Day 2 / Session 7 / Space A

Session Title: OpenID4VP and OpendID4VP over Browser API

Session Convener: Joseph Heenan, Kristina and Torsten
Session Notes Taker(s): Jin Wen; Albert Wu (screen shots)

Tags / links to resources / technology discussed, related to this session:

- Digital Credentials API explainer
- Digital Credentials (This document specifies an API to enable user agents to mediate access to, and presentation of, digital credentials such as a driver's license, government-issued identification card, and/or other types of digital credential. The API builds on Credential Management Level 1 as a means by which to request a digital credential from a user agent or underlying platform.)
- Digital Credentials API Web Platform and App Platform Layering / Interactions

Slides: OID4VC_20240410_OSW.pptx-2.pdf

PR to review in OIDF DCP WG: https://github.com/openid/OpenID4VP/pull/155

work in W3C:
https://github.com/WICG/digital-identities/blob/main/resources/DigitalCredentialsAPI-Layering-v20240301.pdf

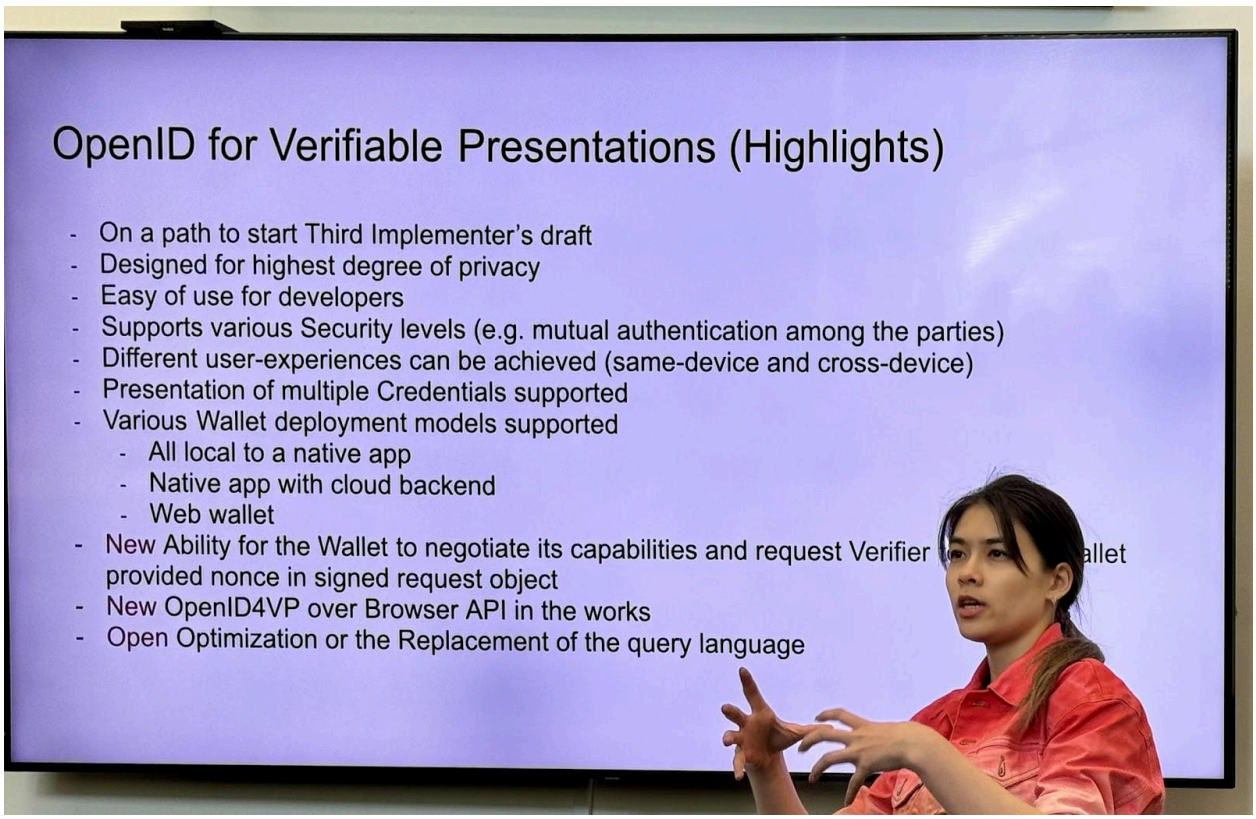
OpenID for Verifiable Presentations (Highlights)

