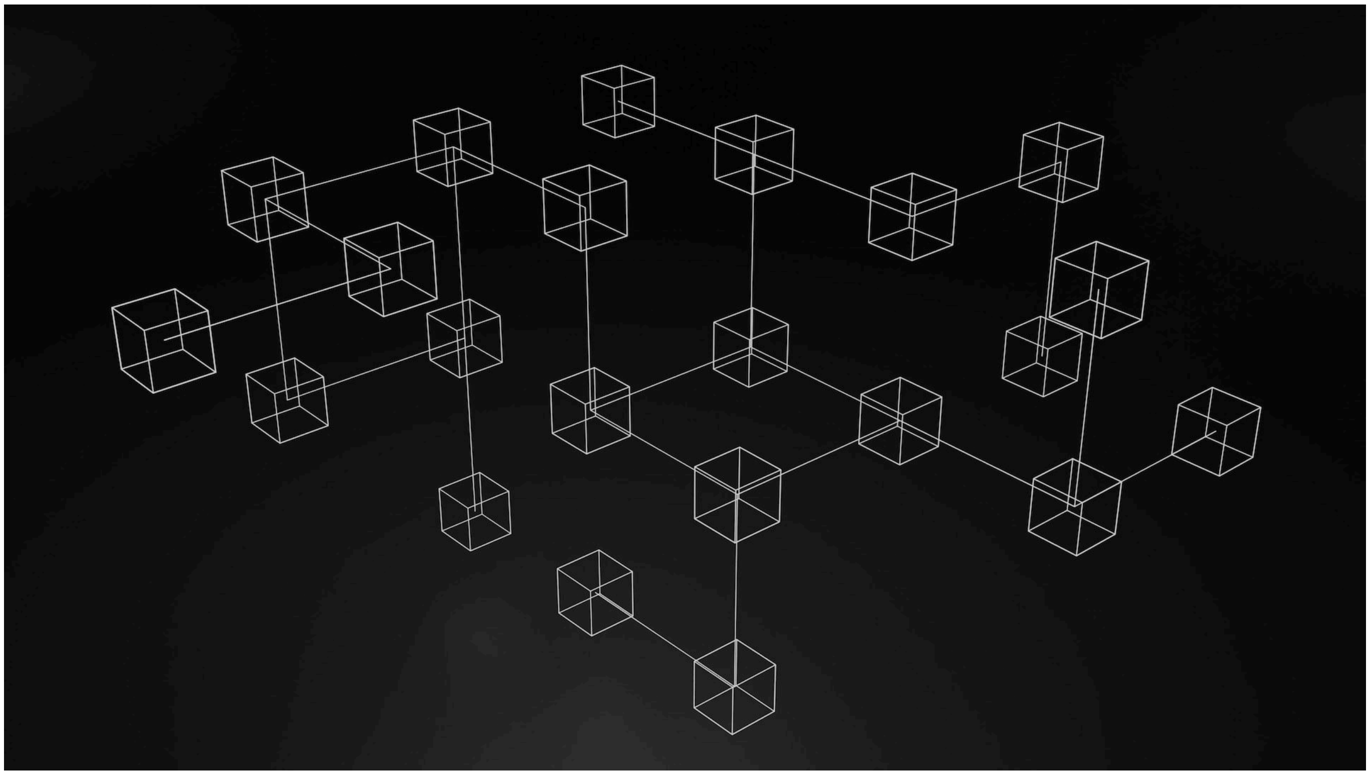
DECEMBER 9, 2022 / #AUTHENTICATION

How to Sign and Validate JSON Web Tokens – JWT Tutorial

Kristofer Koishigawa



securely creating and sending data between two parties, usually a client and a server.

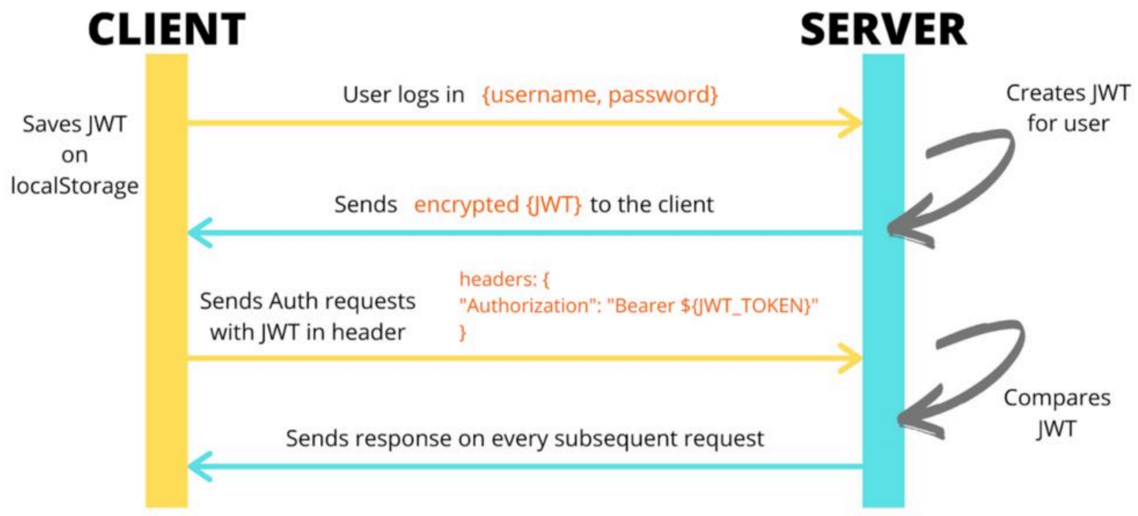
If you've ever signed in to a site like freeCodeCamp with your Google or GitHub account, there's a good chance that you're already using a JWT.

