

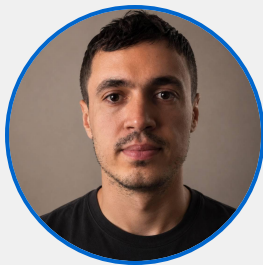


Keycloak for Cross-Domain Access

Secure Identity Propagation between Apps and Agents

KeyConf26 - 08 Oct 2026 - Prague

whoami



Giuseppe Graziano

Senior Software Engineer @ IBM

Keycloak Core Protocols Team



graziang



ggrazian@ibm.com

What this talk covers

01

The problem

why cross-domain identity is still solved badly

02

The pattern

identity chaining: RFC 8693 in A, RFC 7523 in B

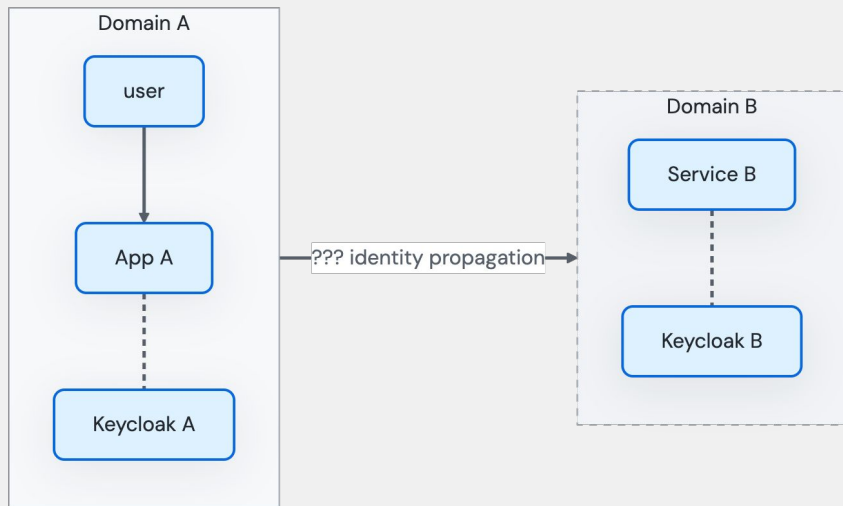
03

Keycloak

what ships today, how to configure it, and a demo

The scenario

A single user request crosses **independent trust domains**.



DOMAIN ISOLATION

Each domain has **its own** authorization server, **its own** user store, **its own** policies.

A TRUST BOUNDARY IS TWO PEOPLE

Two domains who do not report to each other, one decides what may leave, the other what may enter.

THE CROSSING IS NOT NEW

SaaS-to-SaaS integrations, microservices split across business units, overnight batch jobs.

Agents

An agent is a client in domain A



One turn: check an order, process a payment, arrange shipping. **Three trust domains, one authentication, no browser in any of the crossings.**

YOU CANNOT PLAN FOR IT

An agent chooses its next tool **at runtime**, per turn.
Trust and tool permissions still need configuration. Per-call policy decides what the selected tool may do.

