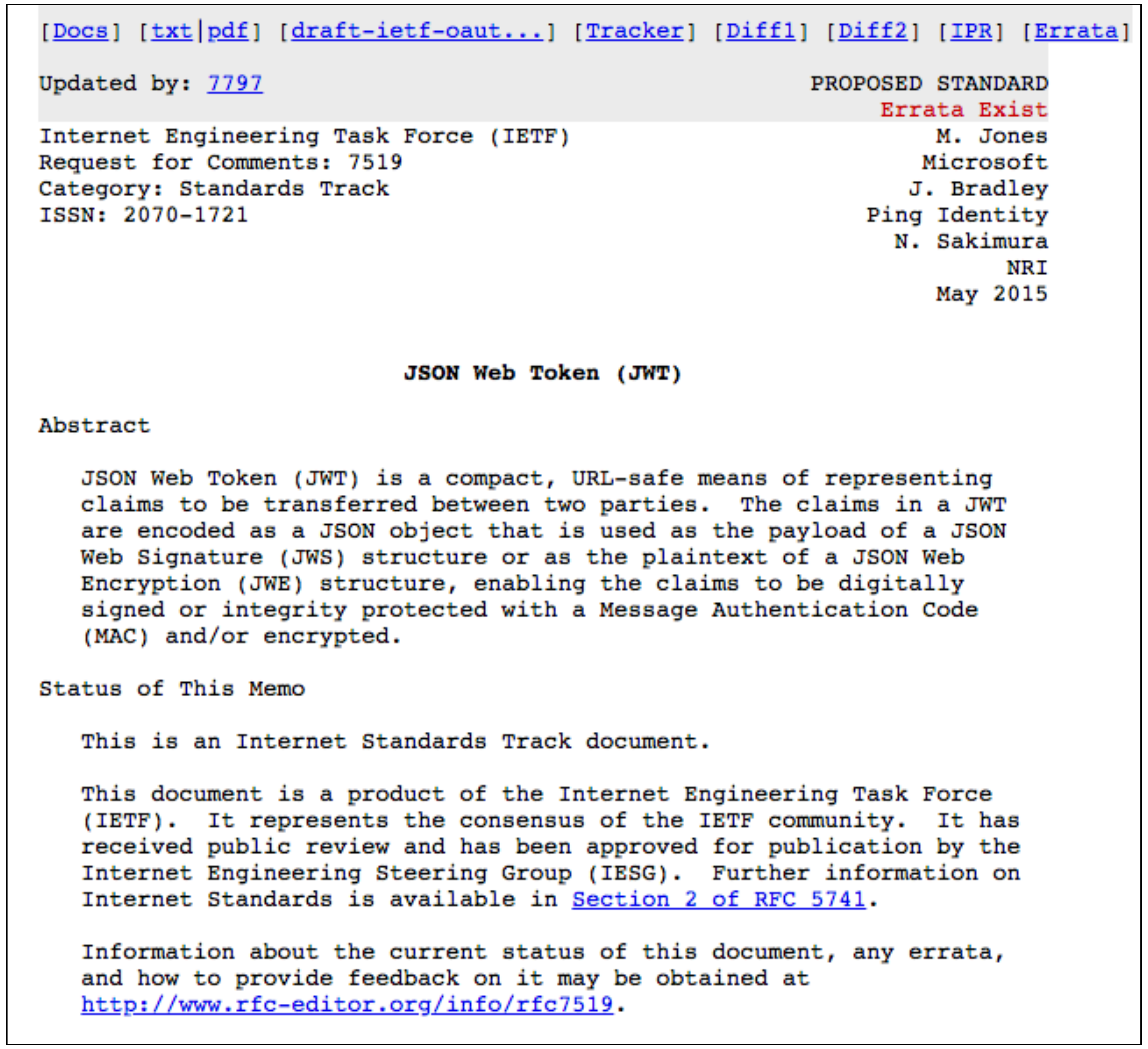
JWT Authentication

Full Stack Web Development

https://tools.ietf.org/html/rfc7519

- What is HTTP Authorisation?
- What is Bearer Authentication?
- What is JSON Web Token?
- When should you use JSON Web Tokens?
- What is the JSON Web Token structure?
- How do JSON Web Tokens work?
- Why should we use JSON Web Tokens?



```
[Docs] [txt|pdf] [draft-ietf-oaut...] [Tracker] [Diff1] [Diff2] [IPR] [Errata]

Updated by: 7797                                          PROPOSED STANDARD
                                                              Errata Exist
Internet Engineering Task Force (IETF)                           M. Jones
Request for Comments: 7519                                      Microsoft
Category: Standards Track                                      J. Bradley
ISSN: 2070-1721                                             Ping Identity
                                                              N. Sakimura
                                                                      NRI
                                                                 May 2015


                          JSON Web Token (JWT)

Abstract

   JSON Web Token (JWT) is a compact, URL-safe means of representing
   claims to be transferred between two parties.  The claims in a JWT
   are encoded as a JSON object that is used as the payload of a JSON
   Web Signature (JWS) structure or as the plaintext of a JSON Web
   Encryption (JWE) structure, enabling the claims to be digitally
   signed or integrity protected with a Message Authentication Code
   (MAC) and/or encrypted.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF).  It represents the consensus of the IETF community.  It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG).  Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7519.
```

What is HTTP Authorisation?

- The HTTP Authorisation request header can be used to provide credentials that authenticate a user agent with a server, allowing access to a protected resource.

- The Authorization header is usually sent after the user agent first attempts to request a protected resource without credentials. .

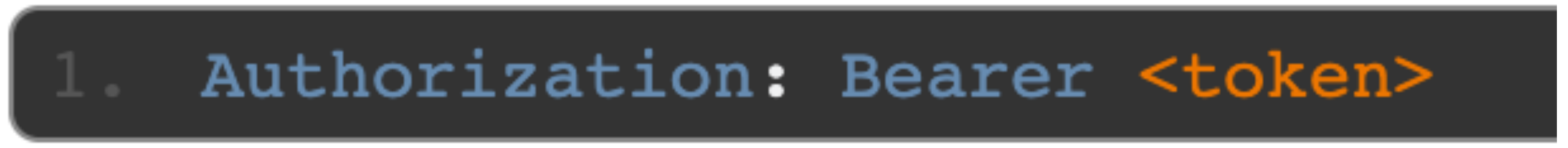
- This header indicates what authentication schemes can be used to access the resource

```
Authorization: <auth-scheme> <authorization-parameters>
```

```
Authorization: Basic <credentials>
```

What is Bearer Authentication?

- Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.
- The name “Bearer authentication” can be understood as “give access to the bearer of this token.”
- The bearer token is generated by the server in response to a login request via the Authenticate route.
- The client must send this token in the Authorization header in order to access API endpoints

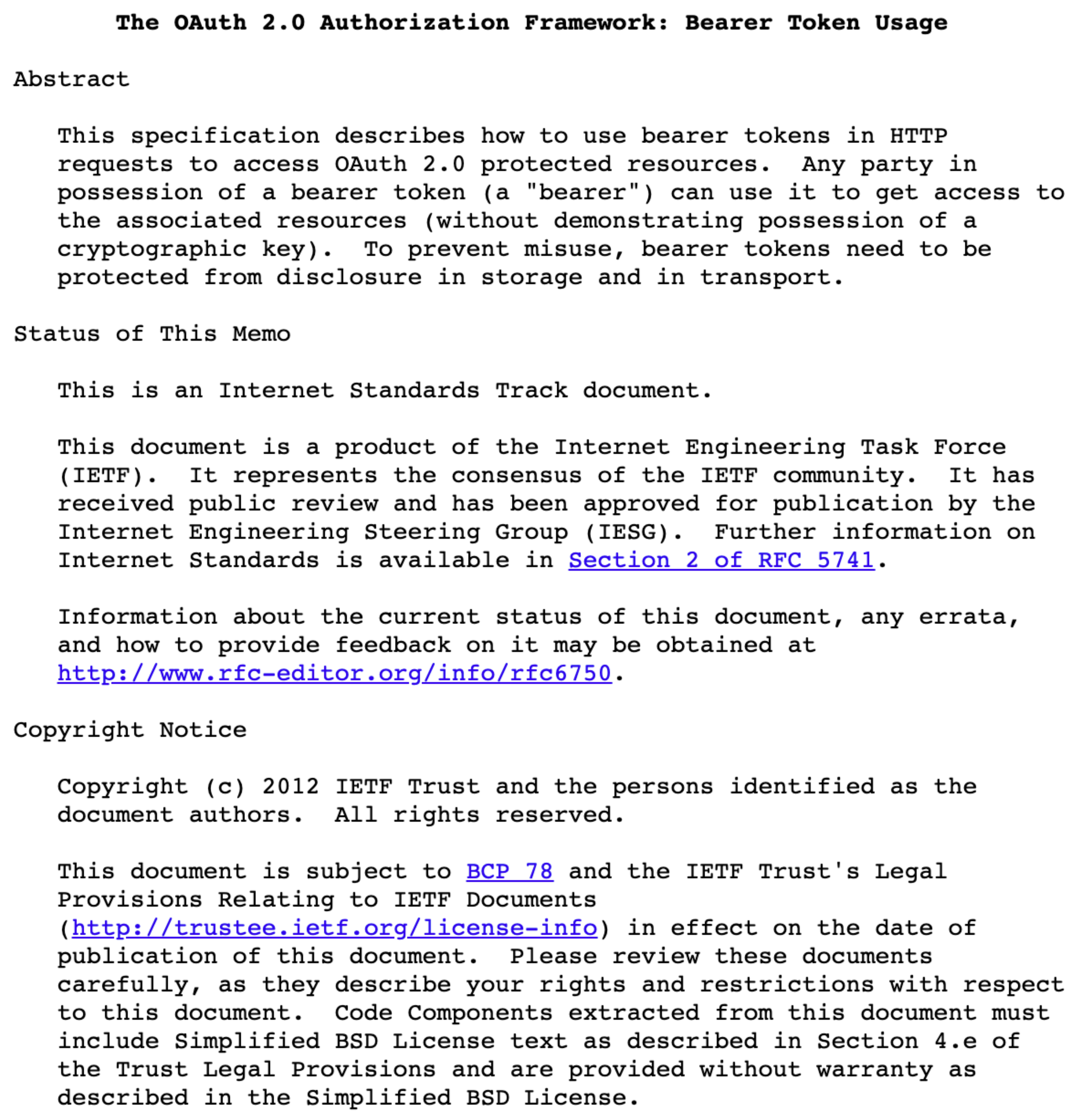


```
Authorization: Bearer <token>
```

OAuth 2.0

- The Bearer authentication scheme was originally created as part of OAuth 2.0 in RFC 6750, but is sometimes also used on its own.

