

Applications & Single Sign-On

Connect applications to ID Everywhere using OIDC and configure secure single sign-on.

- [OpenID Connect \(OIDC\)](#)
 - [Understanding Authentication Methods in ID Everywhere](#)
 - [Choosing the Right OIDC Application Type](#)
 - [Connecting an Application Using OIDC](#)
 - [Understanding OIDC Settings and Security Terms](#)
 - [Testing ID Everywhere OIDC with Postman](#)
 - [Troubleshooting OIDC Connections](#)

OpenID Connect (OIDC)

Learn how ID Everywhere uses OpenID Connect (OIDC) to provide secure application authentication and single sign-on. Explore application types, authorization flows, PKCE, scopes, and integration troubleshooting.

Understanding Authentication Methods in ID Everywhere

Overview

ID Everywhere provides centralized identity management so users can authenticate to supported applications using their ID Everywhere credentials.

Different applications support different authentication technologies. Understanding these technologies helps you choose the correct integration method.

ID Everywhere currently provides two documented integration methods:

- **OpenID Connect (OIDC)** for modern applications that support identity-provider-based sign-in.
- **LDAP** for applications that authenticate users through a directory service.

These methods serve different purposes, but both can use ID Everywhere as the authoritative source for user authentication.

What is OpenID Connect (OIDC)?

OpenID Connect is an identity authentication protocol built on OAuth 2.0.

It allows an application to redirect a user to a trusted identity provider, such as ID Everywhere, to complete sign-in.

How OIDC works

1. A user opens an application that is configured to use ID Everywhere.

2. The application redirects the user to ID Everywhere.
3. ID Everywhere authenticates the user.
4. After successful authentication, the user returns to the application.
5. The application securely validates the authentication response and identifies the signed-in user.

The application does not need to collect or store the user's ID Everywhere password.

Common OIDC use cases

- Web applications with an external identity-provider option.
- Custom business applications.
- Mobile and desktop applications that support OIDC.
- Applications offering configurable OpenID Connect single sign-on.

Choose OIDC when the application explicitly supports OpenID Connect or a compatible custom OIDC identity provider.

What is OAuth 2.0?

OAuth 2.0 is an authorization framework that allows applications to obtain tokens for permitted access.

OIDC uses OAuth 2.0 as its foundation and adds standardized user identity information.

This distinction is important:

- **OAuth 2.0:** Primarily provides authorization.
- **OIDC:** Provides authentication and identity information using OAuth 2.0 mechanisms.

An application advertising OAuth 2.0 support does not necessarily support OIDC sign-in.

ID Everywhere currently supports the **Authorization Code** grant for its OIDC integration.

Support for OIDC does not automatically mean ID Everywhere supports every OAuth 2.0 grant type.

What is LDAP?

Lightweight Directory Access Protocol (LDAP) is commonly used by applications that need to look up directory users and verify credentials.

Unlike OIDC, LDAP generally does not involve redirecting the user's browser to an identity provider. Instead, the application communicates with a directory service.

How LDAP authentication works with ID Everywhere

1. An LDAP-compatible application connects to the configured directory service.
2. The application submits a user authentication request.
3. The LDAP service delegates password verification to ID Everywhere.
4. ID Everywhere evaluates the credentials and account status.
5. The LDAP service returns an authentication result to the application.

ID Everywhere remains the authoritative password provider. The LDAP integration does not maintain a separate database of end-user passwords.

Common LDAP use cases

- Email applications supporting LDAP authentication.
- Legacy business applications.
- Software that requires directory-based user verification.
- Applications that can search an LDAP directory for user information.

Use a secure LDAP connection, such as LDAPS or an appropriately configured StartTLS connection, when transmitting credentials across networks.

What about SAML, OAuth 1.0, RADIUS, and Kerberos?

These are separate authentication or authorization technologies.