THERE IS NOBODY TO ASK

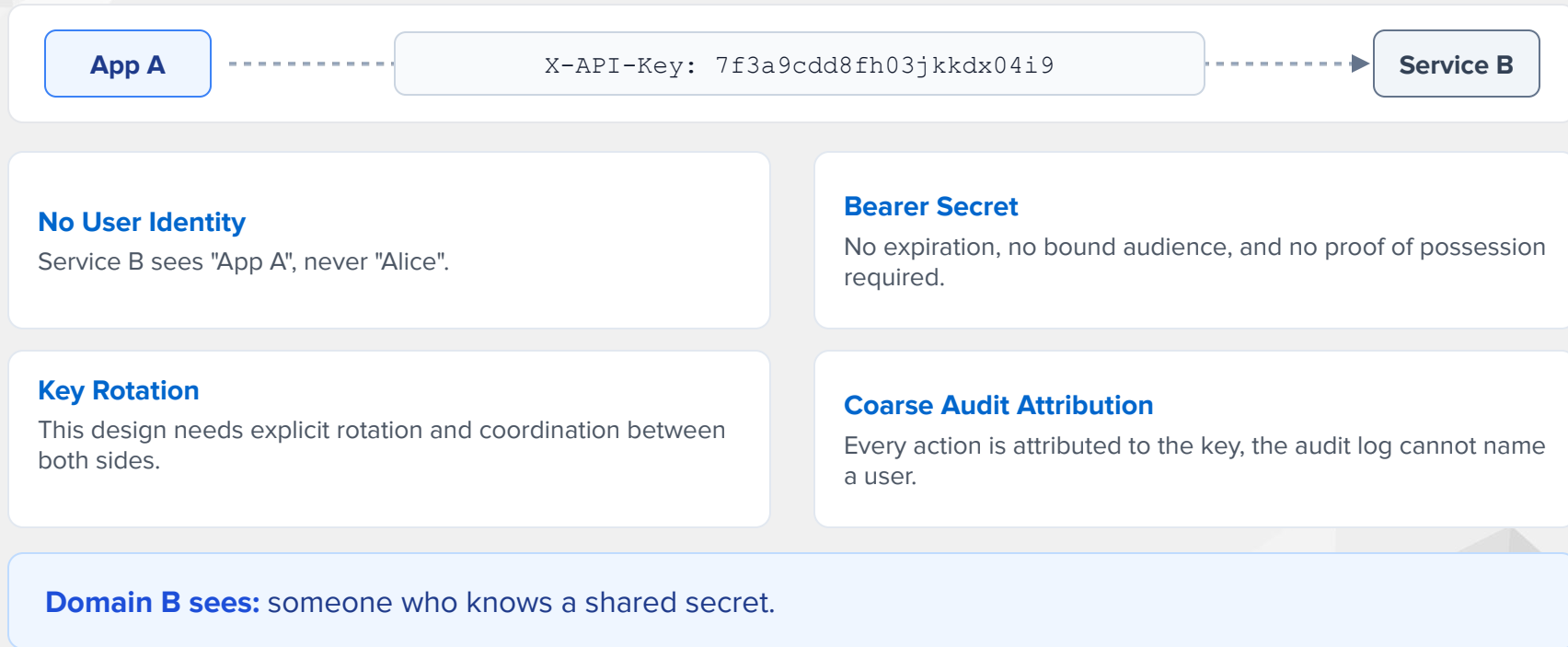
A user may be absent when the agent runs.

Existing grants can allow unattended access. New consent or step-up may still require interaction.

It does not change the problem. It changes how often you have it. Agents increase the frequency of authorization decisions.

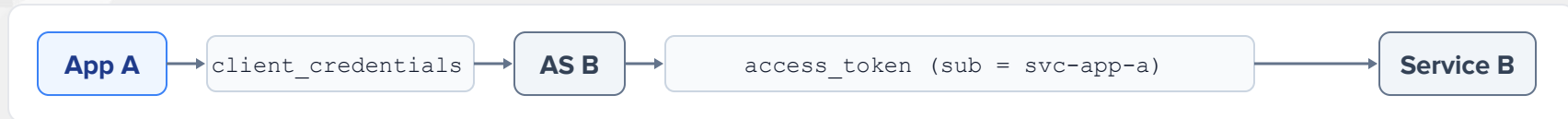
What people do today (1/4)

Static API keys



What people do today (2/4)

Over-privileged service account



Standard OAuth

Looks correct because is standard, but the user is gone.

Service Account Identity

sub is the service account, not Alice.

Over-Broad Scope

B may grant aggregate permissions across all users.

High Vulnerability Impact

A bug in App A reaches everything B exposed to that account.

Domain B sees: a service account with broad permissions, rather than Alice's local permissions.

What people do today (3/4)

Token pass-through



Audience Mismatch

A token intended for A must be rejected by B. Accepting it without audience validation breaks isolation.

Confused deputy Risk

If B ignores the audience, it can accept a token intended for another service.

Replay Risk

A stolen bearer token can be replayed at its allowed recipients until expiry or effective revocation.

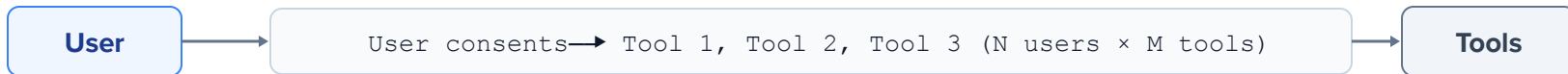
Lifetime & Revocation

A's token lifetime becomes B's problem and B cannot revoke it.

Domain B sees: a user, in a token it was never meant to receive.

What people do today (4/4)

Per-app OAuth consent: one consent per user, per tool



This one is not broken. It is correct OAuth.