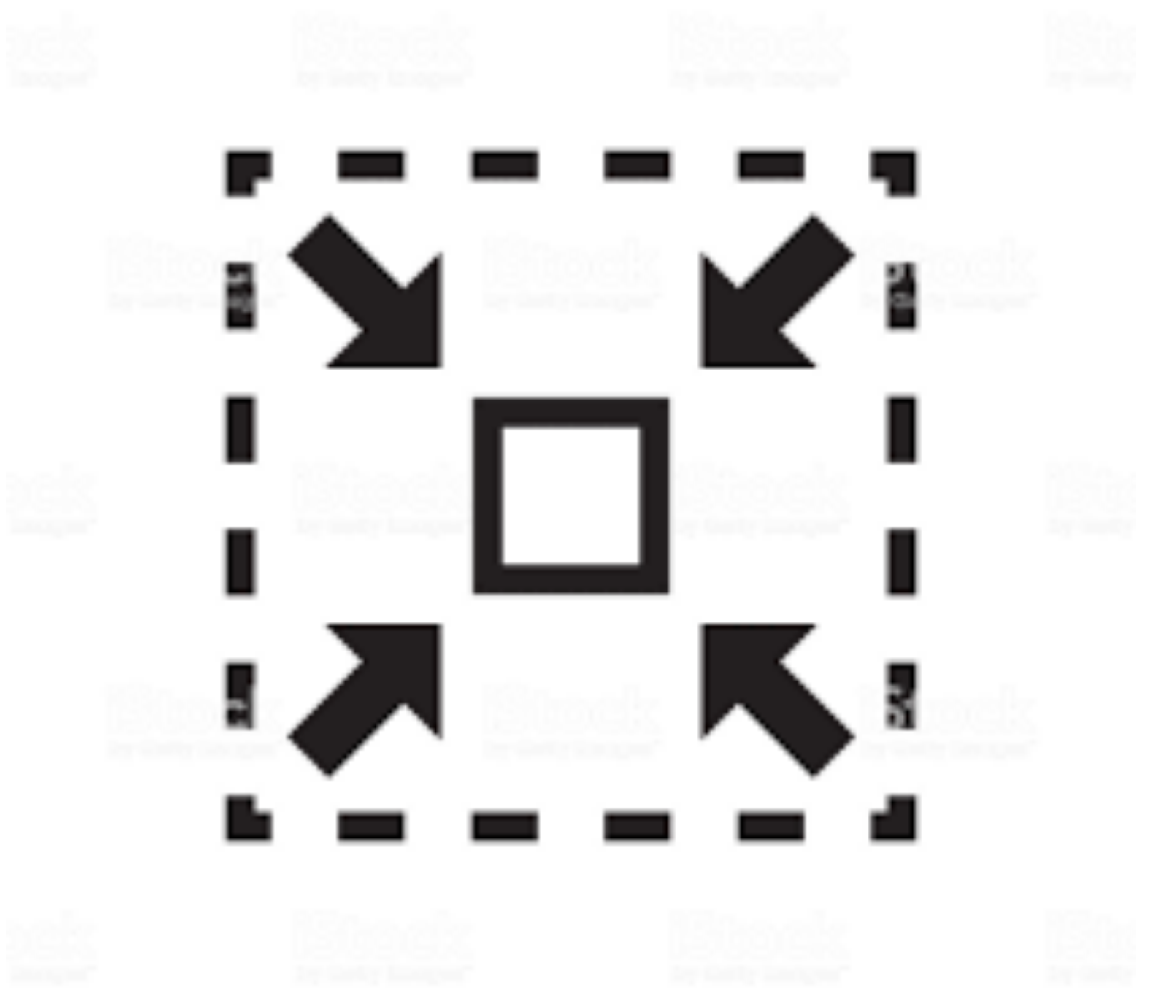
- Similarly to Basic authentication, Bearer authentication should only be used over HTTPS (SSL).



```
        The OAuth 2.0 Authorization Framework: Bearer Token Usage

Abstract

   This specification describes how to use bearer tokens in HTTP
   requests to access OAuth 2.0 protected resources.  Any party in
   possession of a bearer token (a "bearer") can use it to get access to
   the associated resources (without demonstrating possession of a
   cryptographic key).  To prevent misuse, bearer tokens need to be
   protected from disclosure in storage and in transport.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF).  It represents the consensus of the IETF community.  It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG).  Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc6750.

Copyright Notice

   Copyright (c) 2012 IETF Trust and the persons identified as the
   document authors.  All rights reserved.

   This document is subject to BCP 78 and the IETF Trust's Legal
   Provisions Relating to IETF Documents
   (http://trustee.ietf.org/license-info) in effect on the date of
   publication of this document.  Please review these documents
   carefully, as they describe your rights and restrictions with respect
   to this document.  Code Components extracted from this document must
   include Simplified BSD License text as described in Section 4.e of
   the Trust Legal Provisions and are provided without warranty as
   described in the Simplified BSD License.
```

What is JSON Web Token?

- An open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
  - **Compact**: Because of its smaller size, JWTs can be sent through an URL, POST parameter, or inside an HTTP header.
  - **Self-contained:** The payload contains all the required information about the user, avoiding the need to query the database more than once.



When should you use JSON Web Tokens?

- **Authentication**: 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.

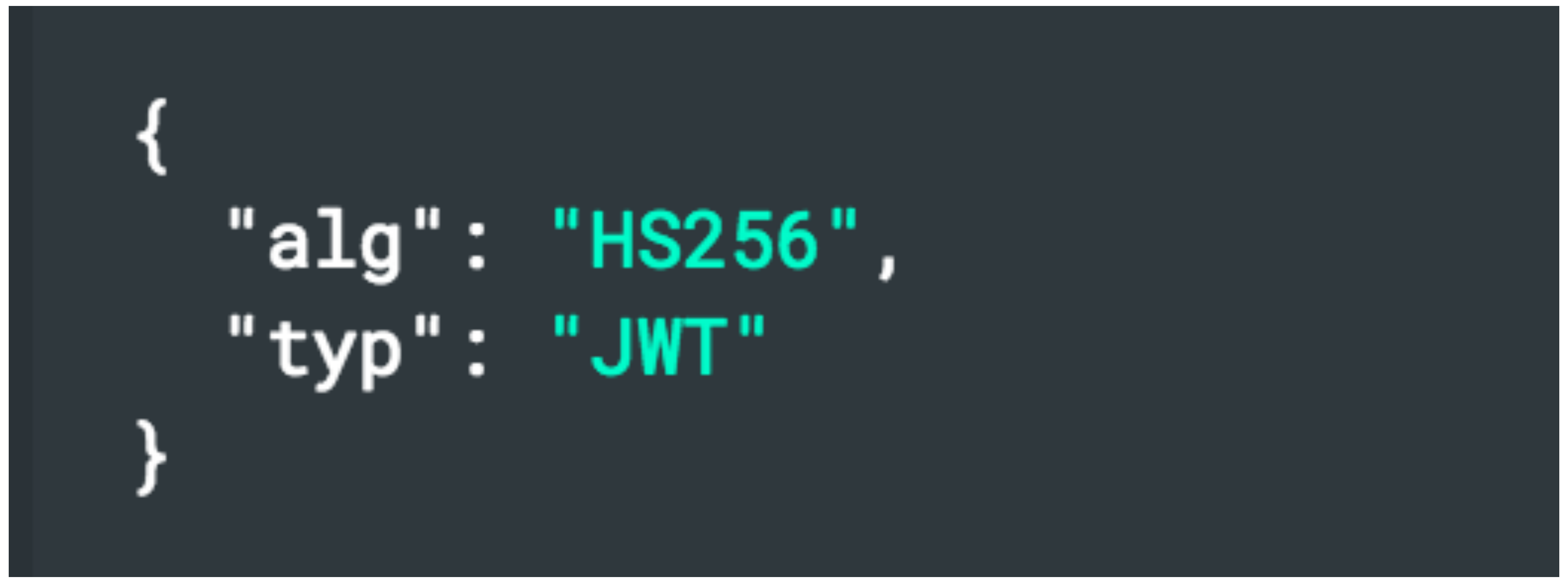
- **Information Exchange:** JSON Web Tokens are a good way of securely transmitting information between parties, because they can be signed.

What is the JSON Web Token structure?

