JSON WEB TOKENS (JWT)

AHMAD HAMO
19-5-2025

Introduction to JWT

- **Definition**: Compact, URL-safe way to transfer claims between client and server
- **Format**: Encoded JSON objects containing claims

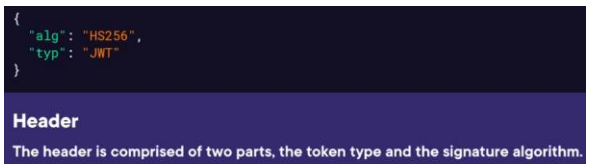
JWT – JSON Web Token

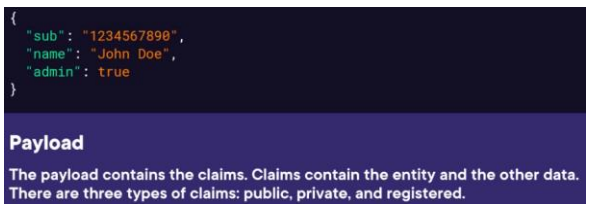
- Instead of repeatedly sending username and password, the server verifies the user once and returns a token.
- From this point, the client sends the token in every API request instead of credentials.
- This improves security, performance, and flexibility.

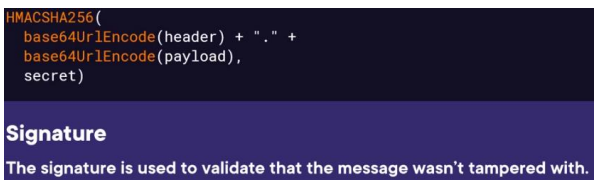
3 biggest advantages of JWT:

- Once the token is issued, your actual password stays out of the network, much safer.
- You can control token expiry so sessions can auto expire, reducing risk.
- And finally, tokens can carry metadata like roles or permissions, making authorization easier.

JSON Web Token Structure

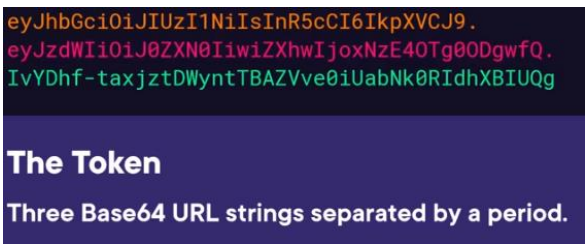






JWT consists of three parts separated by dots:

xxxx.yyyy.zzzz



JWT Structure

Three parts separated by dots:

1. **Header** - Contains token type and signature algorithm (e.g., HS256, RSA)

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

2. **Payload**
   - Contains claims (entity and other data)
   - Three claim types:
     - Registered (predefined)
     - Public (custom)
     - Private (shared between parties)
   - Example: `{"sub": "1234567890", "name": "John Doe", "admin": true}`

JWT Structure

Three parts separated by dots:

3. **Signature**
   - Validates message integrity
   - Created using: **`HMACSHA256(`**`base64UrlEncode(header) + "." + base64UrlEncode(payload),` **`secret`**`)`
   - The **secret** is a shared key(known only to the server).
   - The **HMAC-SHA256** algorithm hashes the concatenated string using the secret.

→JWTs are **signed** (to verify integrity) but **not encrypted** (to hide content). Here's why sensitive data (e.g., passwords, credit card numbers, SSNs) should never be stored in a JWT payload.

JWT Generation using PyJWT

```
import jwt

# Define payload and secret
payload = {
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
}
secret = "your-256-bit-secret"

# Generate JWT
jwt_token = jwt.encode(
    payload,
    secret,
    algorithm="HS256"
)
print(jwt_token)
```

Notice: When you call jwt.encode(), PyJWT internally constructs the header using:

```
{
  "alg": "HS256",  // Taken from the `algorithm` parameter
  "typ": "JWT"     // Always set to "JWT" for JWTs
}
```

Step	Manual Implementation (HMAC + Base64)	PyJWT Library
Header Encoding	Explicit `base64url_encode(header)`	Automatic
Payload Encoding	Explicit `base64url_encode(payload)`	Automatic
Signature Encoding	Manual HMAC + Base64Url	Automatic
JWT Assembly	Manual concatenation with `.`	Automatic

