**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**.

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**.

**JSON Web Token structure**

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

1. **Header**
2. **Payload**
3. **Signature**

Therefore, a JWT typically looks like the following.

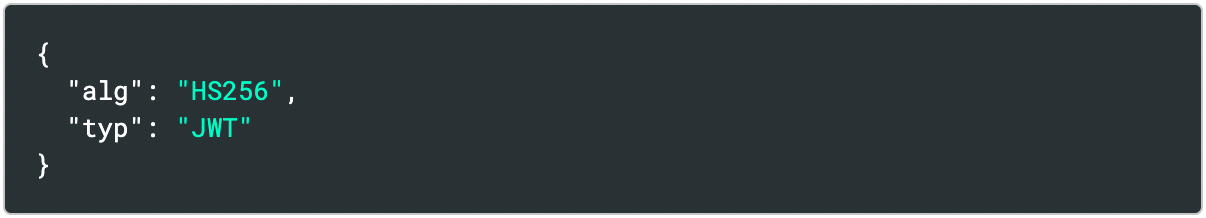


Let's break down the different parts.

**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:



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

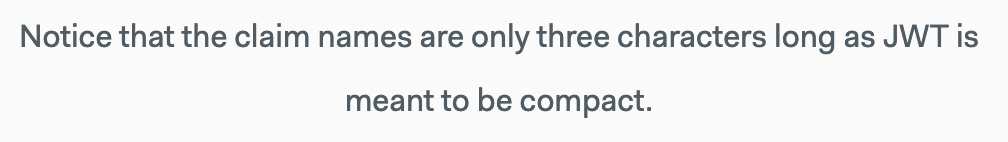
### 

### 

**Payload**

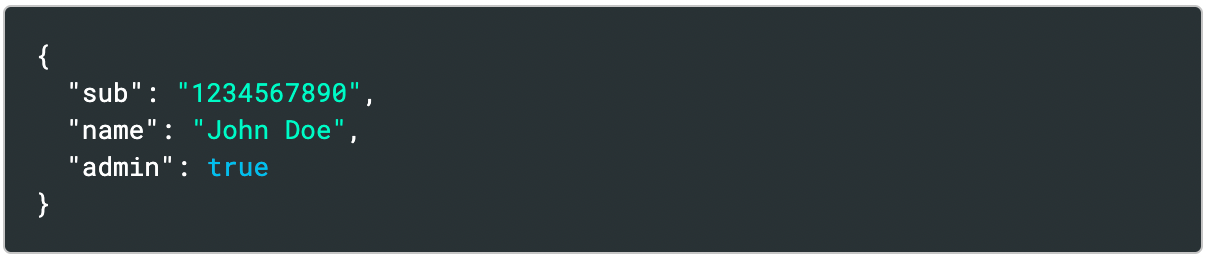
The second part of the token is the payload, which **contains the claims**. **Claims are statements about an entity** (typically, the user) and additional data. There are **three types of claims**: registered, public, and private claims.

1. [**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**), ...

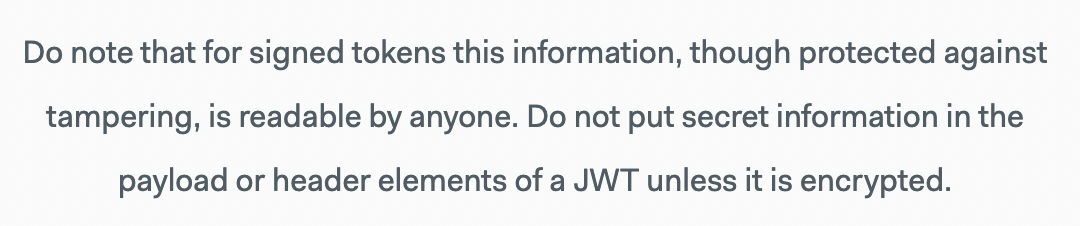


1. [**Public claims**](https://tools.ietf.org/html/rfc7519#section-4.2):
   * These can be **defined at will** by those using JWTs. But **to avoid collisions** they **should be defined in the** [**IANA JSON Web Token Registry**](https://www.iana.org/assignments/jwt/jwt.xhtml) or be **defined as a URI** **that contains a collision resistant namespace**.
2. [**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:



