NETWORK SECURITY

Section-1

Eng. Omar Sameh

INTRODUCTION TO JWT

WHAT IS JWT?

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- This information can be verified and trusted because it is digitally signed.
- JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
- Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties.

WHEN TO USE JSON WEB TOKENS (JWT)?

Here are some scenarios where JSON Web Tokens are useful:

- **Authorization:**
  - This is the most common scenario for using JWT.
  - 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.
  - Single Sign On is a feature that widely uses JWT nowadays.
- **Information Exchange:**
  - JSON Web Tokens are a good way of securely transmitting information between parties.
  - Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are.
  - Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

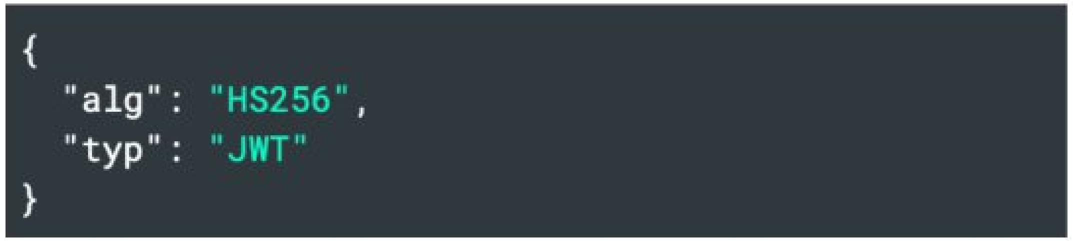
STRUCTURE OF JSON WEB TOKENS (JWT)

- In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:
  - **Header**
  - **Payload**
  - **Signature**
- Therefore, a JWT typically looks like the following:

```
xxxxx.yyyyy.zzzzz
```

STRUCTURE OF JWT: HEADER

- The header *typically* consists of two parts:
  - the type of the token, which is JWT.
  - and the signing algorithm being used, such as HMAC SHA256 or RSA.

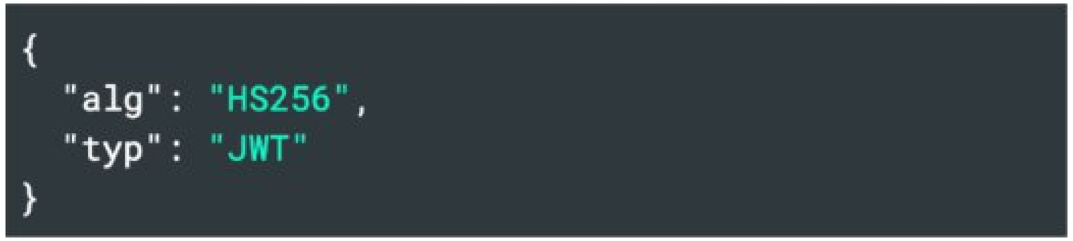


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

STRUCTURE OF JWT: HEADER

- The header *typically* consists of two parts:
  - the type of the token, which is JWT.
  - and the signing algorithm being used, such as HMAC SHA256 or RSA.



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

- Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

STRUCTURE OF JWT: PAYLOAD

- The second part of the token is the payload, which contains the claims.
- Claims are statements about an entity (typically, the user) and additional data.
- There are three types of claims: registered, public, and private claims.
  - **Registered claims**: These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others.
  - **Public claims**: These can be defined at will by those using JWTs.
  - **Private claims**: These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

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

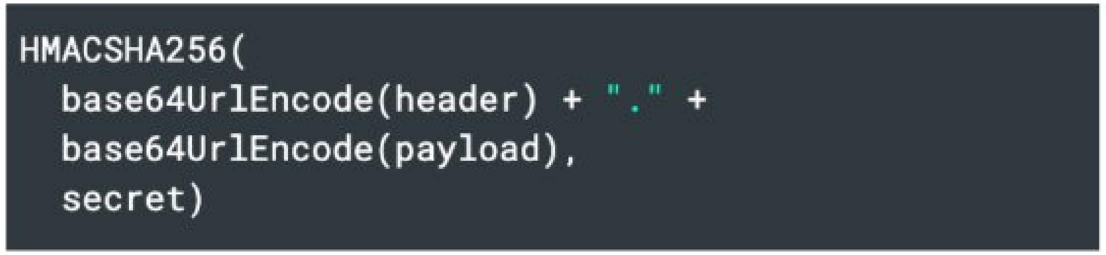
The payload is then **Base64Url** encoded to form the second part of the JSON Web Token.

STRUCTURE OF JWT: PAYLOAD CONT.

```python
payload = {
    "iss": "example.com",  # Public claim - Issuer
    "sub": "user123",       # Public claim - Subject (User ID)
    "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=1),  # Expiry time
    "role": "admin",        # Private claim - Custom role
    "permissions": ["read", "write"]  # Private claim - Custom permissions
}
```

STRUCTURE OF JWT: SIGNATURE

- To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.
- 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 signature is used to verify that the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

STRUCTURE OF JWT CONT.