- Three parts separated by dots (.), which are:
  - Header
  - Payload
  - Signature
- A JWT typically looks like the following.
  - xxxxx.yyyyy.zzzzz

JWT Structure : Header

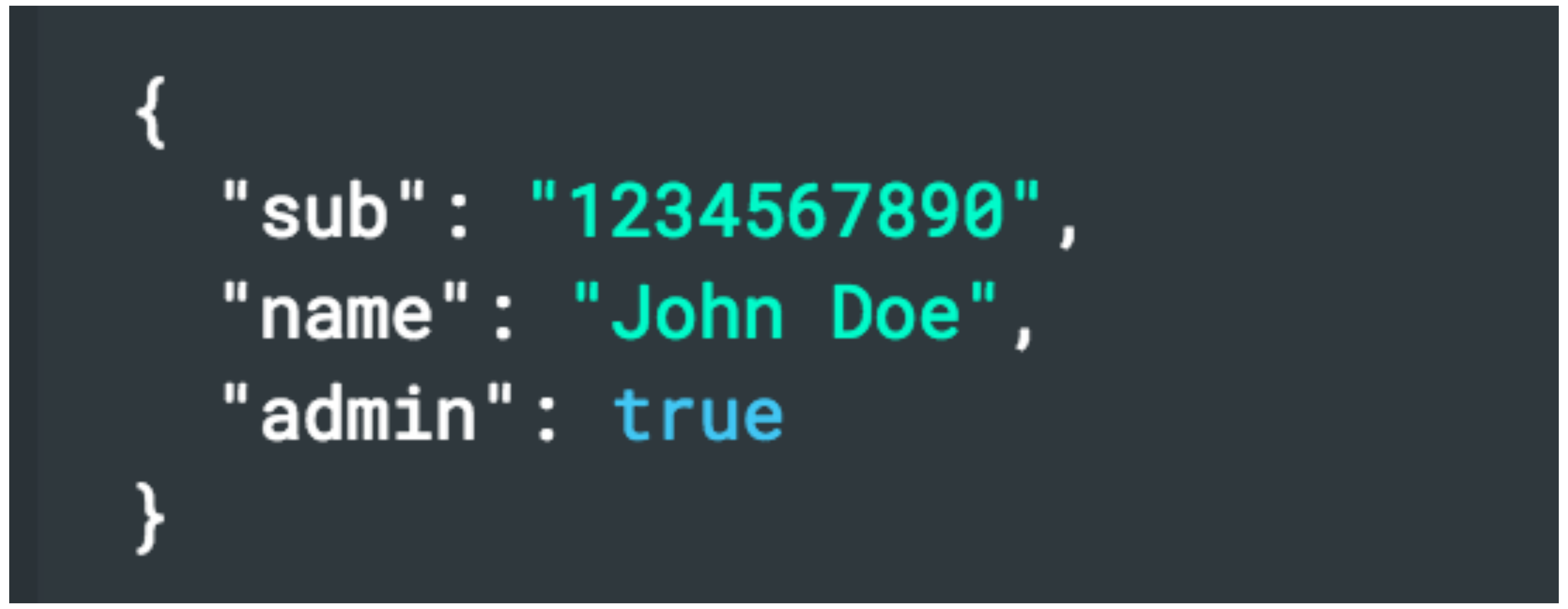
- Typically consists of two parts:
    - hashing algorithm being used, such as HMAC SHA256 or RSA.
    - type of the token, which is JWT,
- This JSON is Base64Url encoded to form the first part of the JWT.



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

JWT Structure : Payload

- Payload contains the “claims” - statements about an entity (typically, the user) and additional metadata. Three types of claims:
  - **Reserved claims**: A set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Examples: iss (issuer), exp (expiration time), sub (subject), aud (audience)
  - **Public claims:** These can be defined at will by those using JWTs. To avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
  - **Private claims:** These are the custom claims created to share information between parties that agree on using them.



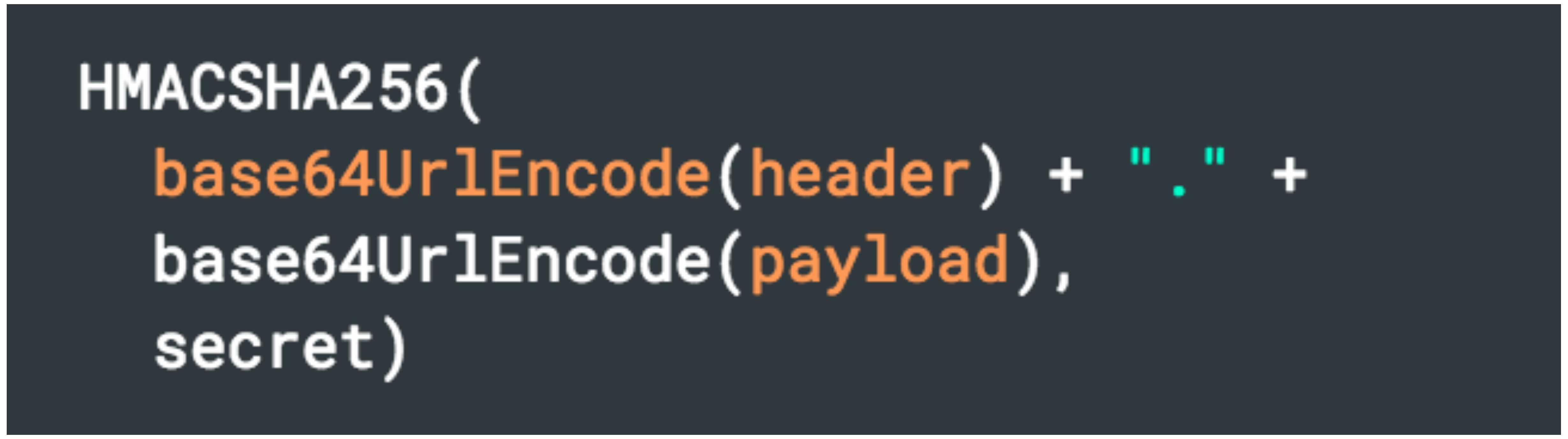
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

JWT Structure : Signature

- Take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign it.