Technology	Purpose	Current ID Everywhere status
OpenID Connect (OIDC)	Modern federated user sign-in	Supported
OAuth 2.0 Authorization Code	Authorization flow used by OIDC	Supported for OIDC
LDAP	Directory access and authentication	Supported
OAuth 1.0	Older authorization protocol	Not supported by the documented integration

SAML 2.0	Enterprise federated authentication	Not currently supported
RADIUS	Network access authentication	Not currently supported
Kerberos	Ticket-based network authentication	Not currently supported

Other OAuth 2.0 grants, including Client Credentials, Implicit, Resource Owner Password Credentials, and Device Authorization, are not part of the currently documented IDE integration.

How do I choose the correct method?

Start by reviewing the application's authentication or SSO settings.

If it supports OpenID Connect: Configure an OIDC application in ID Everywhere.

If it supports LDAP: Configure an LDAP integration using the appropriate directory connection information.

If it only supports SAML, RADIUS, Kerberos, or OAuth 1.0: Do not attempt to configure it as OIDC or LDAP. Those protocols are not interchangeable.

If it says only "OAuth 2.0": Confirm that it supports OIDC authentication, an ID token, and a custom OIDC provider.

Frequently asked questions

Can one user use both OIDC and LDAP?

Yes. A user may authenticate to different compatible applications through either integration, subject to account status and applicable access controls.

Does an OIDC application receive the user's password?

No. The user authenticates with ID Everywhere. The application receives an authorization response and tokens, not the user's password.

Does LDAP maintain a separate password?

No. ID Everywhere's LDAP architecture delegates password verification to ID Everywhere.

Does every application support ID Everywhere?

No. The application must support an authentication method that ID Everywhere provides.

Related articles

- [Choosing the Right OIDC Application Type](#)
- [Connecting an Application Using OIDC](#)
- [Understanding OIDC Settings and Security Terms](#)

Choosing the Right OIDC Application Type

Overview

When registering an application in ID Everywhere, you must choose the appropriate OIDC application type.

The two available types are:

- **Public Application (PKCE)**
- **Web / Server Application (Confidential)**

Both use OpenID Connect. They are not separate authentication protocols.

The correct selection depends primarily on whether the application can securely protect a client secret.

Public Application (PKCE)

A public application cannot reliably keep a client secret confidential.

Examples include:

- Native mobile applications.
- Desktop applications distributed to users.
- Browser-only applications.
- Single-page applications running entirely in the user's browser.

Because users can inspect application code or distributed binaries, storing a permanent client secret inside these applications would be insecure.

How public applications authenticate

Public applications use the Authorization Code flow with **Proof Key for Code Exchange (PKCE)**.

PKCE helps protect the authorization-code exchange.

The application generates a temporary random value called a *code verifier*. It derives a *code challenge* using the S256 method and sends that challenge during authorization.

When exchanging the authorization code for tokens, the application provides the original verifier.

ID Everywhere can then verify that the client exchanging the code possesses the matching verifier.

Public application configuration

Setting	Value
Application type	Public Application (PKCE)
Authorization grant	Authorization Code
PKCE	Required
Code challenge method	S256
Client ID	Required
Client Secret	Not used
Redirect URI	Required

Web / Server Application (Confidential)

A confidential application has a trusted server-side environment capable of protecting a client secret.

Examples include:

- Traditional server-rendered web applications.
- Backend-driven business applications.
- Applications with a secure server responsible for exchanging authorization codes.

How confidential applications authenticate

1. The user starts sign-in from the application.

2. The browser redirects to ID Everywhere.
3. The user authenticates.
4. ID Everywhere redirects the browser back with an authorization code.
5. The application's backend exchanges that code for tokens.
6. The backend authenticates itself using its registered client credentials.

The client secret must remain on the trusted server.

Confidential application configuration

Setting	Value
Application type	Web / Server Application (Confidential)
Authorization grant	Authorization Code
Client ID	Required
Client Secret	Required
Client authentication	Client Secret Basic, where supported
Redirect URI	Required
PKCE	May be used additionally when supported

Important: Confidential applications can also use PKCE. PKCE and confidential client authentication provide different protections and are not mutually exclusive.

Which application type should I select?

Ask this question:

Can the application securely store a secret that ordinary users cannot retrieve?

If the answer is **no**, choose Public Application (PKCE).

If the answer is **yes**, and the application performs the token exchange on a trusted backend, choose Web / Server Application (Confidential).

