**https://jwt.io/introduction**

**How do 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, you should not keep tokens longer than required.**

You 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:

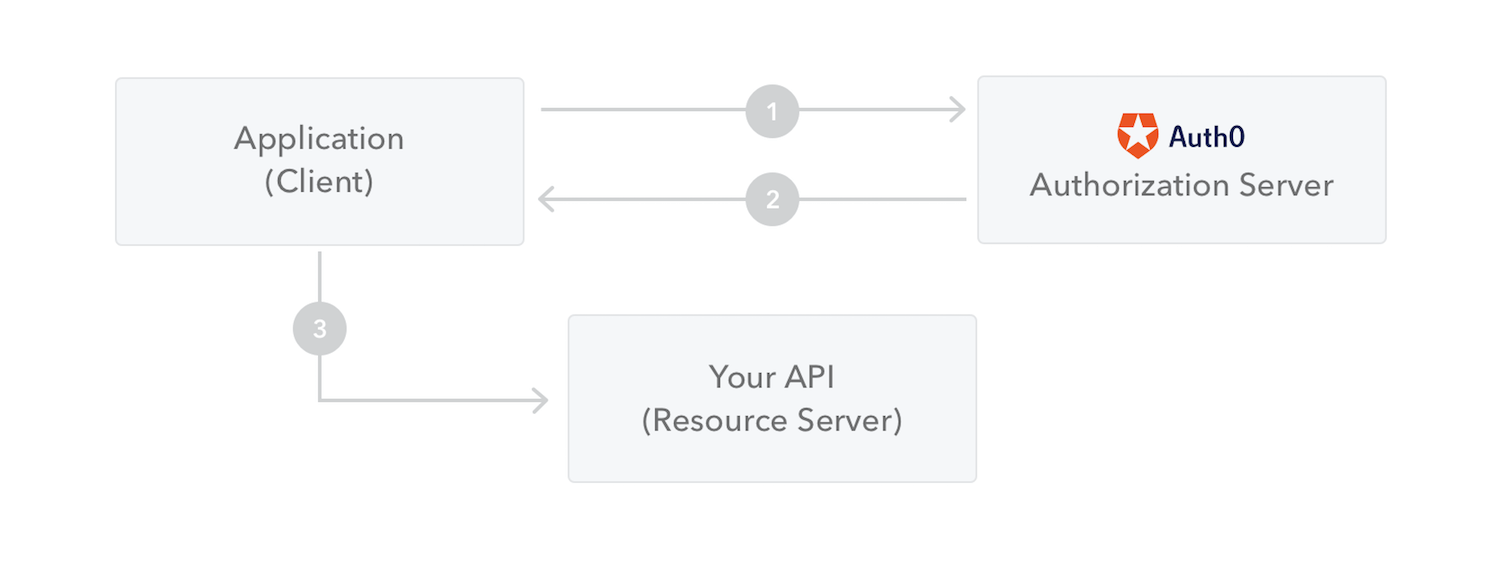
Authorization: Bearer <token>

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.** If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case.

**Note that if you send JWT tokens through HTTP headers, you should try to prevent them from getting too big**. Some servers don't accept more than 8 KB in headers. **If you are trying to embed too much information in a JWT token, like by including all the user's permissions, you may need an alternative solution, like**[**Auth0 Fine-Grained 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 you should not put secret information within the token.**

