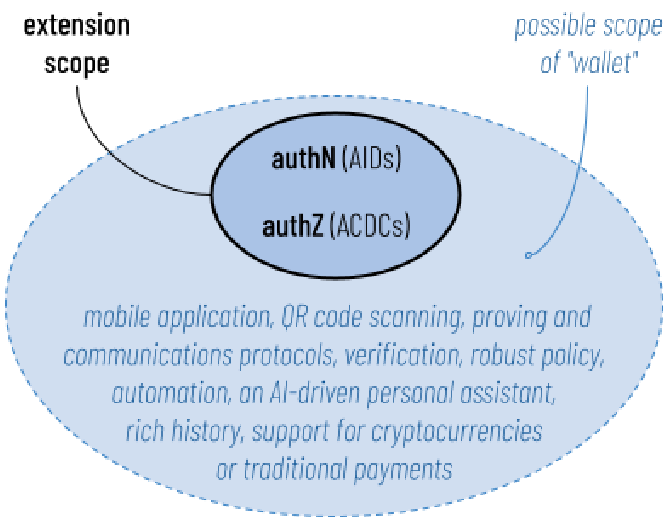
Bounty PB311.1:
Signify+Keria as authX browser extension

This document constitutes a rough specification for some open source software and/or features that Provenant wants. We imagine paying someone a bounty to build what's specified here, as described below.

Overview

Create and publish a browser extension that uses Signify+Keria to do *simple* management of AIDs and their associated keys and ACDCs — just enough to establish trust with websites. (Alternatively, Signify+Keria features could be added into an existing open-source password manager.)

This "authX" feature set is at the heart of what is called a "wallet" in our industry. However, in the minds of many people, "wallet" is a much bigger term.


extension scope
possible scope of "wallet"
authN (AIDs)
authZ (ACDCs)
mobile application, QR code scanning, proving and communications protocols, verification, robust policy, automation, an AI-driven personal assistant, rich history, support for cryptocurrencies or traditional payments

The codebase for this extension is NOT intended to grow into something with all of those features. It should be laser-focused on just authX for websites.

However, the beauty of the Signify+Keria architecture is that the secrets managed by this core feature set are portable to new contexts (new devices, new browsers, new applications capable of running Signify). This means that the extension itself doesn't need to provide those fancy features, ever. A user could easily use the extension to "log in" to a website or mobile application that DOES provide all of the wallet features imagined by the community. (Consider the difference between the popup window of the 1Password tool with the actual featureset available if you browse to yourwallet.1password.com.)

How to Apply

Before 24 Nov, send a proposal to bounty@provenant.net, referencing bounty PB311.1. Please specify who from your team will work on the project, what their qualifications are, and any technical approaches you intend to recommend for the requirements. One page is plenty.

Target Timeframe

Available with at least beta quality by approximately 1 Feb 2024.

Proposed Bounty Amount

30,000 USD. We believe this is an appropriate bounty for about 2 to 3 person-months of work. If applicants believe this is inappropriate, let's talk.

Delivery

Two artifacts:

A. *Repo on Github*. Open source (Apache 2 or equivalent) repo in the WebOfTrust organization (or a similar location to be negotiated). Repo should have Github Issues and should demonstrate beta (~version 0.9) quality by virtue of the issues that are closed and open.

B. *Published software*. Available in the Chrome extension and Firefox extension catalogs.

Proposed Milestones

I. *Award* (target = end of November): Winner of bounty (hereafter, "the team") is chosen by Provenant and additional milestones are scheduled. The team becomes eligible for an initial 20% of the bounty as an advanced payment.

II. *Design Acceptance*: the team presents design as slide deck in an interactive meeting, open to the community (or possibly *on* a community meeting), with Q&A. The team is eligible for another 20% of the bounty.

III. *First Look*: In an interactive presentation, the team demos at least 50% of the features (from the Requirements list below), and becomes eligible for another 20% of the bounty.

IV. *Second Look*: In an interactive presentation, the team demos all of the features from the Requirements list below).

V. *Release Candidate*: In an interactive presentation, the team demos compliance with all of the Requirements and, upon acceptance, becomes eligible for the remaining 40% of the bounty.

