OIDC Federation Setup Guide (SSO)

Setting Up Single Sign-On (SSO) with Adhese

This guide explains what's needed to connect your company's login system (your "Identity Provider" or IdP — e.g. Okta, Azure AD, OneLogin, Keycloak) to the Adhese platform, so your users can log in with their existing company account instead of a separate Adhese password.

We've tried to keep this guide free of unnecessary jargon. Where a technical term is unavoidable, we explain it in plain language the first time it appears.

It is probably best to direct your organizations' IdP responsible to this document, because they will be best suited to go through its steps.

---

1. Overview — how this works

Think of it like using "Sign in with Google" on another website. Instead of Adhese storing a password for your users, your users log in through **your own company system**. Once they're verified there, your system tells Adhese "this person is who they say they are, and here's some basic info about them."

- **Your IdP** = your company's login system. It does the actual authentication (checking passwords, MFA, etc.).
- **Adhese** = the application your users want to reach. We just need to trust your IdP's confirmation.

The technology that makes this handshake work is called **OIDC** (OpenID Connect) — a standard way for two systems to exchange "this user is verified, here's who they are" information.

---

2. What we need from you

To connect to your IdP, we need a small set of technical addresses and credentials. Your IT/identity team will have these, or can generate them when creating a new application/integration for Adhese in your IdP.

Item	In plain terms
Discovery URL	A single web address that automatically tells us all the other addresses below. If your IdP provides this (most modern ones do), you usually don't need to give us anything else in this table.
Authorization endpoint	The web address your users get sent to, to log in. (Only needed if there's no discovery URL.)
Token endpoint	The address we use, behind the scenes, to confirm the login was real. (Only needed if there's no discovery URL.)
UserInfo endpoint	An address we can check for extra user details, if needed. (Only needed if there's no discovery URL.)
JWKS URI	The address holding your IdP's "signature keys," which we use to confirm nothing was tampered with. (Only needed if there's no discovery URL.)
Client ID	An ID number/string your IdP assigns to identify "the Adhese application" specifically.
Client Secret	A password-like secret that goes with the Client ID, so we can prove to your IdP that we really are Adhese. Keep this confidential.
Client authentication method	How we should send the Client ID/Secret when talking to your IdP: `client_secret_post` or `client_secret_basic`.
Supported scopes	Confirmation that your IdP can send us the specific pieces of information we need (see Section 4).

> **Simplest path:** if your IdP has a discovery URL, you really only need to give us: the discovery URL, the Client ID, the Client Secret, and the client authentication method.

3. What we give you

To set things up on your side, we'll send you:

Item	In plain terms
Redirect URI (Callback URL)	The exact web address your IdP must be told to send users back to after login. You'll paste this into your IdP's app configuration, usually a field called "Redirect URI," "Callback URL," or "Allowed callback."
Required scopes	Which permissions/data your IdP setup needs to allow (see Section 4).
Required claims	Which specific pieces of user information must be included (see Section 4).

4. Scopes and claims, explained simply

Think of it this way:

- A **scope** is a *permission you switch on* in your IdP's app configuration. It's like ticking a box that says "yes, allow sharing this category of information."
- A **claim** is the *actual piece of information* that gets sent to us once that permission is switched on, like a labeled field with a value.

For example, ticking the `email` scope is what allows the `email` claim (the user's actual email address) to be included.

4.1 Scopes and claims we require

Scope (the permission)	Claim it unlocks	What that claim contains
`openid`	`sub`	A unique ID number for the user. Always required for any OIDC login — this is the standard itself.

Scope (the permission)	Claim it unlocks	What that claim contains
`email`	`email`	The user's email address.
`email`	`email_verified`	A `true`/`false` flag confirming your IdP has actually verified that email address (e.g. via a confirmation link). **This must come through as `true`.**

> ⚠ **Important:** Some IdPs don't send `email_verified` automatically, even when the `email` scope is enabled — it's technically an optional field in the OIDC standard. Please double-check with whoever manages your IdP that this field is switched on and will read `true` for your users. **If it's missing, or set to `false`, the login will be rejected.**

4.2 Optional: showing real names in Adhese

Enabling the `profile` scope is not required, but lets us show your users' actual names in the Adhese interface instead of just an ID or email address.

Scope	Claim	What it contains
`profile`	`name`	Full name
`profile`	`given_name`	First name
`profile`	`family_name`	Last name
`profile`	`preferred_username`	Username

---

5. Client authentication method (`client_auth_mode`)

When Adhese talks to your IdP to confirm a login, we have to prove we are who we say we are, using the Client ID and Client Secret from Section 2. There are two common ways to send that proof, and **your IdP will only accept one of them** — so we need to know which.