Examples

Example 1: A native mobile application

The application runs on customer-owned phones.

Recommended type: **Public Application (PKCE)**.

Example 2: A PHP web application

The application runs on a secured server. Its backend performs the token exchange and stores the client secret outside the public web directory.

Recommended type: **Web / Server Application (Confidential)**.

Example 3: A browser-only JavaScript application

The application code is delivered to the browser, where users can inspect it.

Recommended type: **Public Application (PKCE)**.

Example 4: A web application with both frontend and backend components

The correct type depends on which component performs the authorization-code exchange and whether the backend securely manages the client secret.

A trusted backend may use a confidential client. A browser-only token exchange should use a public client.

Security recommendations

- Never place a confidential client secret in browser JavaScript.
- Never embed a confidential client secret in a mobile or desktop application.
- Register only the redirect URLs your application needs.
- Use HTTPS for production redirect URLs.
- Use PKCE S256 for public clients.
- Protect client secrets in server-side configuration or a secret manager.
- Rotate a client secret if it may have been exposed.
- Validate received tokens according to OIDC requirements.

Frequently asked questions

Is PKCE a different login protocol?

No. PKCE is a security mechanism used with the OAuth 2.0 Authorization Code flow.

Does a public application have a Client ID?

Yes. Public applications still use a Client ID to identify their registration.

Does a public application need a Client Secret?

No. Public applications cannot securely protect a permanent client secret.

Should every website use Confidential?

Not necessarily. A browser-only website generally cannot protect a secret. The architecture of the application determines the correct client type.

Related articles

- [Understanding Authentication Methods in ID Everywhere](#)
- [Connecting an Application Using OIDC](#)
- [Understanding OIDC Settings and Security Terms](#)

Connecting an Application Using OIDC

Overview

This guide explains how to connect an OpenID Connect-compatible application to ID Everywhere for centralized user authentication.

Before beginning, confirm that your application supports **OpenID Connect (OIDC)** and allows configuration of a custom identity provider.

Prerequisites

You will need:

- Administrative access to your ID Everywhere tenant.
- Permission to configure authentication in the application you're connecting.
- The application's authorized redirect or callback URL.
- A user account authorized to sign in through ID Everywhere.

Step 1 — Open OIDC Applications

Sign in to your ID Everywhere administration portal:

<https://app.ideverywhere.com>

Navigate to **OIDC Applications** in your tenant administration interface.

Choose the option to add a new application.

Step 2 — Enter the application details

Provide a recognizable application name.

For example:

- Customer Portal
- Employee Dashboard
- Internal CRM
- Support Application

Choose a name that helps administrators identify the integration later.

Step 3 — Select the application type

Choose the client type based on the application's architecture.

Public Application (PKCE)

Use for browser-only, desktop, and mobile applications that cannot securely store a client secret.

Web / Server Application (Confidential)

Use for applications with a trusted backend that securely stores a client secret and performs the token exchange.

See *Choosing the Right OIDC Application Type* for additional guidance.

Step 4 — Configure authorized redirect URLs

The redirect URL, sometimes called a callback URL, tells ID Everywhere where to send the browser after authentication.

Obtain this URL from the application's OIDC configuration instructions.

For example:

```
https://portal.example.com/auth/callback
```

Register the exact URL in ID Everywhere.

The redirect URL used during sign-in must match an authorized redirect URL registered for the client.

Pay attention to:

- `https://` versus `http://`
- Domain and subdomain
- URL path
- Trailing slash
- Port number, when applicable
- Query parameters, when applicable

Do not guess the callback URL. Use the value provided by the application.

For production web applications, use HTTPS.

Step 5 — Configure scopes

Scopes tell ID Everywhere what categories of information the application requests.

ID Everywhere advertises these scopes:

Scope	Purpose
<code>openid</code>	Required for OIDC authentication
<code>profile</code>	Requests standard profile information
<code>email</code>	Requests email-related identity information
<code>groups</code>	Requests group-related information where available

A common starting configuration is:

```
openid profile email
```

Always include `openid` when performing OIDC authentication.

Request only the scopes the application needs.