Same Device

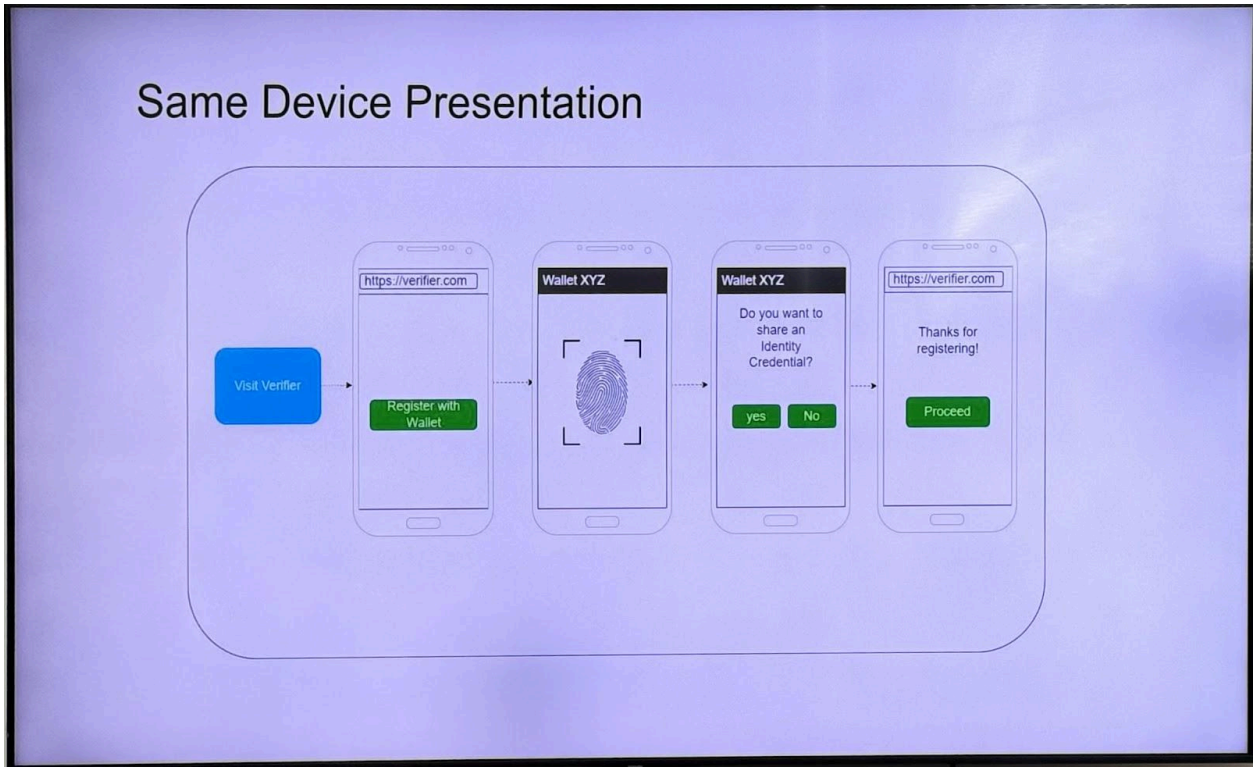
Discussion notes, key understandings, outstanding questions, observations, and, if appropriate to this discussion: action items, next steps:

OpenID for Verifiable Presentations (Highlights)

- On a path to start Third Implementer's draft
- Designed for highest degree of privacy
- Easy of use for developers
- Supports various Security levels (e.g. mutual authentication among the parties)
  Different user-experiences can be achieved (same-device and cross-device)
  Presentation of multiple Credentials supported
  Various Wallet deployment models supported
    - All local to a native app
    - Native app with cloud backend
    - Web wallet
- New Ability for the Wallet to negotiate its capabilities and request Verifier to include wallet provided nonce in signed request object
- New OpenID4VP over Browser API in the works
- Open Optimization or the Replacement of the query language