Mode	In plain terms
`client_secret_basic`	The Client ID and Secret are sent using a built-in web standard called "HTTP Basic Authentication" — essentially packed into a request header, not visible in the main message body.
`client_secret_post`	The Client ID and Secret are sent as regular fields inside the request itself (like filling in a form), rather than in a header.

Neither option is more or less secure in practice — it's purely about which format your IdP expects. Most modern IdPs support both, and their app registration screen will usually have a dropdown or setting labeled something like "Token endpoint authentication method." **Please check this setting and let us know which value it's set to**, so we configure our side to match. If we don't match your IdP's expectation, the login exchange will fail even if everything else is correct.

---

6. Role mapping

How it works

You (or your IdP admin) pick a claim name — for example `adhese_role` — and tell us that name. From then on, whatever value(s) come through in that claim will decide what roles the user gets in Adhese, updated automatically every time they log in.

The value can be:

- **One role**, as plain text, or
- **Several roles**, as a list.

One role:

```
{
  "adhese_role": "admin"
}
```

Multiple roles:

```
{
  "adhese_role": ["viewer", "creative_approver"]
}
```

The important part: **the claim name you chose becomes a normal, top-level field in the token, with the role name(s) as its value.** Nothing more nested or wrapped around it.

Extra scope

If your idp expects us to send a specific scope before it embeds the custom claim in the token, you need to let us know the scope we have to request.

Available roles — Classic UI

Role	Description
`classic_admin`	Full admin access to the Classic UI
`classic_read_only`	Read-only access to the Classic UI

Available roles — New UI

Role	Description
`admin`	Full administrator
`creative_approver`	Can approve creatives
`creative_master`	Full creative management
`managed_ad_master`	Managed advertising management
`self_service_ad_master`	Self-service advertising management
`viewer`	Read-only access
`access_all_advertisers_debtors_brands`	Access across all advertisers, debtors, and brands

Examples

```
{
  "bidfood_role": "adheseAdmin"
}
```

If a user has more than one role:

```
{
  "bidfood_role": ["adheseAdmin", "adheseOtherRole"]
}
```

> **Tip:** Some IdPs also require you to separately enable a scope before a custom claim will actually be included in the token — even if the claim itself looks correctly configured. For example, some setups need a `params` (or similarly named) scope explicitly requested before the custom role claim shows up at all. If your custom role claim isn't arriving, check whether an additional scope needs to be requested, and let us know its exact name so we can add it to our request.

---

7. Setup checklist

Your side (IdP)

- ☐ Register a new OIDC application/client for Adhese
- ☐ Add the redirect URI we provide as an allowed callback
- ☐ Enable the `openid` and `email` scopes
- ☐ Confirm the `email_verified` claim is included and will read `true` | if not, let us know
- ☐ Confirm which client authentication method your IdP uses: `client_secret_post` or `client_secret_basic`
- ☐ *(Optional)* Enable the `profile` scope for real names
- ☐ *(Optional)* Set up a custom claim for role mapping, making sure it outputs a plain value (see Section 6's common mistake)
- ☐ *(Optional)* Confirm whether an extra scope is needed to expose that custom claim
- ☐ Send us: Client ID, Client Secret, discovery URL (or individual endpoints), client authentication method, wether or not `email_verified` claim will be set correctly

Our side (Adhese)

- ☐ Provide the redirect URI
- ☐ Configure the connection with your provided endpoints and credentials

- ☐ Configure the client authentication method to match yours
- ☐ Configure scope requests (`openid`, `email`, optionally `profile`)
- ☐ Configure validation so `email_verified` must be `true`
- ☐ *(Optional)* Configure role mapping based on the agreed claim
- ☐ Perform a test login together

---

8. Testing

Once both sides are set up, we'll do a test login together:

1. Start a login on the Adhese platform
2. Confirm the redirect to your IdP works
3. Log in with a test user
4. Confirm the return trip back to Adhese succeeds
5. Confirm the user's email and name display correctly
6. *(If applicable)* Confirm the correct role(s) were applied

If the test login fails

The most common causes, in order of likelihood:

- The `email` scope isn't actually enabled on the IdP side
- The `email_verified` claim isn't included in the token at all
- The user's email isn't marked verified at the IdP (`email_verified: false`)
- The client authentication method (Section 5) doesn't match what we configured on our side
- A custom role claim contains the IdP's raw parameter configuration instead of a plain value (see the common mistake in Section 6)

Revision #5
Created 2026-04-15 07:41:00 UTC by Wout De Rooms
Updated 2026-08-21 08:45:04 UTC by Wout De Rooms