**HRM CODE TUTORIAL COLLECTION**

**Collection By:** R&D Team

Security Documents

**OAuth 2.01**

Type chapter title (level 2)2

Type chapter title (level 3)3

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

# OAuth and OpenID Overview

# OAUTH 2.0

## Roles

OAuth định nghĩa có 4 Roles

### Resource Owner

Là một thực thể có khả năng nhận được Access tới các Protected Resource.

Trong trường hợp Resource Owner là một người, nó ám chỉ tới một End-user

### Resource Server

Là Server lưu trữ các Protected Resource, có khả năng chấp nhận và phản hồi là các Request có sử dụng Access Token

### Client

Là một Application tạo ra các Protected Resource Request đại diện cho Resource Owner với Authorization. Từ “Client” ở đây không có nghĩa ám chỉ cụ thể một cách triển khai cụ thể nào cả (e.g., có thể là một Application thực thi tại Server, hoặc một Desktop hay thiết bị gì đó khác)

### Authorization Server

Là Server sẽ Issue (có thể hiểu là phát hành) ra các Access Token cho Client sau khi chứng thực thành công Resource Owner và nhận Authorization

### Note

Sự tương tác giữa Authorization Server và Resource Server sẽ được tìm thấy tại các tài liệu khác.

Về cơ bản, Authorization Server có thể cũng là Resource Server mà cũng có thể là các thực thể riêng biệt.

Và một Authorization Server có thể Issue nhiều Access Token được chấp nhận bởi nhiểu Resource Servers.

## Protocol Flow

Về cơ bản, một Abstract OAuth 2.0 Flow sẽ trải qua những bước sau đây

1. Client yêu cầu Authorization từ Resource Owner. Authorization Request có thể được tạo trực tiếp tời Resource Owner, hoặc có thể gián tiếp thông qua Authorization Server như một thực thể trung gian
2. Client nhận một Authorization Grant, nó là một bộ thông tin cá nhân (credential) của Resource Owner người thực hiện Authorization, và Authorization Grant này là một trong 4 Grant Type sẽ được giới thiệu chi tiết ở phần sau. Authorization Grant Type này phụ thuộc và Method được sử dụng bởi Client để yêu cầu Authorization và Grant Type được Support bởi Authorization Server
3. Client yêu cầu một Access Token bằng cách xác thực với Authorization Server và đưa ra Authorization Grant Type
4. Authorization Server xác thực client và Validate Authorization Grant, và nếu nó là hợp lệ thì sẽ Issue một Access Token
5. Client gửi một Request cho Protected Resource từ Resource Server và xác thực thông qua Access Token
6. Resource Server Validates Acess Token, nếu hợp lệ thì sẽ thực thi Request

## Authorization Grant Overview

Một Authorization Grant là một bộ thông tin xác thực (credential ) đại diện cho Authorization của Resource Owner (dùng để Access Protected Resource) sử dụng bởi Client để nhận Access Token.

OAuth cung cấp 4 Grant Types – Authorization code, Implicit, Resource Owner Password Credentials và Client Credentials và một cơ chế mở rộng để định nghĩa các Types khác.

### Authorization Code

Authorization Code có thể nhận được thông qua Authorization Server như một trung gian giữa Client và Resource Owner. Thay vì Resource Owner trực tiếp gửi Authorization Request, Client sẽ điều hướng Resource Owner tới một Authorization Server (thông qua User-agent - <https://tools.ietf.org/html/rfc2616>) và sau đó điều hướng Resource Owner ngược trở lại Client với Authorization Code.

Trước khi điều hướng Resource Owner trở lại Cliient với Authorization Code, Authorization thực hiện xác thực Resource Owner và nhận Authorization. Bởi vì Resource Owner chỉ thực hiện chứng thực tại Authorization Server, thông tin chứng thực của Resource Owner không bao giờ được chia sẻ vơi Client.

