OpenID Connect

OpenID Connect 1.0 può essere visto come un semplice livello di identità in cima al protocollo OAuth 2.0 e consente ai client di verificare l'identità dell'utente finale in base all'autenticazione eseguita da un server di autorizzazione, ma anche di ottenere informazioni di base sul profilo dell'utente finale in modo interoperabile.

Trattandosi di un protocollo standard, OpenID Connect consente ai client di tutti i tipi, compresi quelli basati su Web, dispositivi mobili e JavaScript, di richiedere e ricevere informazioni sulle sessioni autenticate e sugli utenti finali.

OAuth (Open Authorization)

Il framework di autenticazione **OAuth 2.0** è uno **standard aperto** che consente ad un utente di concedere ad un sito Web o ad un'applicazione di terze parti l'**accesso alle proprie risorse protette** tramite un **token** di accesso, senza necessariamente rivelare a terzi le proprie credenziali o addirittura l'identità.

Quando un servizio Web si connette a un altro, viene utilizzato il protocollo OAuth per garantire un'interazione sicura senza richiedere agli utenti di condividere le proprie password

Un token di accesso è una stringa che rappresenta l'autorizzazione all'utente finale ad accedere alle risorse sul Resource Server. OAuth 2.0 non definisce un formato specifico per i token di accesso. Tuttavia, in alcuni contesti, viene spesso utilizzato il formato JSON Web Token (JWT).

Per motivi di sicurezza, i token di accesso potrebbero avere una data di scadenza.

OpenID Connect

Poiché Oauth 2.0 non è capace di fornire informazioni relative all'autenticazione dell'utente finale OpenID Connect implementa l'autenticazione come un'estensione del processo di autorizzazione di Oauth 2.0

L'uso di questa estensione è richiesta dai Client per includere il valore della variabile openid nella richiesta di autenticazione. L'informazione relativa all'esecuzione dell'autenticazione è restituito come un JSON Web Token (JWT) chiamato ID Token. I Server di Autenticazione OAuth 2.0 che implementato OpenID Connect sono anche nominati OpenID Providers (OPs). I Client OAuth 2.0 che usano OpenID Connect sono anche denominati Relying Parties (RPs)

OpenID Connect

L’utente ha un’unica identità digitale per accedere alle risorse dell’intero ecosistema digitale e, nel caso di risorse differenti ma in trust tra loro, effettuando un’unica autenticazione gli sarà consentito l’accesso al pool di queste risorse senza autenticarsi nuovamente. Questo è uno dei risultati del protocollo di autenticazione decentrata definito nello standard OpenID Connect

OpenID Connect

Lo standard OpenID Connect consente parametri aggiuntivi nella richiesta e introduce i concetti di Claims e ID Token

I Claims sono coppie “nome-valore” che forniscono dati di attributo sull’utente che è stato autenticato. Vengono già forniti dei nomi standard per i Claims ma è previsto si possano aggiungere dei nuovi nomi per adattarsi a delle specifiche esigenze.

Terminologia OpenId Connect

- **End-User**: è l'individuo (utente), che richiede l'accesso a servizi online.
- **User Agent**: è il browser web usato dall'utente per accedere alle risorse online.
- **Claims**: è un insieme di informazioni sull'utente di tipo nome-valore.
- **Authorization Server**: è un server che possiede l'identità e le credenziali dell'utente.
- **OpenID Provider**: è l'Authorization Server capace di autenticare l'utente e rilasciare i Claims.
- **Relying Party**: è l'applicazione **Client** che richiede l'autenticazione dell'utente ed i Claims.
- **Resource Server**: è il server che ospita le risorse che saranno accedute.
- **ID Token**: è la stringa che contiene i Claims di autenticazione nel formato JWT (JSON Web Token), coppie del tipo nome:valore.
- **Subject Identifier**: è l'identificativo univoco dell'utente, rilasciato al Client.
- **UserInfo Endpoint**: è l'interfaccia che rilascia le informazioni autorizzate dell'utente.

Funzionamento autenticazione OpenID Connect

Il modello astratto di riconoscimento dell'identità è sintetizzato nei seguenti passi:

1. Il Client prepara una richiesta di autenticazione contenente i parametri di richiesta desiderati tramite lo User Agent.
2. Il Client invia la richiesta all'Authorization Server.
3. L'Authorization Server autentica l'End-User.
4. L'Authorization Server ottiene il consenso/autorizzazione dell'utente.
5. L'Authorization Server ritorna al Client un Access Token e, se richiesto, un ID Token.
6. Il Client richiede una risposta utilizzando l'Access Token al Token Endpoint.
7. L'Authorization Server valida l'Access Token e restituisce l'ID e l'Access Token.
8. Il Client convalida l'ID Token e recupera il Subject Identifier dell'utente.

Funzionamento autenticazione OpenID Connect

L'ID Token è costruito dall'OpenID Provider e restituito al Client come passo finale dell'autenticazione e contiene i Claims di autenticazione e quelli utente. Il token è nella forma di un JSON Web Token (JWT) ed è firmato in modo sicuro

Le caratteristiche standard dell'ID Token sono:

- Dichiara l'identificativo univoco dell'utente, chiamato Subject (*sub*).
- Specifica l'identificatore dell'autorità emittente la risposta (*iss*).
- È generato per un particolare insieme di destinatari (Clients) (*aud*).
- Può contenere una specifica idonea per una sola occasione (*nonce*).
- Può specificare quando (auth_time) e come, in termini di robustezza del metodo (*acr*), l'utente è stato autenticato.
- Stabilisce un tempo di emissione (*iat*) ed uno di scadenza (*exp*).
- Può includere ulteriori dettagli sull'utente, come nome e indirizzo email.
- È firmato digitalmente, quindi può essere verificato solo dai destinatari previsti.
- Può essere opzionalmente crittografato per la riservatezza.