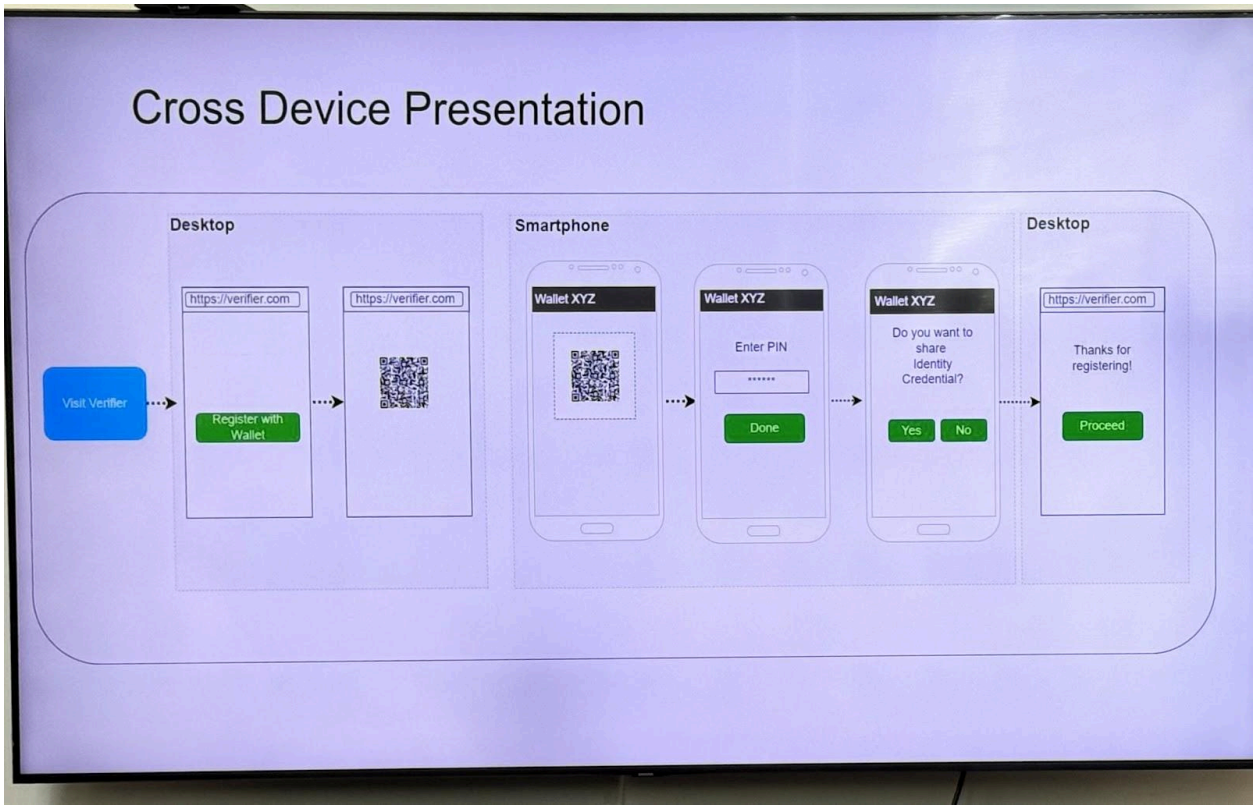
Same Device Presentation



Cross Device Presentation

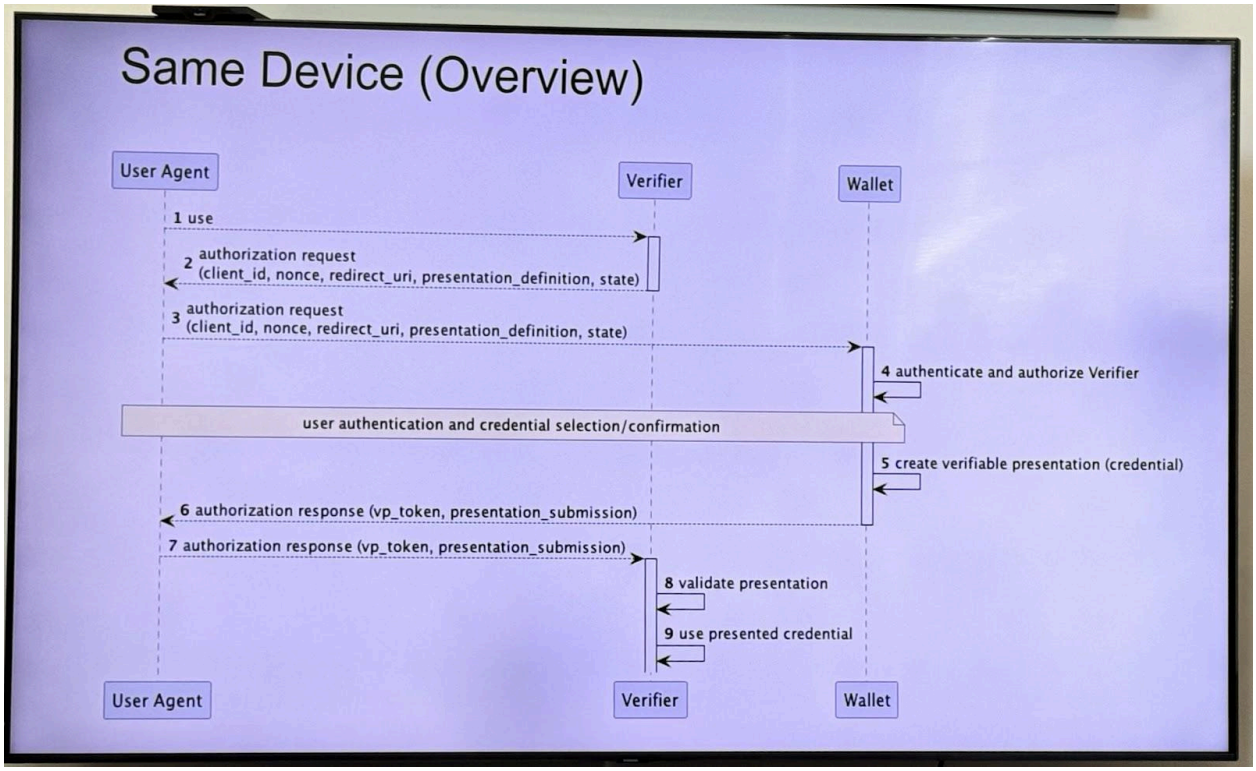
there's confirmation dialog

Q: instead of PIN

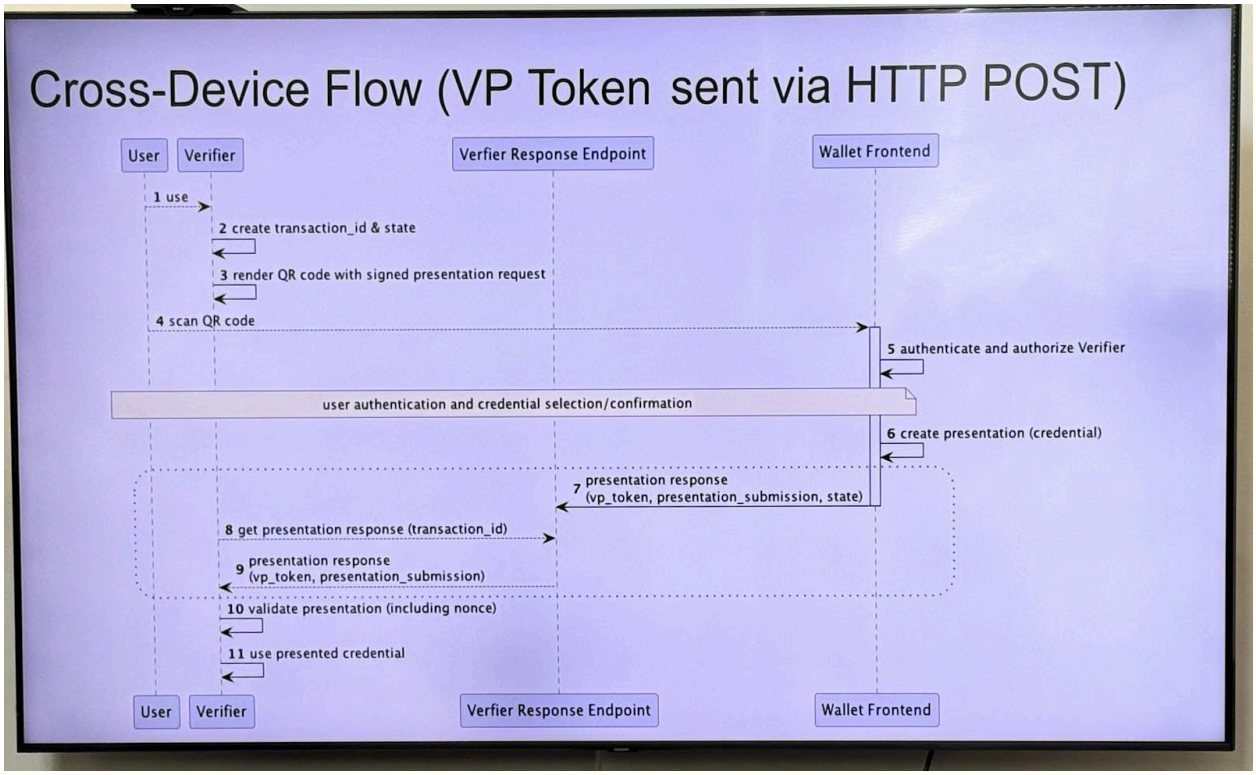


