# Introduction:

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**.

JWTs are commonly used for authentication, authorization, and information exchange in web applications.

<https://jwt.io/introduction/>

# History and Origin

* **Invented by:** JWT was developed by the **Internet Engineering Task Force (IETF)** to address the need for a stateless, secure, and interoperable token format.
* **Why It Was Invented:** Traditional authentication systems relied on server-side sessions or cookies to track users, which had drawbacks like scalability issues and dependence on server state. JWT introduced a stateless authentication mechanism where the client stores all the necessary session data.

# JWT Token Structure

The most commonly used JWT token consists of 3 parts separated by a dot (.).

1. Header
2. Payload
3. Signature

 **Header**  
Contains metadata about the token, such as the signing algorithm and token type . **Base64Url** encoded to form the first part

Example:json

{

"alg": "HS256", // Algorithm used for signing (HMAC-SHA256 in this case)

"typ": "JWT" // Token type

}

 **Payload**  
Contains claims (statements about an entity, like user data or permissions). There are three types of claims: *registered*, *public*, and *private* claims. The payload is then **Base64Url** encoded to form the second part

Example:json

{

"sub": "1234567890", // Subject (user identifier)

"name": "John Doe", // User's name

"iat": 1516239022, // Issued at timestamp

"role": "admin" // User's role

}

 **Signature**  
Used to verify the token's authenticity. It combines:

scss

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret\_key)

# How JWT Works

1. **Authentication:**
   * A user logs in with their credentials.
   * The server validates the credentials and generates a JWT containing user information (e.g., user ID and roles).
   * The JWT is returned to the client.
2. **Storage:**
   * The client stores the JWT, typically in **localStorage** or **sessionStorage** in a browser or in an **Authorization** header.
3. **Authorization:**
   * The client includes the JWT in subsequent requests to protected endpoints.
   * The server verifies the JWT's signature using the secret key or public key.
   * If valid, the server processes the request.
4. **Stateless Nature:**
   * No session state is stored on the server. The token itself carries all necessary information.

# Security Management in JWT

**1. Signature Verification**

* JWTs are signed using either:
  + **Symmetric Key (e.g., HMAC-SHA256):** Both parties share the same secret.
  + **Asymmetric Key (e.g., RS256):** A private key signs the token, and a public key verifies it.

**2. Expiration Time**

* Tokens should include an **exp (expiration)** claim.
* Example:

json

Copy code

{

"exp": 1635609600 // Unix timestamp

}

* Once expired, the token becomes invalid.

**3. Secure Transmission**

* Always use **HTTPS** to prevent tokens from being intercepted.
* Never transmit tokens over unencrypted channels.

**4. Audience and Issuer Validation**

* Include **aud** (audience) and **iss** (issuer) claims in the token to ensure it is intended for the correct party.
* Example:

json

Copy code

{

"aud": "myapp.com",

"iss": "auth.myapp.com"

}

**5. Revocation**

* JWTs are stateless and cannot be "revoked" directly.
* Implement a blacklist or a short token lifetime with refresh tokens for revocation.

**6. Avoid Storing Sensitive Data**

* Do not include sensitive information in the payload, as JWTs are base64-encoded, not encrypted.

**7. Use Strong Secrets**

* Use a sufficiently strong secret key for signing (256-bit or longer for HMAC).

**8. Refresh Tokens**

* Use refresh tokens with short-lived JWTs to minimize risk if a token is compromised.

# Advantages of JWT

1. **Stateless:**
   * No server-side storage required.
   * Ideal for microservices and distributed systems.
2. **Compact and Efficient:**
   * Small size makes it ideal for HTTP headers.
3. **Interoperability:**
   * Easy to use across different programming languages.
4. **Self-contained:**
   * Contains all the required information in the token itself.

# Disadvantages of JWT

1. **No Built-in Revocation:**
   * Tokens cannot be invalidated once issued without additional mechanisms like blacklists.
2. **Payload Exposure:**
   * The payload is base64-encoded but not encrypted, so it can be decoded by anyone.
3. **Overhead:**
   * Large tokens can increase the size of HTTP requests, affecting performance.

# Usage

**When to Use JWT**

* Stateless Authentication (e.g., APIs, microservices).
* Authorization (e.g., user roles and permissions).
* Information exchange between parties.

**When NOT to Use JWT**

* If revocation is critical and frequent.
* For scenarios requiring heavy encryption of data.

# Key Practices for Secure JWT Usage

1. Always use HTTPS.
2. Set short token lifetimes and use refresh tokens.
3. Validate the token's signature and claims.
4. Avoid storing sensitive information in tokens.
5. Use strong cryptographic keys and algorithms.