**HRM CODE TUTORIAL COLLECTION**

**Collection By:** R&D Team

Contents

[OAuth and OpenID Overview 4](#_Toc62652169)

[OpenID 4](#_Toc62652170)

[OAuth 2.0 5](#_Toc62652171)

[1. Introduction 5](#_Toc62652172)

[1.1. Roles 5](#_Toc62652173)

[1.2. Protocol Flow 5](#_Toc62652174)

[1.3. Authorization Grant Overview 6](#_Toc62652175)

[2. Client Registration 6](#_Toc62652176)

[3. Protocol Endpoints 6](#_Toc62652177)

[Authorization Endpoint 7](#_Toc62652178)

[Token Endpoint 7](#_Toc62652179)

[Access Token Scope 7](#_Toc62652180)

[4. Obtaining Authorization 7](#_Toc62652181)

[4.1. Authorization Code Grant 8](#_Toc62652182)

[4.2. Implicit Grant 8](#_Toc62652183)

[4.3. Resource Owner Password Credentials Grant 8](#_Toc62652184)

[4.4. Client Credential Grant 8](#_Toc62652185)

[5. Issuing an Access Token 8](#_Toc62652186)

[Successful Response 8](#_Toc62652187)

[Error Response 9](#_Toc62652188)

[6. Refreshing an Access Token 9](#_Toc62652189)

[7. Accessing Protected Resource 9](#_Toc62652190)

[7.1. Access Token Types 9](#_Toc62652191)

[Grant Types 11](#_Toc62652192)

[What is an OAuth 2.0 Grant Type? 11](#_Toc62652193)

[The Authorization Code Flow 11](#_Toc62652194)

[Authorization Code Flow with Proof Key for Code Exchange (PKCE) 14](#_Toc62652195)

[OpenID Connect Core 19](#_Toc62652196)

[1. Introduction 19](#_Toc62652197)

[2. ID Token 20](#_Toc62652198)

[3. Authentication 21](#_Toc62652199)

[3.1. Authentication using the Authorization Code Flow 21](#_Toc62652200)

[3.2. Authentication using the Implicit Flow 22](#_Toc62652201)

[3.3. Authentication using the Hybrid Flow 22](#_Toc62652202)

[4. Initiating Login from a Third Party 23](#_Toc62652203)

[5. Claims 23](#_Toc62652204)

[6. Passing Request Parameters as JWTs 23](#_Toc62652205)

[7. Self-Issued OpenID Provider 23](#_Toc62652206)

[8. Subject Identifier Types 23](#_Toc62652207)

[9. Client Authentication 23](#_Toc62652208)

[10. Signatures and Encryption 23](#_Toc62652209)

[11. Offline Access 23](#_Toc62652210)

[12. Using Refresh Tokens 23](#_Toc62652211)

[13. Serializations 23](#_Toc62652212)

[14. String Operation 23](#_Toc62652213)

[15. Implemention Considerations 23](#_Toc62652214)

[16. Security Considerations 23](#_Toc62652215)

[17. Privacy Considerations 23](#_Toc62652216)

[18. IANA Considerations 23](#_Toc62652217)

[19. References 23](#_Toc62652218)

[OpenID Connect Single Logout (SLO) 23](#_Toc62652219)

[Overview 23](#_Toc62652220)

[OpenID Connect Logout 25](#_Toc62652221)

[OpenID Connect Back-Channel Logout 1.0 26](#_Toc62652222)

[OpenID Connect Front-Channel Logout 1.0 31](#_Toc62652223)

[OpenID Connect Session Management 1.0 32](#_Toc62652224)

[OAuth 2.0 and OIDC Support 33](#_Toc62652225)

[OAuth and OpenID Connect Concepts 33](#_Toc62652226)

[OAuth 2.0 Endpoints 33](#_Toc62652227)

[OAuth 2.0 and OIDC Workflows 33](#_Toc62652228)

[OAuth State Management 33](#_Toc62652229)

[Trusted Clients Management 33](#_Toc62652230)

[Proof Key for Code Exchange Support 33](#_Toc62652231)

[Reverse Proxy Configuration for OAuth and OIDC Provider 33](#_Toc62652232)

[Configuring OAuth 2.0 API Protection 33](#_Toc62652233)

[OIDC Claims Customization 33](#_Toc62652234)

[… More 33](#_Toc62652235)

[Bearer Token Usage 34](#_Toc62652236)

[Introduction 34](#_Toc62652237)

[Authenticated Requests 34](#_Toc62652238)

[Authorization Code Header Field 34](#_Toc62652239)

[Form-Encoded Body Parameter 34](#_Toc62652240)

[URI Query Parameter 34](#_Toc62652241)

[JSON Web Tokens (JWT) 35](#_Toc62652242)

[What is JWT 35](#_Toc62652243)

[When should you use JWTs 35](#_Toc62652244)

[Authorization 35](#_Toc62652245)

[Information Exchange 35](#_Toc62652246)

[What is the JWT Structure 35](#_Toc62652247)

[How do JWTs work 36](#_Toc62652248)

[Why should we use JWTs 36](#_Toc62652249)

