***What is JWT (JSON Web Token)***

*JSON Web Token (JWT) is an open standard (RFC 7519) for securely transmitting information between parties as JSON object. It is compact, readable, and digitally signed using a private key/ or a public key pair by the Identity Provider (IdP). So, the integrity and authenticity of the token can be verified by other parties involved.*

*The purpose of using JWT is not to hide data but to ensure the authenticity of the data. JWT is signed and encoded, not encrypted.*

*JWT is a token based stateless authentication mechanism. Since it is a client-side based stateless session, server doesn't have to completely rely on a datastore(database) to save session information.*

***Structure of JWT***

*A JSON Web Token consists of 3 parts separated by a period. - header.payload.signature*

* *Header - JWT header consists of token type and algorithm used for signing and encoding. Algorithms can be HMAC, SHA256, RSA, HS256 or RS256.*

{

"sub": "user10001",

"iat": 1569302116

}

* *Payload - Payload consists of the session data called as claims. Below are some of the standard claims that we can use -* 
  + - *Issuer(iss)*
    - *Subject (sub)*
    - *Audience (aud)*
    - *Expiration time (exp)*
    - *Issued at (iat)*

*Custom claims can also be included in the claim set. When using custom claim sets,*

1. *Do not put large data in claim sets. Claim sets meant to be compact.*
2. *Do not put sensitive informations since, JWT can be decoded easily.*

{

"sub": "user10001",

"iat": 1569302116,

"role": "admin",

"user\_id": "user10001"

}

* *Signature - Signature is most important part of a JSON Web Token (JWT). Signature is calculated by encoding the header and payload using Base64url Encoding and concatenating them with a period separator. Which is then given to the cryptographic algorithm.*

*So, when the header or payload changes, signature has to calculated again. Only the Identity Provider (IdP) has the private key to calculate the signature which prevents the tampering of token.*

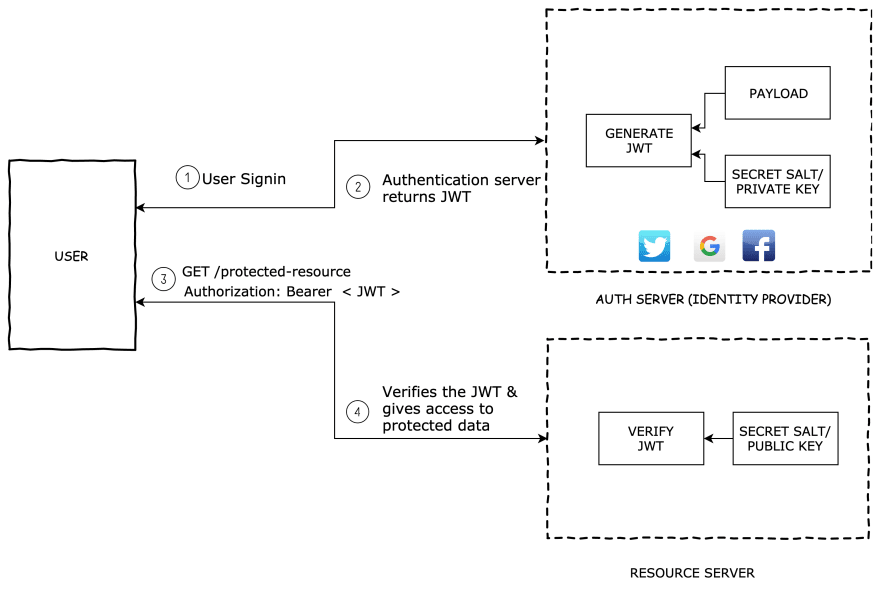
// signature algorithm

data = base64urlEncode( header ) + "." + base64urlEncode( payload )

signature = HMAC-SHA256( data, secret\_salt )

***How it works?***

*Basically the identity provider (IdP) generates a JWT certifying user identity and Resource server decodes and verifies the authenticity of the token using secret salt / public key.*

**

* *User sign-in using username and password or google/facebook.*
* *Authentication server verifies the credentials and issues a jwt signed using either a secret salt or a private key.*
* *User's Client uses the JWT to access protected resources by passing the JWT in HTTP Authorization header.*
* *Resource server then verifies the authenticity of the token using the secret salt/ public key.*

*Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema.*

*This can be, in certain cases, a stateless authorization mechanism. 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. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case.*

*Note that if you send JWT tokens through HTTP headers, you should try to prevent them from getting too big. Some servers don't accept more than 8 KB in headers. If you are trying to embed too much information in a JWT token, like by including all the user's permissions, you may need an alternative solution, like Auth0 Fine-Grained Authorization.*

*If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.*

***Security -*** *Just like any other authentication mechanism, JWT also has its own pros and cons.*

* *Must use HTTPS to secure the Authorization headers.*
* *Validate algorithm name explicitly. Do not completely rely on the algorithm mentioned in the header of JWT. There are a few known attacks based on the header like algo none attack, header stripping.*
* *Revoking the session of a user from backend server is difficult. Since a JWT is set to automatically expire, If an attacker gets the token before it expires It leads to various exploits. Building a token revocation list on your server to invalidate tokens.*
* *If JWT is persisted on cookies, we need to create HttpOnly cookie. This will restrict third party javascripts from reading jwt token from cookie.*
* *XSS - backend servers must always sanitize user generated data.*
* *CSRF  -  If JWT in persisted on cookies, CSRF attacks are possible. We can mitigate CSRF by using origin of request and special request headers.*

*What if JWT token stolen –*

1. *This is what SSL helps prevent against: by encrypting your NETWORK traffic from your computer -> some server when authenticating, a third party who is monitoring your network traffic can NOT see your tokens, passwords, or anything like that unless they're somehow able to get a copy of the server's private SSL key (unlikely). This is the reason SSL is MANDATORY for all forms of authentication.*
2. *The way OAuth2 works is that it was designed to give clients TEMPORARY tokens (like JWTs!) for authentication for a SHORT PERIOD OF TIME ONLY! The idea is that if your token gets stolen, the attacker can only use it for a short period of time. With OAuth2, you have to re-authenticate yourself with the server every so often by supplying your username/password OR API credentials and then getting a token back in exchange. Because this process happens every now and then, your tokens will frequently change, making it harder for attackers to constantly impersonate you without going through great trouble.*