**JSON Web Tokens**

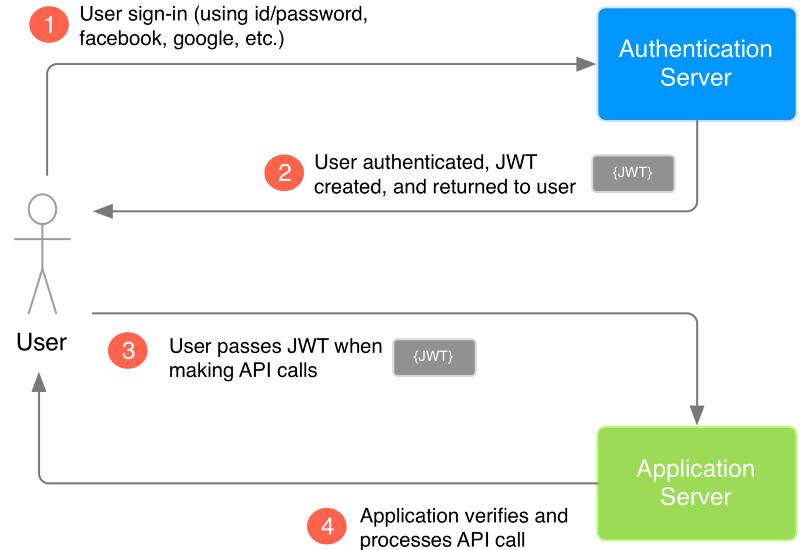
A JSON Web Token (JWT) is a [JSON object](http://www.w3schools.com/json/)  as a safe way to represent a set of information between two parties.

The token is composed of a header, a payload, and a signature.

JWT is just a string with the following format:

header.payload.signature

It should be noted that a double quoted string is actually considered a valid JSON object



**The authentication server will provide the JWT to the user. With the JWT, the user can then safely communicate with the application.**

* In this example, the user first signs into the authentication server using the authentication server’s login system (e.g. username and password, Facebook login, Google login, etc).The authentication server then creates the JWT and sends it to the user.
* When the user makes API calls to the application, the user passes the JWT along with the API call.In this setup, the application server would be configured to verify that the incoming JWT are created by the authentication server (the verification process will be explained in more detail later).
* So, when the user makes API calls with the attached JWT, the application can use the JWT to verify that the API call is coming from an authenticated user.
* **Step 1. Create the HEADER**

The header component of the JWT contains information about how the JWT signature should be computed. The header is a JSON object in the following format:

|  |
| --- |
| { |
|  | "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" |
|  | } |

In this JSON, the value of the “typ” key specifies that the object is a JWT, and the value of the “alg” key specifies which hashing algorithm is being used to create the JWT signature component.

* **Step 2. Create the PAYLOAD**

The payload component of the JWT is the data that‘s stored inside the JWT (this data is also referred to as the “claims” of the JWT).

In our example, the authentication server creates a JWT with the user information stored inside of it, specifically the user ID.

The data inside the payload is referred to as the “claims” of the token.

In our example, we are only putting one claim into the payload. You can put as many claims as you like.

There are several different standard claims for the JWT payload, such as “iss” the issuer, “sub” the subject, and “exp” the expiration time.

Keep in mind that the size of the data will affect the overall size of the JWT.

* **Step 3. Create the SIGNATURE**

The signature is computed using the following pseudo code:

// signature algorithm

data = base64urlEncode( header ) + “.” + base64urlEncode( payload )

signature = Hash( data, secret );

The algorithm then joins the resulting encoded strings together with a period (.) in between them. In our pseudo code, this joined string is assigned to data.

To get the JWT signature, the data string is hashed with the secret key using the hashing algorithm specified in the JWT header.

In our example, both the header, and the payload are base64url encoded as:

// header

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

// payload

eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ

Then, using the joined encoded header and payload, and applying the specified signature algorithm(HS256) on the data string with the secret key set as the string “secret”, we get the following JWT Signature:

// signature

-xN\_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM

* **Step 4. Put All Three JWT Components Together**

Now that we have created all three components, we can create the JWT. Remembering the **header.payload.signature** structure of the JWT, we simply need to combine the components, with periods (.) separating them. **We use the base64url encoded versions of the header and of the payload, and the signature we arrived at in step 3.**

// JWT Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.-xN\_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM

* **How does JWT protect our data?**

**It is important to understand that the purpose of using JWT is NOT to hide or obscure data in any way. The reason why JWT are used is to prove that the sent data was actually created by an authentic source.**

* As demonstrated in the previous steps, the data inside a JWT is **encoded** and **signed**, not **encrypted**. The purpose of encoding data is to transform the data’s structure.
* Signing data allows the data receiver to verify the authenticity of the source of the data. So encoding and signing data does NOT secure the data. On the other hand, the main purpose of encryption is to secure the data and to prevent unauthorized access.

**Since JWT are signed and encoded only, and since JWT are not encrypted, JWT do not guarantee any security for sensitive data.**

* **Step 5. Verifying the JWT**

In our simple 3 entity example, we are using a JWT that is signed by the HS256 algorithm where only the authentication server and the application server know the secret key.

The application server receives the secret key from the authentication server when the application sets up its authentication process.

Since the application knows the secret key, when the user makes a JWT-attached API call to the application, the application can perform the same signature algorithm as in Step 3 on the JWT.

The application can then verify that the signature obtained from it’s own hashing operation matches the signature on the JWT itself (i.e. it matches the JWT signature created by the authentication server).

If the signatures match, then that means the JWT is valid which indicates that the API call is coming from an authentic source.

Otherwise, if the signatures don’t match, then it means that the received JWT is invalid, which may be an indicator of a potential attack on the application.

So by verifying the JWT, the application adds a layer of trust between itself and the user.

**IT SHOULD ALSO BE NOTED THAT JWT SHOULD BE SENT OVER HTTPS CONNECTIONS (NOT HTTP). HAVING HTTPS HELPS PREVENTS UNAUTHORIZED USERS FROM STEALING THE SENT JWT BY MAKING IT SO THAT THE COMMUNICATION BETWEEN THE SERVERS AND THE USER CANNOT BE INTERCEPTED.**