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



### **Signature**

**To create the signature** part we 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:

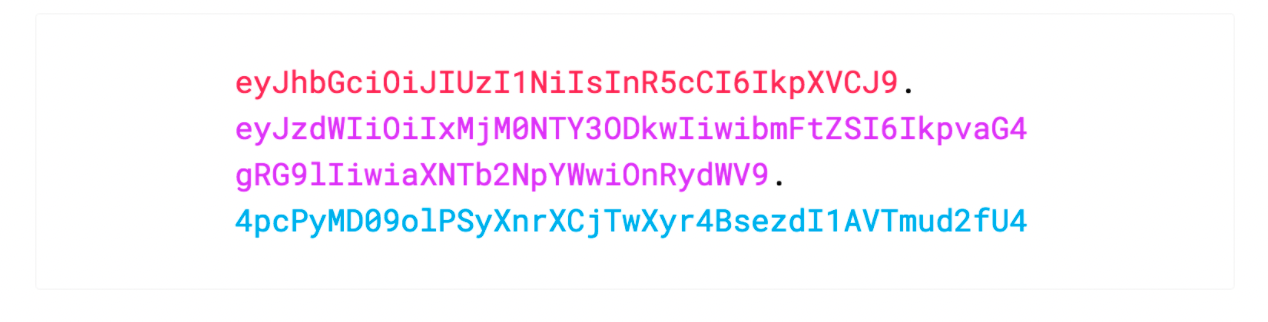


The **signature is used to verify the message wasn't changed along the way**, and, **in the case of tokens signed with a private key**, it can **also verify that the sender of the JWT** is who it says it is.

