Try! JWT

Handcraft your awesome JWT encoder

Agenda

- Intro to JWT
- Implementation details
- Craft your own JWT encoder
- Enhance your JWT encoder with signing (optional)

What’s JWT

it is actually a very compact, printable representation of a series of claims, along with a signature to verify its authenticity.

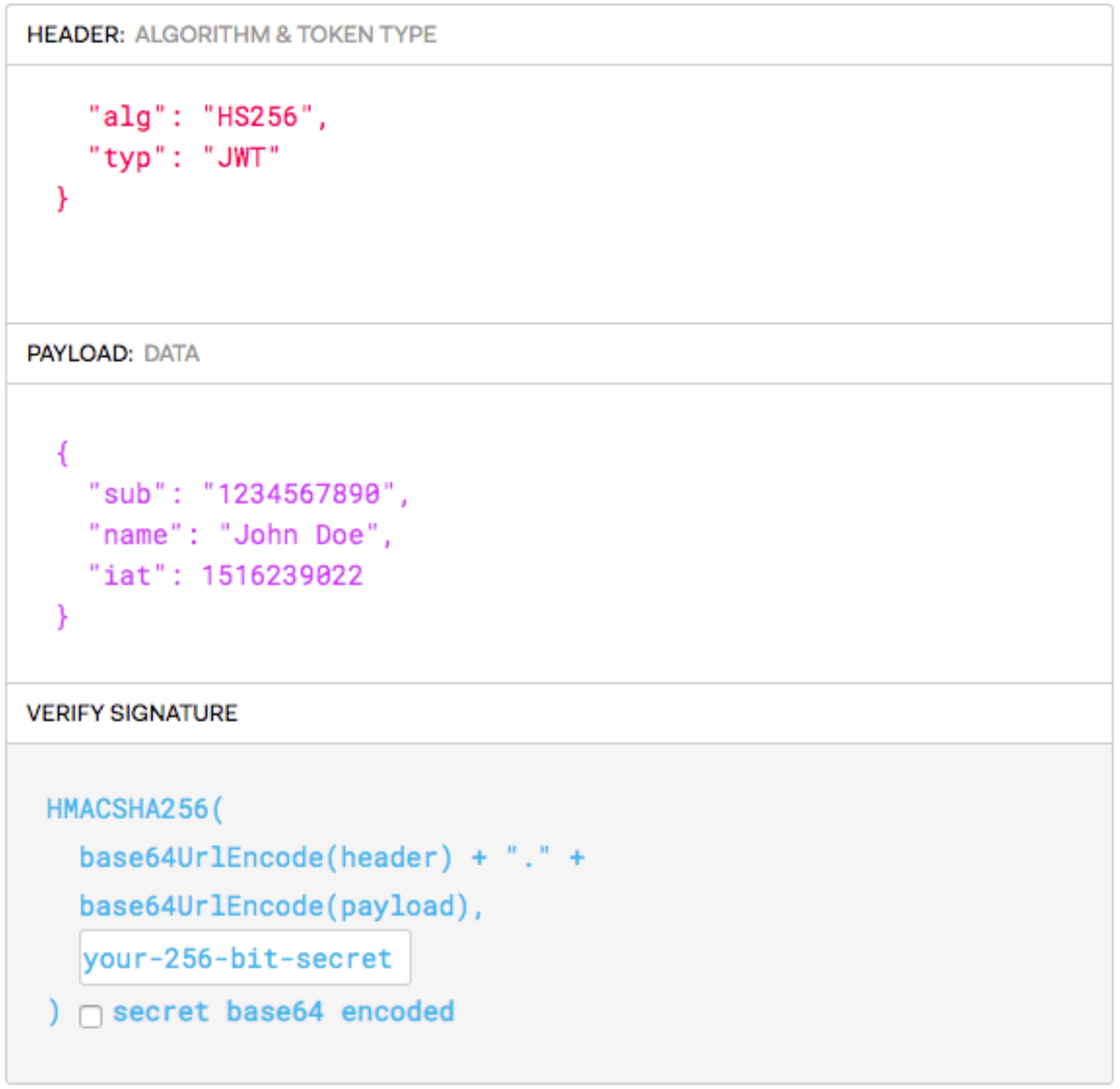
JWT

- Header
- Payload
- Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT

- Header
- Payload
- Signature



HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

RFCs

- JWT https://tools.ietf.org/html/rfc7519
- JWS https://tools.ietf.org/html/rfc7515
- JWE https://tools.ietf.org/html/rfc7516
- JWK https://tools.ietf.org/html/rfc7517
- JWA https://tools.ietf.org/html/rfc7518

Use Cases

Use cases

- Client side state
- Federated identity

JWT in detail

Header

- alg: algorithm use for signing and/or decrypting JWT
- typ: “JWT”
- cty: content type, for nesting JWT