Step 6 — Save the application

After creating the application, ID Everywhere provides its connection information.

Record the **Client ID**.

For a confidential application, securely save the **Client Secret** when it is provided.

Do not place client secrets in public documentation, support tickets, or client-side code.

Step 7 — Configure the third-party application

Open the third-party application's authentication settings.

Where supported, use ID Everywhere's OIDC discovery URL:

```
https://app.ideverywhere.com/.well-known/openid-configuration
```

This discovery document publishes the provider's endpoints and supported OIDC capabilities.

If the application requires individual endpoint URLs, use:

Setting	URL
Issuer	<code>https://app.ideverywhere.com</code>
Authorization Endpoint	<code>https://app.ideverywhere.com/oauth2/authorize</code>
Token Endpoint	<code>https://app.ideverywhere.com/oauth2/token</code>
UserInfo Endpoint	<code>https://app.ideverywhere.com/oauth2/userinfo</code>
JWKS Endpoint	<code>https://app.ideverywhere.com/.well-known/jwks.json</code>
Discovery Document	<code>https://app.ideverywhere.com/.well-known/openid-configuration</code>

Enter the Client ID and, for confidential clients, the Client Secret.

Select **Authorization Code** as the grant type.

For public clients, enable **PKCE** with **S256**.

For confidential clients, use the client authentication method supported by the application and ID Everywhere, such as **Client Secret Basic**.

Step 8 — Test the connection

Start a new sign-in attempt from the connected application.

A successful flow generally looks like this:

1. The application redirects your browser to ID Everywhere.
2. You authenticate using your ID Everywhere credentials.
3. ID Everywhere returns your browser to the registered callback URL.
4. The application completes the token exchange.
5. The application validates the identity response.
6. You are signed in to the connected application.

The application must validate the ID token rather than simply trusting unverified token contents.

Step 9 — Verify user information

Depending on the requested scopes and the application's configuration, ID Everywhere may provide information such as:

- Unique user identifier (`sub`)
- Name
- Email address
- Tenant identifier
- Application role

Not every claim is guaranteed in every response. Applications should request appropriate scopes and handle optional claims correctly.

Security best practices

- Use a separate OIDC registration for each distinct application.
- Restrict redirect URLs to known, trusted destinations.
- Protect confidential client secrets.
- Use HTTPS.
- Request only necessary scopes.

- Validate token signatures, issuer, audience, and expiration.
- Review account access when users are disabled or removed.
- Rotate credentials if they may have been compromised.

Frequently asked questions

Can I use the same Client ID for several applications?

Separate registrations are recommended because each application may have different redirect URLs, credentials, and security requirements.

Can I change the redirect URL later?

Update the authorized redirect URLs in ID Everywhere when the connected application's callback URL changes.

What if the application asks for a SAML metadata URL?

That application is requesting SAML configuration, not OIDC. The current ID Everywhere OIDC integration cannot be substituted for SAML metadata.

What if the application asks for an OAuth 2.0 Client Credentials grant?

The current documented IDE OIDC integration uses Authorization Code. Client Credentials is a different grant and is not currently supported by this integration.

Related articles

- [Choosing the Right OIDC Application Type](#)

- Understanding OIDC Settings and Security Terms
- Troubleshooting OIDC Connections

Understanding OIDC Settings and Security Terms

Overview

When configuring OpenID Connect, applications may ask for settings such as grant type, scopes, client authentication, PKCE, S256, or RS256.

These terms describe different parts of the authentication process. They are not interchangeable.

This guide explains what each setting means and how it relates to ID Everywhere.

Client ID

The Client ID identifies an application registered with ID Everywhere.

It is not a password.

Both public and confidential applications use a Client ID.

Client Secret

A Client Secret is a credential used by a confidential application to authenticate itself to the token endpoint.

A Client Secret must be stored securely on a trusted server.

Public applications do not use a Client Secret.

Never publish or embed a Client Secret in browser JavaScript, mobile applications, or desktop software distributed to users.

Authorization Code grant

A grant type describes how an application obtains tokens.

ID Everywhere's current OIDC integration uses the **Authorization Code** grant.