Same Device Flow

Sequence diagram for Same Device

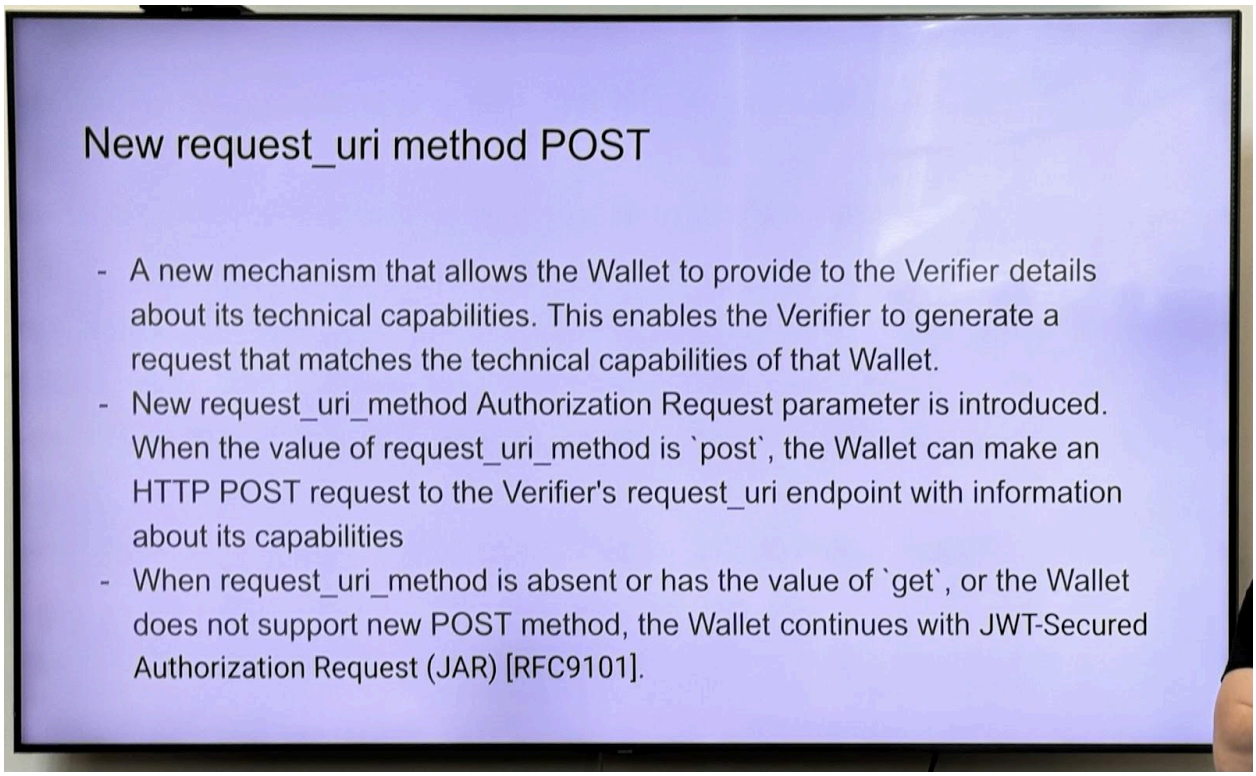


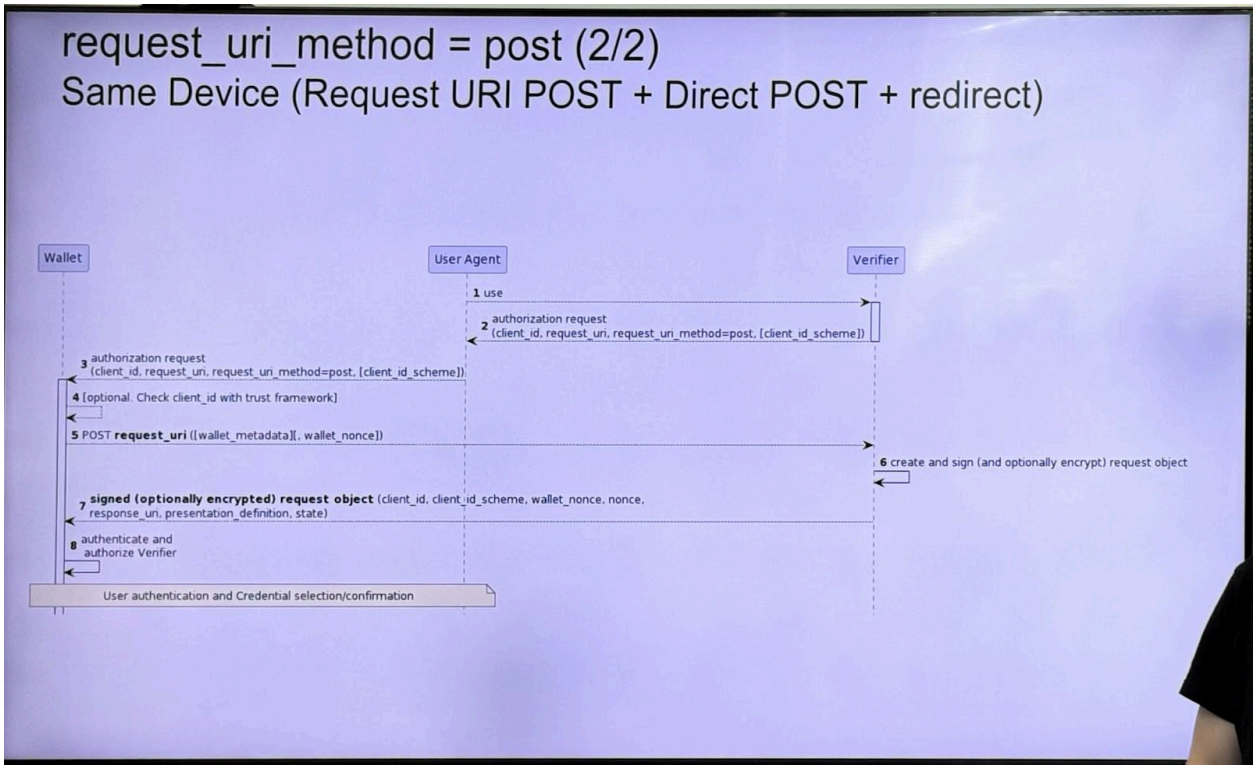
Cross-Device Flow



New request_uri method POST

purpose: allow the wallet to provide to the Verifier details about its technical capabilities. This enables the Verifier to generate a request that matches the technical capabilities of that wallet allow encryption



