LEARNING

1. Security

**JWT (JSON Web Token) -**

JSON Web Token (JWT) is a widely used method for securely exchanging information as a compact JSON object. It adheres to the **RFC 7519** standard and includes features that make it secure and trustworthy.

### **Key Characteristics of JWT:**

1. Compact:

JWTs are small and easy to transfer over HTTP headers, query strings, or other channels. They consist of three parts: Header, Payload, and Signature.

1. Self-Contained:

A JWT includes all the information needed to verify its integrity and claims.

1. Digitally Signed:

JWTs are signed to ensure the data hasn’t been tampered with.

Two main signing methods:

* HMAC (Hash-based Message Authentication Code): Uses a secret key shared between parties.
* RSA or ECDSA: Utilizes a public/private key pair where the private key signs the token, and the public key verifies it.

**JWT Structure :**

<Header>.<Payload>.<Signature>

**Signing and Encrypting with JSON Web Tokens (JWTs) :**

**1. Signed JWTs**

A signed JWT guarantees that the claims within the token have not been tampered with and are from a trusted source. However, the content of a signed JWT is not encrypted, so it is visible to anyone with access to the token.

Structure of a Signed JWT:

* Header - Specifies the algorithm used for signing (e.g., HS256 or RS256).
* Payload - Contains the claims (data) to be transmitted.
* Signature- A cryptographic signature generated by applying the signing algorithm to the encoded header and payload, using a secret or private key.

**2. Encrypted JWTs**

An encrypted JWT ensures that the claims are confidential and cannot be read by unauthorized parties. Encryption transforms the token's content into ciphertext that only intended recipients can decrypt.

Structure of an Encrypted JWT

An encrypted JWT (or JWE) has five parts:

* Header - Specifies the encryption algorithm (e.g., RSA-OAEP) and content encryption method (e.g., A256GCM).
* Encrypted Key - The encryption of a symmetric key using the public key of the recipient.
* Initialization Vector (IV) - A random value used for encryption to ensure uniqueness.
* Ciphertext - The encrypted version of the payload.
* Authentication Tag - Used to verify that the encrypted content has not been tampered with.

Encryption Process

1. Generate a symmetric key for encrypting the payload.
2. Encrypt the payload using the symmetric key and content encryption algorithm.
3. Encrypt the symmetric key using the recipient's public key.
4. Combine the encrypted key, ciphertext, IV, and authentication tag.

Decryption Process

1. Decrypt the symmetric key using the recipient's private key.
2. Use the symmetric key to decrypt the payload.

**OAUTH (Open Authorization) -**

OAuth (Open Authorization) is an open standard protocol for authorization. It allows third-party applications to access user resources hosted on a service provider (e.g., Google, Facebook) without sharing the user's credentials. OAuth focuses on authorization, not authentication, and facilitates secure, delegated access.

For example, a user can log in to a third-party app using their Google account without providing their Google password.

**Advantages of OAuth 2.0**

* Enhanced Security: Sensitive credentials (e.g., passwords) are not shared directly with third-party applications.
* Granular Permissions: Allows users to grant specific permissions to applications (e.g., access to contacts but not emails).
* Token-Based Access: Uses access tokens that can expire, improving security in case of breaches**.**
* Ease of Use: Users can log in to applications using existing accounts, reducing registration friction.
* Cross-Platform Compatibility: Works seamlessly with mobile apps, web apps, and desktop apps.
* Third-Party Integration: Facilitates integration with multiple service providers (e.g., social logins, API access).

**Grant Types in OAuth 2.0**

OAuth 2.0 provides various grant types to suit different scenarios and ensure secure access to resources.

**1. Authorization Code Grant**

* The Authorization Code grant type is used by confidential and public clients to exchange an authorization code for an access token.
* After the user returns to the client via the redirect URL, the application will get the authorization code from the URL and use it to request an access token.
* It is recommended that all clients use the [PKCE](https://oauth.net/2/pkce/) extension with this flow as well to provide better security.