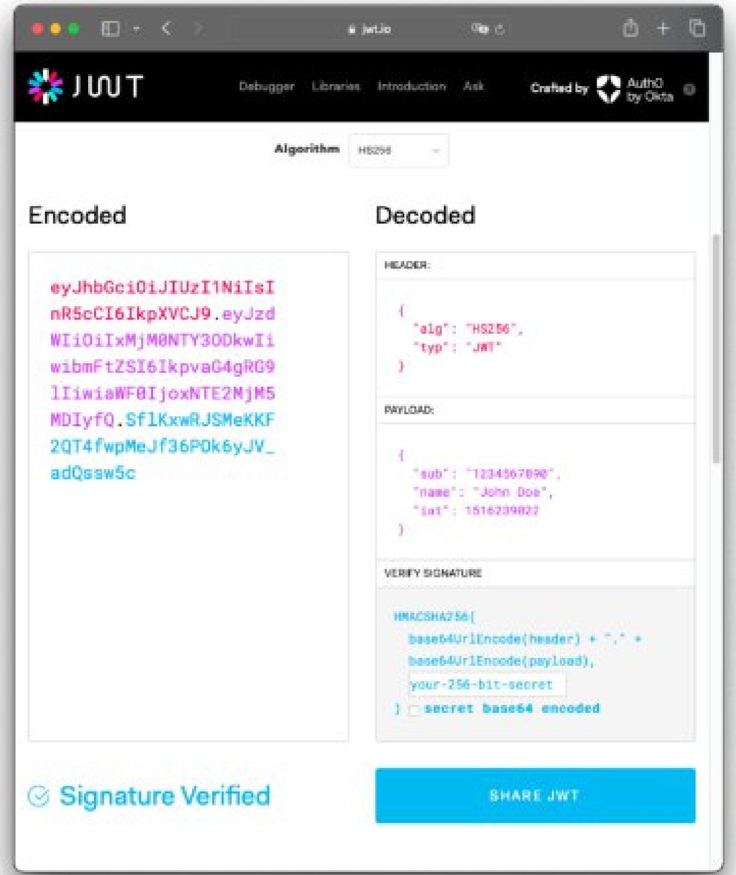
- The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments.
- The following shows a JWT that has the previous **header** and **payload** encoded, and it is signed with a **secret key**:

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

STRUCTURE OF JWT CONT.

- If you want to play with JWT and put these concepts into practice, you can use jwt.io Debugger to decode, verify, and generate JWTs.