Redirect, consent, authorization code, refresh token. By the book.

Initial consent needs a present user

Later calls can run unattended using a valid refresh token.

Separate credentials per tool

Each tool may issue a refresh token. Enterprise visibility and control depend on the integration.

SSO logs may not show the API grant

It logs "Alice signed in to App". Not "an AI agent can now act as Alice in App, indefinitely".

Separate grants need coordinated policy, revocation and inventory.

What we actually want

01

Domain B issues its own token

B's lifetimes, B's scopes, B's revocation.

02

User preserved across the hop

sub at B identifies Alice, not a service.

03

One intended recipient

Audience-restricted, single-use where possible.

04

Trust is administered

A configured relationship, not a secret someone put in a vault.

05

No new login

The user is not present to consent again; this may run at 3 a.m.

06

It is a standard

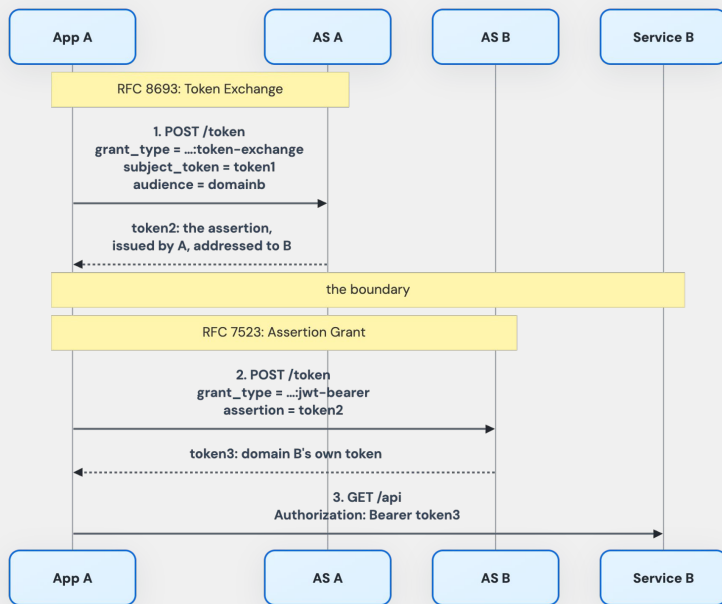
Because domain B is not run by you.

Identity chaining

RFC 8693 + RFC 7523

Identity chaining

Two calls to two token endpoints, one per domain. Nothing else changes.



token1 never leaves A

token2 **the assertion** carries A's identity statement to AS B.

token3 is minted by B, under B's policies.

ISOLATION The ASs keep private signing keys separate. B trusts A's public keys and maps the subject locally.

Step 1: Token Exchange in domain A (RFC 8693)

In one sentence: *hand the authorization server a token you already have, and ask it for a different one.*

```
POST /realms/domaina/protocol/openid-connect/token
grant_type =
urn:ietf:params:oauth:grant-type:token-exchange
subject_token = <token1>
└ the token you already have
subject_token_type =
urn:ietf:params:oauth:token-type:access_token
└ REQUIRED
audience = http://.../realms/domainb
└ names the AS in domain B (audience/resource REQUIRED)
scope = access-domainb
└ scopes to carry into the grant
```

Re-authorizes Request

The AS **re-authorizes** the request. This is not a re-signing service.

Policy-Controlled Output

A selects the audience, fewer scopes, shorter life.

Names the User

Still names **the user**, not caller, sub is Alice; azp records client.

Runs entirely inside **domain A**. Domain B is not involved.

The assertion: what the two standards pass to each other

```
{
  "iss": "https://.../domaina", // who is saying this
  "sub": "9c1e42a7-...", // who it is about
  "aud": "https://.../domainb", // who it is for
  "exp": 1789000300, // until when
  "jti": "b0e3...", // and only once
  "scope": "access-domainb" // which scopes
}
```

💡 **aud** is the one claim the identity chaining makes **mandatory**
└─ MUST identify the requested authorization server in domain B

Question 1

Who is allowed to produce this?

Domain A answers this · **RFC 8693**

Question 2

Who is allowed to accept it?

