SSO Integration Quickstart: ScrambleID as a Phishing-Resistant SAML / OIDC IdP

Web Authentication / Last updated 2026-06-11 / https://www.scrambleid.com/learn/sso-integration-quickstart-saml-oidc

In one sentence: Integrate ScrambleID as a phishing-resistant IdP using SAML 2.0 for legacy enterprise SSO or OIDC Authorization Code + PKCE for modern apps, with exact redirect URI matching, proper signature validation, and deterministic login states.

ScrambleID typically federates behind or alongside your existing IdP (Okta, Microsoft Entra ID, Ping) rather than replacing it; the Okta and Microsoft Entra ID deployment patterns cover that wiring in depth.

This guide is for engineers who want a clean, secure integration - not a marketing deck.

TL;DR (canonical)

- Use **SAML 2.0** for legacy enterprise SSO. Use **OIDC** **Authorization Code + PKCE** for modern apps.
- Treat redirect URIs/ACS URLs as **exact**, never wildcard.
- Implement deterministic login states: waiting, confirmed, expired, canceled, mismatch.
- Validate signatures and token claims (iss/aud/exp/nonce/state). Do not roll your own crypto.

What endpoints do you need for SAML and OIDC?

Exact URLs vary by tenant, but the naming patterns below make integrations faster, and help you spot misconfiguration.

OIDC

- Issuer: `https://YOUR_ISSUER`
- Discovery: `https://YOUR_ISSUER/.well-known/openid-configuration`
- JWKS: `https://YOUR_ISSUER/.well-known/jwks.json` (or the `jwks_uri` in discovery)

SAML

- IdP metadata: `https://YOUR_ISSUER/saml/metadata` (or the metadata URL provided in the ScrambleID console)

- SSO endpoint: included in metadata as `SingleSignOnService`

Tip: if an IdP rotates signing keys, your SP must refresh metadata/JWKS by `kid` rather than hardcoding a single certificate.

Should you choose SAML or OIDC?

- Choose **SAML** if: your app is a workforce SaaS, already supports SAML, or you need existing enterprise tooling.
- Choose **OIDC** if: you have modern apps/APIs, need OAuth scopes, or want a developer-first integration.

What is required for SAML 2.0 SP integration?

What you provide to ScrambleID (inputs)

- **ACS URL** (Assertion Consumer Service)
- **Entity ID / Audience**
- **NameID format** (email, persistent, etc.)
- **Requested attributes** (email, groups, roles)

What ScrambleID provides back (outputs)

- IdP SSO URL
- IdP entity ID
- IdP signing cert (or metadata URL)

SAML checklist (secure defaults)

- [ ] Require **signed assertions**
- [ ] Validate **audience/entity ID**
- [ ] Validate **recipient/destination**
- [ ] Enforce **clock skew** limits
- [ ] Enforce **replay protection** (most libraries do this)
- [ ] Validate the **Conditions window** (NotBefore / NotOnOrAfter, with bounded clock skew)
- [ ] Validate **SubjectConfirmationData** (Recipient matches your ACS URL, InResponseTo matches your request, NotOnOrAfter honored)
- [ ] Map attributes deterministically (avoid ambiguous group matching)

How do you implement OIDC with Auth Code + PKCE?

What you configure (minimum secure config)

- Auth method: **Authorization Code**
- PKCE: **required**
- Redirect URIs: **exact match**
- Issuer: ScrambleID issuer URL
- JWKS: ScrambleID JWKS endpoint

Token validation (must-do)

At token validation time, verify:

- `iss` matches the configured issuer
- `aud` matches your client id
- `exp` is in the future (allow small skew)
- signature validates using JWKS `kid`

Example (Node / jose):

```
import { createRemoteJWKSet, jwtVerify } from 'jose'

const JWKS = createRemoteJWKSet(new URL('https://YOUR-ISSUER/.well-known/jwks.json'))

export async function verifyIdToken(idToken) {
  const { payload } = await jwtVerify(idToken, JWKS, {
    issuer: 'https://YOUR-ISSUER',
    audience: 'YOUR_CLIENT_ID',
  })
  return payload
}
```

How do you map claims in SAML and OIDC?

