JWT TOKENS

Qinshift Academy

WHAT IS A JWT BEARER TOKEN?

A **JWT** or **J**SON **W**eb **T**oken is a json that follows certain rules, is digitally signed and is used for securely exchanging information between client and server.

JWT Tokens can be signed digitally by a secret which only the server knows or with public/private key pairs.

The JWT token has a defined structure consisting of 3 parts:

JWT

- **Header** - Data about the token it self (Encoded in Base64Url)
  - alg - Algorithm used for signing hash ("SHA256")
  - typ - Type of cookie ("JWT")
- **Payload** - Statements for our business logic called claims (Encoded in Base64Url)
  - iss - issuer of the token (Who is issuing the token)
  - exp - what time does the token expire
  - sub - subject of the token (Some data that is important for the business logic)
  - aud - audience of the token (Who is the token meant for)
- **Signature** - a unique signature hashed with the encryption written in the header. The input data that is hashed is the header, payload and a secret (a string that only the server knows)

JWT

- **Payload*** (a bit more for this part)

The payload will carry the bulk of our JWT, also called the JWT Claims. This is where we will put the information that we want to transmit and other information about our token.

There are multiple claims that we can provide. This includes registered claim names, public claim names, and private claim names.

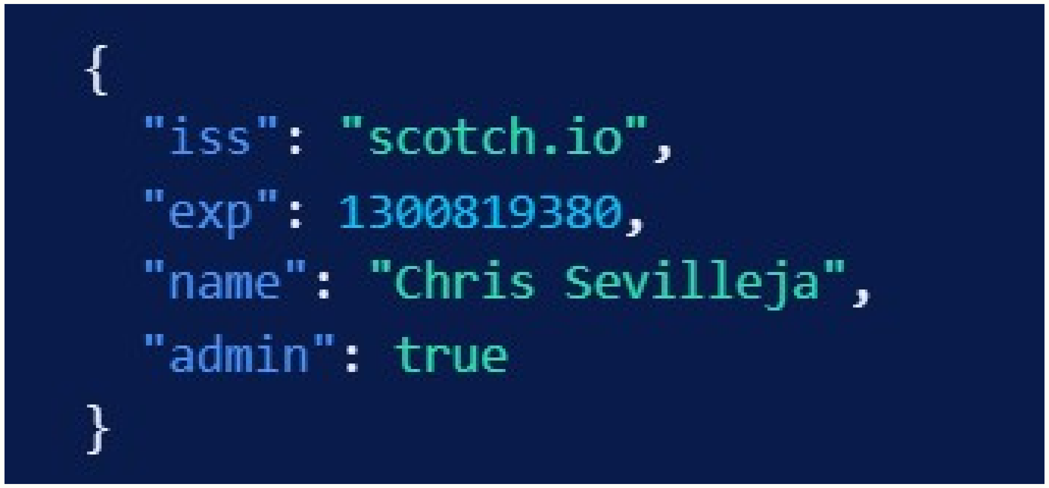
JWT

- **Registered Claims** - Claims that are not mandatory whose names are reserved for us. These include:
  - iss: The issuer of the token
  - sub: The subject of the token
  - aud: The audience of the token
  - exp: This will probably be the registered claim most often used. This will define the expiration in NumericDate value. The expiration MUST be after the current date/time.
  - nbf: Defines the time before which the JWT MUST NOT be accepted for processing
  - iat: The time the JWT was issued. Can be used to determine the age of the JWT
  - jti: Unique identifier for the JWT. Can be used to prevent the JWT from being replayed. This is helpful for a one-time use token.

