tokens, while anyone can check if the token is valid using the public key.
- With RS256, if the private key is compromised, you can implement key rotation without having to re-deploy your application or API with the new secret (which you would have to do if using HS256).

Access Token in JWT format

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

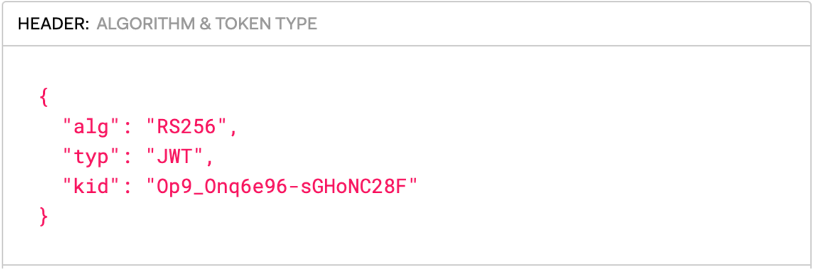
JWT structure

Structure

- Header
- Paylaod
- Signature

JWT structure: Header

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA. It is Base64Url encoded.

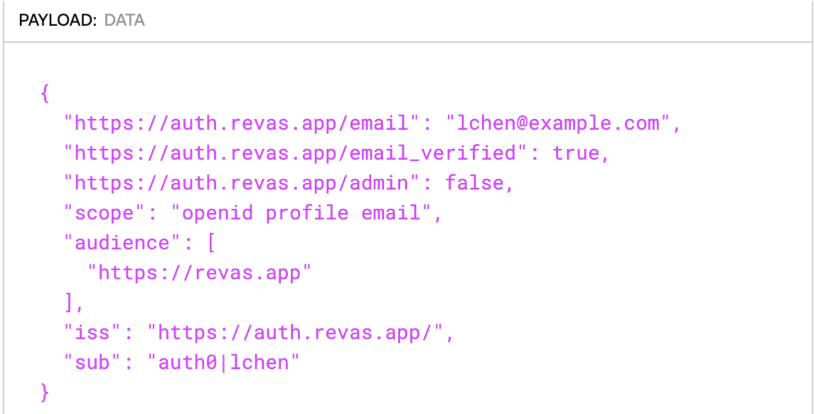


HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "Op9_Onq6e96-sGHoNC28F"
}
```

JWT structure: Payload

Contains the claims which are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.



PAYLOAD: DATA