The process is:

1. The application redirects the user to ID Everywhere.
2. The user authenticates.
3. ID Everywhere redirects back with an authorization code.
4. The application exchanges the code for tokens.
5. The application validates the authentication result.

An authorization code is temporary and intended for a single exchange.

Other OAuth grant types should not be selected unless ID Everywhere explicitly supports them.

PKCE — Proof Key for Code Exchange

PKCE adds protection to the Authorization Code flow.

It helps prevent a stolen or intercepted authorization code from being exchanged by a party that does not possess the original code verifier.

PKCE uses two important values:

Code Verifier: A cryptographically random value generated by the client.

Code Challenge: A value derived from the verifier and included in the authorization request.

During the token exchange, the client sends the verifier. The authorization server checks that it matches the original challenge.

PKCE is required for public clients in the documented IDE configuration.

Confidential clients may also use PKCE when supported.

SHA-256 and S256

SHA-256 is a cryptographic hashing algorithm.

S256 is the PKCE code challenge method that uses SHA-256.

Conceptually:

```
code_challenge = BASE64URL(SHA256(code_verifier))
```

In an application's configuration screen, you might see:

- S256
- SHA-256
- PKCE SHA-256

These generally refer to the same PKCE challenge method.

SHA-256 is not an authentication protocol or OAuth grant type.

RS256

RS256 is a digital signature algorithm based on RSA and SHA-256.

ID Everywhere advertises RS256 for signing OIDC ID tokens.

Applications can retrieve the provider's public signing keys through the JWKS endpoint and use them to verify token signatures.

RS256 and S256 are different:

Setting	Purpose
S256	Protects the authorization-code exchange through PKCE
RS256	Signs and verifies identity tokens
SHA-256	Underlying cryptographic hash algorithm

Do not select RS256 as the PKCE challenge method.

Scopes

Scopes indicate which information or permissions the client requests.

ID Everywhere advertises:

`openid`

Required to request OIDC authentication.

`profile`

Requests standard profile claims, such as name information.

`email`

Requests email-related identity claims.

`groups`

Requests group-related information where available.

A typical sign-in request uses:

`openid profile email`

Additional claims depend on the granted scopes and the provider's implementation.

ID Token

An ID Token communicates authentication and identity information to the OIDC client.

ID tokens are commonly represented as JSON Web Tokens (JWTs).

Typical claims include:

- `iss` — Issuer
- `sub` — Unique user subject identifier
- `aud` — Intended audience, typically the Client ID
- `exp` — Expiration time
- `iat` — Issued-at time

An application must validate the ID token's signature, issuer, audience, expiration, and other applicable requirements, including nonce validation when used.

Access Token

An Access Token is used to access a protected endpoint.

For example, an application can send an access token to ID Everywhere's UserInfo endpoint to request authorized user information.

An access token is not the same as a Client Secret or an ID Token.

Do not assume access tokens and ID tokens can be used interchangeably.

Refresh Token

A Refresh Token can allow an application to request new access tokens without repeating interactive sign-in.

However, **refresh-token support is not currently advertised in ID Everywhere's documented OIDC discovery capabilities.**

Do not enable refresh-token-dependent behavior without confirming that the integration supports it.

Redirect URI

A Redirect URI is the registered destination that receives the browser after authorization.

Example:

```
https://portal.example.com/auth/callback
```

The redirect URI must match an authorized URL registered in ID Everywhere.

It is a security-sensitive setting.

Issuer

The issuer identifies the OIDC provider that created the identity token.

ID Everywhere's issuer is:

```
https://app.ideverywhere.com
```

The application should validate that the ID token's issuer matches the expected issuer.

Discovery Document

The discovery document provides standardized information about the identity provider.

ID Everywhere discovery URL:

```
https://app.ideverywhere.com/.well-known/openid-configuration
```

Applications supporting OIDC discovery can use this URL to learn the provider's supported endpoints and capabilities.

JWKS — JSON Web Key Set

JWKS publishes public keys that applications can use to verify token signatures.

ID Everywhere JWKS URL:

```
https://app.ideverywhere.com/.well-known/jwks.json
```

Public signing keys can be shared. Private signing keys must remain protected.