[OpendID Connect Client Initiated Backchannel Authentication Flow - Draft 37](#_Toc62652250)

[Introduction 37](#_Toc62652251)

[OpendID Connect Note 38](#_Toc62652252)

[Authentication with OpenID Connect Relying Party 38](#_Toc62652253)

[Note 38](#_Toc62652254)

[Note 39](#_Toc62652255)

# OAuth and OpenID Overview

## OpenID

# OAuth 2.0

## Introduction

### 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

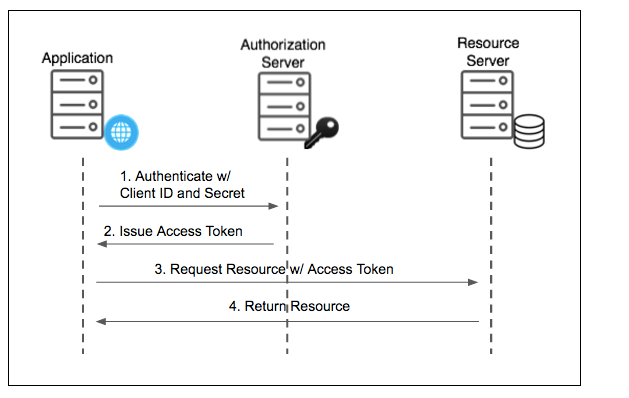
Về cơ bản, một Abstract OAuth 2.0 Flow sẽ trải qua những bước sau đây

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.



Nói rõ hơn thì Grant Type này được sử dụng cho những Applications mà có thể gửi Request một Access Token và truy cập tới Resource bởi chính nó. Những Applications đó thường sử dụng những Services có thể gọi API mà không cần Users

#### 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 Credentials (hoặc cái Forms khác của Client Authentication) có thể được sử dụng như một Authorization Grant khi Authorization Scope tới Protected Resource bị giới hạn dưới sự quản lý của Client, hoặc Protected Resources đã được sắp xếp trước đó với Authorization Server. Client Credentials được sử dụng như một Authorization Grant phổ biến khi Client hoạt động đại diện cho chính nó (Client cũng là Resource Owner) hoặc khi yêu cầu truy cập tới Protected Resources dựa trên một Authorization trước đây đã được sắp xếp với Authorization Server

## 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

## Obtaining Authorization

### Authorization Code Grant

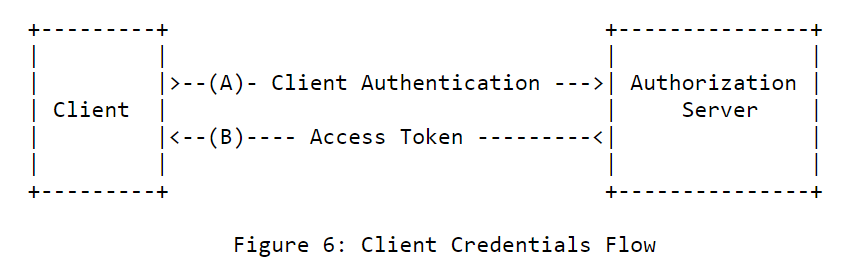
### Implicit Grant

### Resource Owner Password Credentials Grant

### Client Credential Grant

Client có thể request một Access Token sử dụng duy nhất một Client Credentials (của chính Client đó) khi Client request quyền truy cập tới Protected Resource dưới sự quản lý của nó hoặc những Resource Owner khác trước đây được sắp xếp với Authorization Server.

Client Credentials Grant Type PHẢI (MUST) duy nhất được sử dụng bởi các Confidential Clients



1. Client chứng thực với Authorization Server và requests một Access Token từ Token Endpoint
2. Authorization Server chugns thực Client, và nếu hợp lệ thì sẽ issue một Access Token

#### Authorization Request and Response

Bởi vì Client Authentication được sử dụng như Authorization Grant nên việc thêm các Authorization Request là không cần thiết.

#### Access Token Request

## 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

### Error Response

## Refreshing an Access Token

## Accessing Protected Resource

Client access tới Protected Resource thông qua việc đưa ra Access Token cho Resource Server. Resource Server PHẢI (MUST) xác thực Access token và đảm bảo rằng nó chưa hết hạn và nó có một Scope cụ thể. Cách thức mà Resource Server xác thực Access Token (cũng như bất kỳ mọi Error Response) có thể xem tại …, và tổng quát thì quá trình này liên quan tới sự tương tác và kết hợp giữa Resource Server và Authorization Server.

Phương pháp mà Client sử dụng để dùng Access Token xác thực với Resource Server phụ thuộc vào kiểu của Access Token Issued bởi Authorization Server. Và đặc trưng của quá trình này là sử dụng HTTP “Authorization” Request Header Field với một Authentication Scheme được định nghĩa tại Access Token Type được sử dụng.

### Access Token Types

Access Token Type cung cấp cho Client thông tin được yêu cầu để có thể sử dụng Access Token để tạo một Protected Resource Request (cùng với các type-specific Attribtues). Client KHÔNG ĐƯỢC (MUST NOT) sử dụng Token nếu nó không thể hiểu Token Type.