```
{
  "https://auth.revas.app/email": "lchen@example.com",
  "https://auth.revas.app/email_verified": true,
  "https://auth.revas.app/admin": false,
  "scope": "openid profile email",
  "audience": [
    "https://revas.app"
  ],
  "iss": "https://auth.revas.app/",
  "sub": "auth0|lchen"
}
```

JWT structure: Signature

The purpose of the signature is to protect/validate the integrity of the data (e.g. the user id) that the server sends and eventually gets back.
In the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

How to sign tokens

Symmetric key example

- What we do is we take the **Header**, the **Payload** and we add also a **secret password**, and then we hash everything together.
- The result of that function can only be reproduced by someone in possession of all the above 3 things.

How to validate tokens

Symmetric key example

- To check the signature we simply take the JWT header + the payload and hash it together with the password.
- And if we get back the same hash as in the signature it means that the token must be valid, because only someone with the password could have come up with that signature

Encryption and Hashing

Encryption is a two-way function; what is encrypted can be decrypted with the proper key. Reversible key systems can be used for confidentiality, integrity, and non-repudiation.

Hashing, however, is a one-way function that scrambles plain text to produce a unique message digest. Irreversible key systems can only be used for integrity and non-repudiation.

Signature-based communication relies on a one-way hashing function.

JWT: self-contained

The key property of JWTs is that in order to confirm if they are valid we only need to look at the token itself. We don’t have to contact a third-party service or keep JWTs in-memory between requests to confirm that the claim they carry is valid. The signed JWT acts effectively as a temporary user credential, that replaces the permanent credential which is the username and password combination.

JWT best practices

- Use secure connection when transferring tokens;
- Never transfer users’ sensitive data in the tokens;
- Limit JWT lifespan and use “refresh tokens” mechanism;
- Use long key phrases (Mining the key for signature symmetric algorithm);
- Keep a white list of authorised signature algorithms on the application side;
- Work, ideally, with one signature algorithm only;
- Choose well-known and reliable libraries for JWT operation;
- Always validate and sanitise the data received from users.

Where to store JWT?

It is not recommended to store the JWT in the browser local storage:
It will remain if the user closes the browser so the session can be restored until the JWT expires. Any JavaScript code on your page can access local storage: it has no data protection whatsoever. It can’t be used by web workers Storing JWT in session cookie may be the solution

OAuth2.0

Is a security standard that defines an **authorization** framework to allow access to specific information without sharing passwords.

OAuth Roles

- **Resource Owner**: Entity that can grant access to a protected resource. Typically, this is the end-user. (Mike)
- **Client**: Application requesting access to a protected resource on behalf of the Resource Owner. (Revas Platform)
- **Authorization Server**: Server that authenticates the Resource Owner and issues access tokens after getting proper authorization. For Revas is, Auth0.
- **Resource Server** : Server hosting the protected resources. This is the API you want to access. (Revas API)

OAuth Scopes

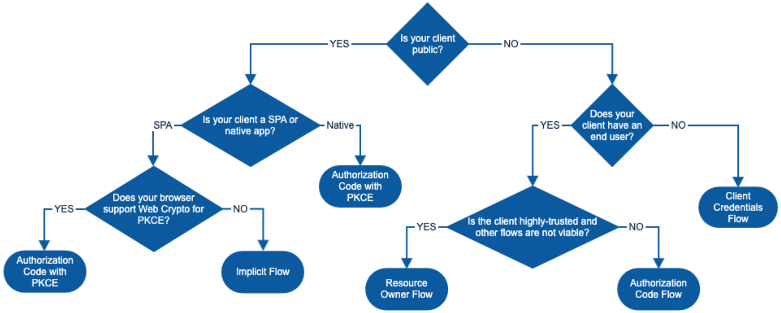
They are used to specify exactly the reason for which access to resources may be granted.

OAuth Access Tokens

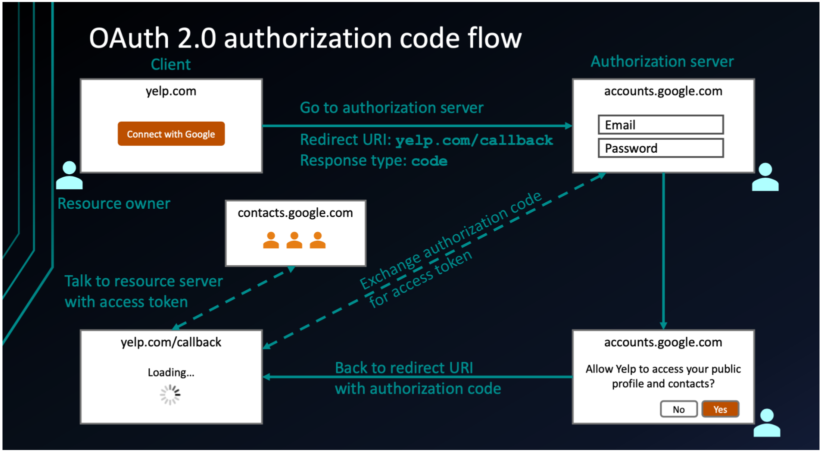
An Access Token is a piece of data that represents the authorization to access resources on behalf of the end-user.
Refresh tokens are used to obtain a renewed access token without having to re-authenticate the user. They are better than long lived access token because they can be revoked in case on misuse.

OAuth Flows