VIEW ON JWT.IO



Algorithm HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsI
nR5cCI6IkpXVCJ9.eyJzd
WIiOiIxMjM0NTY3ODkwIi
wibmFtZSI6IkpvaG4gRG9
lIiwiaWF0IjoxNTE2MjM5
MDIyfQ.SflKxwRJSMeKKF
2QT4fwpMeJf36POk6yJV_
adQssw5c
```

Decoded

HEADER:

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

PAYLOAD:

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

VERIFY SIGNATURE

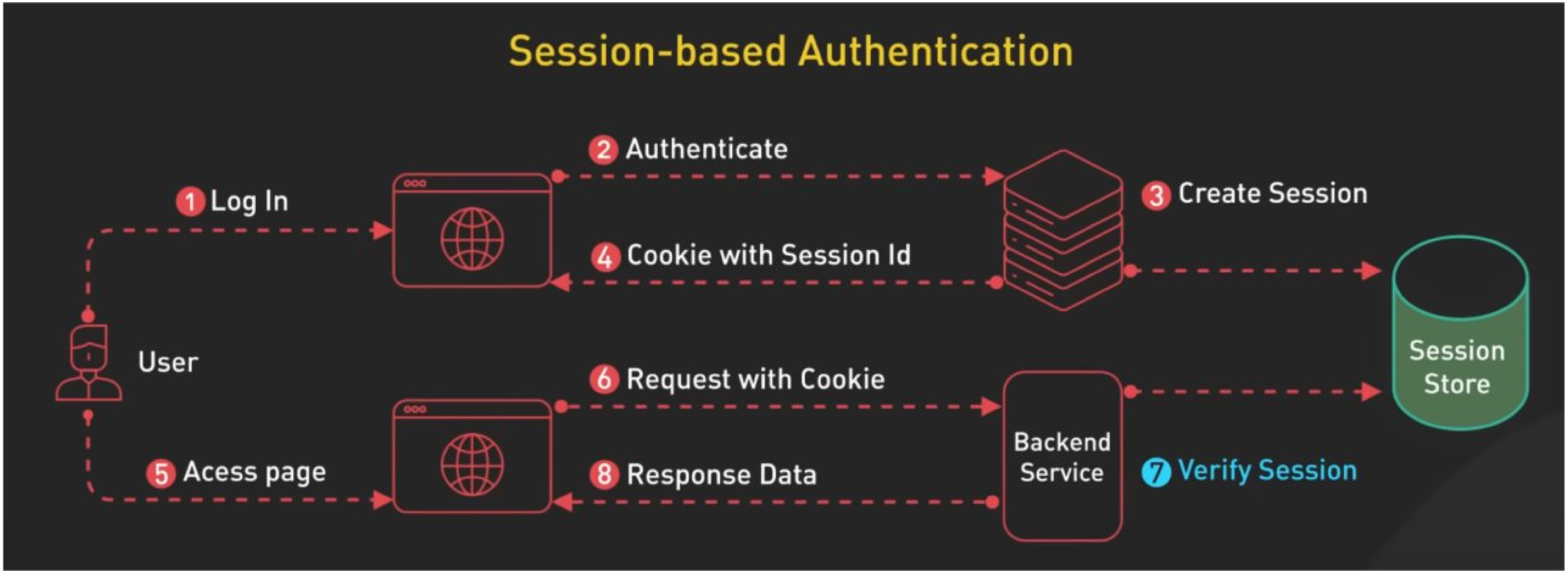
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Signature Verified