Authorization Code cung cấp một vài lợi ích quan trọng trong quá trình bảo mật, như là khả năng để chứng thực Client, cũng như khả năng truyển đổi trực tiếp thành Access Token cho Client mà không cần thông qua User-agent của Resource Owner và bị phơi bày kể cả với Resource Owner

### Implicit

### Resource Owner Password Credentials

### Client Credentials

## Client Registration

## Protocol Endpoints

Quá trình Authorization tận dụng hai Authorization Server Endpoints

* Authorization Endpoint – Được sử dụng bởi Client để nhận Authorization từ Resource Owner thông qua User-agent Redirection
* Token endpoint – được sử dụng bởi Client để trao đổi một Authorization Grant lấy một Access Token, điển hình là với Client Authentication

Và còn một Client Endpoint với

* Redirection Endpoint – được sử dụng bởi Authorization Server để trả về Response chứa Authorization Credentials cho Client thông qua User-agent của Resource Owner

Tuy nhiên không phải mọi Authorization Grant đều sử dụng hai Endpoints, và Extension Grant Type CŨNG CÓ THỂ định nghĩa thêm các Endpoint nếu cảm thấy cần thiết

### Authorization Endpoint

Authorization Endpoint được sử dụng để tương tác với Resource Owner và nhận một Authorization Grant. Authorization Server trước tiên PHẢI xác thực Identity của Resource Owner

#### Response Type

#### Redirection Endpoint

### Token Endpoint

#### Client Authentication

### Access Token Scope

Authorization và Token Endpoints cho phép Cliient chỉ định Scope của Access Request sử dụng “scope” Request Parameter. Tới lượt mình, Authorization Server sẽ sử dụng “scope” Response Parameter để thông báo cho Client về Scope mà Access Token được Issued.

Giá trị của Scope Parameter được thể hiện như là một danh sách Space-delimited (phân định bằng khoảng trắng), Case-sensitive String. Strings được định nghĩa bởi Authorization Server. Nếu giá trị chứa nhiều Space-delimited Strings thì thứ tự của chúng không quan trọng, và mỗi String lại thêm một Access Range vào Scope của Request

* scope = scope-token \* (SP scope-token)
* scope-token = 1\*(%x21 / %x23-5B / %x5D-7E)

Authorization Server CÓ THỂ (MAY) bỏ qua toàn bộ hoặc một phần Scope được yêu cầu bởi Client dựa trên Authorization Server Policy hoặc Resource Owner’s Instruction. Nếu Scope của Access Token được Issued khác với Scope mà Client yêu cầu, Authorization Server PHẢI (MUST) Include “scope” Response Parameter để thông báo cho Client về Scope chính xác đã được cấp.

Nếu Client bỏ qua Scope Parameter khi yêu cầu Authorization, Authorization Server PHẢI (MUST) hoặc là xử lý Request sử dụng một Pre-defined Value hoặc trả về kết quả thất bại với lý do Invalid Scope. Authorization Server NÊN (SHOULD) Document Scope Requirements và Default Value

## Grant Types

### What is an OAuth 2.0 Grant Type?

Trong OAuth 2.0, thuật ngữ “Grant Type” chỉ ra cách mà Applications của chúng ta lấy một Access Token. OAuth 2.0 định nghĩa một vài Grant Types, trong đó bao gồm Authorization Code Flow. OAuth 2.0 Extensions cũng có thể định nghĩa các Grant Types mới.

Mỗi Grant Type được tối ưu cho các Use Case cụ thể, có thể là cho một Web App, ,một Native App, một thiết bị không có khả năng chạy các Browser, hoặc một vài Server-to-server Applications.

Authorization Code Grant Type là Grant Type phổ biến nhất của OAuth 2.0 mà chúng ta có thể sẽ gặp. Nó được sử dụng cho cả Web Apps và Native Apps để lấy Access Token sau khi User cho phép ứng dụng.

### The Authorization Code Flow

Authorization Code Grant Type được sử dụng bởi Web và Mobile Apps. Nó khác với đa phần các Grant Type khác khi điều đầu tiên chúng yêu cầu là chạy một Browser để bắt đầu. Tại High Level, Flow có một vài bước như sau:

Application mở một Browser để đưa User tới OAuth Server

User sẽ thấy Authorization Prompt và chấp nhận Request của Application

User được Redirect trở lại Application với Authorization code trong Query String

Application trao đổi Authorization Code để trả lại một Access Token

#### Get the User’s Permission

Tất cả những gì OAuth làm là cung cấp cho User một Limited Access tới Applications. Điều đầu tiên Application cần là quyết định là những yêu cầu gì cần sự cho phép của User, sau đó đưa User tới một Browser và nhận Permission từ User.