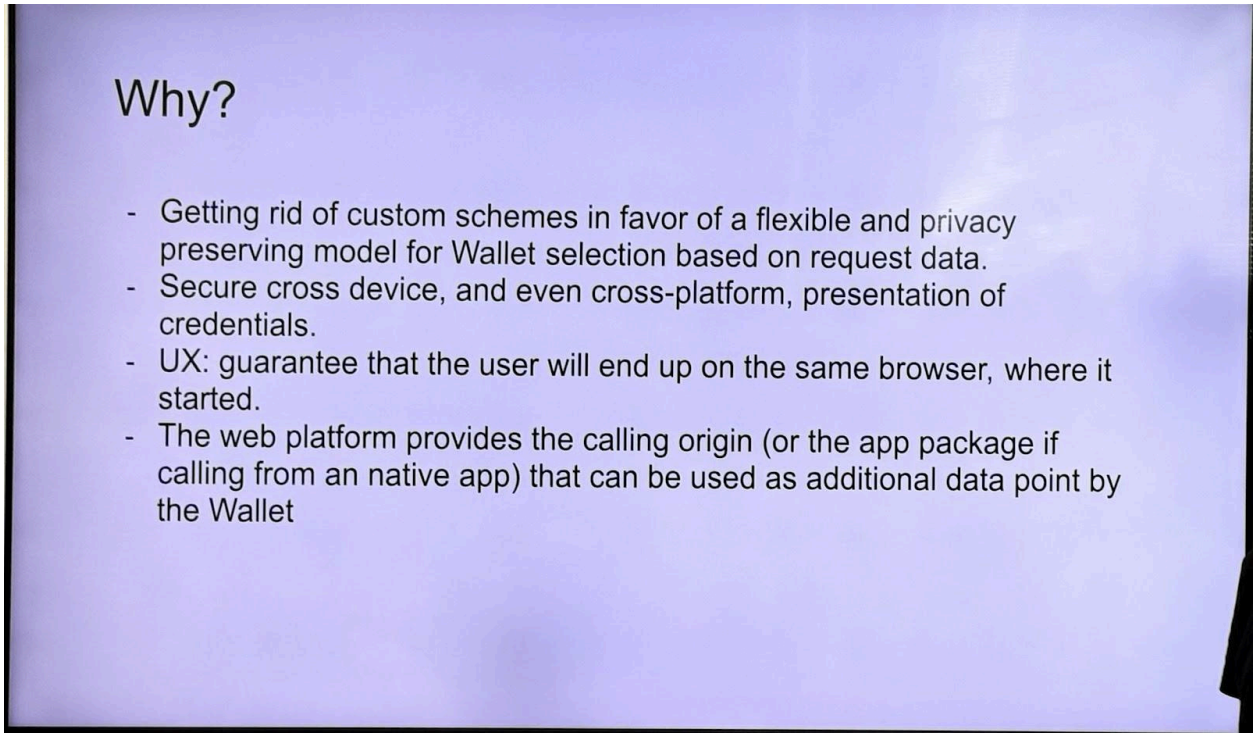

request_uri_method = post (2/2)
Same Device (Request URI POST + Direct POST + redirect)
Wallet
User Agent
Verifier
1 use
2 authorization request
(client_id, request_uri, request_uri_method=post, [client_id_scheme])
3 authorization request
(client_id, request_uri, request_uri_method=post, [client_id_scheme])
4 [optional. Check client_id with trust framework]
5 POST request_uri ([wallet_metadata][, wallet_nonce])
6 create and sign (and optionally encrypt) request object
7 signed (optionally encrypted) request object (client_id, client_id_scheme, wallet_nonce, nonce,
response_uri, presentation_definition, state)
8 authenticate and
authorize Verifier
User authentication and Credential selection/confirmation

Same Device

OpenIDVP over Browser API

Why

- getting rid of custom schemes in favour of a flexible and privacy preserving model for Wallet selection based on the request data
- Secure cross device, and even cross-platform, presentation of credentials
- UX: guarantee that the user will end up on the same browser, where it started
- The web platform provides the calling origin (or the app package if calling from an active app) that can be as additional data point by the Wallet



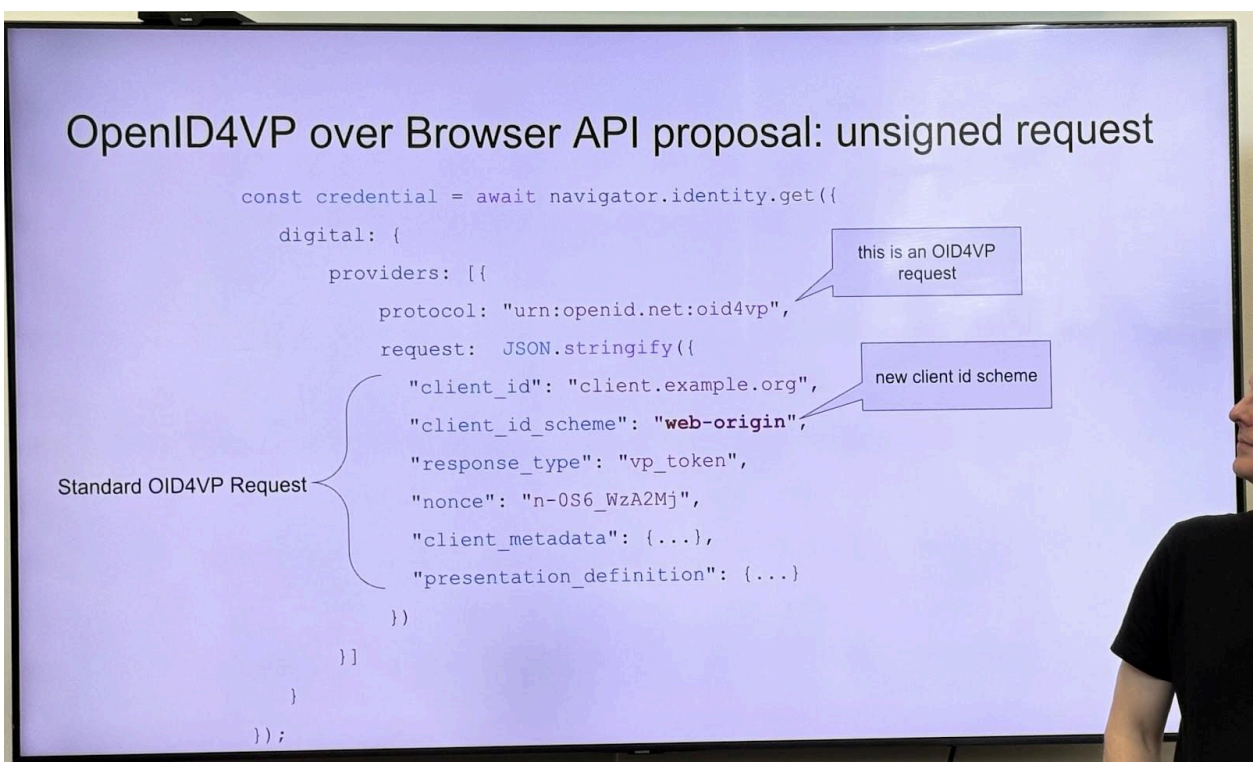
Unsigned request OpenID4VP over Browser API proposal:

Source: [Digital Credentials API explainer.md]

```
const credential = await navigator.identity.get ({
        digital: {
                providers: [{

                        protocol: "urn:openid. net: oid4vp", //this is an OID4VP request
                        //Standard OID4VP Request within the request block
                        request: JSON. stringify ( {
                          "client_id": "client.example.org",
                          "client_ id _scheme": "web-origin" //new client id scheme
                          "response_type": "vp_token",
                          "nonce": "n-0S6_WzA2Mj",
                          "client _metadata": (...),
                          "presentation _definition": (...)
                           // Presentation Exchange request, omitted for brevity
                        })
                }]//providers block
        } //digital block
}) ; //credential block
```

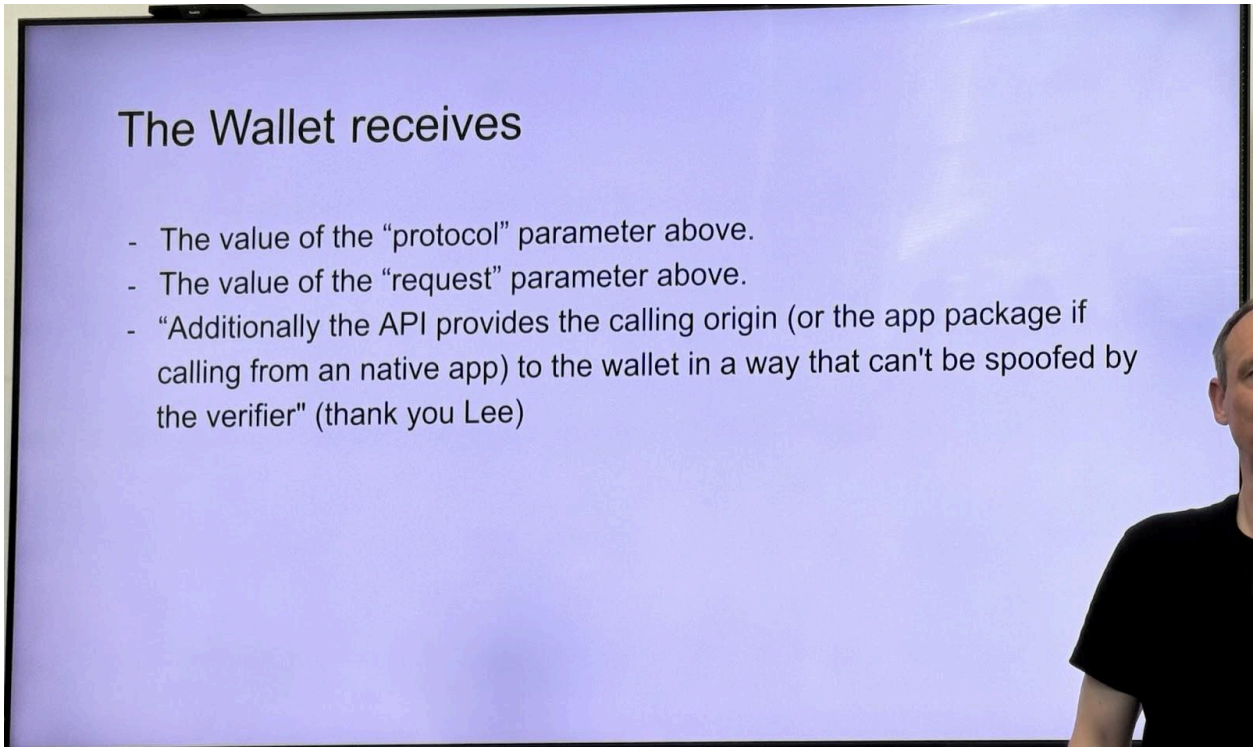
first the request unsigned version



note: presentation_definition is pointing to various credentials formats: mdoc, anoncred, etc.