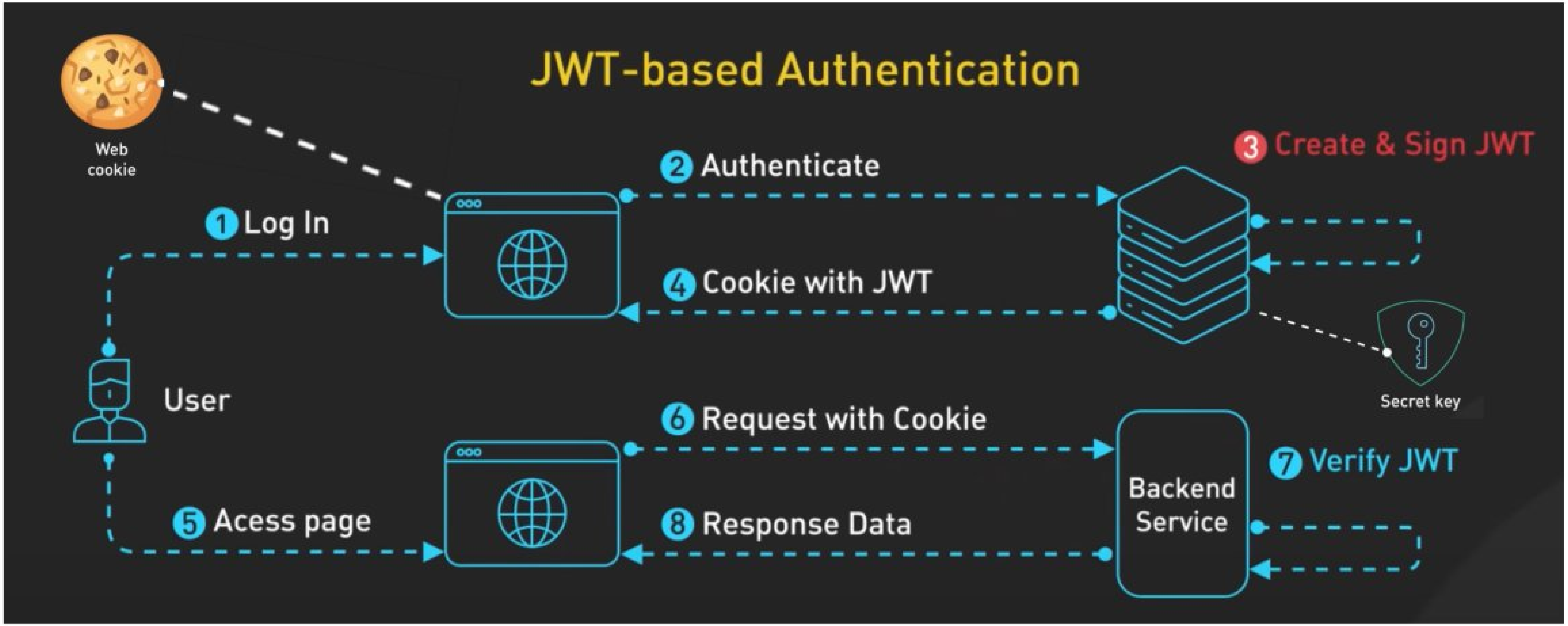
SHARE JWT

JWT

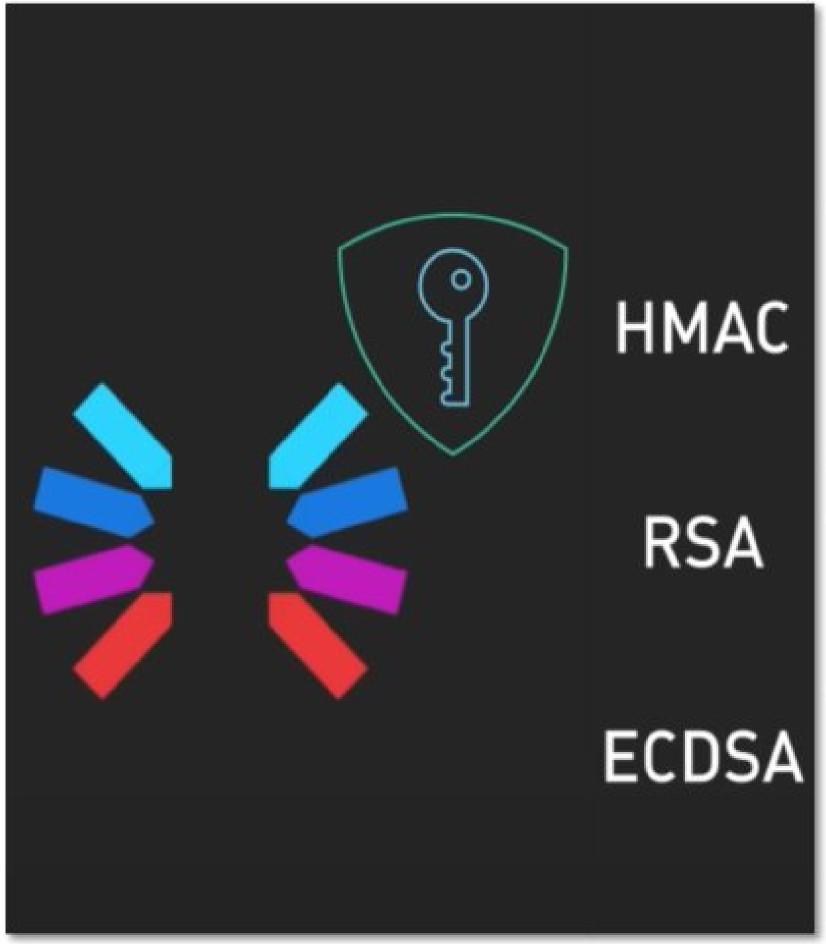
HOW JWT vs SESSION WORK



HOW JWT vs SESSION WORK CONT.



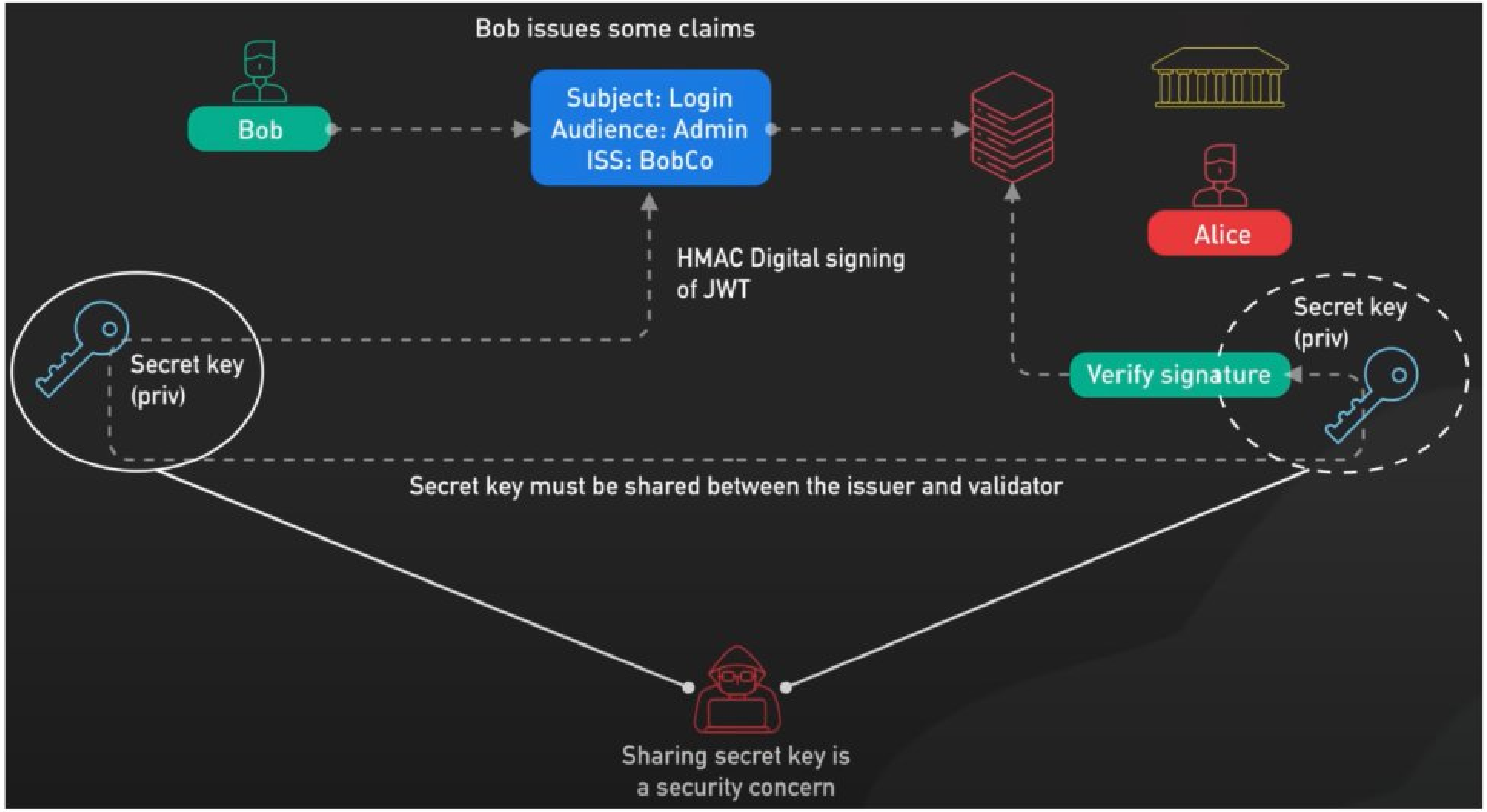
TYPES OF SIGNING ALGORITHMS



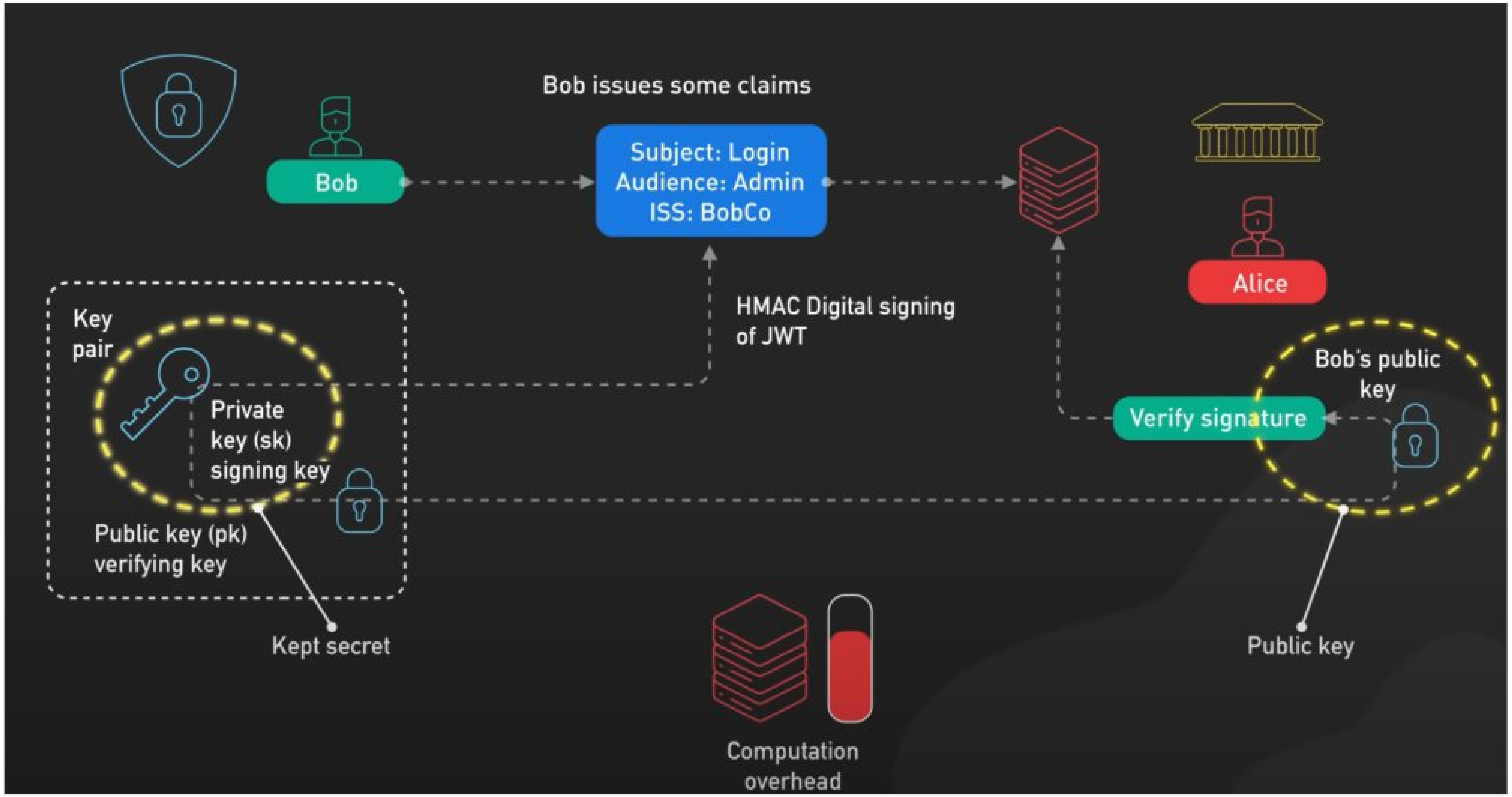
- HMAC is a symmetric signing method, which means the same secret key is used to sign and verify the token. This is simpler and more efficient,
- but it requires sharing the secret key with any service that needs to verify the token, which can be a security concern.