Payload

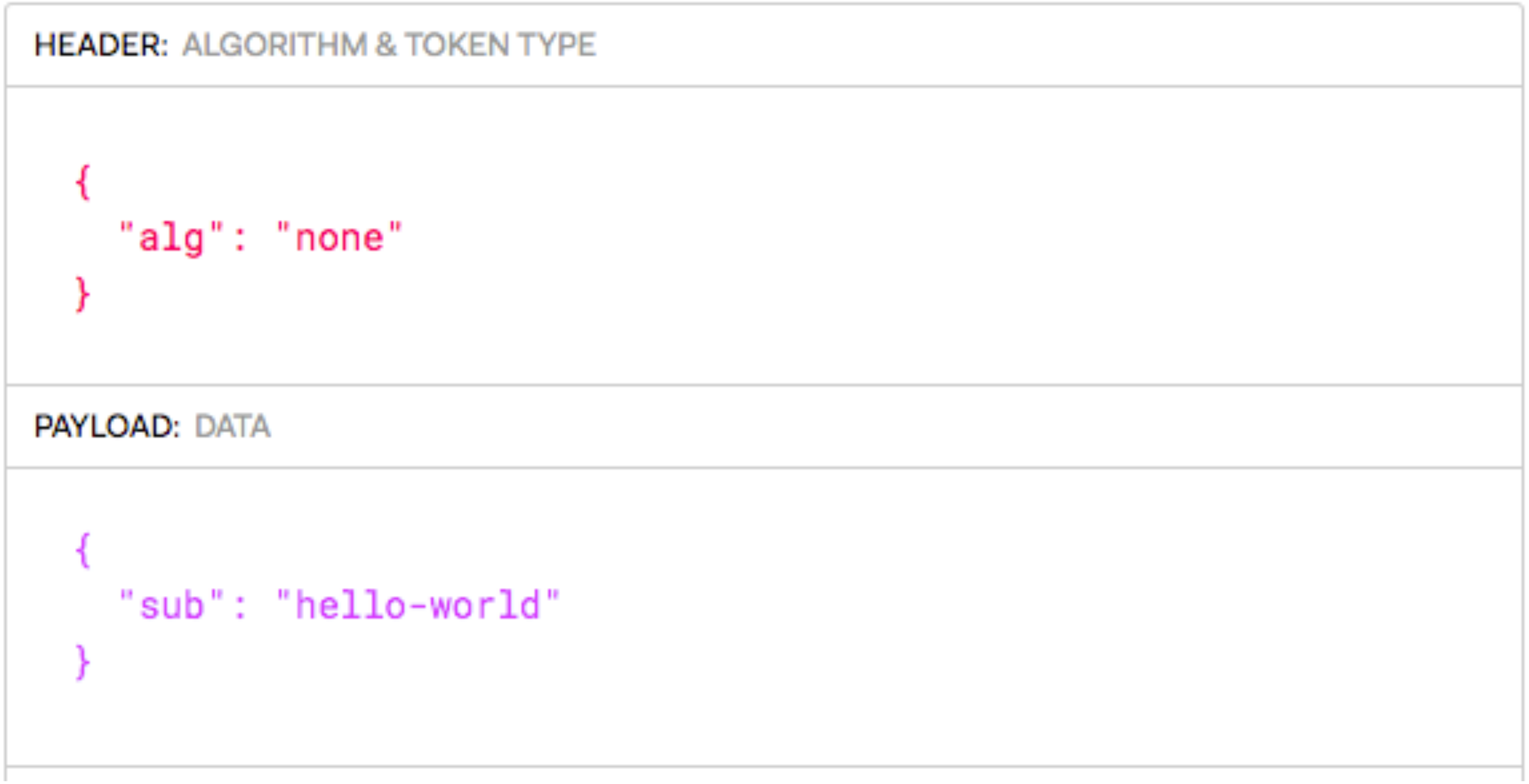
Registered Claims

- iss: Issuer. Uniquely identifies the party that issued the JWT
- sub: Subject. Uniqueness in the context of issuer, or globally
- aud: Audience. Intended recipients
- exp: Expiration (time). Seconds since epoch
- nbf: Not Before (time). Seconds since epoch
- iat: Issued At (time). Seconds since epoch
- jti: JWT ID. Unique identifier for this JWT

Public and Private Claims

- Public: registered with IANA JSON Web Token Claims registry or collision resistant name
- Private: defined by users (consumers and producers)

💻 Implement a JWT encoder


HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "none"
}
```

PAYLOAD: DATA

```
{
  "sub": "hello-world"
}
```

1. Take the header as a byte array of its UTF-8 representation. The JWT spec does not require the JSON to be minified or stripped of meaningless characters (such as whitespace) before encoding.

2. Encode the byte array using the Base64-URL algorithm, removing trailing equal signs (=).

3. Take the payload as a byte array of its UTF-8 representation. The JWT spec does not require the JSON to be minified or stripped of meaningless characters (such as whitespace) before encoding.

4. Encode the byte array using the Base64-URL algorithm, removing trailing equal signs (=).

5. Concatenate the resulting strings, putting first the header, followed by a “.”character, followed by the payload.

Header:

```
{
  "alg": "none"
}
```

Payload:

```
{
  "sub": "hello-world"
}
```

🎉 Congratulations

JWS

Probably the single most useful feature of JWTs

Purpose

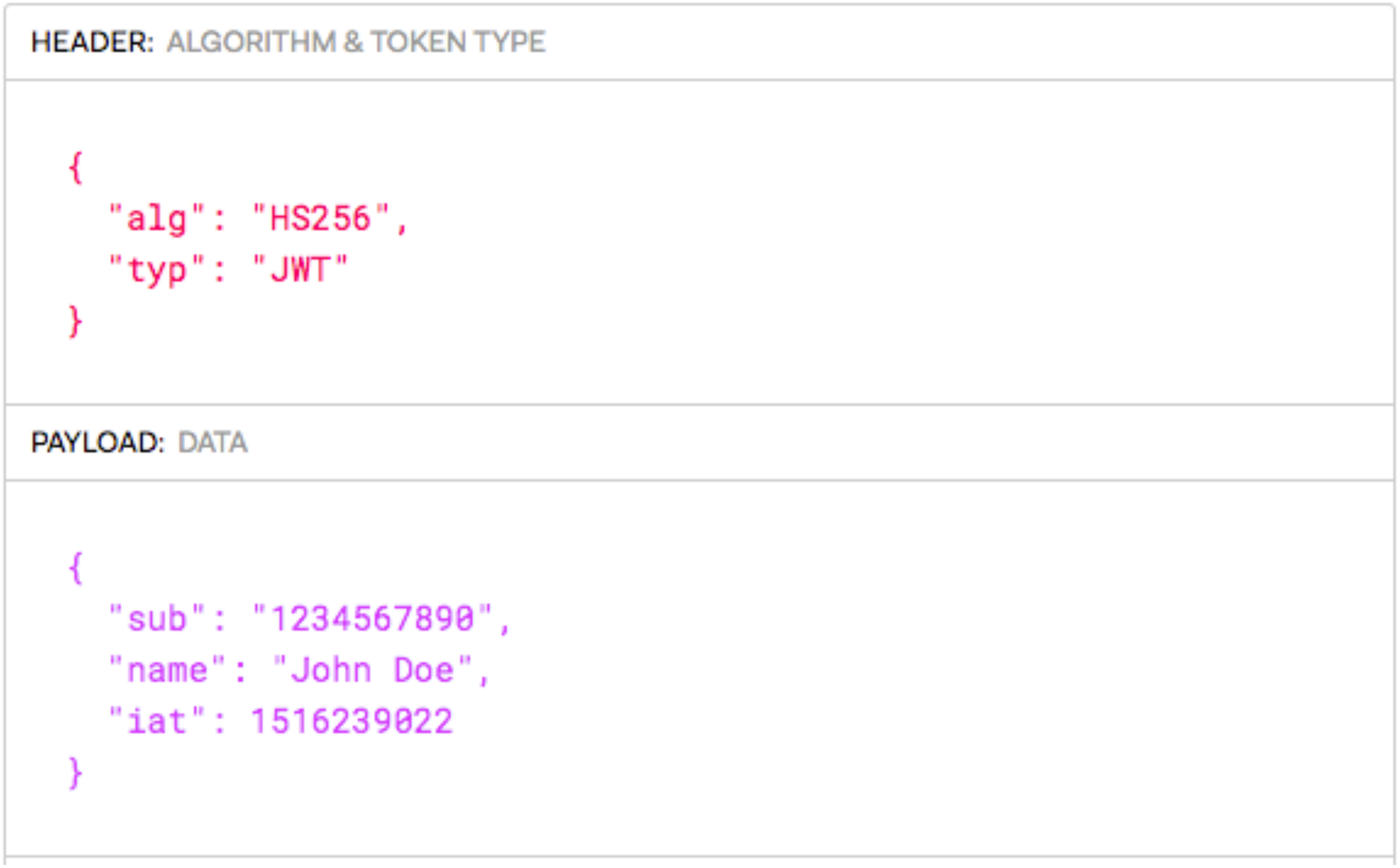
- Authenticity: in this context means the data contained in the JWT has not been tampered with

JWT

- Header
- Payload
- Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

💻 JWT encoder with signing



HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

SHA-256

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

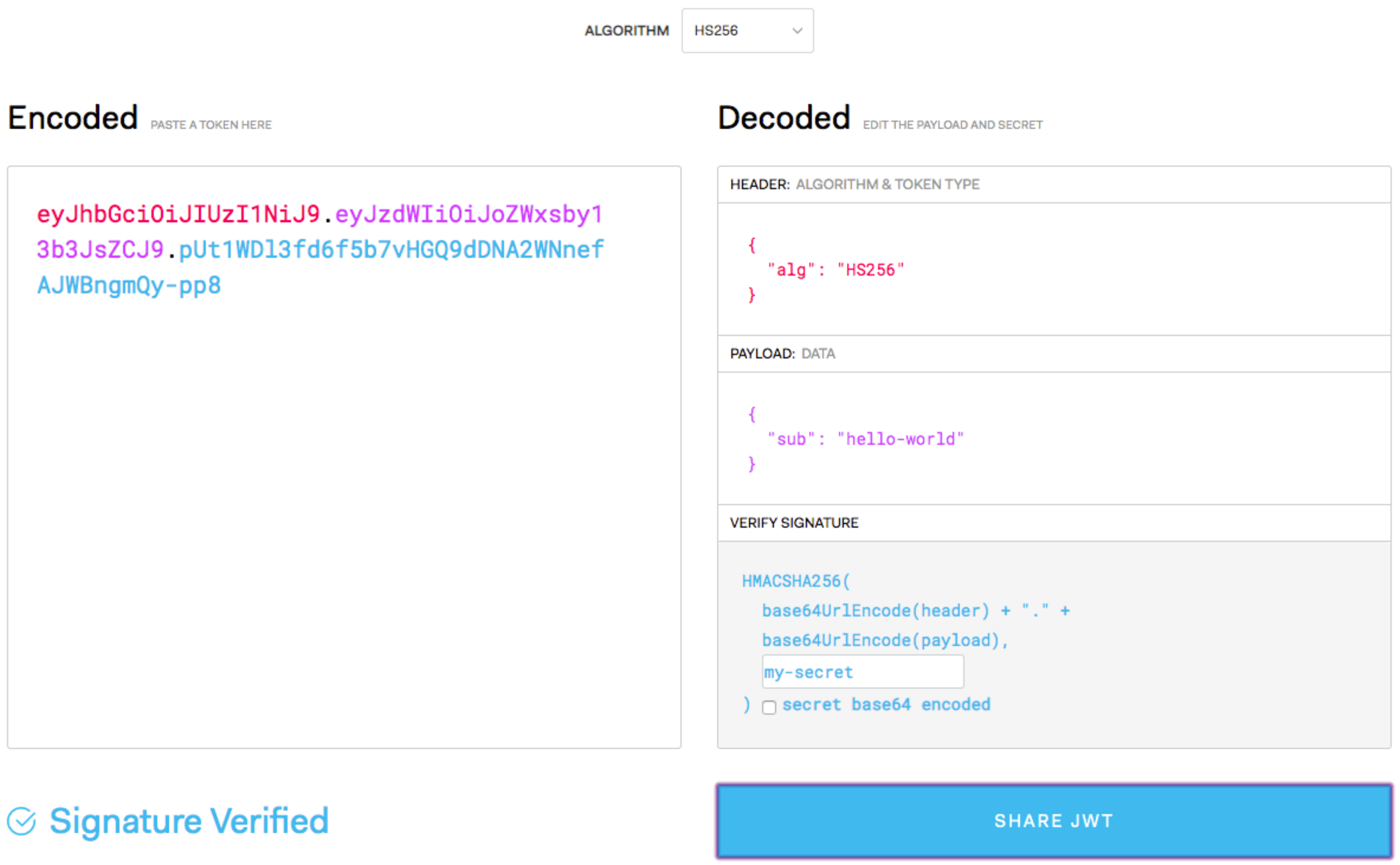
```
sign = HMACSHA256 (
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  my-secret
)
new_token = base64UrlEncode(header) + “.”
 + base64UrlEncode(payload) + “.”
 + sign
```

Header:

```
{
  "alg": “HS256"
}
```

Payload:

```
{
  "sub": "hello-world"
}
```


ALGORITHM HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJoZWxsby1
3b3JsZCJ9.pUt1WDl3fd6f5b7vHGQ9dDNA2WNnef
AJWBngmQy-pp8
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "hello-world"
}
```

VERIFY SIGNATURE

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

Signature Verified

SHARE JWT

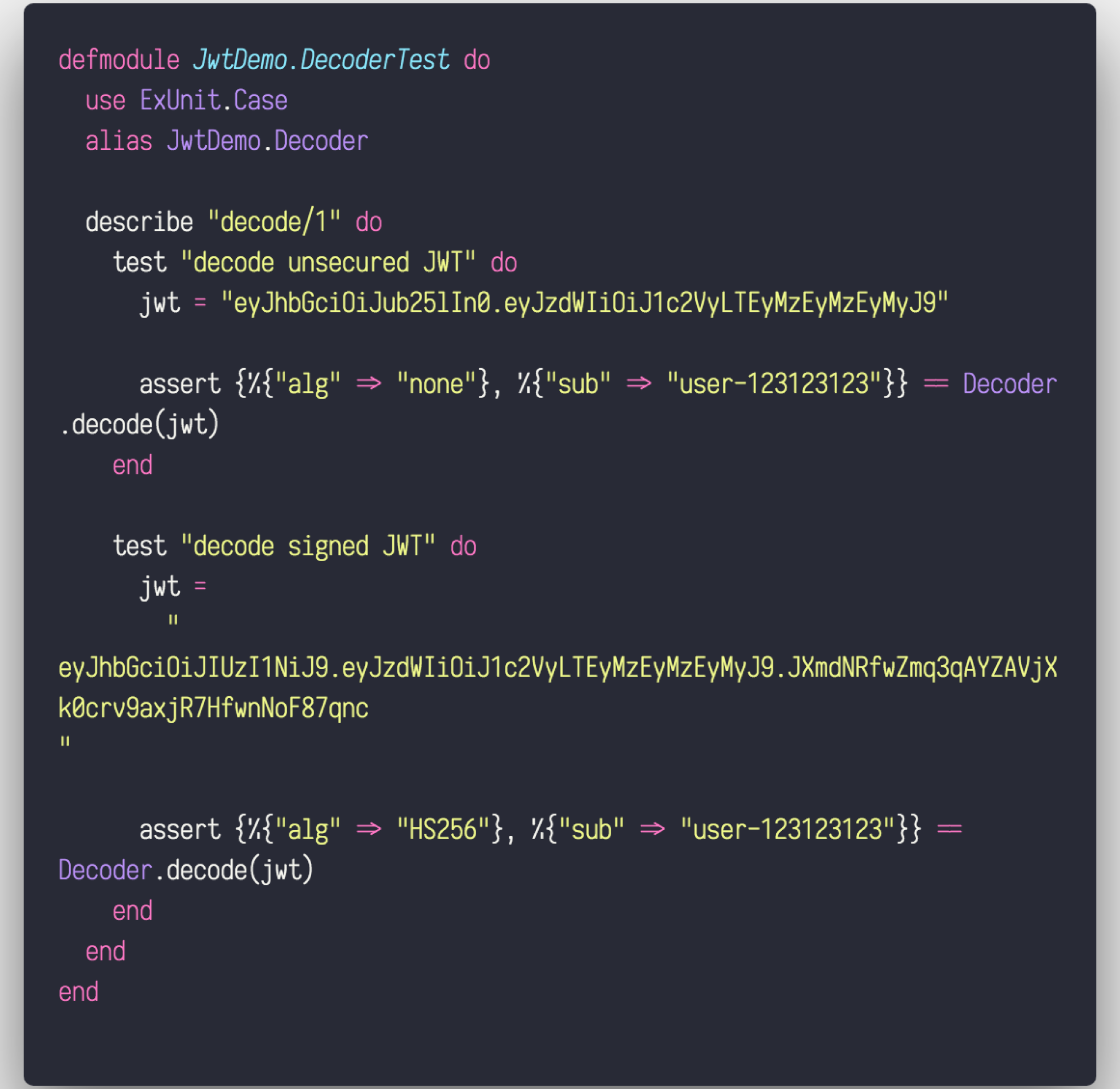
🎉 Congratulations

⚠️

A signature does not prevent other parties from reading the contents inside the JWT

Homework

Decoder



```elixir
defmodule JwtDemo.DecoderTest do
  use ExUnit.Case
  alias JwtDemo.Decoder

  describe "decode/1" do
    test "decode unsecured JWT" do
      jwt = "eyJhbGciOiJub25lIn0.eyJzdWIiOiJ1c2VyLTEyMzEyMzEyMyJ9"

      assert {%{"alg" => "none"}, %{"sub" => "user-123123123"}} == Decoder
.decode(jwt)
    end

    test "decode signed JWT" do
      jwt =
        "
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyLTEyMzEyMzEyMyJ9.JXmdNRfwZmq3qAYZAVjX
k0crv9axjR7HfwnNoF87qnc
"

      assert {%{"alg" => "HS256"}, %{"sub" => "user-123123123"}} ==
Decoder.decode(jwt)
    end
  end
end
```

More specs

- JWE https://tools.ietf.org/html/rfc7516
- JWK https://tools.ietf.org/html/rfc7517
- JWA https://tools.ietf.org/html/rfc7518

Thanks

References

- https://auth0.com/resources/ebooks/jwt-handbook
- https://jwt.io/