JSON Web Token (JWT)

Introduction

JWT stands for JSON Web Token. It's a compact and self-contained way to transmit information between two parties as a JSON object securely. JWTs are often used for authentication and authorization in web applications. They are commonly used to verify the identity of a user.

JWTs are digitally signed and optionally encrypted. They can be sent via URL, POST, or using an HTTP header.

1. Structure of JWT:

A JWT consists of three parts, separated by dots (e.g., xxxxx.yyyyy.zzzzz):

1. **Header**: Contains metadata about the type of token and the signing algorithm. Example:

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

In this code:

.

- The JWT starts with a header, a JSON object containing metadata about the token.
- "alg" specifies the algorithm used for signing the token, in this case, "HS256," which stands for HMAC-SHA256. This algorithm uses a secret key for both signing and verifying the token.
- "typ" indicates the token type, which is "JWT."

2. **Payload**: Contains claims. Claims are statements about an entity (typically, the user) and additional data. For example

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

In this code:

- The payload is another JSON object that contains claims. Claims are statements about an entity (typically, the user) and additional data.
- "sub" (subject) identifies the subject of the JWT, often represented as the user's unique identifier.
- "name" may contain additional information about the user.
- "iat" (issued at) represents the timestamp when the token was created, specified in seconds since the Unix epoch (January 1, 1970).

There are three types of claims:

- **Registered Claims**: These are a set of predefined claims, not mandatory but recommended. Examples include iss (issuer), exp (expiration time), and sub (subject).
- **Public Claims**: These can be defined at will and need to be shared between parties using the token. They should be defined in the IANA JSON Web Token Registry or as a URI.
- **Private Claims**: These are the custom claims created to share information between parties that agree to use them.

For example:

```
{
  "iss": "your_issuer",
  "sub": "1234567890",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "role": "user",
  "exp": 1645872000,
  "nbf": 1645795600,
  "iat": 1645795600,
  "jti": "a1b2c3d4e5f6g7h8i9j0"
}
```

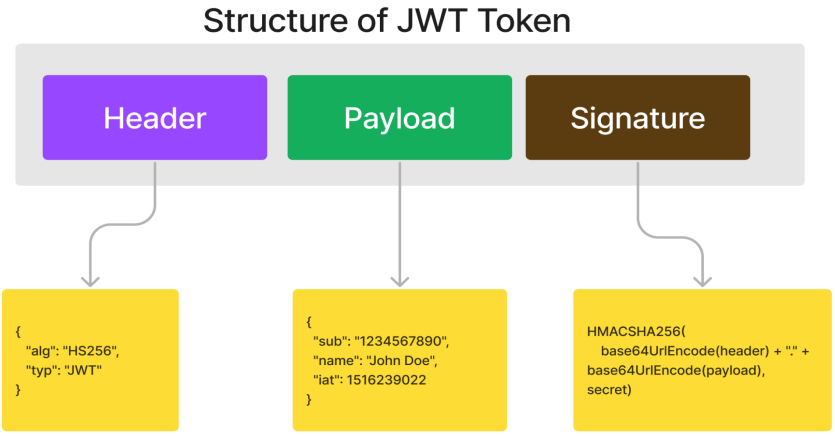
3. **Signature**: To create the signature part, you have to take the encoded header, the encoded payload, a secret, and the algorithm specified in the header, and sign that. Example:

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

In this code:

- The signature is created by combining the encoded header and payload with a dot ('.'), then applying the HMAC-SHA256 hashing algorithm using a secret key.
- The base64UrlEncode function converts the header and payload into a URL-safe base64-encoded string.

- The secret key is shared between the server (issuer) and the party verifying the token.



2. How JWT Works

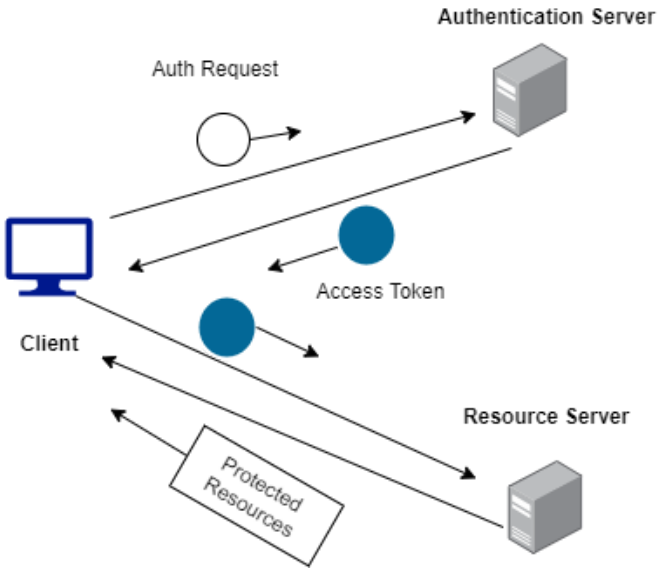
1. **Authentication**: When a user logs in, the server generates a JWT and sends it back to the client.

2. **Authorization**: The client stores the JWT and sends it with every subsequent request to the server.

3. **Validation**: The server validates the JWT to ensure it's not tampered with and is still valid. It checks the signature and expiration time.

4. **Access Control**: The server extracts user information from the JWT and uses it for access control. For example, the user's roles and permissions can be included in the payload.

   Refer to the flow diagram below for more clarity.



3. Use Cases

1. **Single Sign-On (SSO)**: JWTs are commonly used for SSO. Once a user logs in to one application, the JWT can be used to access other related applications without the need to log in again.

2. **API Authentication:** JWTs are used to secure APIs. Clients (e.g., mobile apps or other servers) send JWTs in API requests to prove their authenticity.

3. **Stateless Authentication:** JWTs make authentication stateless. The server doesn't need to keep session information, which is beneficial for scalability.

4. Security Considerations

- Don't store sensitive data in the JWT payload; it's not encrypted.

- Use HTTPS to protect the JWT during transmission.

- Set appropriate expiration times to limit the lifetime of a JWT.

5. Conclusion

- JSON Web Tokens (JWTs) are a powerful and widely adopted method for securely transmitting information between parties, commonly used for authentication and authorisation in web applications and APIs.

- The structure of a JWT consists of three parts: the Header, Payload, and Signature. The Header specifies the signing algorithm and token type, while the Payload contains claims, including information about the subject and additional data. The Signature ensures the integrity and authenticity of the token.

- JWTs offer numerous benefits, including stateless authentication, Single Sign-On (SSO) support, and secure API authentication. They are flexible and versatile, allowing custom claims to be included, making them suitable for various use cases.

- When implementing JWT-based authentication, it is crucial to protect the secret key, set appropriate expiration times, use HTTPS for transmission, and avoid storing sensitive data in the token's payload. By following these best practices, JWTs become a robust solution for secure data exchange in modern web applications.

6. References

1. JWT in Springboot
2. More on JWT.
3. JWT: Official Documentation