**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** or **ECDSA**.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on *signed* tokens. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

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

Here are some scenarios where JSON Web Tokens are useful:

* **Authorization**: 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 JWTs can be signed—for example, using public/private key pairs—you can be sure 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?**

In its compact form, 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 signing 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 data. There are three types of claims: *registered*, *public*, and *private* claims.

* [**Registered claims**](https://tools.ietf.org/html/rfc7519#section-4.1): These are 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](https://tools.ietf.org/html/rfc7519#section-4.1).

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

* [**Public claims**](https://tools.ietf.org/html/rfc7519#section-4.2): 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](https://www.iana.org/assignments/jwt/jwt.xhtml) or be defined as a URI that contains a collision resistant namespace.
* [**Private claims**](https://tools.ietf.org/html/rfc7519#section-4.3): These are the custom claims created to share information between parties that agree on using them and are neither *registered* or *public* claims.

An example 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.

Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted.

**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 the message wasn't changed along the way, and, 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. 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** or **ECDSA**.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on *signed* tokens. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

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

Here are some scenarios where JSON Web Tokens are useful:

* **Authorization**: 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 JWTs can be signed—for example, using public/private key pairs—you can be sure 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?**

In its compact form, 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 signing 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 data. There are three types of claims: *registered*, *public*, and *private* claims.

* [**Registered claims**](https://tools.ietf.org/html/rfc7519#section-4.1): These are 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](https://tools.ietf.org/html/rfc7519#section-4.1).

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

* [**Public claims**](https://tools.ietf.org/html/rfc7519#section-4.2): 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](https://www.iana.org/assignments/jwt/jwt.xhtml) or be defined as a URI that contains a collision resistant namespace.
* [**Private claims**](https://tools.ietf.org/html/rfc7519#section-4.3): These are the custom claims created to share information between parties that agree on using them and are neither *registered* or *public* claims.

An example 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.

Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted.

**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 the message wasn't changed along the way, and, 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.

**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** or **ECDSA**.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on *signed* tokens. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

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

Here are some scenarios where JSON Web Tokens are useful:

* **Authorization**: 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 JWTs can be signed—for example, using public/private key pairs—you can be sure 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?**

In its compact form, 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 signing 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 data. There are three types of claims: *registered*, *public*, and *private* claims.

* [**Registered claims**](https://tools.ietf.org/html/rfc7519#section-4.1): These are 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](https://tools.ietf.org/html/rfc7519#section-4.1).

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

* [**Public claims**](https://tools.ietf.org/html/rfc7519#section-4.2): 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](https://www.iana.org/assignments/jwt/jwt.xhtml) or be defined as a URI that contains a collision resistant namespace.
* [**Private claims**](https://tools.ietf.org/html/rfc7519#section-4.3): These are the custom claims created to share information between parties that agree on using them and are neither *registered* or *public* claims.

An example 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.

Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted.

**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 the message wasn't changed along the way, and, 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.