- RSA and ECDSA, on the other hand, are asymmetric signing methods.
- They use a private key to sign the token and a public key to verify it.
- This allows for a more secure architecture where the private key is kept secret and only used for signing, while any service can verify the token using the public key.
- However, this adds some complexity and computational overhead compared to HMAC.

TYPES OF SIGNING ALGORITHMS : HMAC

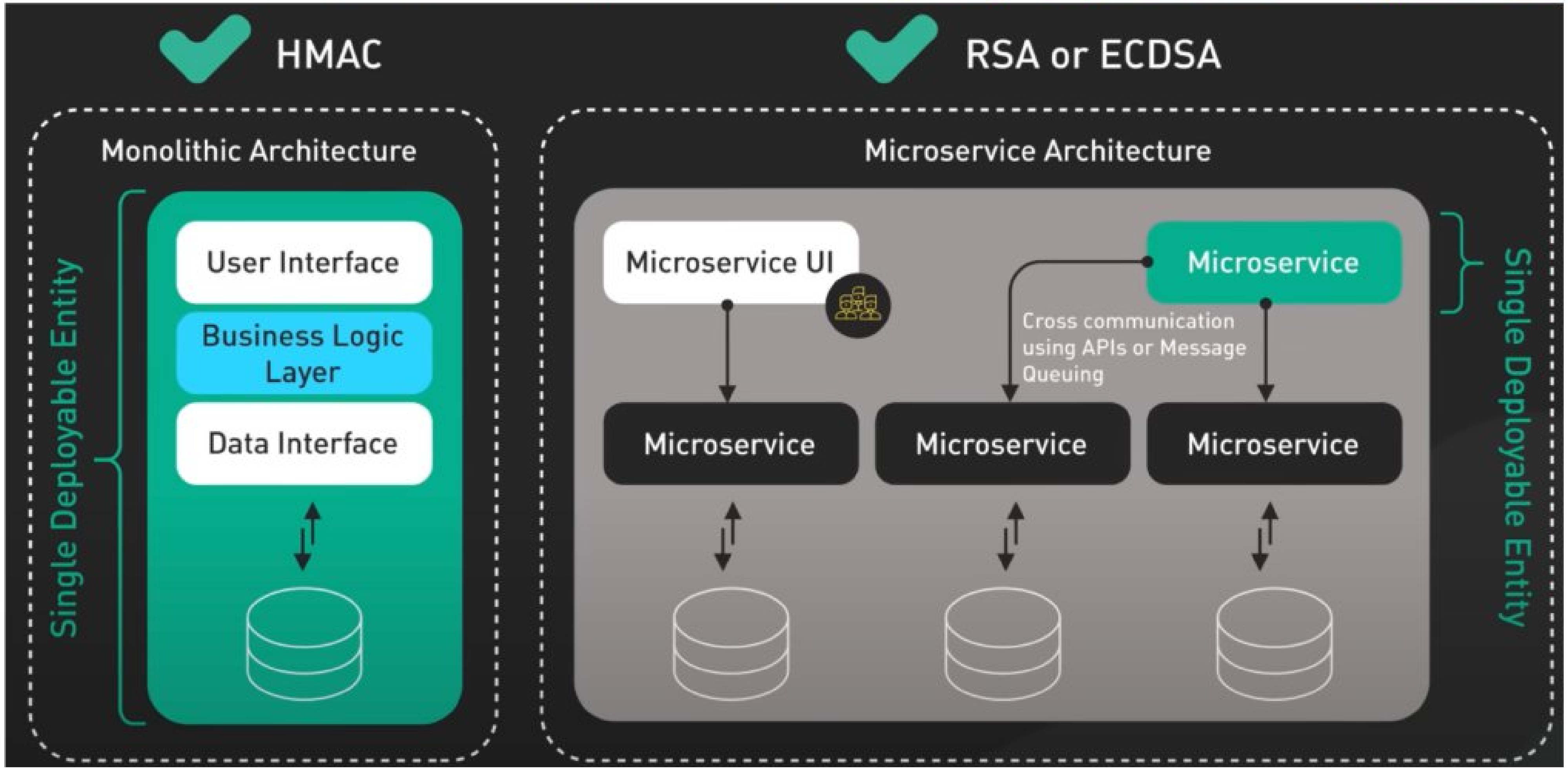

Bob issues some claims
Bob
Subject: Login