Client authentication methods

Client authentication describes how a confidential application proves its identity when calling the token endpoint.

ID Everywhere advertises:

- `client_secret_basic` — Sends client credentials using HTTP Basic authentication.
- `none` — Used by public clients that do not authenticate using a client secret.

These methods should not be confused with the user's sign-in method.

Frequently asked questions

Is PKCE required for all applications?

It is required for public applications in the documented IDE integration. It can also provide additional protection for confidential clients.

Should I select S256 or RS256 in Postman?

For the **PKCE Code Challenge Method**, select **S256 / SHA-256**.

RS256 relates to ID-token signatures, not the PKCE challenge setting.

Is OAuth 2.0 the same as OIDC?

No. OIDC adds authentication and identity information to OAuth 2.0.

Is the Client ID secret?

No. The Client ID identifies the application. The Client Secret is confidential.

Related articles

- [Understanding Authentication Methods in ID Everywhere](#)
- [Choosing the Right OIDC Application Type](#)
- [Testing ID Everywhere OIDC with Postman](#)

Testing ID Everywhere OIDC with Postman

Overview

Postman can be used to test an ID Everywhere OpenID Connect integration before connecting a production application.

This guide demonstrates the **Authorization Code flow with PKCE**, using a confidential test client.

This configuration has been successfully tested against ID Everywhere, including token retrieval and a UserInfo request.

Prerequisites

You will need:

- Access to ID Everywhere.
- An account permitted to create OIDC applications.
- Postman with OAuth 2.0 authorization support.
- An active ID Everywhere user for testing.

Never use production client secrets in screenshots or publicly shared Postman collections.

Step 1 — Register an OIDC test application

Sign in at:

<https://app.ideverywhere.com>

Open your tenant's OIDC Applications management page.

Create an application with:

Application Name: Postman OIDC Test

Application Type: Web / Server Application (Confidential)

Authorized Redirect URL:

`https://oauth.pstmn.io/v1/browser-callback`

Save the application and securely record its Client ID and Client Secret.

Important: The Postman browser callback URL ends in `/v1/browser-callback`, not `/v1/callback`.

Step 2 — Open Postman's authorization settings

In Postman, create or open an HTTP request.

Select the **Authorization** tab.

Set the authorization type to **OAuth 2.0**.

Choose the option to configure a new token.

Step 3 — Configure OAuth 2.0

Enter the following settings:

Setting	Value
Grant Type	Authorization Code (With PKCE)
Callback URL	<code>https://oauth.pstmn.io/v1/browser-callback</code>
Authorization URL	<code>https://app.ideverywhere.com/oauth2/authorize</code>
Access Token URL	<code>https://app.ideverywhere.com/oauth2/token</code>

Client ID	Your IDE application's Client ID
Client Secret	Your IDE application's Client Secret
Scope	<code>openid profile email</code>
Code Challenge Method	SHA-256 / S256
Client Authentication	Send as Basic Auth header

The exact names of settings may vary slightly between Postman versions.

Why these settings matter

Authorization Code: The supported OIDC grant type.

PKCE S256: Protects the authorization-code exchange.

Client Secret: Authenticates the confidential client to the token endpoint.

Scope: Requests OIDC authentication and identity information.

Basic Auth Header: Sends the confidential client's credentials using the supported `client_secret_basic` method.

Step 4 — Request an access token

Click **Get New Access Token** or the equivalent action in Postman.

Postman should open the ID Everywhere authorization page.

Sign in using your ID Everywhere account.

After authentication, Postman should receive the authorization response and exchange the code for tokens.

If successful, Postman displays the resulting token information.

Do not share these tokens publicly.

Step 5 — Test the UserInfo endpoint

Create a new HTTP request in Postman.

Method: GET

URL:

```
https://app.ideverywhere.com/oauth2/userinfo
```

Under Authorization, select **Bearer Token** and use the access token returned by the successful authorization flow.

Alternatively, configure the request to use the OAuth 2.0 token already obtained in Postman.

Send the request.