Requirements

1. *General password manager behaviors*. The browser extension SHOULD behave approximately like the familiar software category of "password manager", of which 1Password and BitWarden are notable examples. This means that it can be pinned to the browser's URL bar, and it normally runs invisibly, monitoring browsing activity. If a user activates the extension, it displays a UI that allows the user to select an AID (and possibly associated credentials) that can be used to authenticate (and possibly authorize) with the active web site. Associations between websites and AIDs are remembered by the extension, so this kind of "login" process can be performed automatically thereafter. A user can create a new AID for a website. The extension is unlocked by a Signify passcode, and remains unlocked until it's been idle for 10 minutes or longer — whereupon it locks again.

2. *Password manager security*. The browser extension will be operating in a highly privileged mode, analogous to the one used by password managers. Such extensions typically trigger a warning from the browser, saying that they can see all browsing behavior and modify the DOM of any web page. This extension is likely to be similar, although it seems possible that it may not need to be able to see any secrets or passwords persisted by the browser. The extension MUST use appropriate cybersecurity techniques (as carefully copied from existing password manager extensions) to prevent malicious javascript in web pages from exfiltrating secrets, etc.

   Because it may be challenging to get the security right, contributing the Signify+Keria authX features to either BitWarden or 1Password (or maybe MetaMask) would also be an acceptable way to guarantee correct security posture. (In such a case, deployment requirements would change, since these password managers have their own repos and are already in the browser extension catalogs.)

3. *Automated login or manual authentication*. The extension MUST support a mode whereby it can earn the trust of a website that requests "HTTP WOT authX". This would be a new HTTP authentication scheme, a cousin of Basic Auth that conforms to RFC 7235. The scheme supports pure authN, but (despite the the terminology of its governing standards) also ACDC-based authZ. Specifying this scheme is a community task out of

scope for this extension, but is generally described in Appendix 2 for reference. It's expected that little effort will be required to support the scheme; it's extremely close to what Signify and Keria already do.

If the extension was previously configured by the user to automatically login to this website, it uses this AID to sign requests to the website from then on. If the extension has not been configured for this website, it informs the user that the website is asking for authentication, allows the user to select an appropriate AID and/or credentials, and remembers the user's decision for subsequent browsing on the site.

4. *Configuration with a vendor (Keria provider)*. Upon install, the extension MUST allow a person to provide a configuration URL for a vendor that will provide the Keria agent for the Signify instance that runs in the extension. Without a vendor who hosts Keria, the extension is not functional.

   This URL is NOT the URL of the agent, but rather a URL that MUST be browsable by a person. It should point to an onboarding web page hosted by the vendor, that walks the user through a configuration experience and that ultimately redirects the user to a URL that serves a JSON object with the following configuration parameters:

   - `agent_url`: where Signify can call the user's Keria agent.
   - `vendor_css`: URL where the vendor's CSS can rebrand the UI of the extension.
   - `strings_url`: A nullable URL that includes the token "lang=??". This URL returns a JSON object that contains localized strings for the extension; the extension will call this URL, substituting a lang code appropriate to the user's browser settings.
   - `support_url`: A URL where the user can get support from the vendor.
   - `docs_url`: A URL where the user can browse documentation about the extension as supported by that vendor.
   - `management_url`: A URL which allows the vendor to override the basic UI of the extension. It may be null. If it's null, the extension displays a default UI in a popup window, to help users to manage, create, and select identifiers and credentials to authenticate to a web site. If it's NOT null, the extension loads HTML from this URL to perform management tasks.

   (Note: need to discuss overall config strategy. There are cybersecurity implications if we get too fancy, but there are problems with overreach in the UI of this extension if we don't give vendors enough freedom.)

5. *Alias and identifier management*. If the extension has an active management URL, it doesn't do management of aliases and identifiers. But if it displays its default UI for management of aliases and identifiers, the UI MUST provide a login feature that allows a

user to select an AID by a human-friendly alias, and that optionally allows them to associate the AID with ACDCs. The extension SHOULD encourage the user to choose a new AID for each website.