Để bắt đầu Authorization Flow, Application sẽ xây dựng lên một URL như phía dưới và mở một Browser với URL này

<https://xauthorization-server.com/auth>

?response\_type=code

&client\_id=299354654564641231

&redirect\_uri=https%3A%2F%2Fexample-app.com%2Fcallback

&scope=create+delete

&state=n2lb13o32h32vvzap1321

##### Query Parameters

* response\_type=code – nói cho Authorization Server là chúng ta sẽ sử dụng Authorization Code Flow
* client\_in – Public Identifer của Application
* redirect\_uri – Nói cho Authorization Server nơi mà nó có thể đưa User trở lại sau khi đã Approve Request (ví dụ Client có địa chỉ là <https://localhost:44303> thì redirect\_uri có thể là <https://localhost:44303/signin-oidc>)
* scope – Một hoặc nhiều String cách nhau bởi khoảng trắng chỉ ra Permission của Application đang Request.
* state – Application sẽ tạo ra một chuổi bất kỳ và thêm nó vào Request. Nó nên được kiểm tra để kết quả trả về là tương tự sau khi User Authorize Application. Được sử dụng để chặn CSRF Attacks

Sau khi User tới địa chỉ trên, Authorization Server sẽ tạo ra một Prompt để hỏi xem User có muốn Authorize những Request của Application hay không

#### Redirect Back to the Application

Nếu User Approve Request, Authorization Server sẽ Redirect Browser trở lại redirect\_uri được chỉ định trong Application và kèm theo code và state

Ví dụ:

<https://example-app.com/redirect>

?code=fsdFDS4FDS421FsfDFdw43Gffd

&state=n2lb13o32h32vvzap1321

state sẽ có giá trị giống như như Application ban đầu set nó trong Request. Application sẽ cần phải kiểm tra trạng thái trong Redirect khớp với lúc ban đầu để chống lại CSRF và các kiểu tấn công tương tự

code ở đây là Authorization Code được tạo ra bởi Authorization Server. Code này sẽ có Lifetime ngắn, thông thường kéo dài khoảng 1 tới 10 phút tùy vào OAuth Service

#### Exchange the Authorization Code for an Access Token

Sau khi qua các bước ở trên, Application của chúng ta hiện đã có một Authorization Code, và giờ chúng ta có thể sử dụng nó để đổi lấy một Access Token.

Application sẽ tạo ra một POST Request tới Token Endpoint tại Authorization Server với những Parameters như sau

* grant\_type=authorization\_code – có tác dụng báo cho Token Endpoint rằng Application đang sử dụng Authorization Code Grant Type
* code – Application sẽ thêm Authorization Code mới nhận được vào Redirect
* redirect\_uri – Tương tự URI sử dụng khi Request Authorization Code. Một vài APIs không yêu cầu parameter này, nên chúng ta đôi khi cần kiểm tra lại Docs của các API cụ thể
* client\_id – Application’s Client Id
* client\_secret – Application’s Client Secret. Secret có tác dụng đảm bảo Request được sử dụng để lấy Access Token được tạo ra bởi duy nhất một Application, và không tạo ra điều kiện để Attacker can thiệp vào Authorization Code

Token Endpoint sẽ xác thực tất cả Parameters trong Request, kiểm tra Code hết hạn hay chưa hay ClientId và ClientSecret có khớp với nhau hay không. Nếu xác thực thành công, Token Endpoint sẽ tạo ra một Access Token và trả về trong Response

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3",

"token\_type":"bearer",

"expires\_in":3600,

"refresh\_token":"IwOGYzYTlmM2YxOTQ5MGE3YmNmMDFkNTVk",

"scope":"create delete"

}

Và Authorization Code Flow đã hoàn tất, bây giờ Application có Access Token và có thể sử dụng chúng để tạo các API Request.

#### When to use the Authorization Code Flow

Authorization Code Flow có hiệu quả tốt nhất khi sử dụng trên Web và Mobile Apps. Bởi vì Authorization Code Grant có thêm bước trao đổi Authorization Code để lấy Access Token, từ đó cung cấp thêm một lớp bảo mật mà trong Implicit Grant Type không có.

Nếu chúng ta sử dụng Authorization Code Flow trong Mobile App, hoặc bất kỳ một Application nào không có khả năng lưu trữ một Client Secret, chúng ta nên sử dụng PKCE extension (Proof Key for Code Exchange), điều này cung cấp khả năng bảo về Application khỏi bị tấn công bởi vì Authorization Code có thể bị can thiệp.