Step 6 — Review the response

A successful UserInfo response may contain information similar to:

```
{
  "sub": "example-unique-user-id",
  "name": "Example User",
  "given_name": "Example",
  "family_name": "User",
  "tenant_id": 2,
  "role": "domain_admin",
  "email": "user@example.com",
  "email_verified": true
}
```

This is illustrative data. Actual values depend on the authenticated user and the granted scopes.

A successful response confirms that the access token was accepted by the UserInfo endpoint and that identity information was returned.

It does not, by itself, prove that a third-party application correctly validates ID tokens.

Common Postman errors

invalid_request

Verify that the Scope includes:

openid

This is required for OIDC authentication.

Also confirm the grant type, callback URL, and other required authorization parameters.

Redirect URL mismatch

Ensure the callback URL registered in ID Everywhere exactly matches the one used in Postman.

invalid_client

Verify the Client ID, Client Secret, and client authentication method.

For this confidential-client example, use **Basic Auth header**.

invalid_grant

Start a new authorization flow.

Authorization codes cannot be reused. Also confirm the PKCE verifier and redirect URI are consistent across the authorization and token requests.

No UserInfo response

Confirm that the request uses the **access token**, not the ID token or Client Secret.

Check whether the token is expired.

Testing public applications

For a public-client test:

1. Create a Public Application (PKCE) registration in ID Everywhere.
2. Register Postman's exact callback URL.
3. Select Authorization Code with PKCE S256.
4. Supply the Client ID.
5. Do not supply a Client Secret.
6. Use a client authentication method of None, where available.
7. Complete the authorization flow.

Security reminders

- Do not publish Client Secrets.
- Do not paste live tokens into support tickets.
- Use dedicated test applications where possible.
- Revoke or rotate exposed credentials.
- Remove unused test registrations.

Related articles

- [Connecting an Application Using OIDC](#)
- [Understanding OIDC Settings and Security Terms](#)
- [Troubleshooting OIDC Connections](#)

Troubleshooting OIDC Connections

Overview

This guide covers common problems when connecting an application to ID Everywhere using OpenID Connect.

Many connection issues are caused by incorrect redirect URLs, missing scopes, mismatched client credentials, or unsupported OAuth settings.

Start by identifying the exact error returned by the application or identity provider.

1. Error: `invalid_request`

Possible causes

- The authorization request is missing required parameters.
- The `openid` scope was omitted.
- The application is using an unsupported authorization flow.
- Required PKCE parameters are missing or malformed.

Recommended steps

1. Confirm the application uses the Authorization Code flow.
2. Include `openid` in the requested scopes.
3. Verify the Client ID.
4. Confirm the redirect URI is registered.
5. For public clients, confirm PKCE S256 is enabled.
6. Retry with a new authorization request.

A common working scope configuration is:

2. Redirect URI mismatch

Symptoms

Authentication is rejected before completion, or the application reports an invalid callback URL.

Possible causes

The callback URL supplied by the application does not match the registered redirect URL.

Recommended steps

Compare the URLs character by character.

Check:

- HTTP versus HTTPS.
- Hostname and subdomain.
- Port number.
- URL path.
- Trailing slash.
- Query parameters.

For example, these are different redirect URLs:

```
https://portal.example.com/auth/callback
```

```
https://portal.example.com/auth/callback/
```

Do not add wildcard redirect URLs to work around mismatches.

3. Error: invalid_client

Possible causes

- Incorrect Client ID.
- Incorrect or outdated Client Secret.
- Wrong application type.
- Unsupported client authentication method.
- Credentials belonging to another application registration.

Recommended steps

1. Verify the Client ID in ID Everywhere.
2. Confirm whether the client is Public or Confidential.
3. For confidential clients, verify the Client Secret.
4. Confirm the application uses an appropriate token endpoint authentication method.
5. For public clients, remove any unnecessary Client Secret.
6. If the secret may have been exposed, rotate it and update the connected application.

4. Error: invalid_grant

Possible causes

- The authorization code has expired.
- The code has already been used.
- The PKCE verifier does not match the original challenge.
- The redirect URI differs between authorization and token exchange.
- The code was issued to a different client.