Ví dụ, với “bearer” Token Type được sử dụng đơn giản bằng cách thêm Access Token String vào Request

GET /resource/1 HTTP/1.1

Host: example.com

Authorization: Bearer mF\_9.B5f-4.1JqM

Trong khi “mac” Token Type được sử dụng bằng cách Issue một Message Authentication Code (MAC) Key cùng với Access Token được sử dụng để đăng ký một Component cụ thể của HTTP Request

GET /resource/1 HTTP/1.1

Host: example.com

Authorization: MAC id="h480djs93hd8",

nonce="274312:dj83hs9s",

mac="kDZvddkndxvhGRXZhvuDjEWhGeE="

Ví dụ ở trên chỉ có mục đích phác họa. Developer nên tìm hiểu Bearer Token Usage và OAuth-HTTP-MAC trước khi sử dụng

Mỗi Access Token Type định nghĩa cụ thể những Attribute (nếu có) gửi tới Client cùng với “access\_token” Response Parameter. Nó cũng định nghĩa HTTP Authentication method sử dụng để thêm Access Token khi tạo một Protected Resource Request.

## 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

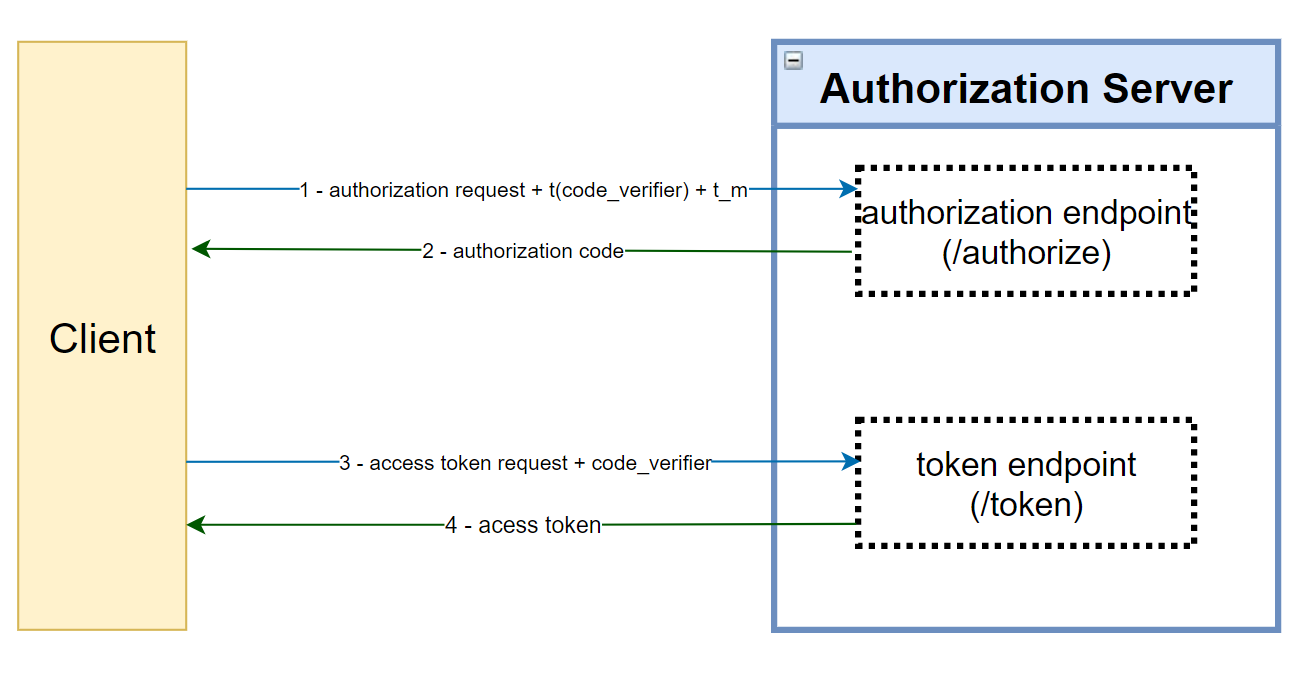
#### Solution

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

# OpenID Connect Core

## Introduction

## ID Token

## Authentication

### Authentication using the Authorization Code Flow

Session này mô tả việc triển khai Authorization Code Flow. Khi sử dụng Flow này, tất cả Token sẽ được trả về từ Token Endpoint.

Authorization Code Flow trả về một Authorization Code cho Client, thứ mà sau này có thể đổi lấy một ID Token và một Access Token một cách trực tiếp. Việc này cung cấp khả năng không cần expose bất kỳ Tokens nào cho User Agent và các Malicious Applications có thể Access tới User Agent. Authorization Server cũng có thể chứng thực Client trước khi thực hiện việc trao đổi Authorization Code cho một Access Token. Authorization Code Flow thích hợp cho các Clients có khả năng lưu trữ Client Secret giữa Client và Authorization Server một cách an toàn.