Bước tra đổi Code đảm bảo rằng Attacker sẽ không thể can thiệp vào Access Token, bởi vì Access Token luôn luôn được gửi đi thông qua một Secure Backchannel giữa Application và OAuth Server

### Authorization Code Flow with Proof Key for Code Exchange (PKCE)

#### Problems

Khi chúng ta có một Public Client (e.g., Native và Single-page Applications), chúng ta sẽ gặp một vài vấn đề với Authorization Code Flow liên quan tới việc lưu trữ Client Secret như ở phần trên có đề cập, và cụ thể hơn là

##### Native Apps

* Không thể lưu trữ Client Secret. Decompiling Appliction sẽ Reveal Client Secret. Secret này sẽ được đính kèm với Application và luôn luôn giống nhau với mọi Users và Devices
* Có thể sử dụng một Custom URL Scheme để Capture Redirects (e.g., MyApp://) và dễ có thể bị Malicious Applications sử dụng để nhận một Authorization Code từ Authorization Server

##### Single-page Apps

* Không thể lưu trữ một Client Secret vì toàn bộ Source có thể tìm thấy trên Browser

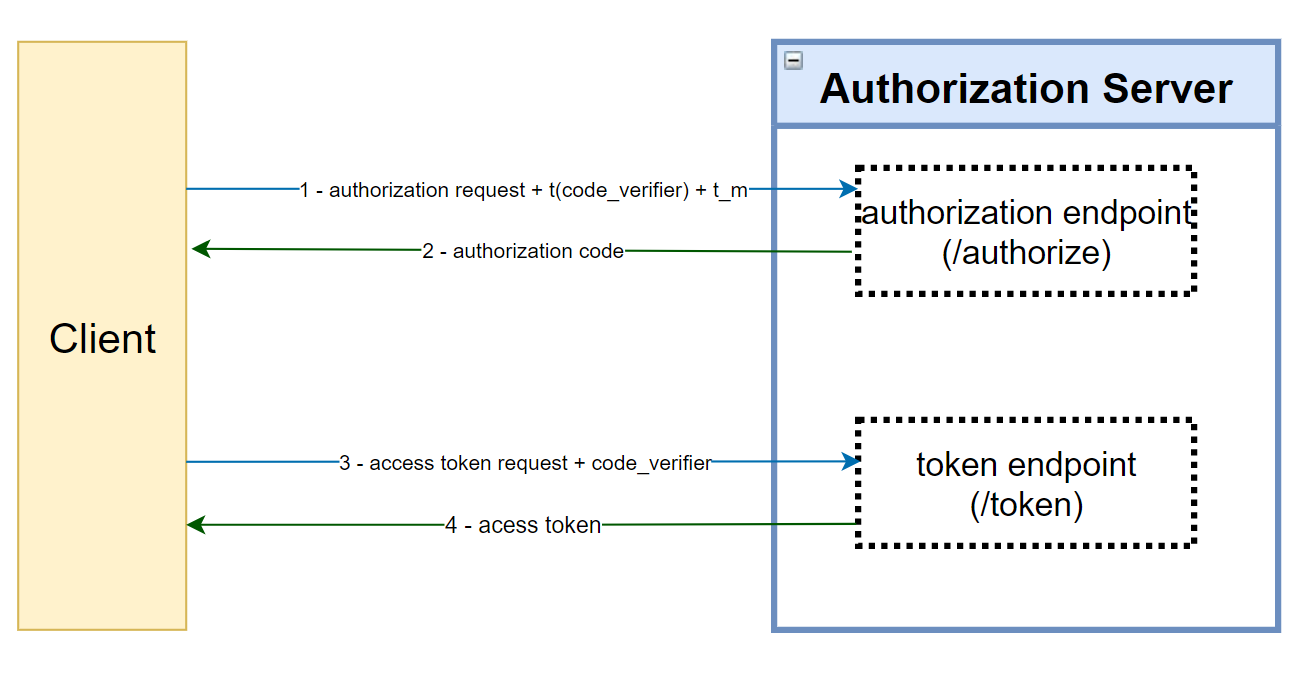
#### Solution

Và với tất cả những vấn đề trên, OAuth 2.0 cung cấp một Version của Authorization Code Flow trong đó sử dụng PKCE.