Recommended steps

1. Start a completely new sign-in attempt.
2. Do not reuse an authorization code.
3. Confirm the same Client ID is used throughout the flow.
4. Verify the redirect URI remains consistent.
5. Confirm PKCE S256 is configured correctly.
6. Ensure the application and server clocks are accurate.

5. Error: invalid_scope

Possible causes

The application requests a scope that is not supported or permitted.

Recommended steps

Start with:

openid profile email

ID Everywhere currently advertises:

- openid
- profile
- email
- groups

Remove unsupported scopes and retry.

Only request `groups` if the application needs group-related information and the integration supports the required claim behavior.

6. Error: Token signature validation failed

Possible causes

- The application is using the wrong issuer.
- It has outdated signing keys.
- The expected audience does not match.
- The token is expired.
- The application is configured with the wrong signing algorithm.

Recommended steps

Verify the expected issuer:

<https://app.ideverywhere.com>

Verify the JWKS endpoint:

```
https://app.ideverywhere.com/.well-known/jwks.json
```

Confirm that the application:

1. Retrieves the correct public signing keys.
2. Validates the token signature.
3. Validates the issuer (`iss`).
4. Validates the audience (`aud`).
5. Checks expiration (`exp`).
6. Performs any other required OIDC validation.

ID Everywhere advertises RS256 for ID-token signatures.

Never disable signature verification as a troubleshooting workaround.

7. Error: Unauthorized UserInfo request

Possible causes

- Missing Bearer token.
- Expired or invalid access token.
- Sending the wrong token type.
- User or client access restrictions.

Recommended steps

Make a GET request to:

```
https://app.ideverywhere.com/oauth2/userinfo
```

Include:

```
Authorization: Bearer YOUR_ACCESS_TOKEN
```

Replace the placeholder with a valid access token.

Do not use the Client Secret or ID Token in place of the access token.

8. User cannot sign in

Possible causes

- Incorrect credentials.
- Disabled or otherwise ineligible account.
- Application or tenant access restrictions.
- Incomplete authentication requirements.

Recommended steps

1. Confirm the user can sign in to ID Everywhere normally.
2. Verify the user's account is active and authorized.
3. Confirm the correct tenant and application registration.
4. Review relevant authentication and audit information available to administrators.
5. Retry after resolving the underlying account issue.

Do not repeatedly retry a locked or restricted account without determining the cause.

9. Application asks for unsupported settings

Some applications offer multiple authentication protocols and grant types.

SAML metadata URL

SAML is not interchangeable with OIDC. The documented IDE integration does not currently provide SAML configuration.

OAuth 1.0 credentials

OAuth 1.0 is not supported by the documented integration.

Client Credentials grant

The documented OIDC integration uses Authorization Code, not Client Credentials.

Implicit or Password grant

These are not supported by the current documented IDE integration.

Refresh Token required

Refresh-token support is not currently advertised. Confirm the connected application's requirements before proceeding.

10. Third-party testing tool reports an error

Some online OIDC testing tools use their own backend services to exchange authorization codes.

An error from the testing tool does not necessarily indicate an error in ID Everywhere.

For example, a tool may reject a URL through its own security validation before contacting the provider.

If this happens:

1. Identify which service generated the error.
2. Verify ID Everywhere's discovery document.
3. Retry using Postman or another trusted OIDC-compatible client.
4. Compare the authorization and token requests.
5. Avoid exposing credentials in publicly accessible testing tools.

Information to provide when requesting support

To help diagnose a connection issue, provide:

- Application name.
- Client ID (not the Client Secret).

- Application type (Public or Confidential).
- Requested scopes.
- Grant type.
- Redirect URL.
- Exact error code and message.
- Approximate time of failure, including timezone.
- Whether the failure occurs before sign-in, after sign-in, or during token exchange.

Never send passwords, Client Secrets, authorization codes, access tokens, ID tokens, or private signing keys to support.

Related articles

- [Understanding Authentication Methods in ID Everywhere](#)
- [Choosing the Right OIDC Application Type](#)
- [Connecting an Application Using OIDC](#)
- [Understanding OIDC Settings and Security Terms](#)
- [Testing ID Everywhere OIDC with Postman](#)