#### Authorization Code Flow Steps

1. Client chuẩn bị một Authentication Request chứa những Parameters cần thiết
2. Client gửi Request tới Authorization Server
3. Authorization Server chứng thực End-user
4. Authorization Server nhận End-user Consent/Authorization
5. Authorization Server đưa End-user trở lại Client với một Authorization Code
6. Client request một Response sử dụng Authorization Code tại Token Endpoint
7. Client nhận một Response chứa một ID Token và Access Token trong Response Body
8. Client validates ID Token và nhận Subject Identifier của End-user

#### Authorization Endpoint

##### Authentication Request

Một Authentication Request là một OAuth 2.0 Authorization Request và với Request đó End-user có thể được chứng thực bởi Authorization Server.

Authorization Server PHẢI (MUST) hỗ trợ việc sử dụng HTTP GET và POST Methods tại Authorization Endpoint. Client CÓ THỂ (MAY) sử dụng HTTP GET hoặc POST methods để gửi Authorization Request tới Authorization Server. Nếu sử dụng HTTP GET Method, Request Parameters sẽ được Serialized sử dụng URI Query String Serialization. Nếu sử dụng HTTP POST Method, Request Parameters sẽ được Serialized sử dụng Form Serialization.

OpenID Connect sử dụng những Request Parameters sau của OAuth 2.0 với Authorization Code Flows

* scope – REQUIRED. OpenID…
* response\_type – REQUIRED.
* client\_id – REQUIRED.
* redirect\_uri – REQUIRED.
* state – RECOMMENED.

OpenID Connect cũng sử dụng các OAuth 2.0 Request Parameter được định nghĩa trong OAuth 2.0 Multiple Response Type Enconding Practices

* response\_mode – OPTIONAL. Sử dụng để thông báo cho Authorizatoin Server cơ chế cần được sử dụng để cho Returning Parameters từ Authorization Server. Việc sử dụng Parameter này KHÔNG ĐƯỢC KHUYẾN KHÍCH (NOT RECOMMENDED) khi Response Mode có thể ???

Các Parameter được định nghĩa thêm

* nonce
* display
* prompt
* max\_age
* ui\_locales
* id\_token\_hint
* login\_hint
* acr\_values

Các Parameters khác cũng CÓ THỂ (MAY) được gửi kèm

### Authentication using the Implicit Flow

### Authentication using the Hybrid Flow

Session này mô tả cách triển khải Authentication sử dụng Hybrid Flow. Khi sử dụng Hybrid Flow, Tokens có thể được trả về từ Authorization Endpoint và Token Endpoint. Cơ chế trả về những Tokens này được mô phân tích tại OAuth 2.0 Multiple Response Type Encoding Practices

#### Hybrid Flow Steps

Triển khai Hybrid Flow sẽ có những bước như sau

1. Client chuẩn bị một Authentication Request chứa những Parameters cần thiết
2. Client gửi Request tới Authorization Server
3. Authorization Server chứng thực End-user
4. Authorization Server nhận End-user Consent/Authorization
5. Authorization Server đưa End-user trở lại Client với một Authorization Code, và tùy thuộc vào Response Type sẽ có thêm một hoặc nhiều Parameters
6. Client request một Response sử dụng Authorization Code tại Token Endpoint
7. Client nhận một Response chứa một ID Token và Access Token trong Response Body
8. Client validates ID Token và nhận Subject Identifier của End-user

#### Authorization Endpoint

Khi sử dụng Hybrid Flow, Authorization Endpoint sẽ được sử dụng tương tự với trường hợp Authorization Code Flow, ngoài ra các trường hợp ngoại lệ sẽ được nêu rõ tại S này

##### Authentication Request

Authentication Requests được tạo như trong Section 3.1.2.1, ngoại trừ các Authentication Parameters được sử dụng như sau

response\_type – REQUIRED. OAuth 2.0 Response Type Value chỉ ra Authorization Processing Flow sẽ được sử dụng. Khi sử dụng Hybrid Flow, giá trị này là code id\_token, code token, hoặc code id\_token token. Các giá trị này được định nghĩa tại OAuth 2.0 Multiple Response Type Encoding Pratices

bdgf

#### Token Endpoint

## Initiating Login from a Third Party

## Claims

## Passing Request Parameters as JWTs

## Self-Issued OpenID Provider

## Subject Identifier Types

## Client Authentication

## Signatures and Encryption

## Offline Access

## Using Refresh Tokens

## Serializations

## String Operation

## Implemention Considerations

## Security Considerations

## Privacy Considerations

## IANA Considerations

## References

## OpenID Connect Single Logout (SLO)

### Overview

Single Logout (SLO) được sử dụng như là một bộ với Single Sign On (SSO). SSO tăng trải nghiệm người dùng bằng việc cung cấp khả năng sử dụng tính năng Sign On một lần duy nhất với các Clients khác nhau mà không cần phải lặp việc cung cấp Credentials. Việc này cũng đúng với SLO. Bằng việc hỗ trộ một SLO Protocol một người dùng có thể thực hiện việc Logout và tất cả Sessions sẽ được kết thúc mà không cần phải thực hiện hành động này tại tất cả Clients. Kết quả là một Client có triển khai SLO có thể bảo việc người dùng của nó và toàn bộ Data qua toàn bộ hệ thống bởi vì nó đảm bảo sẽ không còn bất kỳ Session nào được tạo bởi SSO còn hoạt động.