Funzionamento autenticazione OpenID Connect

JSON Web Token:

è un sistema di cifratura e di contatto in formato JSON per lo scambio di informazioni tra i vari servizi di un server. Si genera così un token che può essere cifrato e firmato tramite una chiave disponibile solo a colui che lo ha effettivamente generato.

L'algoritmo di firma viene elaborato tramite HMAC (Keyed-hash message authentication code) o con chiavi pubbliche e/o private con standard RSA o ECDSA.

I JWT vengono utilizzati nei web services per autenticare un client. Il sistema di funzionamento è abbastanza semplice: il client invia una richiesta al server e questo genera un token di autenticazione che il client utilizzerà tutte le volte che andrà a collegarsi allo stesso nodo.

Funzionamento autenticazione OpenID Connect

HMAC

E' una modalità utilizzata per l'autenticazione dei messaggi. Si ricorre all'HMAC quando si ha necessità di garantire sia l'autenticità, sia l'integrità di un dato messaggio, per essere certi che non sia stato modificato/manipolato in alcun modo.

La modalità HMAC si basa su una funzione di hash. Utilizzata nei vari ambiti della sicurezza informatica, l'hash è una funzione che trasforma qualsiasi stringa di lunghezza variabile in una stringa di lunghezza predefinita. Ad esempio, "firma digitale" (in Adler32, una delle varie funzioni crittografiche di hash) diventa 288e0573.

L'hash è, quindi, una sorta di codice che permette di ricavare il messaggio originale dalla stringa originata dalla funzione. La peculiarità di HMAC è di non essere legato a nessuna funzione di hash in particolare, che quindi può essere sostituita in base alle necessità.

Funzionamento autenticazione OpenID Connect

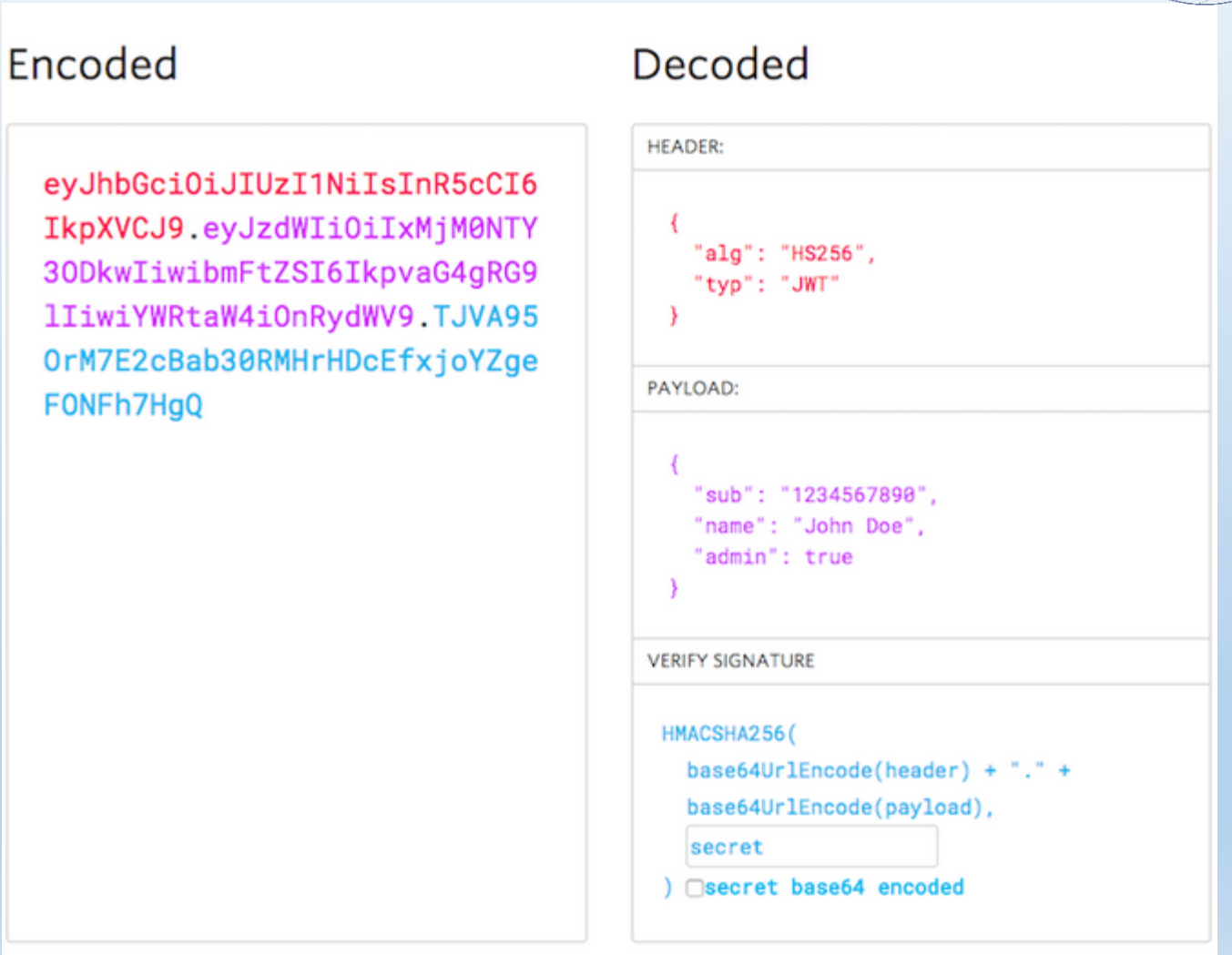
Il token si compone di tre corpi essenziali:

1. **Header:** Oggetto JSON contenente i parametri che descrivono le operazioni crittografiche e i parametri utilizzati. L'intestazione JOSE (JSON Object Signing and Encryption) è composta da un insieme di parametri di intestazione che in genere consistono in una coppia nome/valore: l'algoritmo di hashing utilizzato (ad esempio, HMAC SHA256 o RSA) e il tipo di JWT.;
2. **Payload**: contiene dichiarazioni sull'entità (in genere, l'utente) e attributi aggiuntivi dell'entità, chiamati attestazioni (Claims).
3. **Signature**: La firma viene utilizzata per verificare che il mittente del JWT sia chi dice di essere e per garantire che il messaggio non sia stato modificato lungo il percorso. Per creare la firma, vengono presi l'header e il payload con codifica Base64 e firmati con l'algoritmo specificato nell'intestazione.

Funzionamento autenticazione OpenID Connect

Esempio

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```



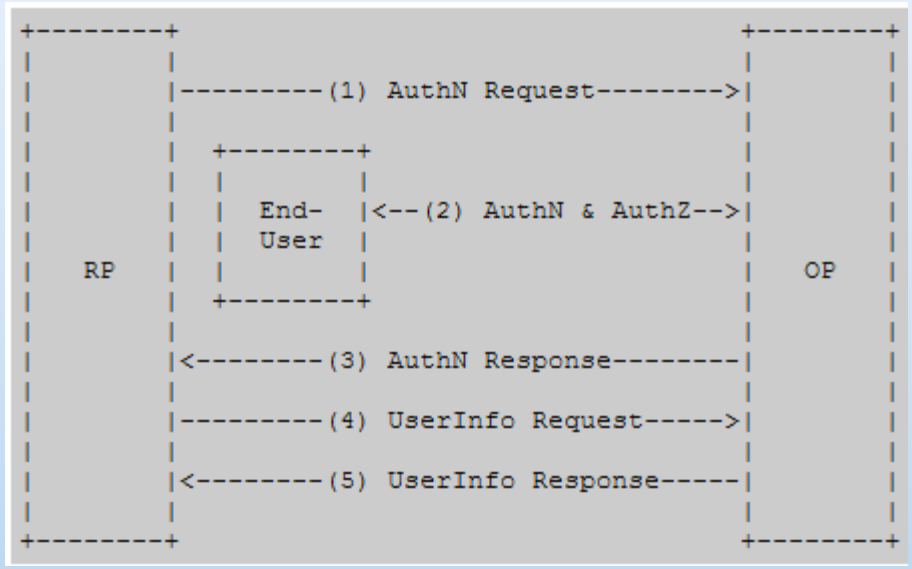
Funzionamento autenticazione OpenID Connect

Flussi di autenticazione di OpenID Connect

L'iniziale Authentication Request è strutturalmente simile ad una OAuth 2.0 Authorization Request il cui compito è semplicemente di richiedere che l'End-User sia autenticato dall' Authorization Server. Il processo di autenticazione può seguire uno dei tre flussi autorizzativi che si distinguono per delle peculiari caratteristiche, riassunte in tabella.

Property	Authorization Code Flow	Implicit Flow	Hybrid Flow
Tutti i token sono ritornati dal Authorization Endpoint	no	yes	no
Tutti i token sono ritornati dal Token Endpoint	yes	no	no
I token non sono rivelati al User Agent	yes	no	no
Il Client può essere autenticato	yes	no	yes
Il Refresh token è possibile	yes	no	yes
La comunicazione è in un solo viaggio andata/ritorno	no	yes	no
Principalmente le comunicazioni sono di tipo server-to-server	yes	no	varies

Funzionamento protocollo OpenID Connect



1. Il client (RP) invia una richiesta al OpenID Provider (OP) chiedendo l'autorizzazione e l'autenticazione utente
2. L'OP autentica l'utente finale e ottiene l'autorizzazione
3. L'OP risponde con un ID Token e di solito con un Access Token
4. L'RP può inviare una richiesta con l'Access Token all'UserInfo Token
5. Il UserInfo Endpoint restituisce asserzioni relative all'utente finale

Asserzioni (claims)

iss

OBBLIGATORIO. L'identificatore dell'emittente per l'Issuer della risposta. Il valore ISS è un URL case sensitive che utilizza lo schema https.

sub

OBBLIGATORIO. L'oggetto dell'identificatore. Un identificatore localmente unico e mai riassegnato all'interno dell'Emittente per l'Utente Finale, che è destinato ad essere consumato da parte del Client.

Asserzioni (claims)

aud

OBBLIGATORIO. Destinatario (i) per il quale questo ID Token è indirizzato. Esso deve contenere l'OAuth 2.0 client_id del Relying Party come un valore audience. Esso può anche contenere identificatori di altri tipo audience. Nel caso generale, il valore aud è un array di stringhe case sensitive. Nel caso comune speciale comune quando c'è un solo pubblico, il valore di aud può essere una singola stringa case sensitive.

exp

OBBLIGATORIO. Il tempo di scadenza oltre il quale l'ID Token non deve essere accettato per l'elaborazione. L'elaborazione di questo parametro richiede che la data e l'ora corrente deve essere antecedente alla data e ora della scadenza indicata nel valore. Gli implementatori possono prevede un margine di tolleranza di pochi minuti che tenga conto dello scostamento degli orologi. Il valore è un JSON number che rappresenta il numero di secondi dal 1970-01- 01T0:0:0Z calcolato in UTC fino alla data/ora

Asserzioni (claims)

iat

OBBLIGATORIO. L'ora in cui il JWT è stato rilasciato. Il suo valore è un JSON number.

auth_time

L'ora in cui è avvenuta l'autenticazione dell'Utente Finale. Il suo valore è un JSON number. Quando è fatta una richiesta max_age o quando auth_time è una richiesta come un asserzione essenziale, allora questo Claim è OBBLIGATORIO; in caso contrario, la sua inclusione è FACOLTATIVA.