6. *Credential management*. If the extension has an active management URL, it doesn't do management of credentials. But if it displays its default UI for credentials, the UI SHOULD group credentials by which alias/identifier they belong to, automatically bringing to the top any that pertain to the AID used with the current website, and any that would satisfy the credential demands of the current website.
7. *Technology*. The UI for the extension should be simple, small, and lean. It MUST use HTML5 and Javascript, and MAY use React. It MAY NOT use other UI frameworks like Angular or Flutter.

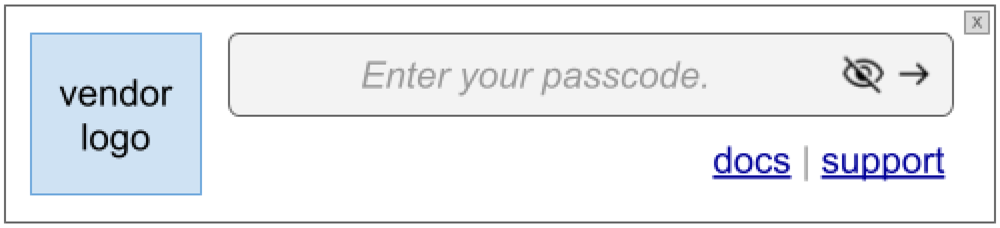
Non-requirements

7. Passcode rotation
8. Key rotation
9. Use of multisig AIDs with websites
10. Import or export
11. Backup

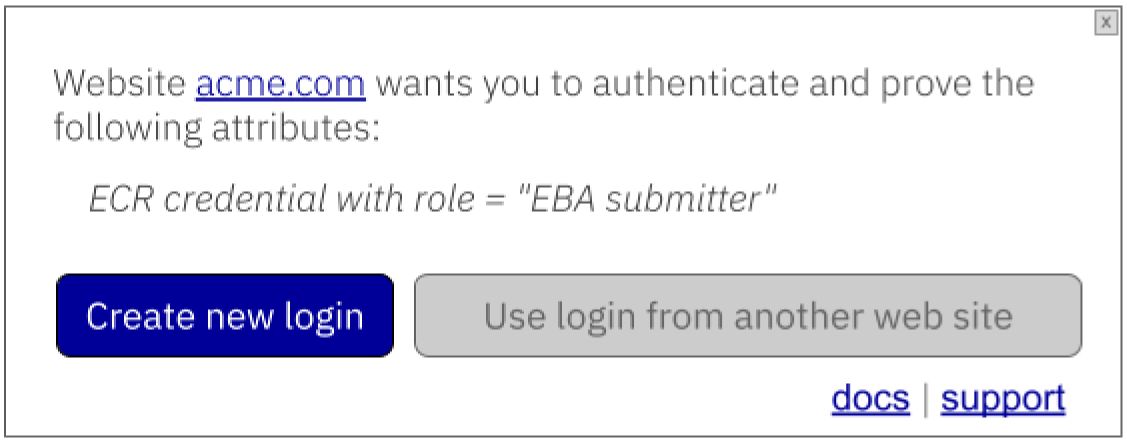
UI Concepts

The default management UI doesn't have to look like this or work exactly like this, but this is an example of a UI that would satisfy the requirements. Imagine each of these concepts is located in a popup window below the browser URL bar.

1. If the extension is locked because no passcode has been entered yet (or because it's been idle for too long), the extension prompts the user to unlock.