### OpenID Connect Logout

Logout ở đây là hướng tới việc làm cho các Active Session trở nên không còn hợp lệ. Tùy thuộc vào kiểu triển khai, thông tin Session sẽ được lưu trữ tại các nơi khác nhau.

* Session được lưu trữ tại User Agent (e.g. tại một Cookie)
* Session được lưu tại Server Side (e.g. tại Database)

OpenID Connect hiện định nghĩa 3 cơ chế cho Logout là

* OpenID Connect Session Management 1.0
* OpenID Connect Front-Channel Logout 1.0
* OpenID Connect Back-Channel Logout 1.0

Session Management định nghĩa một cơ chế cho OpenID Client (Relying Party, RP) có thể theo dõi một người dùng Login Status tại OpenID Provider. Khi người dùng sử dụng logout của OpenID Provider Client nên chấm dứt mọi Session có liên quan.

Front-Channel Logout được handled thông qua User Agent. Cho mỗi Client có một Session của User đượic cung cấp bởi OpenID Provider mà nó hỗ trợ cơ chế Front-channel Logout một iframe sẽ được rendered. Điều này nghĩa là logout Requests của tất cả Clients sẽ được thực hiện song song.

### OpenID Connect Back-Channel Logout 1.0

#### Introduction

Back-Channel Logout là một cơ chế sử dụng sự giao tiếp trực tiếp thông qua Back-channel của OP và RPs đang được logged out. Việc này có khác biệt so với phương pháp Front-channel, vì Front-channel sử dụng User Agent để thực hiện giao tiếp giữa OP và RPs.

Ưu điểm của việc giao tiếp thông qua Back-channel là việc nó đáng tin cậy hơn giao tiếp thông qua User Agent, bởi vì tại Front-channel, Session của RP Browser phải trong trại thái Active để việc giao tiếp có thể thành công.

Bất lợi của giao tiếp Back-channel là việc Session State được lưu lại giữa OP và RP thông qua Front-channel, như là Cookies và HTML5 Local Storage sẽ không thể sử dụng khi sử dụng Back-channel. Và kết quả là tất cả những State cần thiết cần phải được chỉ định cụ thể khi giao tiếp giữa các Parties. Hơn nữa, RPs phải triển khải một Application-specific Method để kết thúc Session của RP với OP dựa trên việc nhận Back-channel Logout Requests, điều này sẽ khiến việc triển khai trở nên phức tạp hơn là đơn giản xóa các Cookies và HTML5 Local Storage State, và đây là toàn bộ những gì chúng ta cần để triển khai Logout tại một Response tới Front-channel Logout Requests

Một điểm bất lợi lớn nữa của Back-channel Logout là backchannel\_logout\_uri của RP phải luôn có thể truy cập từ Ops. Điều này nghĩa là, ví dụ RP không thể ẩn sau Firewall hoặc NAT khi sử dụng Public Ops

#### Back-Channel Logout

##### Indicating OP Support for Back-Channel Logout

Nếu OpenID Provider hỗ trợ OpenID Connect Discovery, nó sẽ sử dụng giá trị Metadata này để chỉ ra việc nó có hỗ trợ Back-channel Logout hay không

Ví dụ: Identity Server Document



* backchannel\_logout\_supported – OPTIONAL. Giá trị Boolean chỉ ra việc OP có hỗ trợ việc sử dụng Back-channel Logout hay không, giá trị mặc định là false.  
  Và NÊN (SHOULD) đăng ký (register) giá trị Metadata backchannel\_logout\_session\_supported
* backchannel\_logout\_session\_supported – OPTIONAL. Giá trị Boolean chỉ ra việc OP có thể truyền một sid (session ID) Claim vào Logout Token để định danh RP Session với OP. Nếu được hỗ trợ, sid sẽ được chứa trong ID Token được issued bởi OP, nếu bị bỏ qua giá trị mặc định là false

sid (Session ID) Claim được sử dụng trong ID Tokens và như là một Logout Token Parameter được định nghĩa như sau (định nghĩa này cũng tương tự như tại Front-Channel Logout)

sid – OPTIONAL. Một chuỗi (string) định danh cho một Session. Nó đại diện cho một Session của User Agent hoặc một thiết bị mà End-User sử dụng đã Logged-in tại RP. Các giá trị sid khác nhau sử dụng để phân biệt các Session khác nhau tại một OP. Giá trị sid này chỉ duy nhất cần phải Unique trong Context của một Issuer cụ thể. Nội dung của sid không được hiểu tại RP, và có Syntax giống với OAuth 2.0 Client Identifier.