OAuth defines **flows**, also called grants, which are mechanisms for a client to get credentials (an access token) from an Authorization Server to access a protected resource.

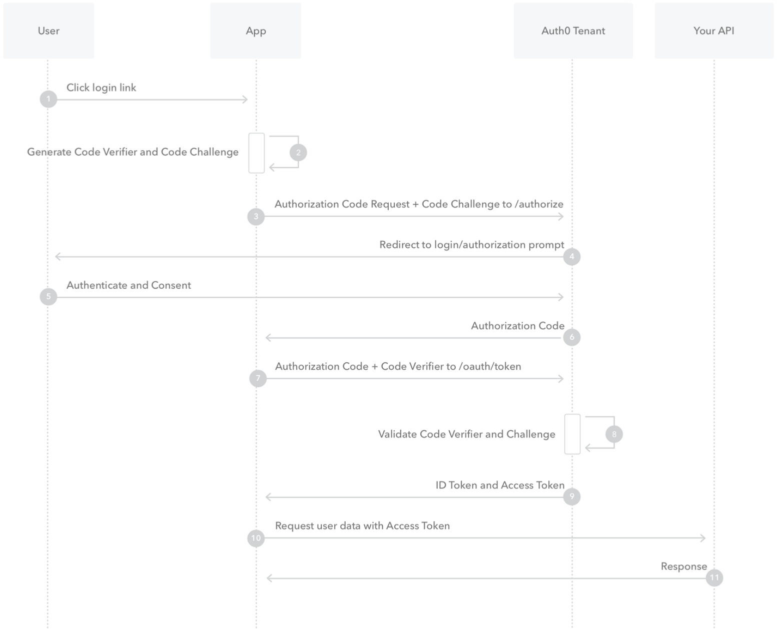

Is your client public?
YES
NO
Is your client a SPA or native app?
SPA
Native
Authorization Code with PKCE
Does your browser support Web Crypto for PKCE?
YES
NO
Authorization Code with PKCE
Implicit Flow
Does your client have an end user?
YES
NO
Client Credentials Flow
Is the client highly-trusted and other flows are not viable?
YES
NO
Resource Owner Flow
Authorization Code Flow

Authorization Code Flow


OAuth 2.0 authorization code flow
Client
Authorization server
yelp.com
Connect with Google
Go to authorization server
Redirect URI: yelp.com/callback
Response type: code
accounts.google.com
Email
Password
Resource owner
contacts.google.com
Exchange authorization code for access token
Talk to resource server with access token
yelp.com/callback
Loading...
Back to redirect URI with authorization code
accounts.google.com
Allow Yelp to access your public profile and contacts?
No
Yes

PKCE: Proof key for code exchange

All the principle of code flow still applies. The PKCE makes this more safe for native and web applications (public clients) by generating a code exchange key, that ensures that the authorization request and the token request is done by the **same client** (aka not intercepted by a man in the middle). Because web applications can’t store secrets, PKCE allows for creating a secret dynamically at the beginning of the authorization flow as a contrast to the static secret in code flow (can only be used for private/server clients).


User
App
Auth0 Tenant
Your API
1 Click login link
2 Generate Code Verifier and Code Challenge
3 Authorization Code Request + Code Challenge to /authorize
4 Redirect to login/authorization prompt
5 Authenticate and Consent
6 Authorization Code
7 Authorization Code + Code Verifier to /oauth/token
8 Validate Code Verifier and Challenge
9 ID Token and Access Token
10 Request user data with Access Token
11 Response

OpenID Connect for authentication

In the world of consent and authorization, identity was missing. OIDC adds

- ID Token
- UserInfo endpoint for getting more user information
- Standard set of scopes
- Standardized implementation

The OIDC looks like OAuth the only difference are in the initial request, a specific scope of openid is used and in the final exchange the client receives both an access token and an ID token.

Thanks to OAuth and OIDC

- Authorizing third party services to access information in another service. For example letting Facebook post your wonderful score on Candy Crush.
- Single Sign-on (SSO) occurs when a user logs in to one application and is then signed in to other applications automatically, regardless of the platform. For example, if you log in to a Google service such as Gmail, you are automatically authenticated to YouTube, AdSense, Google Analytics, and other Google apps.
- Sign-in to other service using an Identity Providers without the need to create another account.

Identity Provider

An identity provider is a system entity that creates, maintains, and manages identity information for principals and also provides authentication services to relying applications.
A single, consistent identity that can be used across platforms, applications and networks is called a federated identity.
Most used Identity Providers

- Google
- Facebook
- Instagram
- Fitbit.
- Microsoft

How do OpenID providers authenticate users?

OpenID Connect leaves this entirely up to the particular IdP. Implementors may use any method for authenticating users, or combine two methods for stronger security (2FA).

- Username / password
- Hardware tokens
- SMS confirmation
- Biometrics
- Etc.

To summarize

Revas uses

- Auth0 as Identity Provider delegating most of the authentication managment
- RSA with SHA-256 to sign tokens which uses the asymmetric algoritm, the most secure one
- PCKE flow to obtain the access token, most secure flow for now…
- Passwordless to handle user credentials