Standard JWT Claims

Claim	Name	Purpose
iss	Issuer	Indicates who created the JWT. This is a single string and often the URI of the authentication service.
aud	Audience	Indicates who the JWT is for. An array of strings identifying the intended recipients of the JWT. If there is only a single value, then it can be a simple string value rather than an array. The recipient of a JWT must check that its identifier appears in the audience; otherwise, it should reject the JWT. Typically, this is a set of URIs for APIs where the token can be used.
iat	Issued-At	The UNIX time at which the JWT was created.
nbf	Not-Before	The JWT should be rejected if used before this time.
exp	Expiry	The UNIX time at which the JWT expires and should be rejected by recipients.
sub	**Subject**	The identity of the subject of the JWT. A string. Usually a username or other unique identifier.
jti	JWT ID	A unique ID for the JWT, which can be used to detect replay.

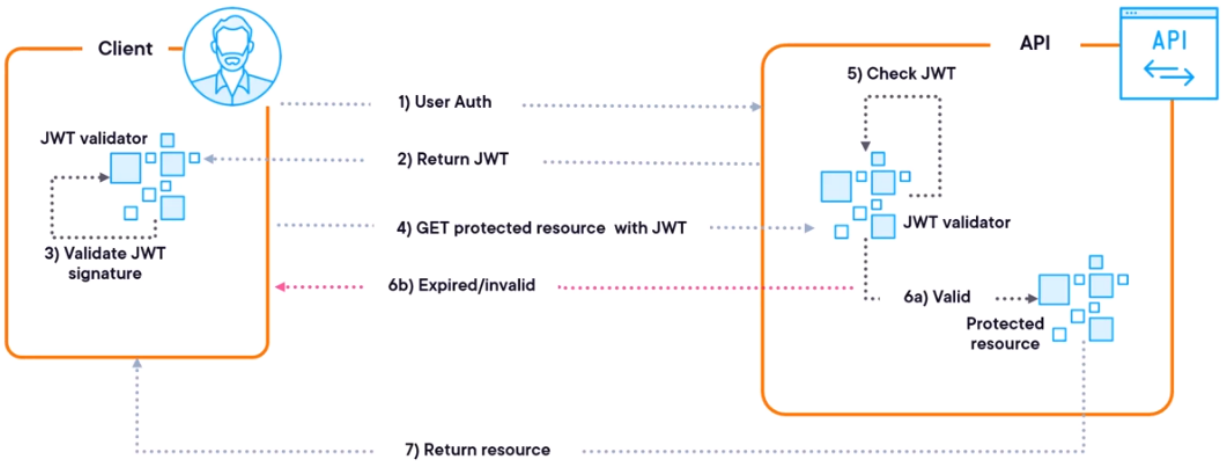
JWT Use Cases

1. **Authentication**
   - Stateless authentication for APIs
   - Cross-domain authentication
   - Mobile and single-page applications (SPAs)
2. **Information Exchange**
   - Securely transmit information between parties
   - Signed tokens allow verification of content integrity

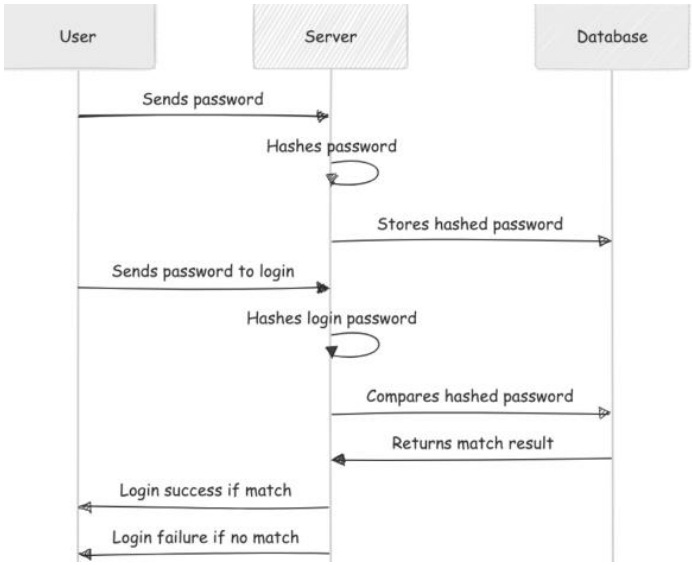
JWT Lifecycle

1. User logs in with credentials
2. Server validates credentials and issues JWT
3. Client stores JWT securely
4. Client includes JWT in Authorization header for protected resources:
   - `Authorization: Bearer <token>`
5. Server validates JWT signature and claims
6. If valid, server grants access to protected resource