Asserzioni (claims)

nonce

Un valore stringa utilizzato per associare una sessione Client con un ID Token e per mitigare i replay attacks. Il valore è scambiato immodificato dall'Authentication Request all'ID Token. Se presente nell'ID Token, i Client devono verificare che il valore nonce è uguale al valore del parametro nonce inviato nell'Authentication Request. Se presente nell'Authentication Request, gli Authorization Server devono includere un claim con valore nonce nell'ID Token con il valore claim inviato nella richiesta di autenticazione. Gli Authorization Server non devono eseguire nessun altra elaborazione sui valori di nonce utilizzati. Il valore nonce è una stringa case sensitive.

Asserzioni (claims)

acr

FACOLTATIVO. Riferimento all'Authentication Context Class. Una stringa che specifica un valore di riferimento alla classe di contesto di autenticazione che identifica l'esecuzione dell'autenticazione. Il valore acr è una stringa case sensitive.

amr

FACOLTATIVO. Riferimento ai metodi di autenticazione Un JSON array di stringhe che sono identificatori per i metodi di autenticazione utilizzati. Il valore amr è un'array di stringhe case sensitive.

azp

FACOLTATIVO. Soggetto autorizzato – l'entità al quale l'ID Token è stato rilasciato. Se presente deve contenere l'OAuth 2.0 Client ID dell'entità. Questa affermazione è necessaria solo quando l'ID Token ha un singolo valore di audience e quando l'audience è diverso dal soggetto autorizzato. Può essere incluso anche quando il soggetto autorizzato è lo stesso come unico audience. Il valore azp è una stringa case sensitive che contiene un valore StringOrURI

Asserzioni (claims)

Gli ID Token possono contenere altre entità. Eventuali Claims utilizzati che non sono compresi devono essere ignorati. Inoltre devono essere firmati utilizzando JWS23 e facoltativamente possono essere sia firmati e successivamente criptati usando rispettivamente JWS e JWE, fornendo in tal modo l'autenticazione, l'integrità, il non ripudio e, facoltativamente, la riservatezza. Se l'ID Token è crittografato, deve essere firmato e poi cifrato con il risultato che deve essere in JWT nidificato, come definito in JWT. Gli ID

Token non devono utilizzare none come il valore alg a meno che il tipo di risposta utilizzata ritorni senza ID Token dall'Authorization Endpoint (come ad esempio quando si utilizza il flusso Authorization Code) ed il Client esplicitamente ha richiesto l'uso di none al momento della registrazione

Esempio asserzioni in un ID Token


```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969,
  "acr": "urn:mace:incommon:iap:silver"
 }
```

Autenticazione

OpenID Connect esegue l'autenticazione per loggare l'utente finale o per determinare se l'utente finale è già connesso. Egli restituisce il risultato dell'autenticazione effettuata dal Server al client in maniera sicura in modo tale che il client possa contare su di esso. Per questo motivo in questo caso il Client è chiamato Relying Party (RP).

Il risultato di autenticazione viene restituito sotto forma di un ID token. Esso contiene asserzioni che esprimono informazioni quali l'Emittente (Issuer) , il Subject Identifier, quando scade l'autenticazione, ecc.

L'autenticazione può seguire uno dei tre seguenti percorsi:

o Authorization Code Flow (response_type=code),

o Implicit Flow (response_type=id_token token or response_type=id_token),

o Hybrid Flow (usando altri valori tipo di risposta definiti nella Oauth 2.0 Multiple Response Type Encoding Practices)

Authorization Code Flow

Quando si utilizza questo flusso, tutti i token vengono restituiti dal Token Endpoint. L'Authorization Code Flow restituisce un codice di autorizzazione (*Authorization Code)* al Client, che può successivamente scambiarlo direttamente per un ID Token e un Access Token.

Passi dell'Authorization Code Flow

- 1. Il Client prepara una Richiesta di Autenticazione (*AuthenticationRequest)* contenete i parametri di richiesta desiderati.
- 2. Il Client invia la richiesta all'Authorization Server.
- 3. L'Authorization Server autentica l'End-User.
- 4. L'Authorization Server ottiene il consenso/autorizzazionedall'End-User.
- 5. L'Authorization Server invia la risposta dell'End-User al Client con un *Authorization Code*.
- 6. Il Client richiede una risposta usando *l'Authorization Code* al TokenEndpoint.
- 7. Il Client riceve una risposta che contiene un *ID Token* e un *AccessToken* nel corpo della risposta.
- 8. Il Client convalida *l'ID Token* e recupera il *Subject Identifier* dell'End-User.

Subject Identifier è un identificatore unico all'interno dell'Emittente per l'End User. E' destinato ad essere consumato da parte del Client

Authentication Request

Una richiesta di autenticazione è un richiesta di autenticazione OAuth 2.0 che richiede che l'End-User sia autenticato dall'Authorization Server.

I Server di Autenticazione devono sostenere l'uso dei metodi HTTP GET e POST definiti nell'RFC 2616 presso l'Authorization Endpoint. I Client possono usare i metodi HTTP GET e POST per inviare la richiesta di autorizzazione al Server di Autenticazione. Se si usa il metodo http GET, i parametri della richiesta vengono serializzati mediante URI Query String Serialization. Se si utilizza il metodo http PUT, i parametri della richiesta vengono serializzati utilizzando il Form Serialization.

