Problem: FedCM does a lot more than the minimum required to enable federated login, including offering a sense of "identity"

e.g. custom service APIs, passing new tokens around, new browser UI paradigms, weird Fetch operational modes, .well-known files, site-controlled content in browser UI, token as a concept independent of cookies, etc.

Question: What is a minimum version of a Credential that enables federated login while "stepping out of the way"?

Clarification: A Credential is a Javascript object that *may (if defined)*:

Have some associated data
Be "created" by javascript
Be "stored" in the browser's credential store
Be "collected" from that credential store by some pages
Be "discovered" from somewhere (not the store)
Have custom arguments for all of those functions.
Be bound to a particular origin for use.

Straw-man solution: CrossOriginStorageAccessCredential:

Semantically: Holding this object allows you to use cross-origin cookies for the IDP. Each Credential is bound to an IDP-RP pair.

How it would work: if one is held, it can be Stored, and if it's been Stored it can be Collected. Creation doesn't happen directly on the RP, only through Discovery.

Discovery is complicated because (1) the IDP needs to be able to opt-in per RP, (2) it may require a redirect away and back mid-request, and (3) this is the point at which we need to show UI to break down the privacy barrier.

Controversial part:
Discovery can only be single-IDP. We give UI that looks like "Try to log in with your account at "foo.com". It must redirect the current navigable to the IDP's origin. Once there, the IDP can see what origin wants to create a credential on them, can Create and Store a Credential. Also during this process the RP should allow silent access to the Credential.

First improvement:
This makes creating a credential only happen amid a redirect-flow. We can improve this by allowing an IDP to pre-register what RPs are acceptable (a list of < M origins, or an endpoint that responds to CORS Origin headers with ACAO). This lets a pre-registered IDP skip the redirect flow.

Magic moment:
Stored COSACs, stored passwords, WebAuthN, and try another site for a COSAC can all be presented with each other.

Thoughts?

Intermediary problem: RP -> A -> IDP -> A -> RP
Who needs cookies on RP?
Logout is the case where we may need some, otherwise it is probably first-party cookies - optionally symmetric, by control of the RP, maybe enable front channel logout
Whose name appears in the UI?
Not a problem then, but ideally UX shows IDP

Chrome: Don't want to fracture from FedCM?

Ben: Line of identity

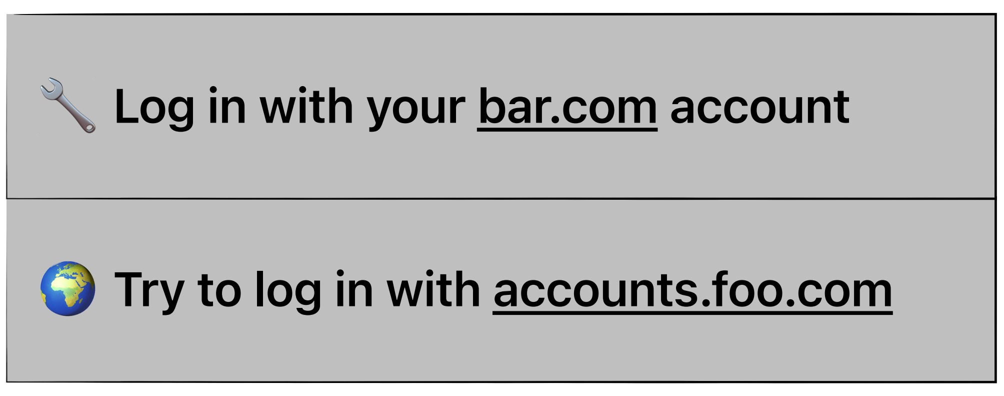
Chrome: Enable onramp (graceful upgrade) to Identity-rich features from this? 👍Achim

NPM: Friendly UI when compared to Storage Access, will this induce tracker use?

Maybe? But seems fine because the UI is login-specific

Open question:
Where do the pre-registered IDPs go?

Proposed doorhanger


🔧 **Log in with your bar.com account**

🌍 **Try to log in with accounts.foo.com**

Already linked IDP

Not yet linked IDP,
will cause redirect flow, unless IDP is registered

Collusion vs non collusion privacy models - this enables non-colluding privacy

Do we need all intermediaries to pop up in a UI? Is that reasonable to expect intermediaries to tolerate?

Johann: Reiterate that additional API surface is a negative. Chrome working on a similar idea, but more integrated into FedCM. Let's break this down into smaller proposals that can change FedCM.

Ben: Action Item: UI and code examples for next call, and provide a whiteboard export