Domain B answers this · **RFC 7521 / 7523**

Step 2: JWT Authorization Grant in domain B (RFC 7523)

In one sentence: *bring an assertion signed by an issuer I trust, and I issue you a token.*

```
POST /realms/domainb/protocol/openid-connect/token

grant_type = urn:ietf:params:oauth:grant-type:jwt-bearer
assertion = <token2>
└ signed JWT from domain A
scope = payments:write
```

iss + signature

Do I know this issuer, and is this really from them?

aud

MUST identify AS B, issuer identifier or token endpoint

exp / jti

Unexpired? Reject reuse when replay protection is enabled.

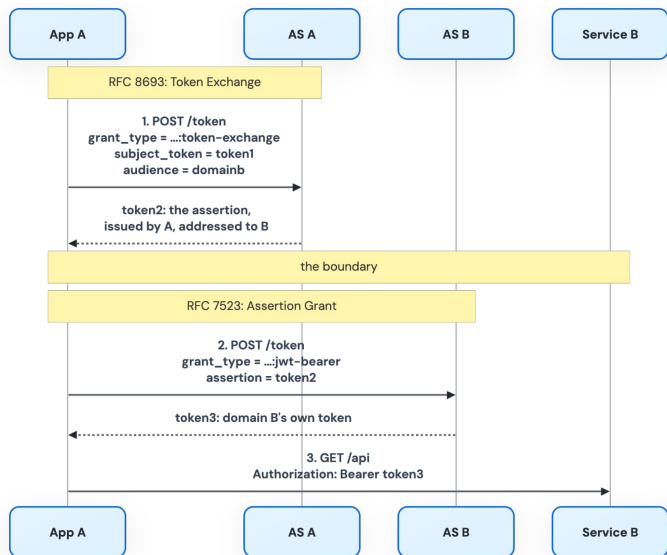
sub / scope

Which **local** user is this, and what does he get **here**? B MUST deny the request if it cannot identify the subject

💡 B is **not re-authenticating a user**. It is deciding whether to accept an assertion about one, and applies local authorization without prompting the user again.

Runs entirely inside **domain B**. Domain A is not involved.

Identity chaining: the complete flow



1. RFC 8693 in A

Alice's local token becomes an **assertion addressed to B**

2. RFC 7523 in B

B validates the assertion and issues **its own** token for Alice

3. API call

Service B sees a token **from its own issuer**

A sends an assertion for AS B. B issues the access token used at its API.

Two authorization servers that share no keys, and sub at B still means the same person.

The specification

OAuth Identity and Authorization Chaining Across Domains

DRAFT

`draft-ietf-oauth-identity-chaining-17`

WORKING GROUP

IETF OAuth WG adopted

IESG STATE

RFC EDITOR QUEUE

INTENDED STATUS

Standards Track (Proposed Standard)