Authorization Endpoint

L’Authorization Endpoint esegue l’autenticazione dell’End-User. Ciò viene fatto con l’invio dell’User Agent all’Authorization Server's Authorization Endpoint per l’autenticazione e l’autorizzazione, utilizzando i parametri definiti dall’OAuth 2.0 ed i parametri aggiuntivi e i valori dei parametri definiti dall’OpenID Connect.

La comunicazione con Authorization Endpoint utilizza il protocollo TLS

Parametri per la richiesta Oauth 2.0

scope

OBBLIGATORIO. Le richieste di OpenID Connect devono contenere valorizzare il campo scope con openid. Se il valore di scope openid

non è presente, il comportamento è completamente indeterminato. Altri valori di scope possono essere presenti. I valori di scope utilizzati che non sono riconosciuti da un'implementazione dovrebbero essere ignorati.

response_type

OBBLIGATORIO. Il valore dell'OAuth 2.0 Response Type che determina il flusso di elaborazione di autorizzazione da utilizzare, includendo quali parametri vengono restituiti dagli endpoint utilizzati. Quando si utilizza l'Authorization Code Flow questo valore è code.

client_id

OBBLIGATORIO. Un identificatore OAuth 2.0 Client valido per l'Authorization Server.

Parametri per la richiesta Oauth 2.0

redirect_uri

OBBLIGATORIO. L'URI di Reindirizzamento a cui verrà inviata la risposta. Questo URI deve corrispondere esattamente a uno dei valori URI di reindirizzamento per il Client preregistrato presso il OpenID Provider, con l'abbinamento eseguita come descritto nella RFC3986 (Simple String Comparison). Quando si utilizza questo flusso, L'URI di Reindirizzamento deve utilizzare lo schema HTTPS; tuttavia, può utilizzare lo schema http, a condizione che il Client Type è confidenziale, e purchè l'OP consenta l'utilizzo di URI HTTP di reindirizzamento. Il rendirizzamento può utilizzare un sistema alternativo, come uno che ha lo scopo di identificare un callback in un'applicazione nativa

Parametri facoltativi

response_mode

Nonce

Display

Prompt

max_age

ui_locales

id_token_hint

login_hint

acr_values

Esempio

Location: https://server.example.com/authorize?

response_type=code

&scope=openid%20profile%20email

&client_id=s6BhdRkqt3

&state=af0ifjsldkj

&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb

Validazione Authentication Request

L’Authorization Server deve convalidare la richiesta ricevuta come segue:

1. L’Authorization Server deve convalidare tutti i parametri OAuth 2.0 secondo la specifica OAuth 2.0.

2. Verificare che un parametro di scope è presente e contiene il valore OpenID. (Se nessun valore di scope OpenID è presente, la richiesta può essere ancora una richiesta OAuth 2.0 valido, ma non è una richiesta di OpenID Connect.)

3. L’Authorization Server deve verificare che tutti i parametri richiesti sono presenti e il loro uso è conforme a questa specifica.

4. Se è richiesto il sub (soggetto) Claim con un valore specifico per l'ID Token, il server di autorizzazione deve inviare solo una risposta positiva se l'utente finale identificato da quel valore sub ha una sessione attiva con il server di autorizzazione o è stata autenticata come a seguito della richiesta. Il server di autorizzazione non deve rispondere con un ID Token o Access Token per un utente diverso, anche se hanno una sessione attiva con l'Authorization Server.

Come specificato nell’OAuth 2.0 [RFC6749], gli Authorization Server dovrebbero ignorare i parametri di richiesta non riconosciuti.
Se l’Authorization Server rileva qualche errore, deve restituire una risposta di errore

Authorization Server Authenticates End-User

Se la richiesta è valida, il server di autorizzazione tenta di autenticare l'utente finale o determina se l'utente finale è autenticato, a seconda dei valori dei parametri di richiesta utilizzati. I metodi utilizzati dal server delle autorizzazioni per autenticare l'utente finale (ad esempio nome utente e password, cookie di sessione, ecc) non vengono trattati in quanto gestiti dai vari OpenId Provider. Un'interfaccia utente di autenticazione può essere visualizzato dal server di autorizzazione, a seconda dei valori dei parametri di richiesta utilizzati ed i metodi di autenticazione utilizzati. Il server di autorizzazione deve tentare di autenticare l'utente finale nei seguenti casi:

• L'utente finale non è già autenticato.

• La richiesta di autenticazione contiene il parametro di richiesta con il valore login. In questo caso, il server di autorizzazione deve autenticare nuovamente l'utente finale, anche se l'utente finale è già autenticato.

Il server di autorizzazione non deve interagire con l'utente finale nel seguente caso:

• La richiesta di autenticazione contiene il parametro di richiesta con il valore none. In questo caso, il server di autorizzazione deve restituire un errore se l'utente finale non è già autenticato o non può essere in silenzio autenticati.

Quando si interagisce con l'utente finale, il server di autorizzazione deve assumere misure adeguate nei confronti di Cross-Site Request Forgery e Clickjacking come, descritto nei paragrafi 10.12 e 10.13 di OAuth 2.0

Authorization Server Obtains End-User Consent/Authorization

Una volta che l'utente finale è autenticato, il server di autorizzazione deve ottenere una decisione di autorizzazione prima di rilasciare le informazioni al Relying Party. Quando consentito dai parametri di richiesta utilizzati, questo può essere fatto attraverso un dialogo interattivo con l'utente finale che rende chiaro ciò che viene consentito o stabilire il consenso attraverso le condizioni per l'elaborazione della richiesta o di altri mezzi

Successful Authentication Response

Esempio di risposta positiva

Location: https://client.example.org/cb?
code=SplxlOBeZQQYbYS6WxSbIA
&state=af0ifjsldkj

Authentication Error Response

Una risposta di errore di autenticazione è un messaggio OAuth 2.0 Authorization Error Response restituito dall'OP's Authorization Endpoint in risposta al messaggio di richiesta di autorizzazione inviato dal Relying Party (RP).