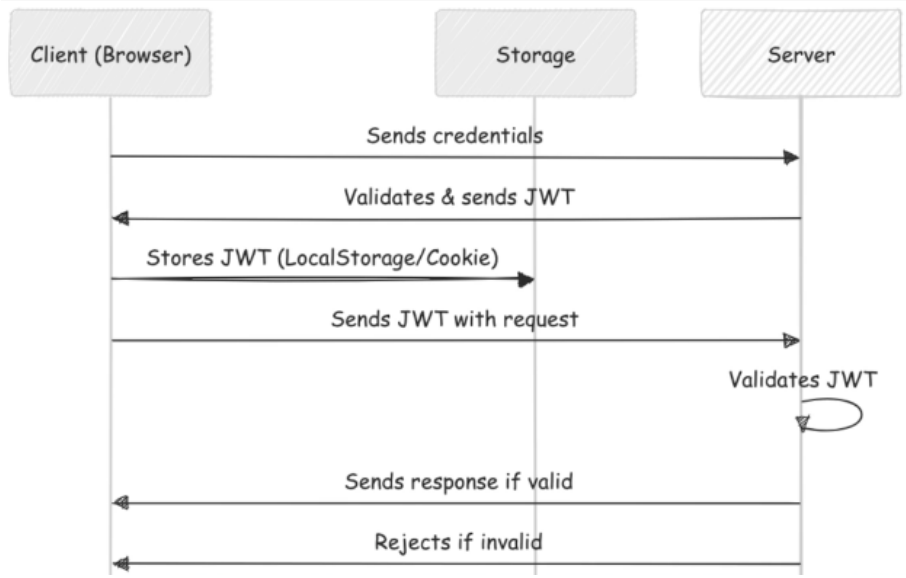
JWT - Lifecycle



Login & Password Hashing



JWT Auth using Storage



Refresh Tokens

- **Purpose**: Renew short-lived access tokens without requiring user to re-login
- **Implementation**:
  - Issued alongside access token
  - Longer expiration (e.g., 1 week)
  - Used to generate new access token when expired
- **Security**:
  - Must be stored securely
  - Should be revocable

Best Practices

1. **Security**:
   - Use strong algorithms (RSA256, ES256 preferred over HS256)
   - Keep signing keys confidential and rotate regularly
   - Always use HTTPS
   - Validate tokens properly
2. **Token Management**:
   - Use short expiration times (15-60 minutes for access tokens)
   - Implement token revocation mechanisms
   - Monitor and log token usage
3. **Storage**:
   - Avoid localStorage/sessionStorage (XSS vulnerability)
   - Use secure, HTTP-only cookies when possible

secure, HTTP-only cookies

- **Secure Cookies:** uses HTTPS
  - Example: `Set-Cookie: sessionId=abc123; Secure`
- **HTTP-Only Cookies**
  - A cookie marked as HttpOnly cannot be accessed by client-side JavaScript.
  - Mitigates (XSS) attacks by preventing malicious scripts from stealing cookies (e.g., session hijacking).
  - Example: `Set-Cookie: sessionId=abc123; HttpOnly`
- Combine both flags for sensitive cookies (e.g., authentication tokens):
  - `Set-Cookie: sessionId=abc123; Secure; HttpOnly; SameSite=Lax`
  - `SameSite Attribute: Prevents cross-site request forgery (CSRF) by restricting when cookies are sent (e.g., SameSite=Lax/Strict).`

When NOT to Use JWT

- For sensitive data in payload (JWT is signed but not encrypted)
- Long-term sessions (use refresh token strategy instead)
- When complex token management is required
- When token size might exceed server header limits

Conclusion

JWTs provide a powerful, standardized way to handle authentication and information exchange in modern applications when implemented correctly with proper security measures.

The combination of access tokens and refresh tokens offers both security and good user experience.