##### Indicating RP Support for Back-Channel Logout

RPs hỗ trợ Back-channel-based Logout sẽ đăng ký một Back-channel Logout URI với OP như là một phần của quá trình đăng ký (Registration)

* Back-channel Logout URI PHẢI (MUST) là một Absolute Path
* Back-channel Logout URI CÓ THỂ (MAY) chứa một application/x-www-form-urlencoded Formatted Query Component, và nó PHẢI (MUST) được giữ lại khi thêm các Query Parameters khác.
* Back-channel Logout URI KHÔNG ĐƯỢC PHÉP (MUST NOT) chứa một Fragment Component

Nếu một RP có hỗ trợ OpenID Connect Dynamic Client Registration, nó sẽ sử dụng giá trị của Metadata này để đăng ký Back-channel Logout URI

* backchannel\_logout\_uri – OPTIONAL. RP URL sẽ sử dụng để Logout bản thân RP khi nhận được một Logout Token từ OP. URL này NÊN (SHOULD) sử dụng https Scheme và CÓ THỂ (MAY) Port, Path và Query Parameter Components; tuy nhiên, có cũng CÓ THỂ (MAY) sử dụng http Scheme, nhưng cần sử dụng Client Type là confidential, và OP cho phép sử dụng http RP URIs  
  NÊN (SHOULD) đăng ký (register) giá trị Metadata backchannel\_logout\_session\_required
* backchannel\_logout\_session\_required – OPTIONAL. Giá trị Boolean chỉ ra việc RP có yêu cầu một sid (session ID) Claim được included trong Logout Token hay không

##### Remembering Logged-In RPs

…

##### Logout Token

OPs sẽ gửi một JWT tương tự như ID Token tới RPs được gọi là Logout Token để request việc Logout.

Logout Token sẽ chứa các Claim sau đây

* iss – REQUIRED. Issuer Identifier
* sub – OPTIONAL. Subject Identifier
* aud – REQUIRED. Audience(s)
* iat – REQUIRED. Issued at time
* jti – REQUIRED. Unique identifier for the token
* events – REQUIRED. Là một Claim trong với giá trị là một JSON Object chứa Member Name <http://schemas.openid.net/event/backchannel-logout>. Việc này để chỉ ra JWT là một Logout Token. Giá trị tương ứng PHẢI (MUST) là một JSON Object và NÊN (SHOULD) là một Empty JSON Object {}
* sid – OPTIONAL. Session ID, chuỗi (string) định danh cho một Session

Một Logout Token PHẢI (MUST) chứa sub hoặc sid Claim, hoặc CÓ THỂ (MAY) cả hai. Nếu một sid Claim không xuất hiện, nó chỉ ra việc tất cả Sessions tại RP để xác định End-user định danh bởi iss và sub Claims đã được logged out

Logout Token không được chứa các Claims sau

nonce – PROHIBITED. Một nonce Claim KHÔNG ĐƯỢC PHÉP (MUST NOT) xuất hiện….

Logout Tokens CÓ THỂ (MAY) chứa các Claims khác. Bất kỳ Claims nào không được hiểu PHẢI (MUST) bị bỏ qua.

Một Logout Token PHẢI (MUST) được signed và CÓ THỂ (MAY) được mã hóa. Keys được sử dụng để sign và mã hóa (encrypt) Logout Tokens và ID Tokens là như nhau. Nếu Logout Token được mã hóa, nó NÊN (SHOULD) thay thế iss (Issuer) Claim tại JWT Header Parameters

Note: Logout Token tương thích với Security Event Token (SET)

Ví dụ

{

"iss": "https://server.example.com",

"sub": "248289761001",

"aud": "s6BhdRkqt3",

"iat": 1471566154,

"jti": "bWJq",

"sid": "08a5019c-17e1-4977-8f42-65a12843ea02",

"events": {

"http://schemas.openid.net/event/backchannel-logout": {}

}

}

##### Back-Channel Logout Request

OP sử dụng một HTTP POST để đăng ký Back-channel Logout URI để trigger Logout Actions bởi RP. POST Body sử dụng application/x-www-form-urlencoded phải phải chứa một logout\_token Parameter chứa Logout Token từ OP cho RP thực hiện việc xác định End-user cần được logged out.

POST Body CÓ THỂ (MAY) chứa các giá trị khác cùng với logout\_token. Các giá trị không được hiểu khi triển khai PHẢI (MUST) bị bỏ qua.

POST /backchannel\_logout HTTP/1.1

Host: rp.example.org

Content-Type: application/x-www-form-urlencoded

logout\_token=eyJhbGci ... .eyJpc3Mi ... .T3BlbklE ...

##### Logout Token Validation

Ngay khi nhận được một Logout Request từ Back-channel Logout URI, RP PHẢI (MUST) validate Logout Token theo các bước sau.

1. Nếu Logout Token được mã hóa, thực hiện giải mã nó sử dụng Keys và thuật toán mà Client đã chỉ ra trong quá trình đăng ký (Registration) mà OP sử dụng để mã hóa ID Tokens. Nếu ID Token mã hóa là Negotiated (thương lượng) với OP tại Registration Time và Logout Token không được mã hóa, RP NÊN (SHOULD) bỏ qua nó.
2. Validate Logout Token Signature với cung cách ID Token Signature được validated với các điều chỉnh sau
3. Validate iss, aud và iat Claims giống như validate tại ID Tokens.
4. Verify Logout Token chứa một sub Claim, một sib Claim, hoặc cả hai
5. Verify Logout Token chứa một events Claim với giá trị là một JSON Object chứa Member Name là <http://schemas.openid.net/event/backchannel-logout>.
6. Verify Logout Token không chứa nonce Claim
7. Optionally verify một Logout Token khác với cùng giá trị jti không được nhận gần đây
8. Optionally verify iss Logout Token Claim

