Verifier Integration: choosing the right integration mode

- Verifier Integration: choosing the right integration mode

Verifier Integration: choosing the right integration mode

When integrating with the Verifier, pick the mode based on **where your app runs**, **whether you can securely store keys**, and **whether a user is involved**.

Quick decision checklist

- **Browser SPA / Mobile app (without a backend that can securely store keys)?** ? Public + PKCE
- **Backend available and you can store keys securely?** ? Confidential
- **No user, purely service-to-service?** ? M2M (client_credentials)

1) Public client — Authorization Code Flow + PKCE

Choose this if:

- Your client runs in an environment that **cannot keep secrets** (SPA in the browser, mobile app, desktop app).
- You want the simplest setup (no client keys/JWKS to manage).

What it implies:

- Client authentication at `/oidc/token`: **none**
- Security for the code exchange: **PKCE** (`code_challenge` / `code_verifier`)
- You still use the normal user login flow: `/oidc/authorize` ? (wallet presentation) ? `/oidc/token`

Recommended when: **front-end apps** and “quick testing” scenarios.

---

2) Confidential client — Authorization Code Flow + JWT-based client authentication

Choose this if:

- You have a **backend** (or a secure server-side web app) that can **store keys securely** (vault/HSM).
- You want **strong client authentication** at the token endpoint.
- You plan to follow the stricter profile using **signed authorization requests** (`request_uri`).

What it implies:

- Client authentication at `/oidc/token`: **JWT client assertion** (e.g., `client_secret_jwt` in our registrations, typically ES256 + JWKS)
- If you use `request_uri` at `/oidc/authorize`, use a `did:key` **client_id** (signed request object profile).
- User login still applies: `/oidc/authorize` ? (wallet presentation) ? `/oidc/token`

Recommended when: **server-side apps** and partners wanting a “strongest” integration.

3) M2M (Machine-to-Machine) — Client Credentials

Choose this if:

- There is **no user interaction** (background jobs, service-to-service calls).
- You need tokens representing a **machine/service identity**, not a person.

What it implies:

- No `/oidc/authorize` and no redirects.
- Token request is directly to `/oidc/token` with `grant_type=client_credentials`.
- Requires a **machine credential** (LEARCredentialMachine) presented as part of the M2M flow.

Recommended when: **backend integrations** without a human login step.