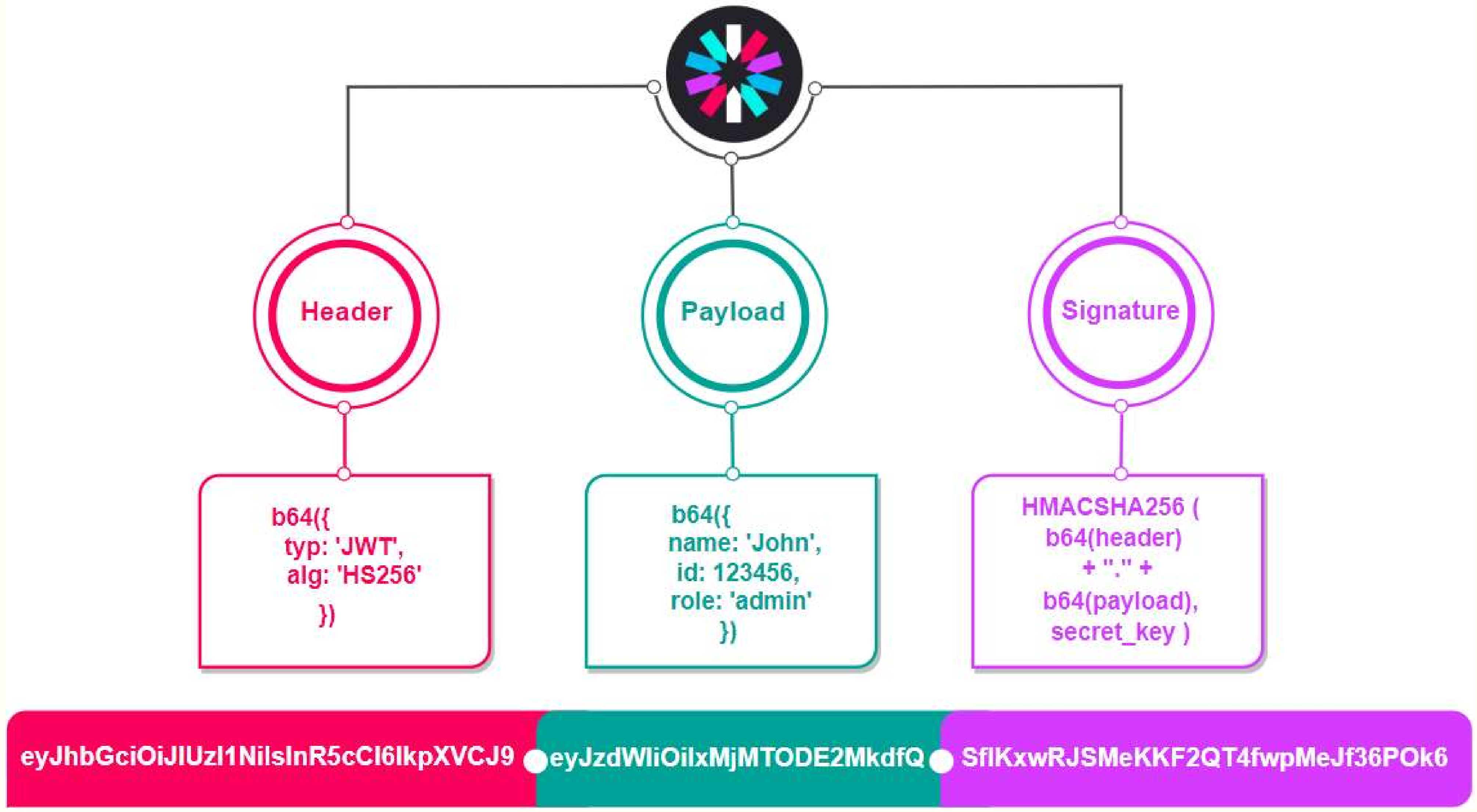
JWT

- **Public Claims** - these are the claims that we create ourselves like user name, information, and other important information.
- **Private Claims** - these are the claims that are private.

Our example payload has two registered claims (iss, and exp) and two public claims (name, admin).



```
{
  "iss": "scotch.io",
  "exp": 1300819380,
  "name": "Chris Sevilleja",
  "admin": true
}
```



```
b64({
  typ: 'JWT',
  alg: 'HS256'
})
```

```
b64({
  name: 'John',
  id: 123456,
  role: 'admin'
})
```

```
HMACSHA256 (
  b64(header)
  + "." +
  b64(payload),
  secret_key )
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMTODE2MkdfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6
```