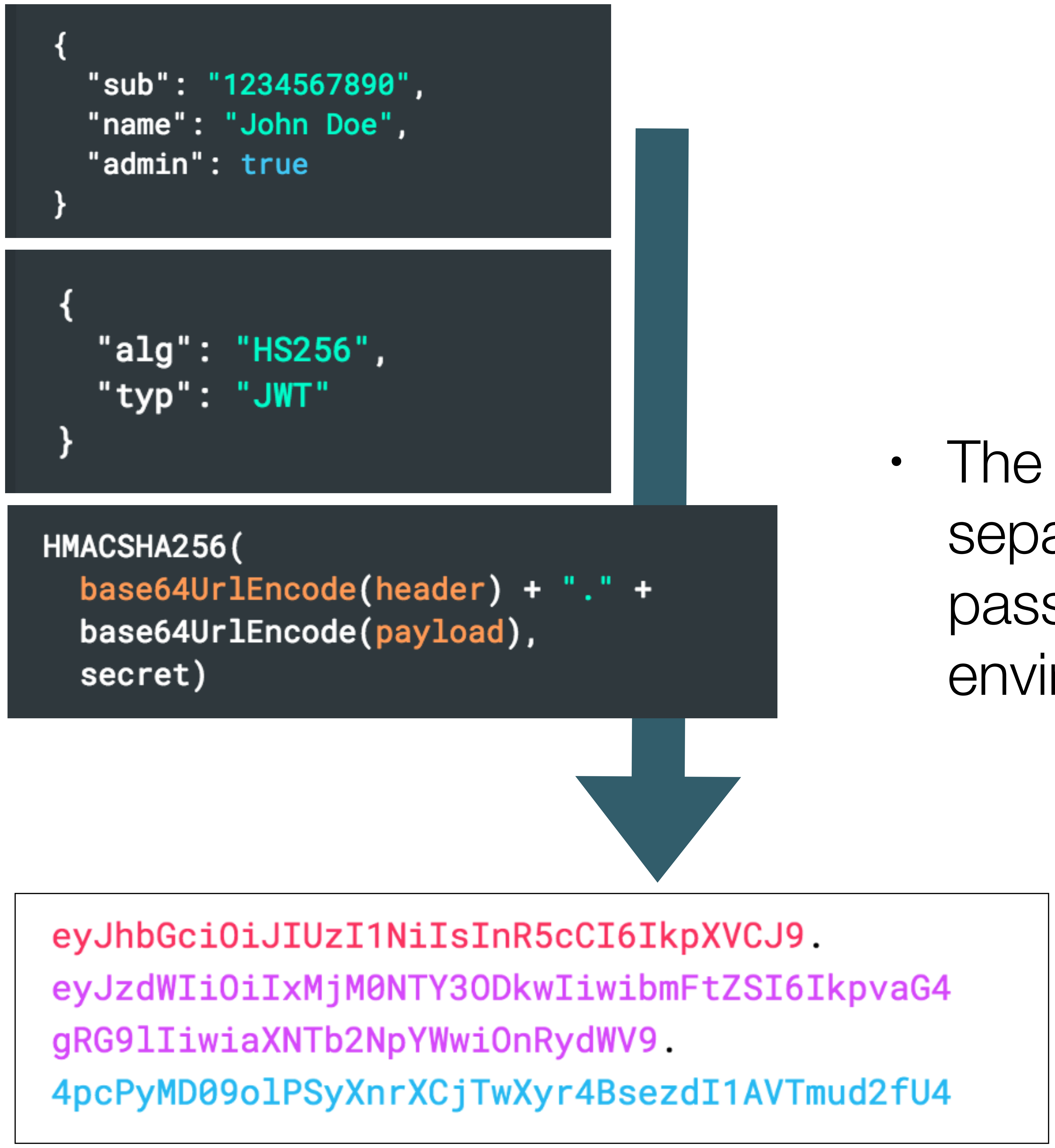
- The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

- 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 Token



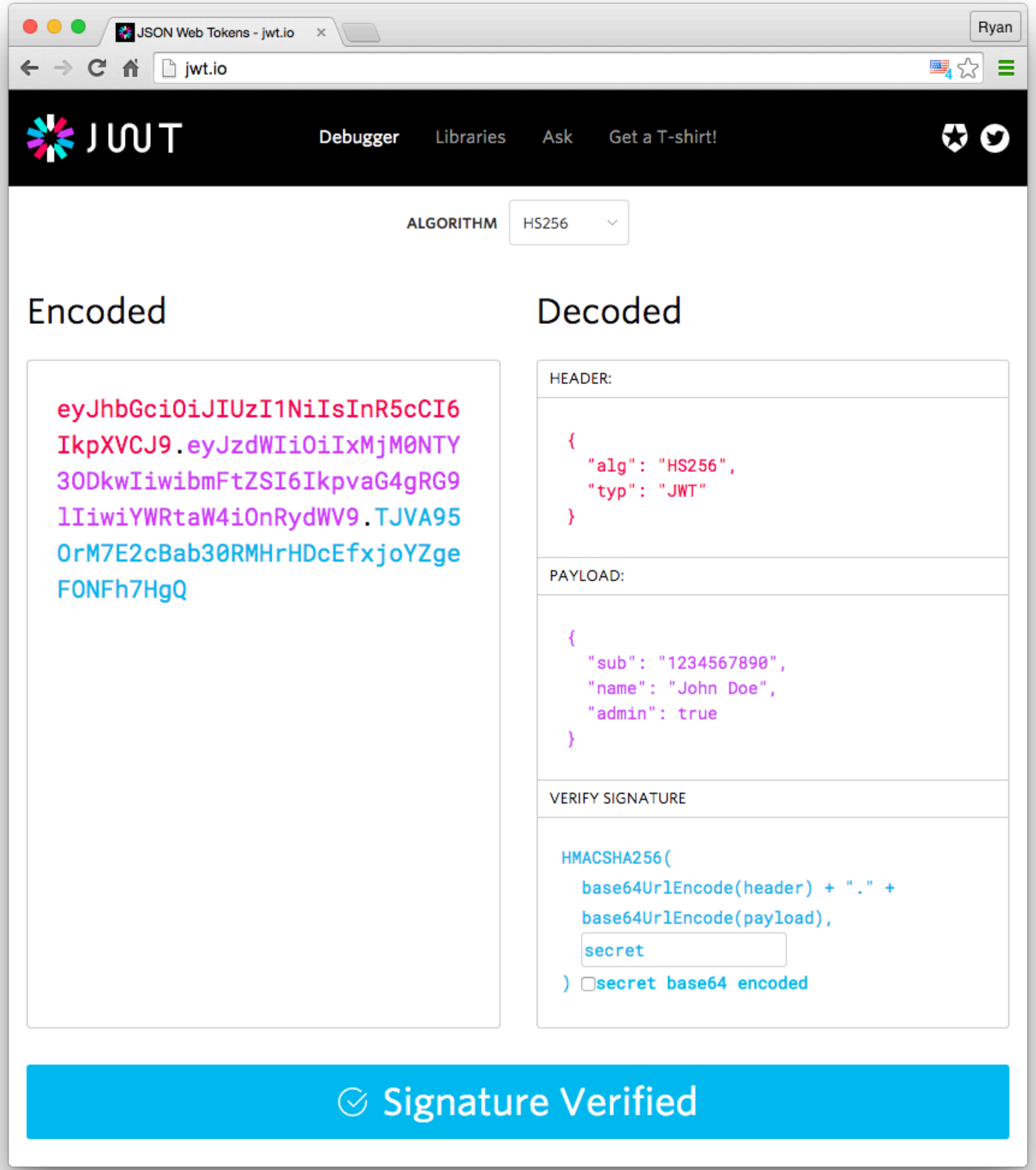
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