**Why should we use JSON Web Tokens?**

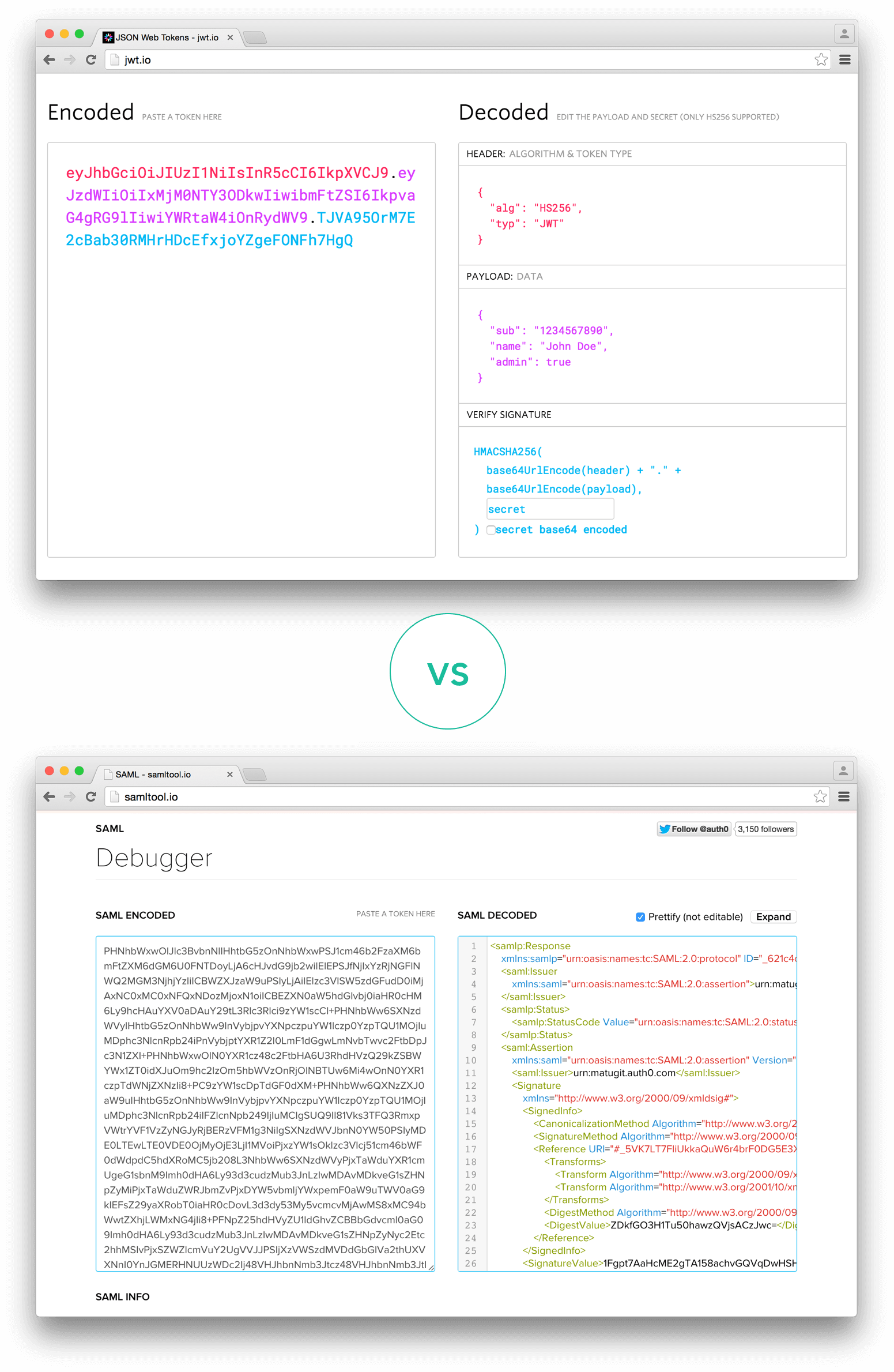
Let's talk about the benefits of **JSON Web Tokens (JWT)** when compared to **Simple Web Tokens (SWT)** and **Security Assertion Markup Language Tokens (SAML)**.

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.

 *Comparison of the length of an encoded JWT and an encoded SAML*

If you want to read more about JSON Web Tokens and even start using them to perform authentication in your own applications, browse to the [JSON Web Token landing page](http://auth0.com/learn/json-web-tokens) at Auth0.