Hashing

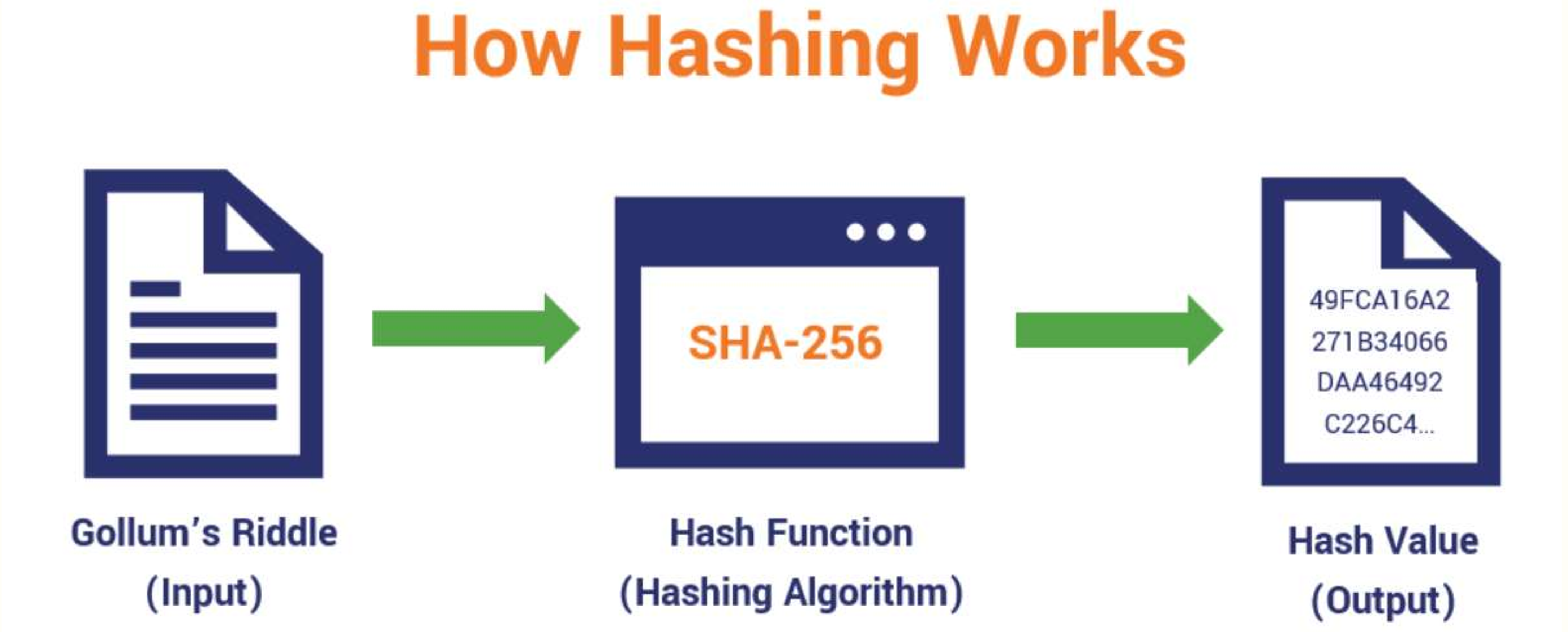
The process of converting an input from a different length to a fixed set of numbers or text of same length (by mathematical algorithm). This process creates a string that is totally different from the string it started. This makes the string useless, unless compared with a hash that came from the same string. Hashing is used for various purposes, mostly for verification of things such as addresses, passwords, file names etc.

Hashing

The hashing process goes like this:

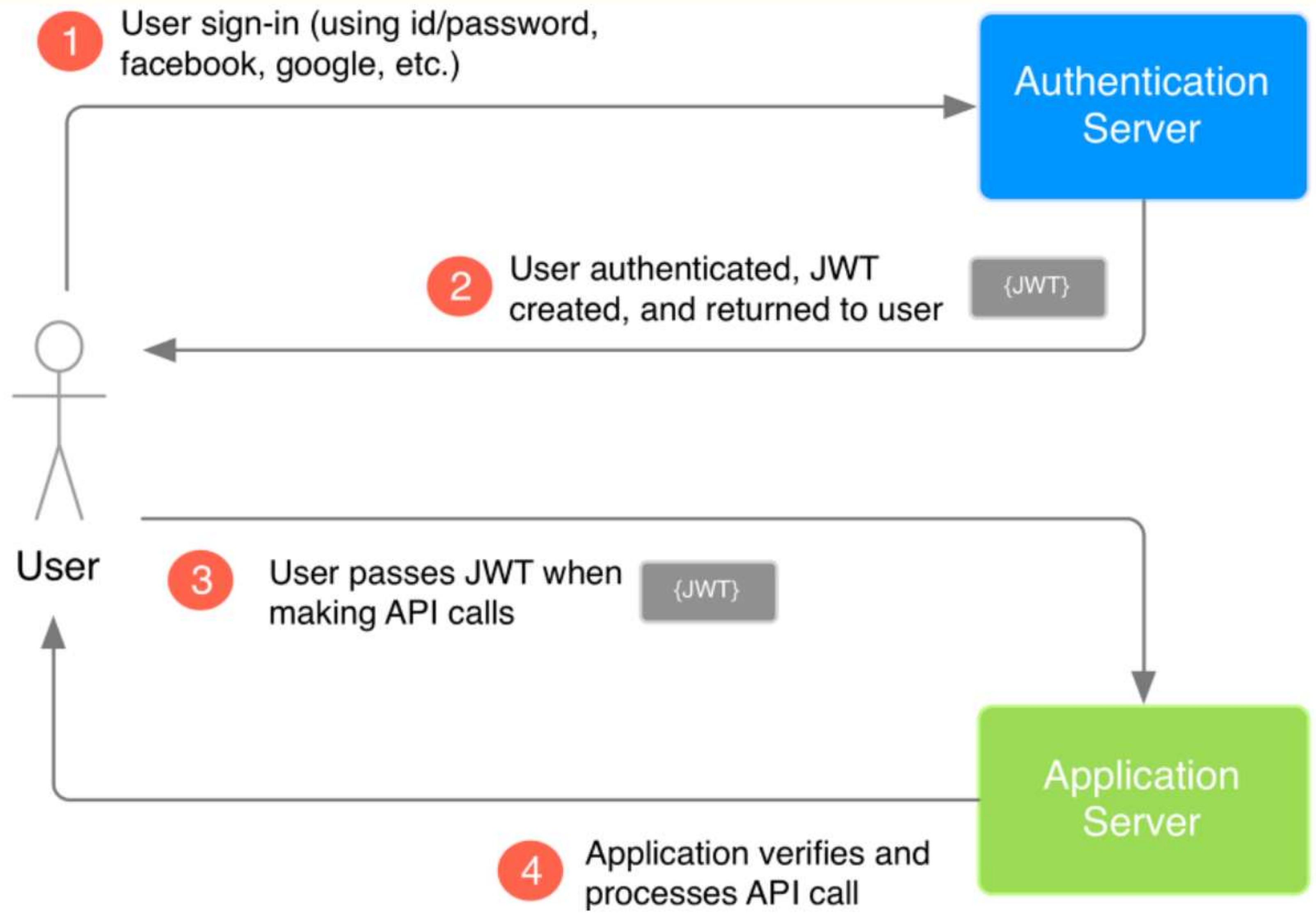
1. **Input**: a string of any length is selected
2. **Hash Method**: A method that executes a mathematical algorithm is selected that will convert the string in to a fixed set of string characters or numbers (MD5, SHA256 etc.)
3. **Hash**: The hash method returns a new string or set of numbers that is called hash