Se l'utente finale nega la richiesta oppure l'autenticazione dell'utente finale non riesce, l'OP (Authorization Server) informa il RP (Client) utilizzando i parametri di risposta di errore di cui alla sezione 4.1.2.1 di OAuth 2.0 [RFC6749]. (Errori HTTP estranei a RFC 6749 vengono restituiti alla User Agent utilizzando il codice di stato HTTP appropriato.)

A meno che il reindirizzamento URI non sia valido, l'Authorization Server restituisce una risposta di errore al Cliente, al URI di reindirizzamento specificato nella richiesta di autorizzazione, con i parametri di stato error. Altri parametri non devono essere restituiti

Codici di errore

- interaction_required
- login_required
- account_selection_required
- consent_required
- invalid_request_uri
- invalid_request_object
- request_not_supported
- request_uri_not_supported
- registration_not_supported

Parametri di risposta di errore

- **Errore**

  RICHIESTO. Codice di errore.
- **error_description**

  FACOLTATIVO. Testo ASCII codificato leggibile dell'errore.
- **error_uri**

  FACOLTATIVO. URI di una pagina web che contiene ulteriori informazioni sull'errore.
- **state**

  OAuth valore di stato 2.0. Obbligatorio se la richiesta di autorizzazione comprendeva il parametro di stato. Impostare il valore ricevuto dal client

```
HTTP/1.1 302 Found
 Location: https://client.example.org/cb?
   error=invalid_request
   &error_description=
    Unsupported%20response_type%20value
   &state=af0ifjsldkj
```

Authentication Response Validation

Quando si utilizza l’Authorization Code Flow, il Client deve convalidare la risposta secondo la RFC 6749

Token Endpoint

Per ottenere un Access Token, un ID Token, e facoltativamente un Refresh Token, l'RP (Client) invia un richiesta di token al Token Endpoint per ottenere un token di risposta

Token Request

Un client effettua una richiesta di Token presentando la sua Authorization Gran (sotto forma di un Authorization Code) al Token Endpoint utilizzando il valore grant_type=authorization_code, come descritto nella Sezione 4.1.3 di OAuth 2.0 [RFC6749]. Se il client è un Confidential Client, allora deve autenticarsi al Token Endpoint utilizzando il metodo di autenticazione registrato per la sua client_id.

Il Client invia i parametri al Token Endpoint utilizzando il metodo HTTP POST e il modulo di serializzazione come descritto nella Sezione 4.1.3 di OAuth 2.0 [RFC6749].

Token Request

- Il seguente è un esempio di una richiesta di Token:

POST /token HTTP/1.1

Host: server.example.com

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

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code&code=**SplxlOBeZQQYbYS6WxS**

bIA&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb

Token Request Validation

L’Authorization Server deve convalidare il Token di richiesta come segue:

- Autenticare il Client se è stato emesso Client Credentials o se si utilizza un altro metodo di autenticazione del client.
- Assicurarsi che l’Authorization Code è stato rilasciato al Client autenticato.
- Verificare che l’Authorization Code sia valido.
- Se possibile, verificare che l’Authorization Code non sia stato utilizzato in precedenza.
- Assicurarsi che il valore del parametro *redirect_uri* è identico al valore del parametro *redirect_uri* che è stato incluso nella richiesta di autorizzazione iniziale. Se il valore del parametro *redirect_uri* non è presente, quando c'è un solo valore *redirect_uri* registrato, l’Authorization Server può restituire un errore (dato che il Client avrebbe dovuto includere il parametro) o può procedere senza un errore (dato che OAuth 2.0 permette l’omissione del parametro).
- Verificare che l’Authorization Code utilizzato sia stato rilasciato in risposta a una richiesta di autenticazione OpenID Connect (in modo che l'ID Token venga restituito dal Token Endpoint)

Successful Token Response

Dopo aver ricevuto e validato una Richiesta di Token dal Client, valida ed autorizzata, l'Authorization Server restituisce una risposta di successo che include un ID Token e un Access Token. La risposta utilizza il tipo di supporto application / json.

L'OAuth 2.0 token_type response parameter deve essere Bearer, come specificato nella OAuth 2.0 Bearer Token Usage a meno che non sia stato negoziato un altro tipo di token con il Client. Il Server dovrebbe sostenere il Bearer Token Type; l'uso di altri tipi di token è al di fuori del campo di applicazione della presente specifica

Oltre ai parametri di risposta specificati da OAuth 2.0 devono essere inclusi i seguenti parametri nella risposta:

id_token: Valore d'ID token associato alla sessione autenticata.