REVISION

17, 19 July 2026

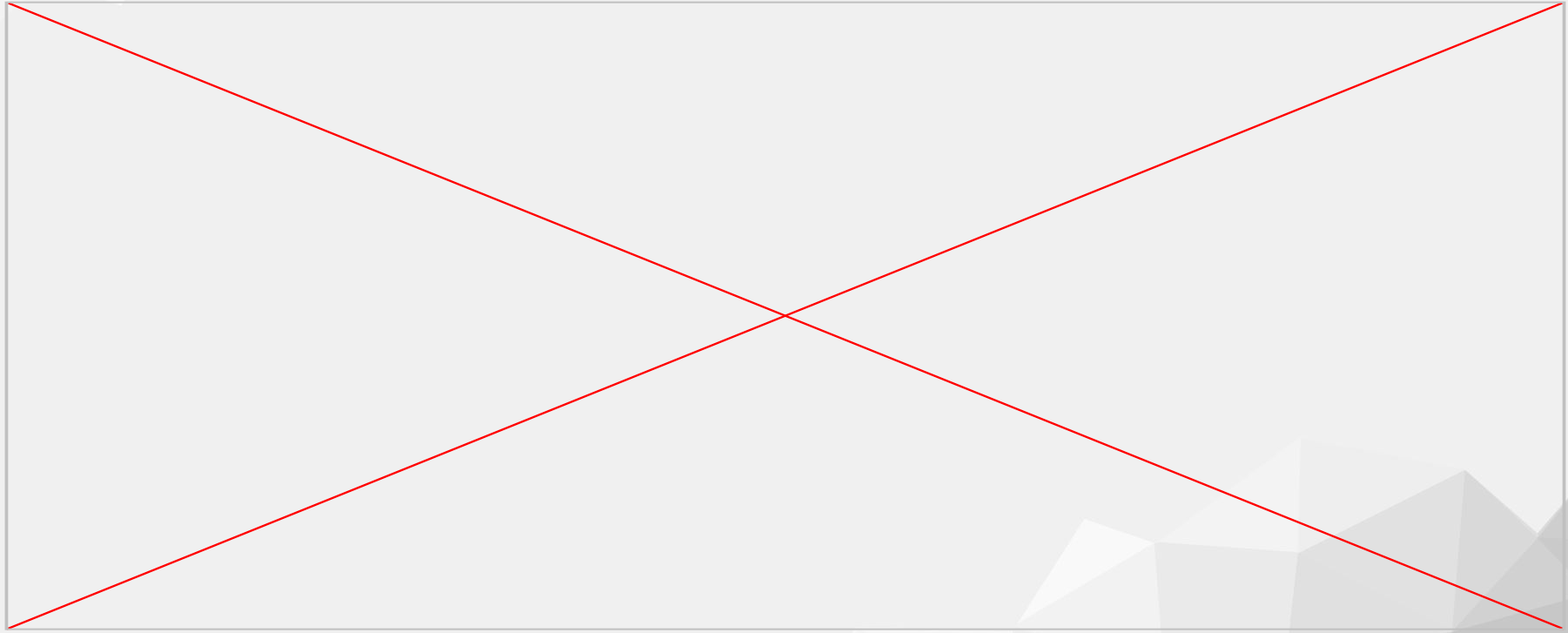
expires 20 January 2027

Doing it in Keycloak

What Keycloak supports today

Capability	Status
Standard Token Exchange (RFC 8693)	✓ Supported
JWT Authorization Grant (RFC 7521 / 7523)	✓ Supported
Identity Chaining (draft)	✓ Supported, by composition
ID-JAG / Cross-App Access	🔧 Experimental

DEMO



Where this is going: ID-JAG /Cross-App Access (XAA)

[draft-ietf-oauth-identity-assertion-authz-grant-04](#) · IETF OAuth WG · Standards Track

A profile of identity chaining, for apps **already configured for SSO against a common IdP**. Reuses existing trust: "XAA extends the IdP's role from single sign-on to brokering cross-domain API access." The IdP brokers, and **B still issues its own tokens**, "applying its own local policy."

Subject Token is the Identity Assertion

OIDC ID Token or **SAML 2.0 assertion**. Optionally a refresh token to save an SSO round trip.

A well defined assertion: the ID-JAG

Type and claims

typ: `oauth-id-jag+jwt`, registered media type, prevents bearer token accidents.
REQUIRED: `iss sub aud client_id jti exp iat` (base chaining required only `aud`).

Security Controls

`aud` is single-valued: "prevents audience injection attacks."
`client_id` names the agent allowed to redeem it.

Subject Resolution

Resolves via `iss+sub(+tenant)`, `aud_sub`, `email`, or `sub_id` (RFC 9493).
JIT provisioning permitted for new users.

Where it is used

MCP Enterprise-Managed Authorization (SEP-990) is **stable**, running on ID-JAG.

IN KEYCLOAK

Experimental:

`--features=identity-assertion-jwt`.

Takeaways

1. One boundary, two decisions

What may leave A, what may enter B. Two administrators who do not report to each other. That is why it takes two standards, one per side.

2. Identity chaining is RFC 8693 in A, then RFC 7523 in B

A sends the assertion to AS B. Keycloak ships both!

3. ID-JAG is the same pattern with the middle artifact specified

Use it where the other side already trusts your IdP for SSO

Agents did not create this problem. They made it hourly.

Each resource domain issues its own access token.

Links & Thank You

Keycloak Documentation

- keycloak.org/securing-apps/oauth-identity-authorization-chaining-across-domains
- keycloak.org/securing-apps/jwt-authorization-grant
- keycloak.org/securing-apps/token-exchange
- keycloak.org/securing-apps/identity-assertion-jwt-authorization-grant

The Drafts

- <https://datatracker.ietf.org/doc/draft-ietf-oauth-identity-chaining>
- [draft-ietf-oauth-identity-assertion-authz-grant](https://datatracker.ietf.org/doc/draft-ietf-oauth-identity-assertion-authz-grant)

MCP Enterprise-Managed Authorization

- <https://modelcontextprotocol.io/extensions/auth/enterprise-managed-authorization>

Thank you!

Questions?

Giuseppe Graziano

Email: ggrazian@ibm.com