### OpenID Connect Front-Channel Logout 1.0

#### Introduction

OpenID Connect 1.0 là một Identity Layer xây dựng trên OAuth 2.0 Protocol. Nó cho phép Clients để xác thực định danh của một End-user dựa trên Authentication đượic thực thi bởi Authorization Server, cũng như nhận các thông tin cơ bản như thông tin Profile của End-user

Tại đây chúng ta sẽ được cung cấp các thông tin về định nghĩa của cơ chế Front-Channel Communication thông qua User Agent giữa OP (OpenID Provider) và RPs (Relying Parties) thực hiện việc logout mà không cần một OpenID Provider iframe tại Relying Party Pages như OpenID Connect Session Management 1.0. Các Protocols khác sử dụng HTTP GTEs tới RP URLs dùng để xóa Login State để thực hiện việc logout, và tại đây chúng ta cũng sử dụng cách tựa như vậy.

#### Relying Party Logout Functionality

RPs hỗ trợ HTTP-based Logout register một Logout URI với OP như là một bộ phận của quá trình đăng ký Client. Domain, Port và Scheme của URL này PHẢI (MUST) giống với Redirection URL Value đã được đăng ký.

Logout URI PHẢI (MUST) là một Absolute URI. Logout URI CÓ THỂ (MAY) include một application/x-www-form-urlencoded Formatted Query Component.

### OpenID Connect Session Management 1.0

…

# OAuth 2.0 and OIDC Support

## OAuth and OpenID Connect Concepts

## OAuth 2.0 Endpoints

## OAuth 2.0 and OIDC Workflows

## OAuth State Management

## Trusted Clients Management

## Proof Key for Code Exchange Support

## Reverse Proxy Configuration for OAuth and OIDC Provider

## Configuring OAuth 2.0 API Protection

## OIDC Claims Customization

## … More

# Bearer Token Usage

## Introduction

OAuth cho phép Client truy access tới một Protected Resource bằng cách nhận một Access Token thay vì sử dụng trực tiếp Resource Owner’s Password.

Tokens được Issued tới Client thông qua Authorization Server với sự chấp nhận của Resource Owner. Client sẽ sử dụng Acess Token để access tới Protected Resource được đặt tại Resource Server. Tại chương này chúng ta sẽ mô tả cách sử dụng OAuth Access Token là một Bearer Token.

## Authenticated Requests

Tại đây chúng ta sẽ mô tả 3 Methods để gửi Bearer Access Token trong Resource Request tới Resource Server. Clients KHÔNG ĐƯỢC (MUST NOT) sử dụng hơn một Method để chuyển Token trong một Request.

### Authorization Code Header Field

### Form-Encoded Body Parameter

### URI Query Parameter

# 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

# Reference Token and Introspection

Access tokens có thể tồn tại dưới hai dạng là Self-contained và Reference

* Self-contained Token sử dụng một Protected, Time-limited Data Structure mà trong nó bao gồm Metadata và Claims để giao tiếp với định danh (identity) của một User hoặc một Client được liên kết. Một Format được sử dụng rộng rãi của Access Token hiện nay là JSON Web Tokens (JWT). Đối với một Self-contained Token, người nhận chúng có thể validate Token locally bằng cách kiểm tra Signature, Expected Issuer Name và Expected Audience hoặc Scope
* Reference Tokens (đôi khi gọi là Opaque Tokens) trái lại chỉ là một định danh cho một Token được lưu trữ tại một Token Service. Token Service lưu trữ nội dung của Token và một và thứ khác, và liên kết với nó thông qua một Id (không thể bị đoán ra – infeasible-to-get) và gửi Id trở lại cho Client. Người nhận sẽ cần phải mở một Back-channel tới Token Service, gửi Token tới một Validation Endpoint, nếu nó hợp lệ, lấy toàn bộ nội dung trả về trong Response

Một trong những tính năng tuyệt vời của Reference Tokens là giúp chúng ta tăng khả năng kiểm soát vòng đời của chúng. Nơi mà một Self-contained Token khó có khả năng bị revoke trước khi chúng hết hạn, một Reference Token chỉ có thể tồn tại nếu như chúng tồn tại trong STS Data Store. Điều này giúp ích chúng ta trong các kịch bản như

* Revoking Token trong những trường hợp “emergency” (mất điện thoại…)
* Invalidate Token tại thời điểm User logout hoặc gỡ cài đặt ứng dụng

Bất lợi của Reference Token là nó cần một Back-channel để kết nối từ Resource Server tới STS

Việc này có thể không khả quan từ phương diện Network, và chúng ta có thể không tránh khỏi lo lắng vì việc tạo thêm một Round-trip và việc này có thể gây áp lực lên STS. Tuy nhiên hai vấn đề sau có thể được xử lý đơn giản bằng cách sử dụng Caching

Và có một hướng đi hiện tại đang được biết đến và sử dụng phổ biến, đại ý là Nếu Token được gửi ra ngoài phạm vi Company Infrastructure (e.g. tới một Browser hoặc một thiết bị di động), sử dụng Reference Tokens để có thể kiểm soát hoàn toàn được vòng đời Tokens đó, ngược lại nếu chỉ sử dụng Tokens trong nội bộ, một Self-contain Token sẽ có ích

# OpendID Connect Client Initiated Backchannel Authentication Flow - Draft

## Introduction

OpenID Connect cho phép Relying Parties (RP) xác thực User của họ cho tất cả các kiểu Client, bao gồm Browser-based JavaScript và Native Mobile Apps bằng cách launch một sign-in flows và nhận verifiable assertions về Identity của signed-in Users.