Audience: Admin
ISS: BobCo
Alice
HMAC Digital signing of JWT
Secret key (priv)
Verify signature
Secret key (priv)
Secret key must be shared between the issuer and validator
Sharing secret key is a security concern

TYPES OF SIGNING ALGORITHM : RSA & ECDSA



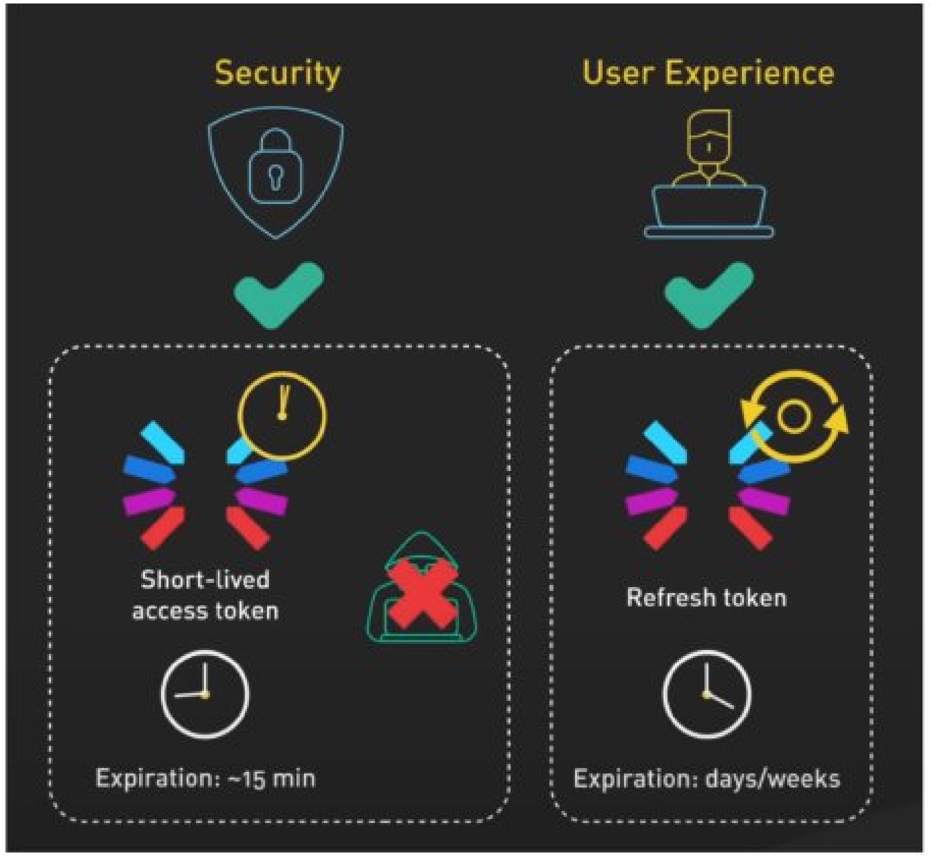
THE CHOICE OF SIGNING ALGORITHM

- The choice of signing algorithm depends on your security requirements and system architecture.

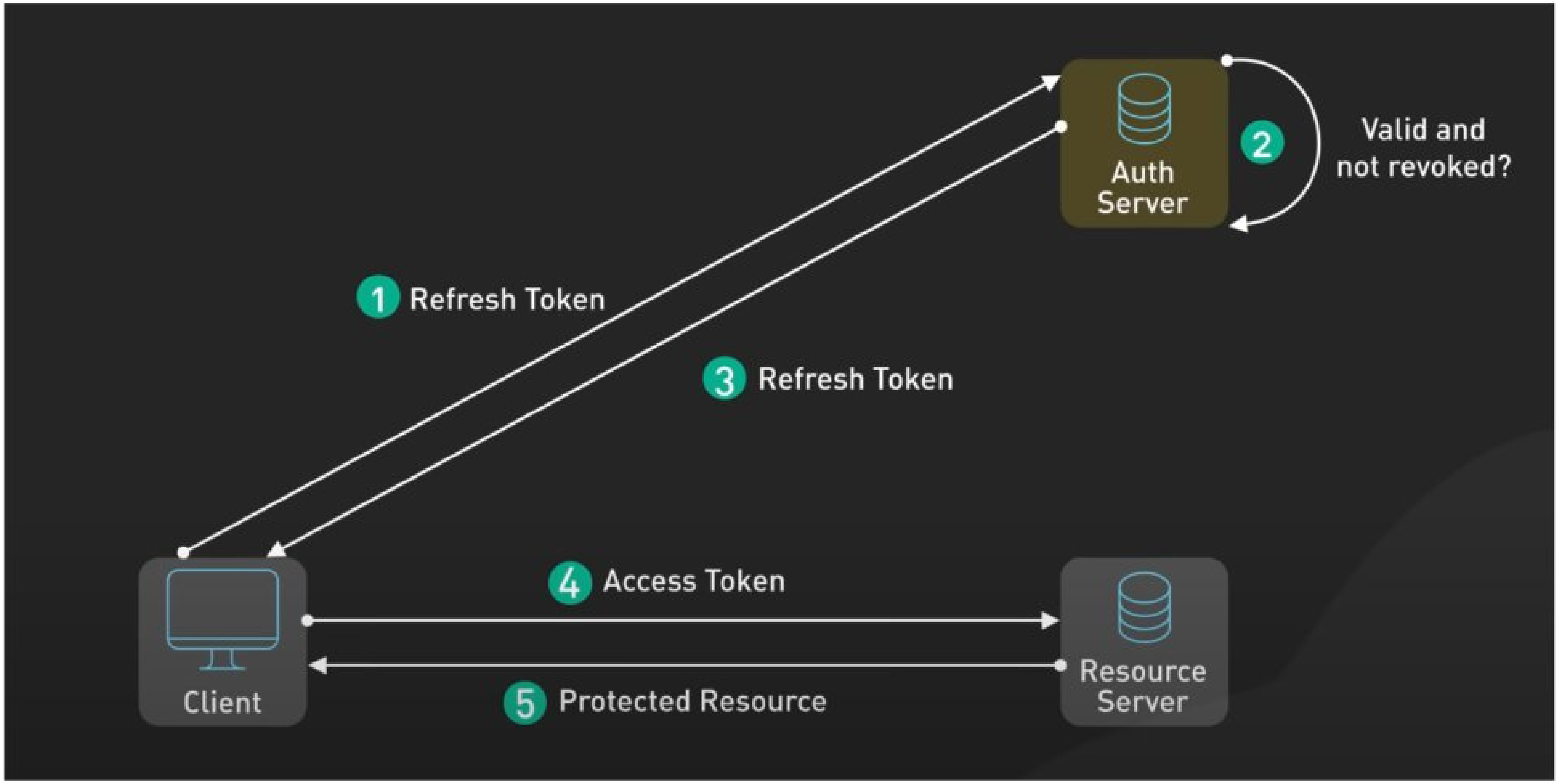


HANDLING TOKEN EXPIRATION.

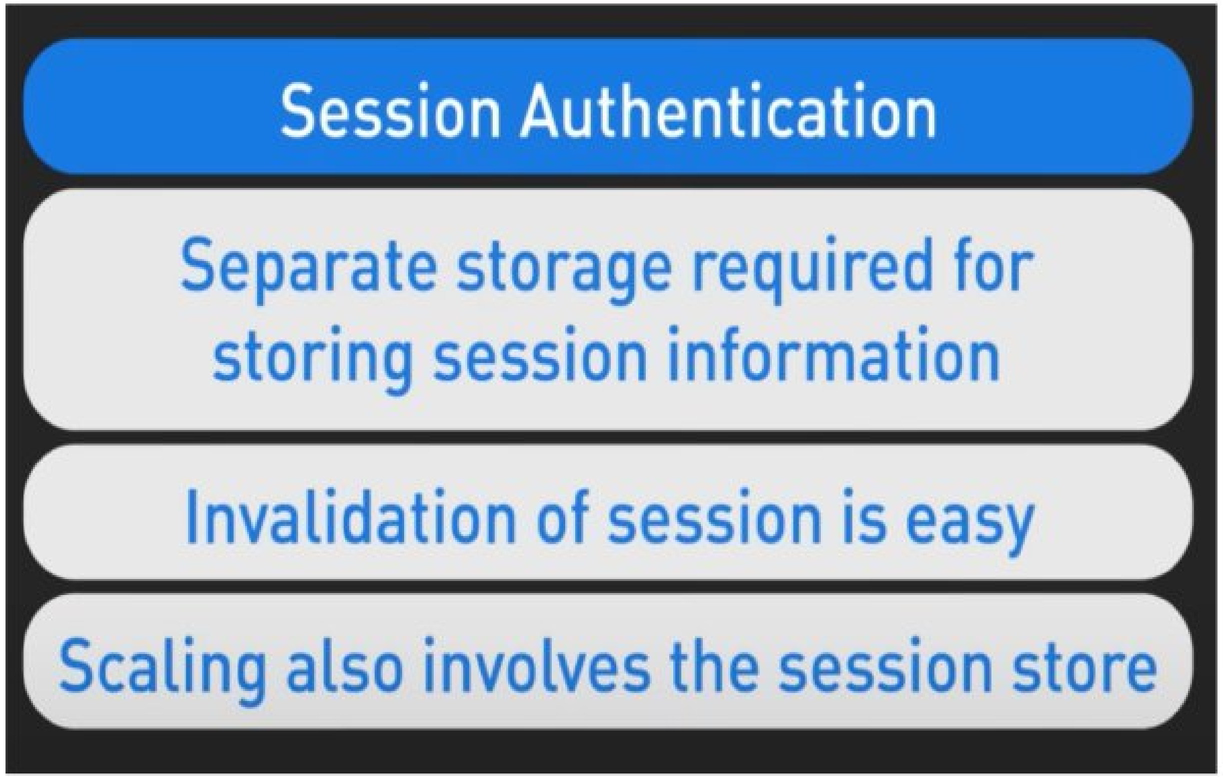
