**What is JSON Web Token?**

JSON Web Token (JWT) is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) 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**.

Let's explain some concepts of this definition further.

* **Compact**: Because of its smaller size, JWTs can be sent through an URL, POST parameter, or inside an HTTP header. Additionally, the smaller size means transmission is fast.
* **Self-contained**: The payload contains all the required information about the user, avoiding the need to query the database more than once.

**When should you use JSON Web Tokens?**

Here are some scenarios where JSON Web Tokens are useful:

* **Authentication**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
* **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties, because as they can be signed, for example using public/private key pairs, you can be sure that the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

**What is the JSON Web Token structure?**

JSON Web Tokens consist of three parts separated by dots (.), which are:

* Header
* Payload
* Signature

Therefore, a JWT typically looks like the following.

xxxxx.yyyyy.zzzzz

Let's break down the different parts.

**Header**

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

For example:

{

"alg": "HS256",

"typ": "JWT"

}

Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

**Payload**

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: *reserved*, *public*, and *private*claims.

* **Reserved claims**: These is a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub**(subject), **aud** (audience), and others.

Notice that the claim names are only three characters long as JWT is meant to be compact.

* **Public claims**: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
* **Private claims**: These are the custom claims created to share information between parties that agree on using them.

An example of payload could be:

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

The payload is then **Base64Url** encoded to form the second part of the JSON Web Token.

**Signature**

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

**Putting all together**

The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.

## How do JSON Web Tokens work?

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally (typically in local storage, but cookies can be also used), instead of the traditional approach of creating a session in the server and returning a cookie.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the **Authorization** header using the **Bearer** schema. The content of the header should look like the following:

Authorization: Bearer <token>

This is a stateless authentication mechanism as the user state is never saved in server memory. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. As JWTs are self-contained, all the necessary information is there, reducing the need to query the database multiple times.

This allows you to fully rely on data APIs that are stateless and even make requests to downstream services. It doesn't matter which domains are serving your APIs, so Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.

The following diagram shows this process:



## Why should we use JSON Web Tokens?

Let's talk about the benefits of **JSON Web Tokens (JWT)** when compared to **Simple Web Tokens (SWT)**and**Security Assertion Markup Language Tokens (SAML).**

As JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, SWT can only be symmetrically signed by a shared secret using the HMAC algorithm. However, JWT and SAML tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Regarding usage, JWT is used at Internet scale. This highlights the ease of client-side processing of the JSON Web token on multiple platforms, especially mobile.