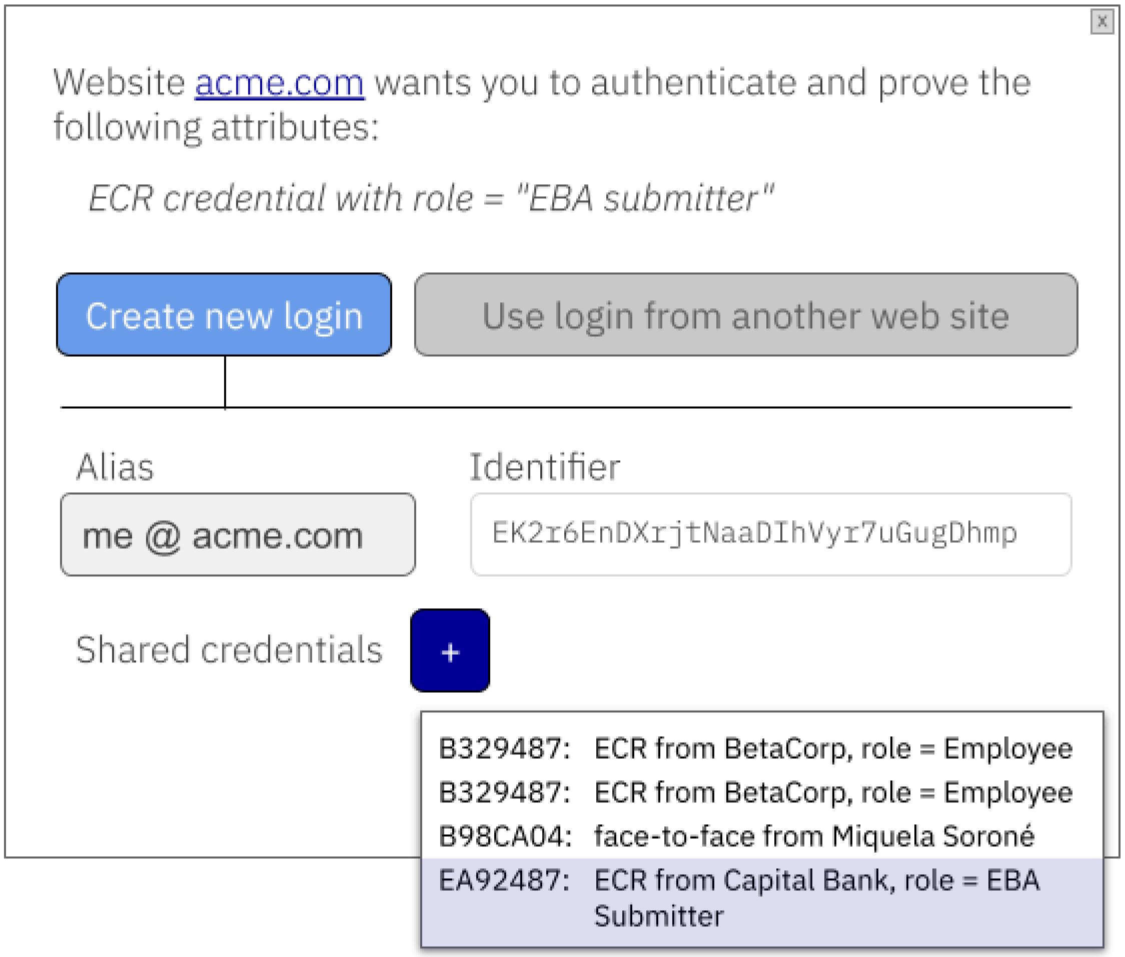
2. If a website communicates a WOT auth challenge (see Appendix), and the extension hasn't yet been configured to automatically log in, the extension explains what would satisfy the web site's trust requirements, and helps the user respond.



Website acme.com wants you to authenticate and prove the following attributes:

ECR credential with role = "EBA submitter"

3. If the user chooses to create a new login, the UI calls Signify to generate an AID and assign it an alias. The pairing of an AID with an alias and its associated website(s) is what this extension considers a "login". Optionally, the extension also allows credentials to be bound to the new alias. (If a credential was issued to an AID other than the one used with the website, the bespoke ACDC presented to the website must reference that credential and be signed by BOTH the new AID and the issuee's AID, to prove that the party behind this new AID controls both.)



Website acme.com wants you to authenticate and prove the following attributes:

ECR credential with role = "EBA submitter"

Alias	me @ acme.com
Identifier	EK2r6EnDXrjtNaaDIhVyr7uGugDhmp

B329487:	ECR from BetaCorp, role = Employee
B329487:	ECR from BetaCorp, role = Employee
B98CA04:	face-to-face from Miquela Soroné
EA92487:	ECR from Capital Bank, role = EBA Submitter

AIDs created by the extension SHOULD go into a "web" namespace by default, and SHOULD have the form "me @ website" (where "website" is the second-level segment of a domain — for nike.com, this is "nike"). Users SHOULD be able to override the alias, but SHOULD NOT have to in most cases.