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

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

- The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments,



ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

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

PAYLOAD:

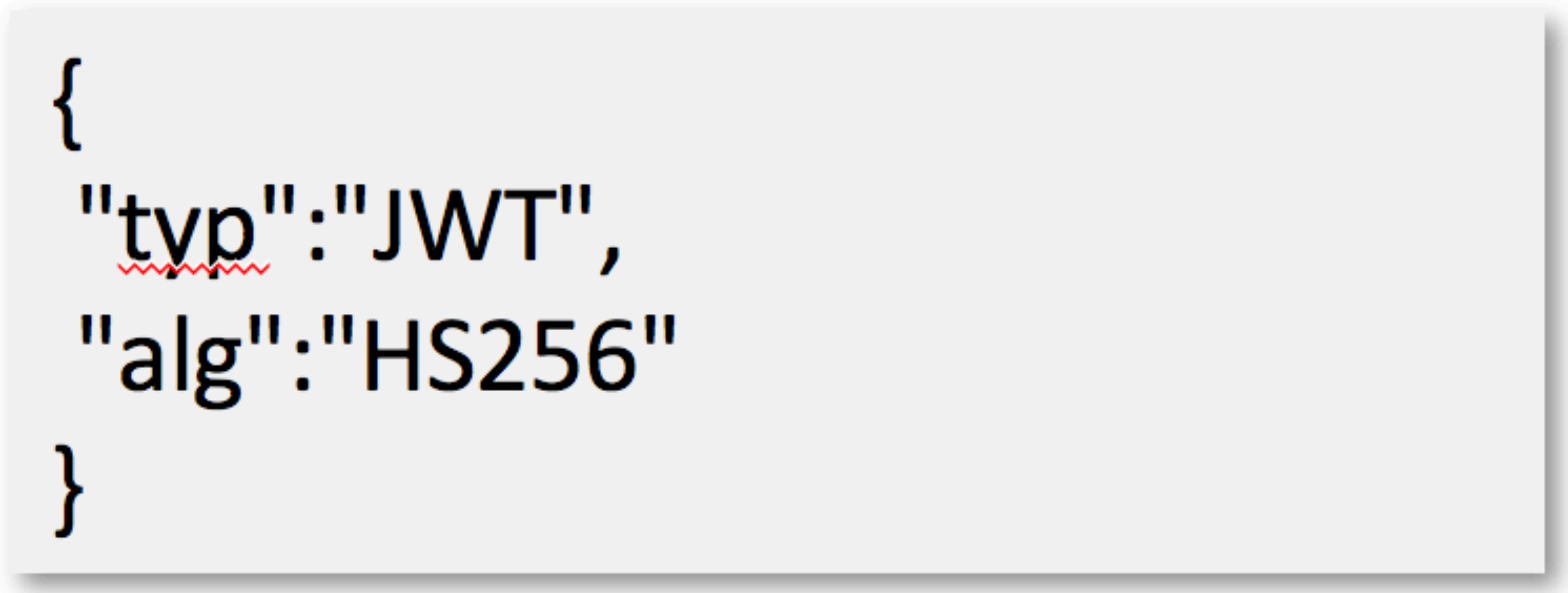
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