- The short-lived access tokens limit the window of potential misuse if a token is stolen,
- while the long-lived refresh tokens allow users to remain authenticated for an extended period without needing to log in repeatedly.



HANDLING TOKEN EXPIRATION: REFRESH TOKEN

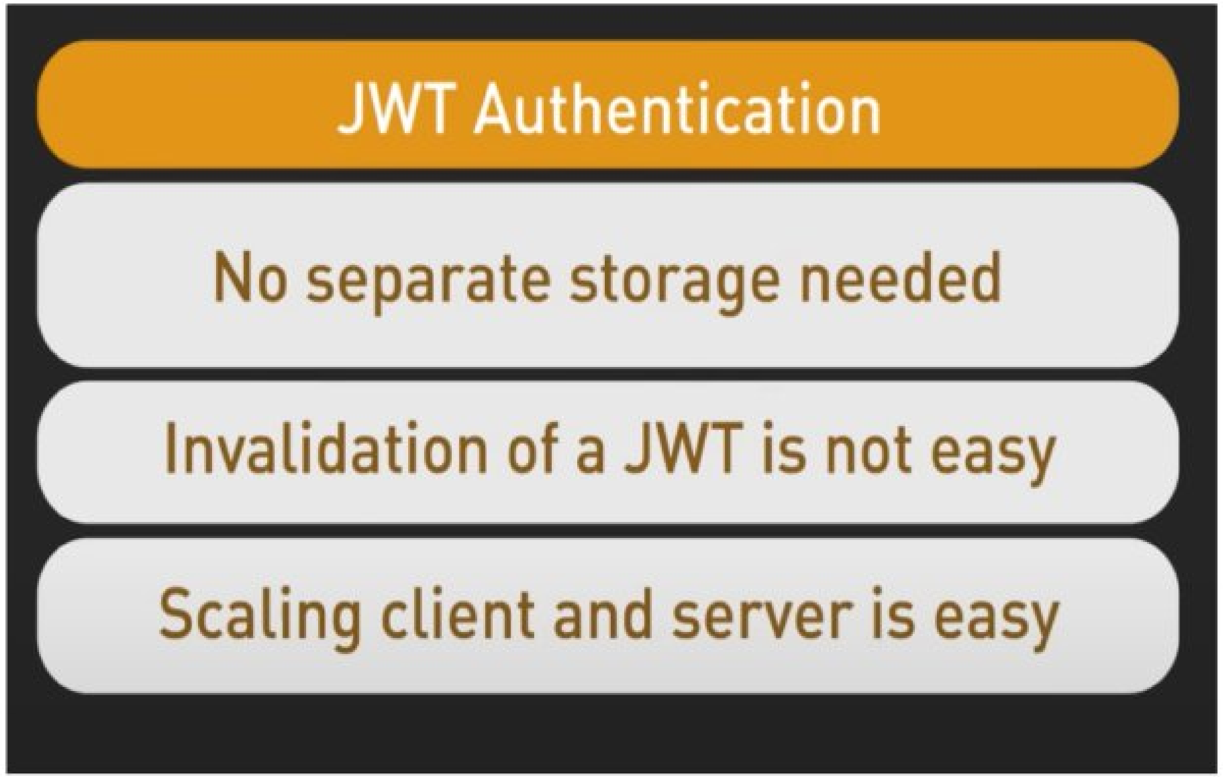


TO CONCLUDE



Session Authentication

- Separate storage required for storing session information
- Invalidation of session is easy
- Scaling also involves the session store



JWT Authentication

- No separate storage needed
- Invalidation of a JWT is not easy
- Scaling client and server is easy

USE ANY FRAMEWORK YOU LIKE TO ACHIEVE AUTHENTICATION THROUGH JWT

Tutorial for using JWT in Laravel

https://youtu.be/KxRtxw6Q-7c?feature=shared

https://laracasts.com/discuss/channels/laravel/best-way-to-implement-jwt-authentication-in-laravel

Tutorial for using JWT in Node.js

https://www.youtube.com/watch?v=mbsmsi7l3r4&t=0s

https://www.geeksforgeeks.org/jwt-authentication-with-node-js/

THANK YOU!