PKCE-enhanced Authorization Code Flow giới thiệu thêm một Secret được tạo bởi Application, và Secret này có thể được Verify bởi Authorization Server, tên gọi của Secret này là Code Verifier. Và thêm nữa, Application từ Code Verifier sẽ tạo thêm một thứ gọi là Code Challenge và gửi Code Challenge này thông qua HTTPS để nhận Authorization Code.

Với phương pháp này, một Malicious Attacker chỉ có thể can thiệp vào Authorization Code, tuy nhiên sẽ không thể sử dụng Code này để đổi lấy một Access Token mà không có Code Verifier.

##### Protocol Flow



###### Client creates a Code Verifier

Client đầu tiên cần tạo ra một Code Verifier “code\_verifier” cho mỗi OAuth 2.0 Authorization Request  
code\_verifier = high-entropy cryptographic random STRING sử dụng Unreserved Characters (Unreserved Characters có thể hiểu đơn giản là một chuỗi ký tự không có bất kỳ ý nghĩa gì) [A-Z] / [a-z] / [0-9] / - / . / \_ / ~ với độ dài tối thiểu là 43 ký tự, tối đa là 128 ký tự

ABNF (Augmented Backus-Naur Form) cho code\_verifier

* code-verifier = 43\*128unreveresd
* unreversed = ALPHA / DIGIT / - / . / \_ / ~
* ALPHA = %x41-5A / %x61-7A
* DIGIT = %x30-39

%x41-5A (mã Hex trong ASCII – A-Z)

%x61-7A (mã Hex trong ASCII – a-z)

%x30-39 (mã Hex trong ASCII – 0-9)

Note

Code verifier NÊN có đủ entropy (sự xáo trộn) để làm nó không thể đoán ra một cách đơn giản, và nó được RECOMMENDED nên là kết quả của một dãy số Random được tạo bởi một 32-octet Sequence. Octet Sequence được base64url-encoded sử dụng để tạo ra một 43-octet URL Safe String để sử dụng như là một code\_verifier

###### Client Creates the Code Challenge

Client sau khi tạo code\_verifier sẽ từ đó tạo Code Challenge “code\_challege” sử dụng một trong những cách Transform code\_verifier sau

* plain – code\_challenge = code\_verifier
* S256 – code\_challenge = BASE64URL-ENCODE(SHA256(ASCII(code\_verifier)))

Nếu Client có thể sử dụng S256, nó PHẢI sử dụng S256, bởi vì S256 là MANDATORY TO IMPLEMENT (bắt buộc phải triển khai) tại Server. Client chỉ trong trường hợp không thể sử dụng S256 vì một vài lý do kỹ thuật gì đó và phải được cấu hình ở Server hỗ trợ “plain”

Plan Transformation được sử dụng để tương thích với Existing Deployments và cho các môi trường rằng buộc không thể sử dụng S256

ABNF cho code\_challenge

* code-challenge = 43\*128unreserved
* unreversed = ALPHA / DIGIT / - / . / \_ / ~
* ALPHA = %x41-5A / %x61-7A
* DIGIT = %x30-39

###### Client Sends the Code Challenge with the Authorization Request

Client sẽ gửi code\_challenge như một phần của OAuth 2.0 Authorization Request sử dụng các Parameters sau

* code\_challenge – REQUIRED
* code\_challenge\_method – OPTIONAL, mặc định là “plain” nếu không gửi kèm Request. Code Verifier Transformation Method là “S256” hoặc “plain”

###### Server Returns the Code

Sau khi Server Issues Authorization Code tại Authorization Response, nó PHẢI được liên kết với code\_challenge và code\_challenge\_method values với Authorization Code để có thể Verify sau này

Thông thường, code\_challenge và code\_challenge\_method Values được lưu trữ trong một Encrypted Form trong chính “code” nhưng có một lựa chọn thay thế là lưu trữ tại Server liên kết với “code”. Server KHÔNG ĐƯỢC Include code\_challenge Value tại Client Requests trong một Form mà nó có thể bị các thực thể khác khai thác.

Method mà chính xác được sử dụng bởi Server để liên kết code\_challenge với Issued “code” không được mô tả tại Document này.