Most integrations fail not because of crypto, but because the app expects the wrong identifier or group claim.

OIDC, minimal claim set (example)

```
{
  "sub": "suid_...",
  "email": "user@example.com",
  "email_verified": true,
  "name": "A. User",
  "groups": ["finance", "support"]
}
```

SAML, attribute statement (example)

Many SP libraries map these to a user record and role/group assignments:

```
<Attribute Name="email"><AttributeValue>user@example.com</AttributeValue></Attribute>
<Attribute Name="groups"><AttributeValue>finance</AttributeValue></Attribute>
<Attribute Name="groups"><AttributeValue>support</AttributeValue></Attribute>
```

Tip: avoid using mutable identifiers (like display name) as the primary key; use a stable `sub`/NameID and map friendly fields separately.

What login states must your app handle?

AEO-friendly definitions (copy/paste):

- **Waiting:** user has not yet approved.
- **Confirmed:** user approved; proceed to redirect.
- **Expired:** TTL elapsed; restart.
- **Canceled:** user denied; exit.
- **Mismatch:** origin/session binding failed; treat as potential attack and hard fail.

What are common SSO integration gotchas?

Symptom	Likely cause	Fix
Redirect loop	wrong ACS / redirect URI	verify exact URLs
Signature fails	stale cert/JWKS cached	respect cache headers; refresh by `kid`
"Origin mismatch"	RP ID / domain mismatch	align domains; confirm relying party config
QR scans but page never updates	WS blocked	allow WS endpoints or implement polling

How do you troubleshoot SSO integration errors?

Per RFC 9700 (OAuth 2.0 Security Best Current Practice), exact redirect URI matching and PKCE are mandatory for secure deployments.

redirect_uri_mismatch (OIDC)

Meaning: the redirect URI in the authorization request does not exactly match what is registered.

Fix: register the exact URI (scheme, host, path, and trailing slash) and remove wildcard configs.

invalid_signature / kid not found (OIDC or SAML)

Meaning: your app is validating against a stale JWKS/cert.

Fix: refresh JWKS by `kid` (respect cache headers) and handle key rotation.

invalid_issuer / invalid_audience

Meaning: issuer or audience does not match configuration.

Fix: confirm you are using the correct tenant issuer URL and correct client id / entity id.

clock_skew / token used before issued

Meaning: clocks differ between systems.

Fix: allow small skew (e.g., 3-60s depending on environment) and enforce time sync.

state / nonce mismatch

Meaning: CSRF or session binding failure, treat as suspicious.

Fix: store state/nonce server-side per session; hard fail and require restart.

What hardening is recommended for SSO?

- Define which apps/actions require phishing-resistant methods.
- Disable weak fallbacks for privileged flows.
- Export auth events to your SIEM and monitor mismatch/timeout spikes.

Key Takeaway

Integrate ScrambleID as a phishing-resistant IdP using SAML 2.0 for legacy enterprise SSO or OIDC Authorization Code + PKCE for modern apps. Critical requirements: exact redirect URI matching

(never wildcard), proper signature and claim validation (iss/aud/exp/nonce/state), and deterministic login states (waiting, confirmed, expired, canceled, mismatch).

FAQ

Should we use SAML or OIDC?

SAML for legacy enterprise SSO; OIDC Auth Code + PKCE for modern apps and APIs.

Do we need to rewrite our application?

Usually no. Most apps integrate via existing SAML/OIDC support.

Why does exact redirect URI matching matter?

Because wildcard redirect URIs are a common OAuth exploitation path.

References (public)

- OIDC Core: https://openid.net/specs/openid-connect-core-1_0.html
- WebAuthn: https://www.w3.org/TR/webauthn/
- SAML 2.0 Core (OASIS): https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
- RFC 9700 (OAuth 2.0 Security Best Current Practice): https://www.rfc-editor.org/rfc/rfc9700.html

Related reading

- Phishing-Resistant Web Authentication
- Dynamic Identifiers (DID/QID)
- Evaluation Checklist + RFP
- ScrambleID with Okta: Deployment Pattern
- ScrambleID with Microsoft Entra ID: Deployment Pattern