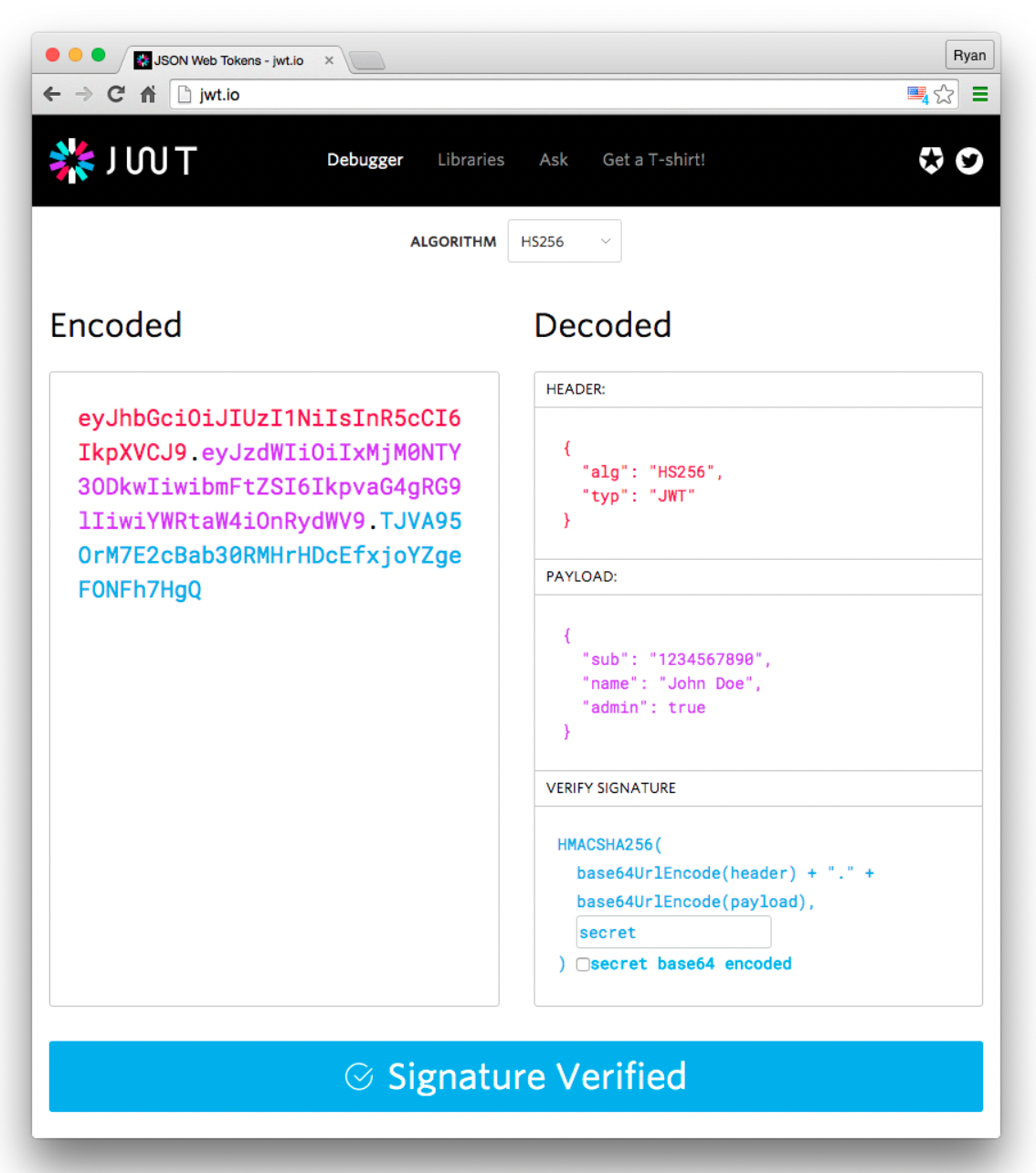
### **Putting all together**

The **output is three Base64-URL 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**.

The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.



If we want to play with JWT and put these concepts into practice, we can use [jwt.io Debugger](https://jwt.io/#debugger-io) to decode, verify, and generate JWTs.



**How JSON Web Tokens work**

In authentication, **when the user successfully logs in using their credentials**, **a JSON Web Token will be returned**. Since tokens are credentials, great care must be taken to prevent security issues. In general, we should not keep tokens longer than required.

We also [**should not store sensitive session data in browser storage due to lack of security**](https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html#local-storage).

**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:

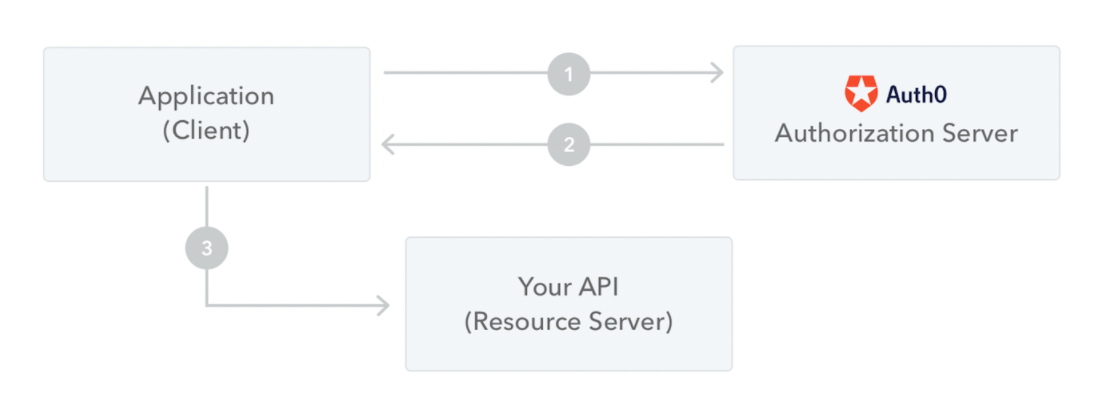


This can be, in certain cases, a **stateless authorization mechanism**. The **server's protected routes** will **check for a valid JWT in the Authorization header**, and **if it's present**, the **user will be allowed to access protected resources**.

Note that **if we send JWT tokens through HTTP headers**, **we should** **try to prevent them from getting too big**. **Some servers don't accept more than 8 KB in headers**. If we are trying to embed too much information in a JWT token, like by including all the user's permissions, we may need an alternative solution, like [Auth0 Authorization](https://fga.dev/).

If the **token is sent in the Authorization header**, **Cross-Origin Resource Sharing (CORS)** **won't be an issue** as it doesn't use cookies.

The following diagram shows how a JWT is obtained and used to access APIs or resources:



1. The **application or client requests authorization to the authorization server**. This is performed through one of the different authorization flows. For example, a typical [OpenID Connect](http://openid.net/connect/) compliant web application will go through the /oauth/authorize endpoint using the [authorization code flow](http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth).
2. **When the authorization is granted, the authorization server returns an access token to the application**.
3. The **application uses the access token to access a protected resource** (like an API).

Do note that **with signed tokens**, **all the information contained within the token is exposed to users** **or other parties**, even though they are unable to change it. This means we should not put secret information within the token.

**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**.