Trong tất cả các Flows được bắt đầu bằng RP thì End-user tương tác từ Consumption Device là bắt buộc và chúng dựa trên HTTP Redirection Mechanism. Tuy nhiên với một số Use Cases không được covered bởi những Flows đó bắt đầu xuất hiện, nơi mà RP cần là nơi phát động của User Authentication Flows và End-user tương tác từ Consumption Device là không còn cần thiết.

Client Initiated Backchannel Authentication (CIBA) là một Authentication Flow mới mà trong nó RPs có thể nhận một định danh hợp lệ của User nó muốn chứng thực sẽ có thể khởi tạo một Interaction Flow để chứng thực Users mà không cần End-user tương tác tự Consumption Device. Flow liên quan trực tiếp tới việc giao tiếp giữa Client tới OpenID Provider mà không xảy ra Redirect trong Browser của User (Consumption Device)

Specification này không thay đổi về mặt ngữ nghĩa của OpenID Connect Authentication Flow. Nó giới thiệu một Endpoint mới mà các Authentication Request được gửi tới, cùng với một Asynchronous method cho Authentication Result Notification hoặc Delivery. Nó không đưa ra Scope Values mới hay thay đổi ngữ nghĩa của Standard OpenID Connect Parameters.

Bởi vì việc User không cung cấp Authentication Credentials trực tiếp từ Consumption Device, để hỗ trợ Flow này đòi hỏi OpenID Provider có một vài cơ chế để kích hoạt User Authentication out-of-band từ việc tương tác với Consumption Device

# OpendID Connect Note

## Authentication with OpenID Connect Relying Party

Security Access Manager hỗ trợ Authentication với OpenID Connect (OIDC) Relying Party

Một OIDC Relying Party là một OAuth 2.0 Client Application đòi hỏi User Authentication và Claims từ OpenID Connect Provider. Security Access Manager hỗ trợ Relying Party (RP) như là một bộ phận của OAuth 2.0 và OpenID Connect (OIDC) Specifications.

Deployment của một Relying Party đòi hỏi kiến thức về các khái niệm OIDC, Work flows và Endpoints. Chi tiết tại OAuth 2.0 and OIDC Support

Một OpenID Connect (OIDC) Relying Party (RP) là một OAuth Client có thêm các một lớp Identity Management. Chúng ta có thể invoke một RP Connection tới Security Access Manager để log User vào WebSEAL. Relying Party triển khai Security Access Manager để sử dụng trong quá trình Authentication, và thêm các Basic RP Functions, một Initiation Delegate, một Reentry Delete và một Context Object. Security Access Manager RP sẽ sử dụng những tính năng sau của Secuity Access Manager

* Security Token Service (STSUU)
* Access Token Manager Credentials (iv-cred)
* JSON Web Token (JWT)
* Identity Mapping
* HTTP Callout
* Attribute Mapping

Security Access Manager RP hỗ trợ những tính năng sau của OpenID Connect (OIDC)

* OIDC Authorization Code, OIDC Implicit và OIDC Hybrid Flows
* 256, 384 và 512-bit SHA Signing Algorithm cho các kiểu HS, RS và ES, chi tiết đầy đủ tại <https://www.ibm.com/support/knowledgecenter/da/SSPREK_9.0.5/com.ibm.isam.doc/config/concept/oidc_rp_merge_partner_props.html#concept_uxk_1xm_nbb>
* Sử dụng response\_mode=form post
* RP luôn gửi một State và Nonce cho cả Implicit và Authorization Code Flow
* Encrypted ID Tokens
* Consumption of OIDC Provider (OP) Metadata at run time, for easy configuration

### Note

OIDC Specification nói rằng với Implicit Flows, chỉ duy nhất một Nonce là bắt buộc, và với Authorization Code, chỉ có State bắt buộc. Và Security Access Manager luôn luôn gửi cả hai

# Note

OpenID Connect Backchannel Authentication là một MODRNA Specification sử dụng cho một Authentication flow cho phép một Relying Party biết User’s Idenitifier nhận có thể nhận Tokens từ OpenID Connect Provider nơi mà User Consent được cấp tại User’s Authentication Device thông qua trung gian là OpendID Connect Provider

OpenID Connect Backchannel Authentication specifically addresses a [Use case](https://ldapwiki.com/wiki/Use%20cases) where the [Relying Party](https://ldapwiki.com/wiki/Relying%20Party) needs to be the initiator of the user [authentication](https://ldapwiki.com/wiki/Authentication) flow and the end-user interaction from the consumption [device](https://ldapwiki.com/wiki/Device) is not needed