Appendix: WOT authX HTTP Authentication Scheme

This HTTP authentication scheme conforms to RFC 7235 and works very much like HTTP Basic Auth (RFC 7617). Like all HTTP authentication schemes, it assumes that servers are properly authenticated in some other way. This assumption is false more than we would like, and it also fosters insidious power imbalances. However, it is the way most web sites are built today, and this mechanism opts to leave that characteristic unchanged, for ease of adoption. Thus, like Basic Auth, the scheme MUST only be used when a properly secured TLS session is active, providing modest confidentiality guarantees.

A user browses to a protected resource and receives a `401 Not Authorized` response that signals a trust boundary. The response is called an "auth challenge" if it includes a standard `Date` header plus a `WWW-Authenticate` header with a scheme token equal to `WOT`, and with additional parameters `realm` and `challenge`. The format of the header is:

```
WWW-Authenticate: WOT realm=<URL pattern of resources that will be unlocked>
   challenge=<SAID of schema>
```

The `realm` parameter is optional. If missing, the entire website is assumed to be the realm. If present, it works exactly the way it does in Basic Auth, including semantics defined in RFC 7617 section 2.2 for reusing credentials.

The `challenge` parameter identifies a credential schema that would satisfy the web site. If the website wants to see attributes from several issued credentials, or if the website wants to see attributes from less than a full credential, this is essentially the schema of a bespoke ACDC that contains what's required from one or more sources (which have their own schemas).

The client responds to this challenge with a new HTTP request that contains an `Authorization` header. The body of the request is whatever the client would otherwise transmit to pursue its goals; the header repeats the `WOT` scheme identifier and provides an ACDC (often, bespoke) plus a signature over the `Date` field from the challenge.

```
Authorization: WOT cred=<ACDC> sig=<sig over server's Date as unix epoch time>
```

The server then evaluates this request and grants access to the realm if the client's signature is valid for the given signature input, including any IPEX response.

Sessionless by default

Most HTTP authentication schemes assume sessions: a client is challenged once, and satisfies the server once, whereupon trust is encoded in a session cookie or similar, and authentication is not repeated.

However, this behavior is NOT the default for this scheme. Instead, the default assumption is that a server will challenge in every response (whether or not the response includes a `401 Not Authorized` status code) — and that a client will authenticate in every request. This achieves the ideals of zero-trust architecture and imposes no session-management assumptions on the server. It also means that clients that support the scheme MUST treat a server response as a challenge in this scheme if it has an appropriate `WWW-Authenticate` header, regardless of its status code.

If a server prefers the lesser security of sessions, it simply ceases to include the `WWW-Authenticate` header in its responses whenever it has established a session that satisfies its trust requirements. A client then has the option (but not the requirement) of ceasing to include the `Authorization` header in its requests.

Non-interactive (unchallenged) mode

As described above, the scheme requires a challenge and a response. This two-part behavior makes it interactive. This is helpful for discovering that a web server uses the scheme and has specific authentication requirements. However, if client software already knows that a web server supports the scheme, and if it knows what proof will satisfy the server, then the client can include the `Authorization` header without being challenged first. This eliminates all interactivity.

However, schemes like this are vulnerable to replay attacks unless there is a nonce, and the nonce should normally come from the server (`Date` header). If there is no challenge, this has not occurred. Thus, in non-interactive mode, the scheme allows *the client's* `Date` header *on the request* to function as the nonce value. In such cases, the server only accepts the `Date` as a legitimate source of entropy for the scheme if it agrees that the delta between the client's `Date` and the current server time is close enough to eliminate enough risk of replay. It is up to each server to decide what "close enough" means, but allowing for a few seconds of discrepancy in system times between the two sides, and allowing a second or two for a request to reach a server, it is probably reasonable to require deltas of less than 30 seconds. The security SHOULD be further enhanced by requiring client `Date` values to increase with each request.