### **2. Authorization Code with PKCE**

* PKCE ([RFC 7636](https://www.rfc-editor.org/rfc/rfc7636)) is an extension to the [Authorization Code flow](https://oauth.net/2/grant-types/authorization-code/) to prevent CSRF and authorization code injection attacks.
* PKCE was originally designed to protect the authorization code flow in mobile apps, but its ability to prevent authorization code injection makes it useful for every type of OAuth client, even web apps that use client authentication.

### **3. Client Credentials Grant**

* The Client Credentials grant is used when applications request an access token to access their own resources, not on behalf of a user.

### **4. Device Authorization Flow**

* The Device Code grant type is used by browserless or input-constrained devices in the [device flow](https://oauth.net/2/device-flow/) to exchange a previously obtained device code for an access token.

### **5. Refresh Token Grant**

* The Refresh Token grant type is used by clients to exchange a refresh token for an access token when the access token has expired.
* This allows clients to continue to have a valid access token without further interaction with the user.

**Different versions of OAuth**

#### **1. OAuth 1.0**

* Introduced: December 2007.
* Key Features: Cryptographic signing of requests using client secrets and access tokens.
* Drawbacks:

Complex implementation due to cryptographic requirements.

Required client secrets for every app, which was unsuitable for public clients like JavaScript apps.

#### **2. OAuth 1.0a**

* Introduced: April 2010.
* Improvements:
  + Introduced a verifier parameter to mitigate session fixation attacks.
* Drawbacks:
  + Retained the complexity of OAuth 1.0 with cryptographic signing.

#### **3. OAuth 2.0**

* Introduced: October 2012.
* Key Features:
  + Eliminated cryptographic signing, simplifying implementation.
  + Introduced multiple authorization flows (e.g., Authorization Code, Implicit, Resource Owner Password Credentials, and Client Credentials).
  + Tokens are bearer tokens (no additional cryptographic validation required).
* Drawbacks:
  + Increased reliance on HTTPS for security.
  + Vulnerable to token theft if not implemented securely.

#### **4. OpenID Connect (OIDC)**

* Introduced: February 2014 (built on OAuth 2.0).
* Key Features:
  + Adds authentication layer to OAuth 2.0.
  + Introduces ID tokens for user identity.
  + Standardized user info endpoint for fetching user details.
* Use Cases:
  + Applications requiring both authentication and authorization.

### **Drawbacks of OAuth 2.0 Compared to OAuth 1.0**

1. Security Trade-Off:
   * By eliminating cryptographic signing, OAuth 2.0 relies entirely on HTTPS, increasing the attack surface if HTTPS is misconfigured.
2. Bearer Token Vulnerability:
   * In OAuth 2.0, possession of a bearer token is enough to access the resource, making it critical to protect the token from theft.
3. Inconsistent Implementations:
   * OAuth 2.0's flexibility leads to varying implementations, increasing the risk of security issues if the protocol is not properly implemented.

**PYCRYPTODOME:**

PyCryptodome is a Python library that provides cryptographic functions and algorithms. It is a self-contained Python package offering a wide range of cryptographic operations, including encryption, decryption, hashing, and signature verification. PyCryptodome is a drop-in replacement for the old pycryptodome library and is designed to address many of its shortcomings, including security vulnerabilities and maintenance issues.

Key Features

* Symmetric Encryption: Provides algorithms for symmetric encryption, such as AES (Advanced Encryption Standard) and DES (Data Encryption Standard), allowing for secure data encryption and decryption using the same key.
* Asymmetric Encryption: Supports asymmetric encryption algorithms, such as RSA (Rivest–Shamir–Adleman), which use a pair of keys (public and private) for secure communication.
* Hash Functions: Includes cryptographic hash functions like SHA (Secure Hash Algorithm) and MD5 (Message Digest Algorithm 5) for generating fixed-size hash values from variable-size input data, useful for data integrity checks.
* Digital Signatures: Enables digital signing and verification using algorithms like RSA and DSA (Digital Signature Algorithm) to ensure the authenticity and integrity of messages or documents.
* Random Number Generation: Provides functionality for generating cryptographically secure random numbers, essential for creating secure keys and tokens.