Hashing

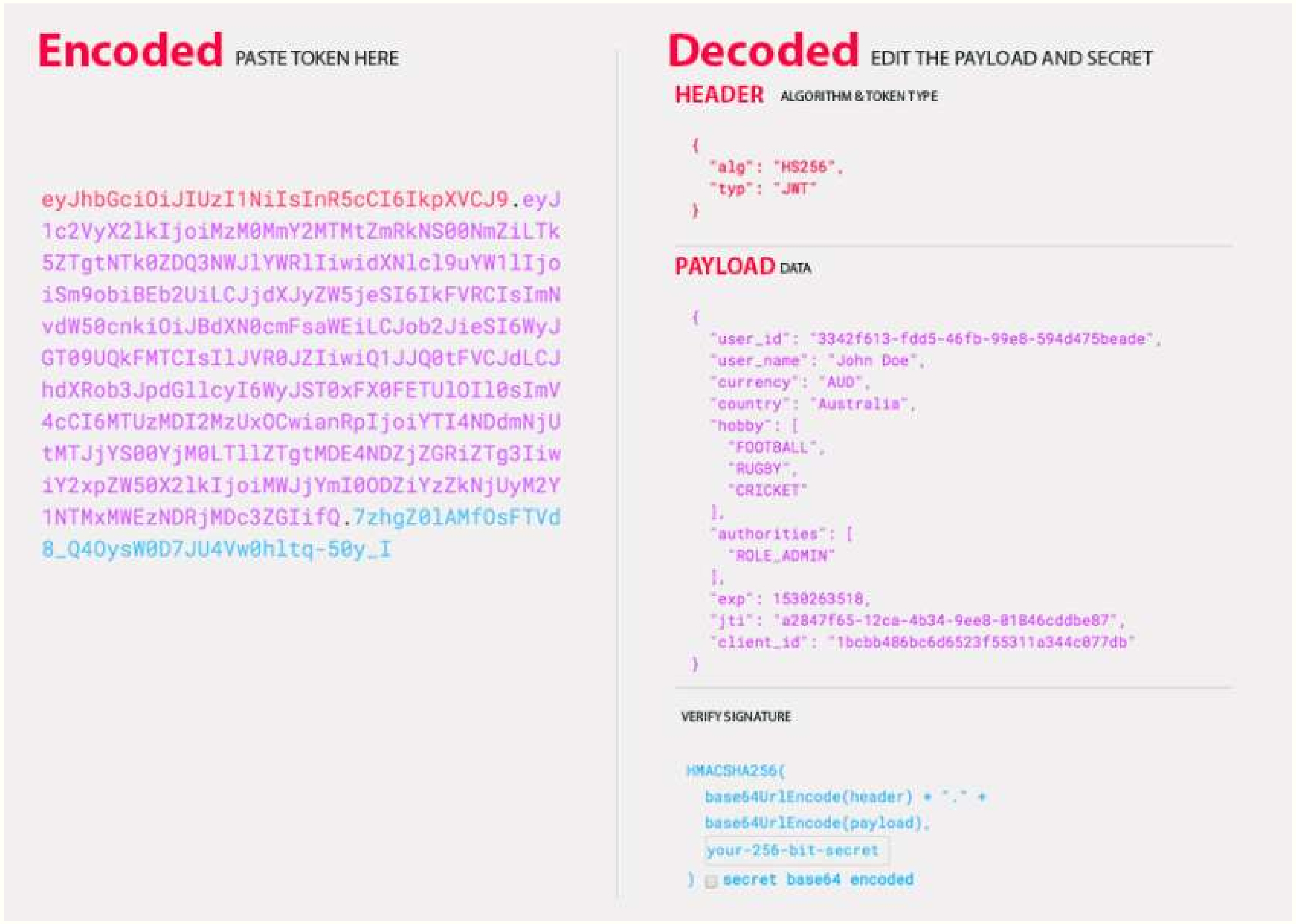

How Hashing Works
SHA-256
49FCA16A2
271B34066
DAA46492
C226C4...
Gollum's Riddle
(Input)
Hash Function
(Hashing Algorithm)
Hash Value
(Output)

How do JWT tokens work

When a Client requests something from a server the server validates the request (Ex: username and password). After that the server creates a token by adding the header, adding some claims about the user that is relevant for when the token is returned back (Ex: Username and UserId). Then it combines them with the secret and creates a hash with the preferred algorithm. It combines everything and creates the JWT string. The token is returned to the client. The client decides how to keep the token. On every other call, the client will just give the token to the server. The token will validate the token to see if it has been changed or tampered. If it is valid it will allow the client request. The token can be invalidated by trying to change it or if the token is expired.


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

Check out jwt.io to play around and check JWT tokens



Encoded PASTE TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
1c2VyX2lkIjoiMzM0MmY2MTMtZmRkNS00NmZiLTk
5ZTgtNTk0ZDQ3NWJlYWRlIiwidXNlcl9uYW1lIjo
iSm9obiBEb2UiLCJjdXJyZW5jeSI6IkFVRCIsImN
vdW50cnkiOiJBdXN0cmFsaWEiLCJob2JieSI6WyJ
GT09UQkFMTCIsIlJVR0JZIiwiQ1JJQ0tFVCJdLCJ
hdXRob3JpdGllcyI6WyJST0xFX0FETUlOIl0sImV
4cCI6MTUzMDI2MzUxOCwianRpIjoiYTI4NDdmNjU
tMTJjYS00YjM0LTllZTgtMDE4NDZjZGRiZTg3Iiw
iY2xpZW50X2lkIjoiMWJjYmI0ODZiYzZkNjUyM2Y
1NTMxMWEzNDRjMDc3ZGIifQ.7zhgZ0lAMfOsFTVd
8_Q4OysW0D7JU4Vw0hltq-50y_I
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER ALGORITHM & TOKEN TYPE

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

PAYLOAD DATA

```
{
  "user_id": "3342f613-fdd5-46fb-99e8-594d475beade",
  "user_name": "John Doe",
  "currency": "AUD",
  "country": "Australia",
  "hobby": [
    "FOOTBALL",
    "RUGBY",
    "CRICKET"
  ],
  "authorities": [
    "ROLE_ADMIN"
  ],
  "exp": 1530263518,
  "jti": "a2847f65-12ca-4b34-9ee8-01846cddbe87",
  "client_id": "1bcbb486bc6d6523f55311a344c077db"
}
```