Tutte le Token Response che contengono Token, Segreti, o altre informazioni sensibili devono includere i seguenti campi e valori di intestazione di risposta HTTP:

Header Name	Header Value
Cache-Control	no-store
Pragma	no-cache

Esempio di Successful Token Response


```
HTTP/1.1 200 OK
 Content-Type: application/json
 Cache-Control: no-store

 {
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "refresh_token": "8xLOxBtZp8",
  "expires_in": 3600,
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
    yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
    NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
    fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
    AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
    Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
    NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
    QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
    K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
    XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
 }
```

Token Errore Response

Se il Token Request non è valido o non autorizzato, l'Authorization Server restituisce una risposta di errore. I parametri della risposta di errore sono definiti come nella sezione 5.2 di OAuth 2.0 [RFC6749]. Il corpo della risposta HTTP utilizza il tipo di supporto application / json con codice di risposta HTTP 400

Esempio di Token Error Response

HTTP/1.1 400 Bad Request

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"error": "invalid_request"

}

Token Response Validation

Il Client deve convalidare la risposta Token come segue:

1. Seguire le regole di convalida in RFC 6749, in particolare quelli nelle sezioni 5.1 e 10.12.
2. Seguire le regole di convalida ID Token.
3. Seguire le regole di convalida di Access Token

ID Token

La struttura è fatta nel seguente modo:

- **iss**
- **sub**
- **aud**
- **exp**
- **iat**
- **auth_time**
- **nonce**
- **acr**
- **amr**
- **Azp**

Quando si utlizza Authorization Code Flow si deve applicare il seguente requisito aggiuntivo

at_hash

FACOLTATIVO. Il valore hash dell'Access Token. Il suo valore è la codifica base64url della metà a sinistra degli ottetti della rappresentazione ASCII del valore ac*cess_tok*en, dove l'algoritmo di hash utilizzato è l'algoritmo hash utilizzato nel parametro alg del'ID Token JWS [JWS]

ID Token Validation

I client devono convalidare l'ID Token nella Token Response nel seguente

modo:

1. Se l'ID Token è crittografato, decifra usando le chiavi e gli algoritmi che il Client ha specificato durante la registrazione, lo stesso che l'OP ha utilizzato per crittografare l'ID Token. Se la crittografia è stata negoziata con l'OP in fase di registrazione e l'ID Token non è crittografato, la RP dovrebbe respingerla.

2. L'Identifier Issuer per l'OpenID Provider deve corrispondere esattamente al valore della ISS (emittente) Claim.

3. Il Client deve convalidare che il Claim aud (audience) contenga il valore client_id registrato presso l'Emittente individuato dal Claim ISS (emittente) come audience. L'aud (audience) Claim può contenere un array con più di un elemento. L'ID Token deve essere respinto se l'ID Token non elenca il Client come un audience valido, o se contiene audience supplementari non attendibile dal Client.

4. Se l'ID Token contiene più audience, il Client deve verificare che un Claim azp sia presente.

5. Se un AZP (soggetto autorizzato) è presente, il Client deve verificare che la sua client_id è il valore del Claim.

ID Token Validation

6. Se l'ID Token viene ricevuto tramite la comunicazione diretta tra il Client e il Token Endpoint (come in questo flusso), la convalida del server TLS può essere utilizzato per convalidare l'emittente al posto di controllare la firma del token. Il Client deve convalidare la firma di tutti gli altri ID Tokens secondo JWS [JWS] utilizzando l'algoritmo specificato nel parametro nell'intestazione JWT alg. Il Client deve utilizzare le chiavi fornite dall'Emittente.

7. Il valore alg dovrebbe essere di default RS256 o l'algoritmo inviato dal Client nel parametro id_token_signed_response_alg durante la registrazione.

8. Se il parametro di intestazione alg JWT utilizza un algoritmo basato su MAC come HS256, HS384, HS512 o, gli ottetti della rappresentazione UTF-8 della client_secret corrispondente al client_id contenute nelle AUD (audience) i Claim sono utilizzati come chiave per convalidare la firma. Per gli algoritmi basati su MAC, il comportamento non è specificato se l'aud è multivalore o se un valore azp è presente diverso rispetto al valore aud.

9. L'ora corrente deve essere prima dell'ora rappresentata dal Claim exp

ID Token Validation

10. L'attestazione iat può essere utilizzata per rifiutare token emessi troppo lontano dall'ora corrente, limitando la quantità di tempo di cui i nonce devono essere archiviati per prevenire attacchi. L'intervallo accettabile è specifico del cliente.

11. Se un valore nonce è stato inviato nella richiesta di autenticazione, DEVE essere presente un'attestazione nonce e il suo valore controllato per verificare che sia lo stesso valore di quello inviato nella richiesta di autenticazione. Il Client DOVREBBE controllare il valore nonce per gli attacchi di replay. Il metodo preciso per rilevare gli attacchi di riproduzione è specifico del client.

12. Se è stato richiesto il Risarcimento acr, il Cliente DOVREBBE verificare che il Valore del Risarcimento dichiarato sia adeguato. Il significato e l'elaborazione dei valori rivendicativi acr non rientrano nell'ambito di questa specifica.

13.Se è stata richiesta la richiesta auth_time, tramite una richiesta specifica per questa richiesta o utilizzando il parametro max_age, il client DOVREBBE controllare il valore della richiesta auth_time e richiedere una nuova autenticazione se determina che è trascorso troppo tempo dall'ultima autenticazione dell'utente finale .

Claim standard restituiti

Di seguito si definiscono un insieme standard di Claim. Possono essere richiesti per essere restituite sia nella risposta UserInfo o nell'ID Token.