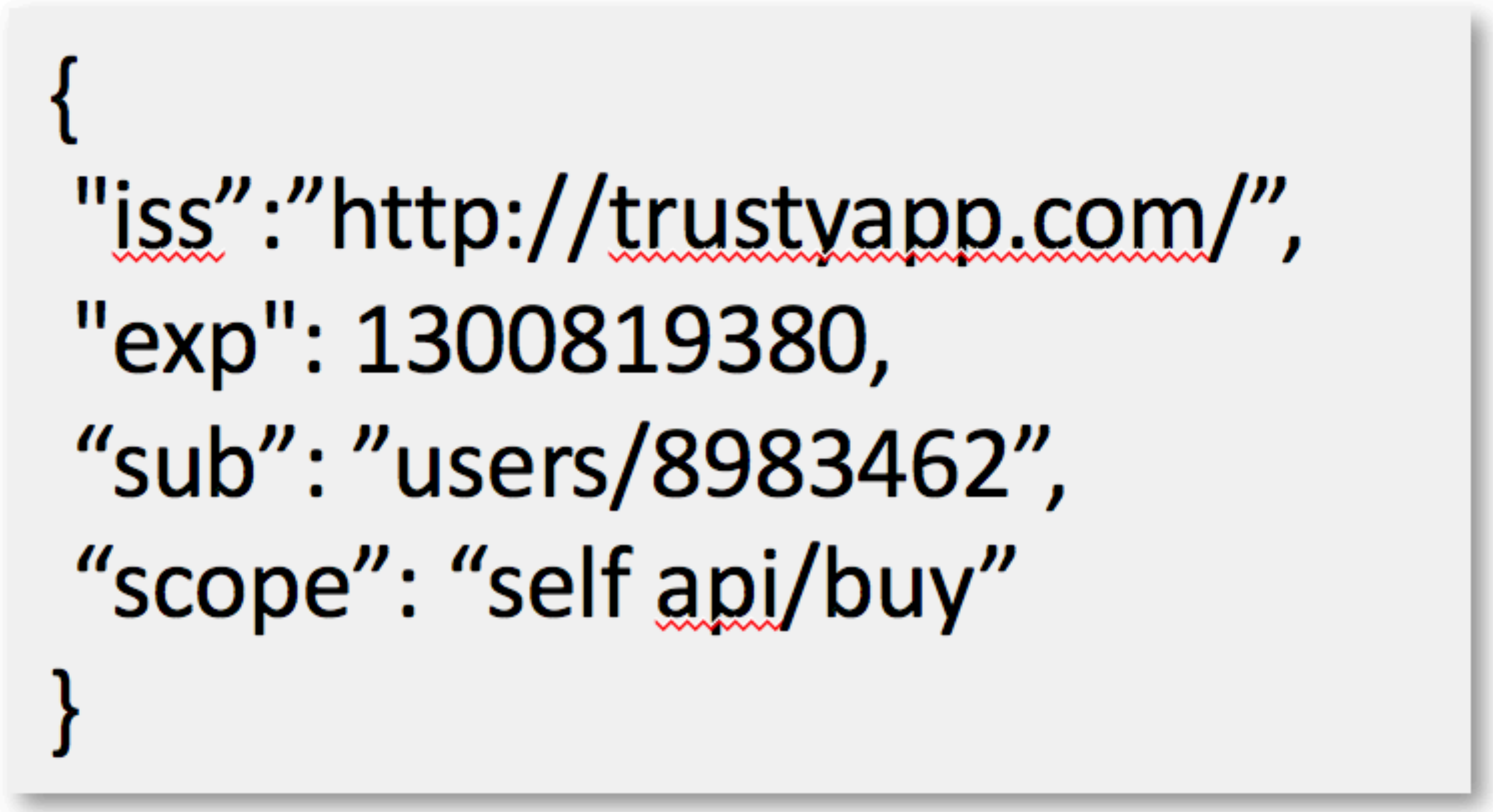
Signature Verified

Another Example



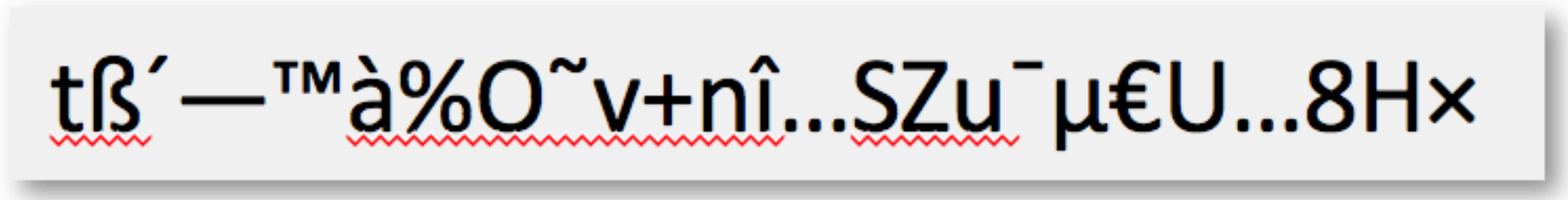
```
{
 "typ":"JWT",
 "alg":"HS256"
}
```

Header



```
{
 "iss":"http://trustyapp.com/",
 "exp": 1300819380,
 "sub": "users/8983462",
 "scope": "self api/buy"
}
```

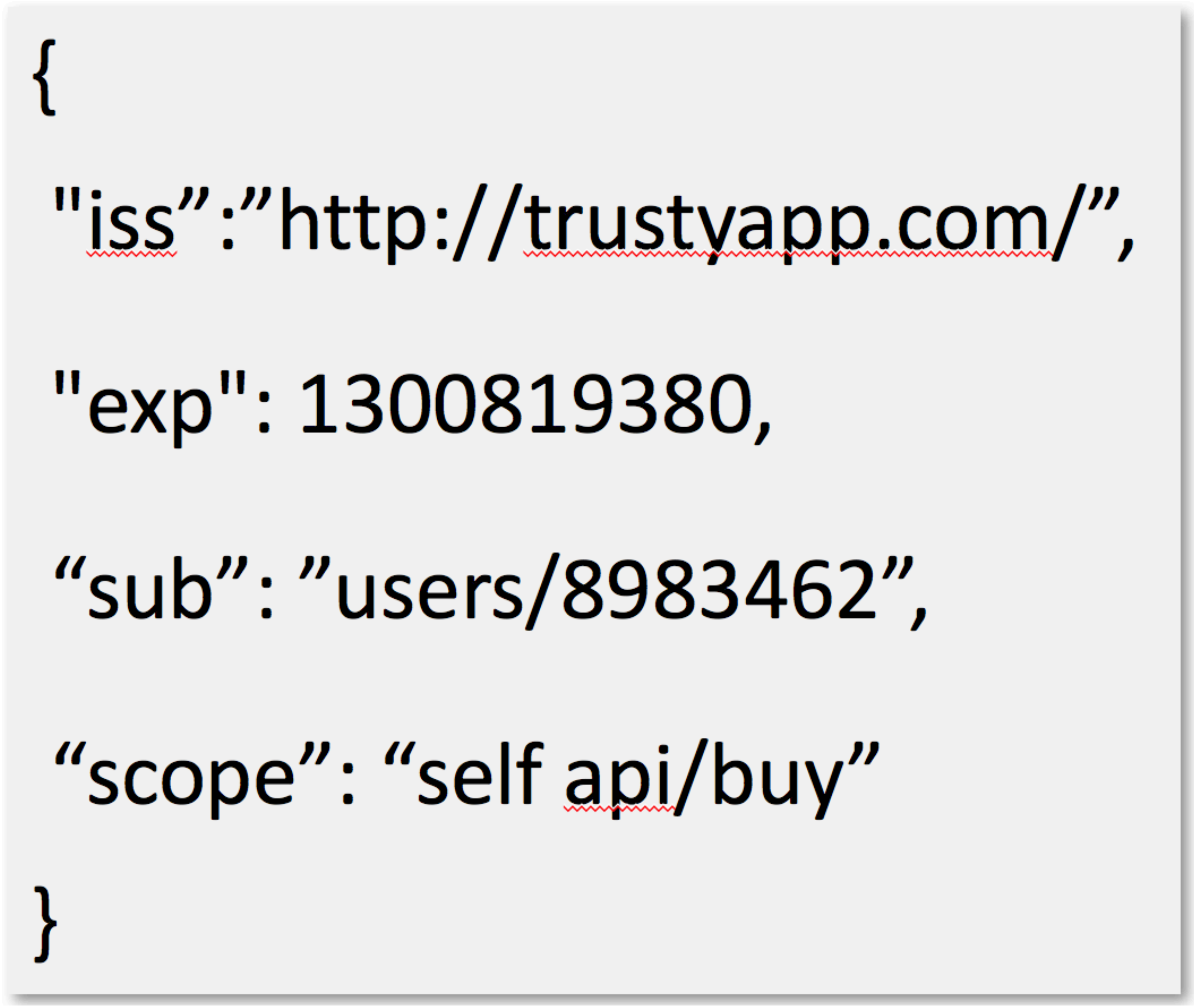
Body ('Claims')



```
tß´—™à%O˜v+nî…SZu¯µ€U…8H×
```