VERIFY SIGNATURE

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

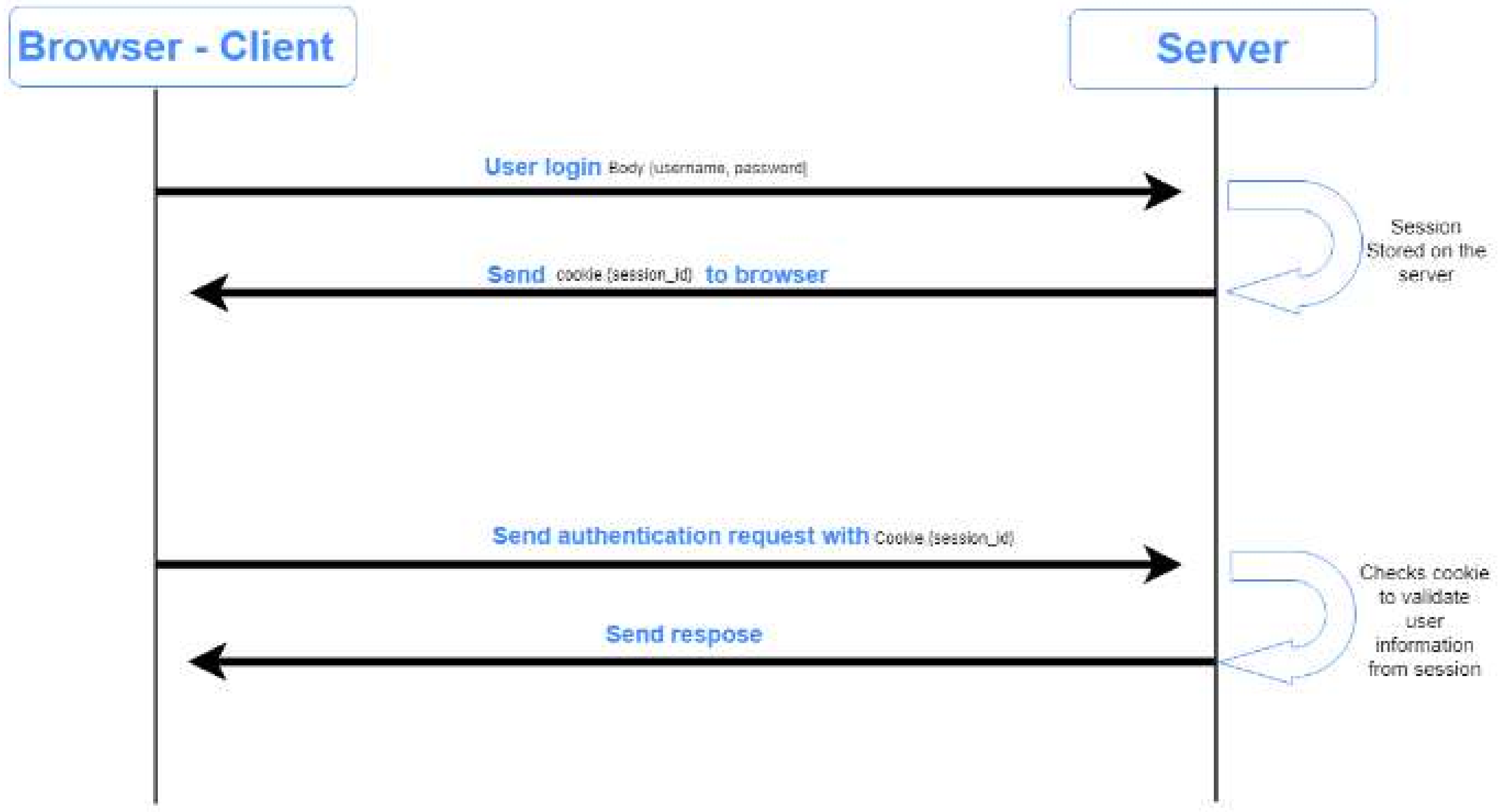
Authentication flow

1. User submits login credentials (username, pass)
2. Server verifies those from the DB
3. Server creates a temporary user session
4. Server issues a cookie with a session ID
5. User sends the cookie with each request
6. Server validates it agains the stored session and grants access
7. When user logs out it destroys the session and clears the cookie

This system is also called stateful because we keep sessions on the server as well as keep data on the client (id).

Stateful Cloud

Cookie-Based


Browser - Client
Server
User login Body (username, password)
Session Stored on the server
Send cookie (session_id) to browser
Send authentication request with Cookie (session_id)
Checks cookie to validate user information from session
Send respose

QUESTIONS?

You can find us at:

mail

main