Member	Type	Description
sub	string	Subject - Identifier for the End-User at the Issuer.
name	string	End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences.
given_name	string	Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	string	Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.
middle_name	string	Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used.
nickname	string	Casual name of the End-User that may or may not be the same as the `given_name`. For instance, a `nickname` value of `Mike` might be returned alongside a `given_name` value of `Michael`.
preferred_username	string	Shorthand name by which the End-User wishes to be referred to at the RP, such as `janedoe` or `j.doe`. This value MAY be any valid JSON string including special characters such as `@`, `/`, or whitespace. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.
profile	string	URL of the End-User's profile page. The contents of this Web page SHOULD be about the End-User.
picture	string	URL of the End-User's profile picture. This URL MUST refer to an image file (for example, a PNG, JPEG, or GIF image file), rather than to a Web page containing an image. Note that this URL SHOULD specifically reference a profile photo of the End-User suitable for displaying when describing the End-User, rather than an arbitrary photo taken by the End-User.
website	string	URL of the End-User's Web page or blog. This Web page SHOULD contain information published by the End-User or an organization that the End-User is affiliated with.
email	string	End-User's preferred e-mail address. Its value MUST conform to the **RFC 5322** [RFC5322] addr-spec syntax. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.
email_verified	boolean	True if the End-User's e-mail address has been verified; otherwise false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed. The means by which an e-mail address is verified is context specific, and dependent upon the trust framework or contractual agreements within which the parties are operating.
gender	string	End-User's gender. Values defined by this specification are `female` and `male`. Other values MAY be used when neither of the defined values are applicable.
birthdate	string	End-User's birthday, represented as an **ISO 8601-1** [ISO8601-1] `YYYY-MM-DD` format. The year MAY be `0000`, indicating that it is omitted. To represent only the year, `YYYY` format is allowed. Note that depending on the underlying platform's date related function, providing just year can result in varying month and day, so the implementers need to take this factor into account to correctly process the dates.
zoneinfo	string	String from IANA Time Zone Database **[IANA.time-zones]** representing the End-User's time zone. For example, `Europe/Paris` or `America/Los_Angeles`.
locale	string	End-User's locale, represented as a **BCP47** [RFC5646] language tag. This is typically an **ISO 639 Alpha-2** [ISO639] language code in lowercase and an **ISO 3166-1 Alpha-2** [ISO3166-1] country code in uppercase, separated by a dash. For example, `en-US` or `fr-CA`. As a compatibility note, some implementations have used an underscore as the separator rather than a dash, for example, `en_US`; Relying Parties MAY choose to accept this locale syntax as well.
phone_number	string	End-User's preferred telephone number. **E.164** [E.164] is RECOMMENDED as the format of this Claim, for example, `+1 (425) 555-1212` or `+56 (2) 687 2400`. If the phone number contains an extension, it is RECOMMENDED that the extension be represented using the **RFC 3966** [RFC3966] extension syntax, for example, `+1 (604) 555-1234;ext=5678`.
phone_number_verified	boolean	True if the End-User's phone number has been verified; otherwise false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this phone number was controlled by the End-User at the time the verification was performed. The means by which a phone number is verified is context specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. When true, the `phone_number` Claim MUST be in E.164 format and any extensions MUST be represented in RFC 3966 format.
address	JSON object	End-User's preferred postal address. The value of the `address` member is a JSON **[RFC8259]** structure containing some or all of the members defined in **Section 5.1.1**.
updated_at	number	Time the End-User's information was last updated. Its value is a JSON number representing the number of seconds from 1970-01-01T00:00:00Z as measured in UTC until the date/time.


spid
Poste ID NUOVO ABILITATO spid
Richiesta di accesso SPID 2 da
INPS
I seguenti dati stanno per essere inviati al fornitore dei servizi
Codice identificativo
Nome
Cognome
Luogo di nascita
Data di nascita
Sesso
Codice fiscale
Provincia di nascita
NON ACCONSENTO
ACCONSENTO
Per consultare l'informativa sul trattamento dei dati personali ai sensi del Regolamento 2016/679/UE clicca qui

UserInfo Endpoint

L’UserInfo Endpoint è un OAuth 2.0 Protected Resource che restituisce Claims relativi all’Utente Finale autenticato. Per ottenere le asserzioni richieste relative all’End-User, il Client fa una richiesta all’UserInfo Endpoint usando un Access Token ottenuto attraverso l’autenticazione OpenID Connect. Queste asserzioni sono normalmente rappresentate da un oggetto JSON che contiene una raccolta di coppie nome e valore di Claims.

La comunicazione con il UserInfo endpoint deve utilizzare il protocollo TLS.

L’UserInfo Endpoint deve sostenere l'uso dei metodi HTTP GET e POST HTTP definiti nella RFC 2616.

L’UserInfo Endpoint deve accettare Access Tokens come definiti in OAuth 2.0 Bearer Token Usage [RFC6750].

Si raccomanda che la richiesta utilizzi il metodo GET HTTP e che l’Access Token sia inviato utilizzando il campo di intestazione dell’autorizzazione

Esempio:

GET /userinfo HTTP/1.1

Host: server.example.com

Authorization: Bearer SlAV32hkKG

Successful UserInfo Response

L’UserInfo Claims devono essere restituiti come membri di un oggetto JSON a meno che sia stata richiesta una risposta firmata o crittografata durante la registrazione del Client.

Esempio di userInfo Response

HTTP/1.1 200 OK
Content-Type: application/json


```
{
 "sub": "248289761001",
 "name": "Jane Doe",
 "given_name": "Jane",
 "family_name": "Doe",
 "preferred_username": "j.doe",
 "email": "janedoe@example.com",
 "picture": "http://example.com/janedoe/me.jpg"
}
```

UserInfo Error response

Quando si verifica una condizione di errore, l'UserInfo Endpoint restituisce una risposta di errore come definito nella Sezione 3 di OAuth 2.0 Bearer Token Usage [RFC6750]

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Bearer error="invalid_token",

error_description="The Access Token expired"

UserInfo Response Validation

Il Client deve convalidare la risposta UserInfo come segue:

1. Verificare che l’OP ha risposto coincida con l'OP previsto attraverso un controllo del certificato del server TLS, per RFC6125.

2.Se il Client ha fornito un parametro userinfo_encrypted_response_alg durante la registrazione, decifrare la risposta UserInfo utilizzando le chiavi specificate durante la registrazione.

3. Se la risposta è stata firmata, il Client deve convalidare la firma

secondo JWS.