In this article, we'll go over how JWTs are used, then dig into what JWTs are, and how they can securely transmit data through the signature and validation process.

How JWTs Are Used

JWTs are usually used to manage user sessions on a website. While they're an important part of the token based authentication process, JWTs themselves are used for authorization, not authentication.

Here's a good overview of how token based authentication works:


CLIENT
SERVER
Saves JWT on localStorage
User logs in {username, password}
Creates JWT for user
Sends encrypted {JWT} to the client
Sends Auth requests with JWT in header
headers: {
"Authorization": "Bearer ${JWT_TOKEN}"
}
Compares JWT
Sends response on every subsequent request

Source

When you sign in to a site with a username and password, or with a third party method like Google, you're proving who you are with those sensitive details or access. This process is called **authentication**.

Once you're signed in, the site's server sends back a JWT that allows you access to things like your settings page, shopping cart, and so on. This process is called **authorization**.

You send your JWT to the server with each request. When the server receives it, it generates a signature using some data from your JWT, verifies it, and if your JWT is valid, it sends back a response.

What are JWTs?

At their core, JWTs are just bits of encoded JSON data with a cryptographic signature at the end.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibn
```

Each JWT is made up of three segments, each separated by a dot (`.`). These three segments are the header, payload, and signature.

If you copy and paste that JWT into the JWT.io Debugger, you can see the decoded versions of those three segments.

Header Segment

The header segment of a JWT contains information about the algorithm and token type.

Here's the header segment of the example JWT token above:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

The header segment is base 64 URL encoded, and decodes to the following JSON object:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

used, though RSA is also common.

`"typ"` is the type of token the segmented string is, which in this case is JWT.

Payload Segment

The payload segment of a JWT contains registered claims or identifying information, usually for a user.

Here's the payload segment of the example JWT token above:

```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlF1aW5jeSBMYXJzb24iLCJpYXQiOjE
```

The payload segment is also base 64 URL encoded, and decodes to the following JSON object:

```
{
  "sub": "1234567890",
  "name": "Quincy Larson",
  "iat": 1516239022
}
```

Since JWTs are usually used as part of the authentication method for sites, the payload segment usually contains identifying information for a user. These claims fall into three categories: registered, public, and private.

Public claims are optional claims, usually from the IANA JSON Web Token Registry.

Private claims are optional, and are any claims that don't fall under the registered or public claims categories.

`"sub"` is the subject of the JWT, and is usually a unique identifying string for a user in an application, usually an email address, username, or id. Subjects are registered claims.

`"name"` is the full name of the user who was issued the JWT, and is a public claim.

`"iat"` is the "issued at" date for the token, and is a registered claim.

Signature Segment

The signature segment of a JWT contains the cryptographic signature of the token.

Here's the signature segment of the example JWT token above:

```
WcPGXClpKD7Bc1C0CCDA1060E2GGlTfamrd8-W0ghBE
```

The signature segment is made up of the base 64 URL encoded header and payload segments, a secret (usually the contents of a key in a

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
)
```

The signature helps ensure that the data in the header and payload segments haven't been tampered with, and the JWT can be trusted.

However, it's important to note that the cryptographic signature at the end of the JWT is just for validation. It doesn't encrypt any data in the header or payload segments of the token. So you should never send sensitive information like a user's password in a JWT – everything in the header and payload can and should be public.

How to Validate JWT Signatures

The exact method for validating a signature depends on the algorithm defined in the header segment and used to generate the signature itself.

For the HS256 signing algorithm, a private key is shared between two entities, say your application's server and an authentication server. This private key is used both to generate signatures for outgoing JWTs, and to validate signatures from incoming JWTs.

When your authentication server receives an incoming JWT, it uses the incoming JWT's header and payload segments and the shared private key to generate a signature.

Another popular signing algorithm is RS256, which uses public and private key pairs to validate signatures. This is similar to the system used for SSH and SSL.

If you'd like to read more about how RS256 works, check out this article:

https://www.freecodecamp.org/news/understanding-encryption-algorithms/#rsa

Other Helpful Tutorials

If you'd like to learn more about JWTs and how to use them in applications, check out these tutorials:

https://www.freecodecamp.org/news/what-are-json-web-tokens-jwt-auth-tutorial/

https://www.freecodecamp.org/news/learn-to-implement-user-authentication-in-node-apps-using-passport-js/

Thanks for Reading

If you found this article on JWTs helpful, consider sharing it so more people can benefit from it.

Kristofer Koishigawa

Read more posts.