Cryptographic Signature

The Claims



```
{
 "iss”:”http://trustyapp.com/”,
 "exp": 1300819380,
 “sub”: ”users/8983462”,
 “scope”: “self api/buy”
}
```

Who issued the token

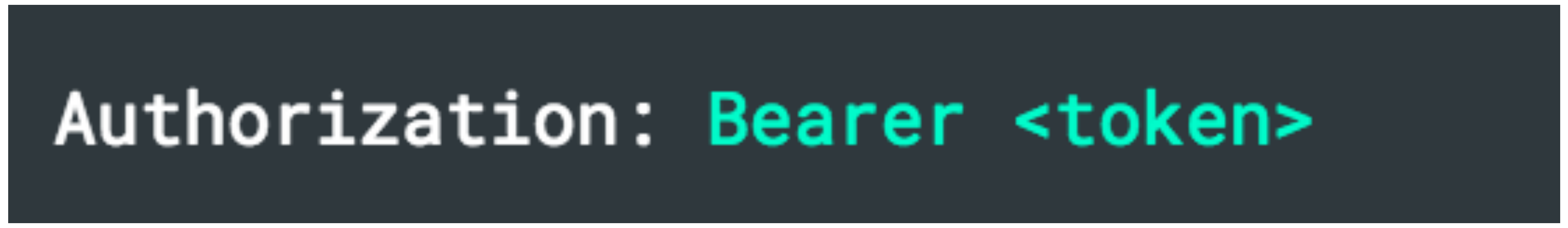
When it expires

Who it represents

What they can do

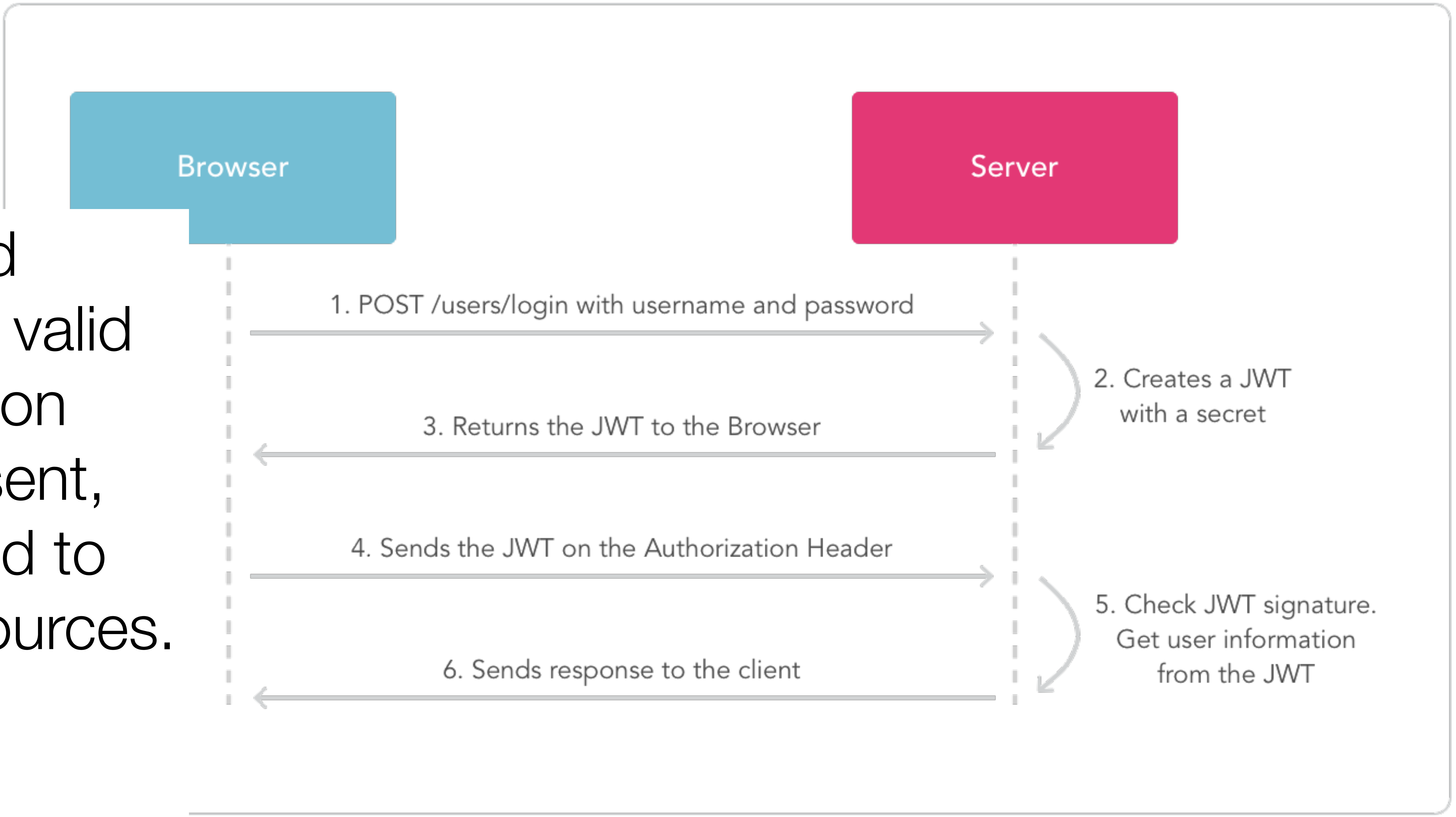
How do JSON Web Tokens work?

- When the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally, perhaps in local storage in a browser.
- If user wants to access a protected route or resource, the the JWT is sent, typically in the Authorization header using the Bearer schema



```
Authorization: Bearer <token>
```

- 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.
- Token contains all the necessary information.
- Token may even make requests to downstream services


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5. Check JWT signature. Get user information from the JWT
6. Sends response to the client

Stateless APIs

Why should we use JSON Web Tokens?

- **Compact** : Less verbose than XML, more compact than Security Assertion Markup Language Tokens (SAML).
- **Security:** JWT tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML can introducing obscure security holes compared to the simplicity of signing JSON.
- **Convenience:** 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

JSON Web Tokens

Full Stack Web Development