###### Error Response

* Nếu Server Requires PKCE được tạo bởi OAuth Public Clients và Client không gửi code\_challenge trong Request, Authorization Endpoint PHẢI trả về Authorization Error Response với “error” Value được Set là “invalid\_request”. Và “error\_description” hoặc “error\_uri” NÊN giải thích nguyên nhân gây ra lỗi một cách thích hợp, ví dụ như “code challenge required”
* Nếu Server hỗ trợ PKCE nhưng không hỗ trợ Requested Transformation, Authorization Endpoint PHẢI trả về Authorization Error với “error” Value được Set là “invalid\_request”. “error\_description” hoặc “error\_uri” nên là “transform algorithm not supported”

###### Client Sends the Authorization Code and the Code Verifier to the Token Endpoint

Sau khi nhận được Authorization Code, Client gửi Access Token Request tới Token Endpoint với các Parameters được định nghĩa tại OAuth 2.0 Access Token Request (<https://tools.ietf.org/html/rfc6749#section-4.1.3>)

Parameters

* code\_verifier – REQUIRED

code\_challenge\_method được gắn với Authorization Code khi Authorization Code được Issued. Đấy là Method Token Endpoint PHẢI sử dụng để Verify code\_verifier

###### Server Verifies code\_verifier trước khi trả về Tokens

Ngay khi nhận được Request tại Token Endpoint, Server sẽ Verifies nó bằng cách tính toán Code Challenge từ code\_verifier và so sánh nó với code\_challenge liên kết trước đấy, sau khi lần đầu Transform nó dựa trên code\_challenge\_method Method được chỉ định bởi Clients

Nếu code\_challenge\_method là S256 nghĩa là code\_verifier vừa nhận được được Hashed bởi SHA-256, base64url-encoded và được so sánh với code\_challege

Nếu code\_challenge\_method là “plain” thì nó sẽ được so sánh trực tiếp code\_verifier == code\_challenge

Nếu kết quả so sánh là giống nhau, Token Endpoint PHẢI tiếp tục tiến trình một cách bình thường. Nếu kết quả của phép so sánh là không giống nhau, một Error Response PHẢI được trả về nói lên việc “invalid\_grant”

#### How it works

Bởi vì PKCE-enhanced Authorization Code Flow được xây dựng dựa trên Standard Authorization Code Flow, nên quá trình xác thực khá là tương tự

1. User Clicks vào Login trong Client Application
2. Auth0’s SDK tạo ra một code\_verifier (cryptographically-random) và từ code\_verifier tiếp tục tạo ra một code\_challenge
3. Auth0’s SDK Redirects User tới Auth0 Authorization Server (/authorize endpoint, với identityserver4 /connect/authorize) cùng với code\_challenge
4. Auth0 Authorization Redirects User tới nới Login và Authorization Prompt
5. User sẽ thực hiện Authenticate thông qua một trong những Login Options, và User cũng có thể thấy một Consent Page liệt kể Permission mà Auth0 sẽ cung cấp cho Application
6. Auth0 Authorization Server sẽ lưu trữ code\_challenge và Redirects User trở lại Application với một Authorization Code (sử dụng một lần)
7. Auth0’s SDK gửi Authorization Code (code) và code\_verifier (được tạo ở bước 2) tới Auth0 Authorization Server ( /oauth/token endpoint, với identityserver4 /connect/token)
8. Auth0 Authorization Server thực hiện Verify code\_challenge và code\_verifier
9. Auth0 Authorization Server Response với một ID Token và Access Token (và có thể có thêm Refresh Token)
10. Application sử dụng Access Token để Call API để lấy Resource
11. API Response Data được Request

##### Note

Nếu chúng ta có Refresh Token Rotation được Enable, một RefreshToken mới sẽ được tạo ra với mỗi Request và được Issue cùng với Access Token. Khi một Refresh Token được trao đổi, Refresh Token trước đó sẽ không còn hợp lệ nhưng các thông tin về mối liên hệ này sẽ được giữ lại bởi Authorization Server

## Issuing an Access Token

Nếu một Acess Token Request là hợp lệ và đã được Authorized, Authorization Server sẽ Issue một Access Token và một Refresh Token (optional). Nếu Request thất bại hoặc không hợp lệ, Authorization Server sẽ trả về một Error Response.

### Successful Response

Authorization Server sẽ Issue một Access Token (+ Refresh Token optional), và xây dựng Response bằng cách thêm các Parameters vào entity-body của HTTP Response với Status Code 200 (Ok).

* access\_token: REQUIRED – Access Token được Issued bởi Authorization Server
* token\_type: REQUIRED – Type của Token được Issued. Value là Case Insensitive
* expires\_in: RECOMMENDED – Vòng đời của Access Token được tính theo giây. Ví dụ với giá trị “3600” chỉ ra Access Token sẽ hết hạn sau 1 giờ từ lúc Response được tạo. Nếu bỏ qua việc set expires\_in, Authorization Servvfer NÊN (SHOULD) cung cấp Expiration Time thông qua các cách khác hoặc tạo ra một Default Value
* refresh\_token: OPTIONAL – Refresh Token, có thế sử dụng để nhận Access Token mới sử dụng cùng Authorization Grant
* scope: OPTIONAL – nếu tương tự như Scope được Request bởi Client. Ngược lại – REQUIRED

Những Parameters này được Included trong entity-body của HTTP Response sử dụng “application/json” Media Type. Parameters được Serialized thành một Javascript Object Notation (JSON) Structure bằng cách thêm những Parameter này tại Structure Level cao nhất. Parameter names và String Values được Included như JSON Strings. Giá trị số được Included như là JSON Numbers. Thứ tự của các Parameters không quan trọng.

Authorization Server PHẢI (MUST) Include HTTP “Cache-Control” Response Header Field với giá trị là “no-store” tại bất kỳ Response nào mang theo Tokens, Credentials, hoặc bất kỳ Sensitive Information nào và cùng với “Pragma” Response Header Field có giá trị là “no-cache”

Ví dụ

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"2YotnFZFEjr1zCsicMWpAA",

"token\_type":"example",

"expires\_in":3600,

"refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",

"example\_parameter":"example\_value"

}

Client PHẢI (MUST) bỏ qua các Value Name không nhận diện được tại Response. Kích thước của Tokens và các giá trị khác nhận được từ Authorization Server được để là undefined. Client nên tránh việc phỏng đoán về Value Sizes. Authorization Server NÊN (SHOULD) Document kích thước của bất kỳ giá trị nào nó Issue

# JSON Web Tokens (JWT)

## What is JWT

JWT là một Open Standard (tiêu chuẩn mở rộng) dùng để định một hướng đi đơn giản và khép kín để truyền thông tin một cách an toàn giữa các Parties như một JSON Object. JWTs có thể được Signed bằng cách sử dụng Secret (với thuật toán HMAC) hoặc với một cặp Public/Private Key sử dụng RSA hoặc ECDSA.

Mặc dù JWTs có thể được mã hóa để cung cấp sự bảo mật giữa các Parties, chúng ta sẽ tập trung vào signedtokens. Signed Tokens có thể xác thực các Claims được tích hợp trong nó, trong khi Encrypted Tokens sẽ ẩn đi Claims từ các Parties khác. Khi Tokens được Signed bằng cách sử dụng cặp Public/Private Key, Signature cũng sẽ chứng nhận rằng chỉ có một Party duy nhất được giữ Private Key là Party đã thực hiện Signed nó.

## When should you use JWTs

Trong một vài ngữ cảnh việc sử dụng JWTs sẽ thực sự có ích

### Authorization

Đây là ngữ cảnh phổ biến nhất chúng ta sử dụng JWT. Một khi User đã được Loged In, mỗi Subsequent Request (các request thực hiện sau đó) sẽ Include JWT, điều này cho phép User truy cập vào các Routes, Services và Resources mà nó cho phép sử dụng Token này. Single Sign On là một tính năng phổ biến sử dụng JWT ngày nay, bởi vì chi phí nhỏ và khả năng dễ dàng sử dụng Across Different Domains (sử dụng được qua các Domains khác nhau).

### Information Exchange

JWTs là một cách rất tốt để truyền dữ liệu một cách an toàn qua các Parties. Bởi vì JWTs có thể được Signed – Ví dụ, sử dụng một cặp Public/Private Key, chúng ta có thể chắc chắn người gửi chắc chắn là họ. Thêm nữa, bởi vì Signature được tính toán sử dụng Header và Payload, chúng ta có thể xác thực rằng nội dung bên trong không phải là giả mạo

### What is the JWT Structure

#### Header

#### Payload

##### Registered Claims

##### Pubic Claims

##### Private Claims

#### Signature

#### Putting all together

### How do JWTs work

### Why should we use JWTs