The Wallet receives

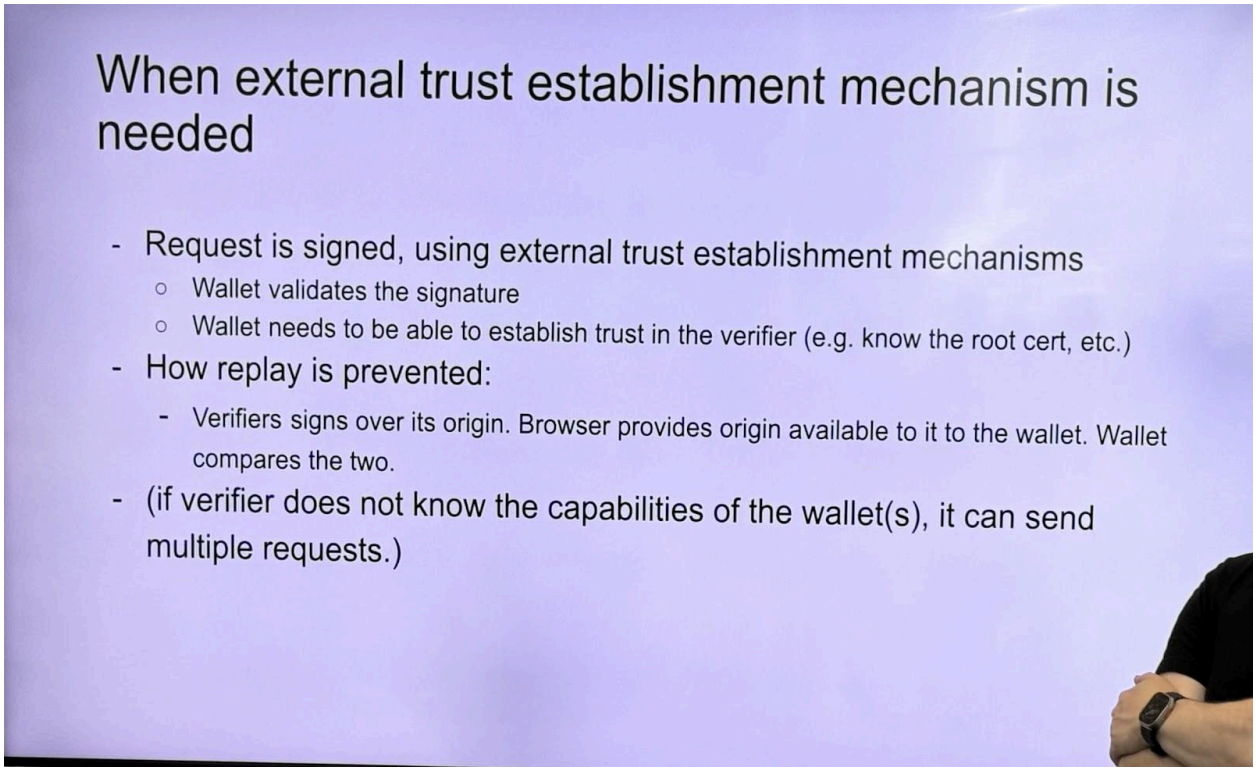
- The value of the “protocol” parameter above.
- The value of the “request” parameter
- - "Additionally the API provides the calling origin (or the app package if calling from an native app) to the wallet in a way that can't be spoofed by the verifier" (thank you Lee)



Signed Request: When external trust establishment mechanism is needed

- Request is signed
  - wallet validate the signature
  - wallet needs to be able to establish trust in the verifier
- How replay is prevented:
  - verifier sign over its origin. Browser provides origin available to it to the wallet. Wallet compares the two.
- (if verifier does not know the capabilities of the wallet(s), it can send multiple requests)

There are robust discussions here between John B, Thorsen, Dirk, Tobias, Sam Goto and others.



Response

- The wallet
  - validates the request / verifier's trust framework
  - prepares the vp_token and presentation_submission
  - MAY/MUST encrypt the response
- The response is sent back through the Browser API

```
const ( data ) = response;
const response = new URLSearchParams (data) ;
```

- The Verifier performs standard OID4VP processing.

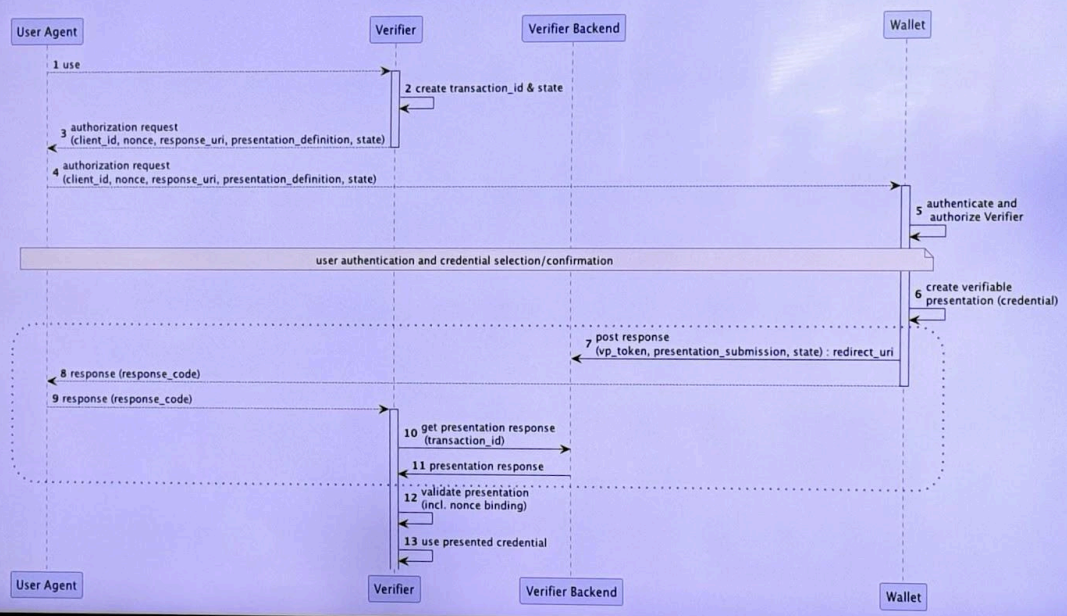
Response

- The wallet
  - validates the request / verifier's trust framework
  - prepares the vp_token and presentation_submission
  - MAY/MUST encrypt the response
- The response is sent back through the Browser API

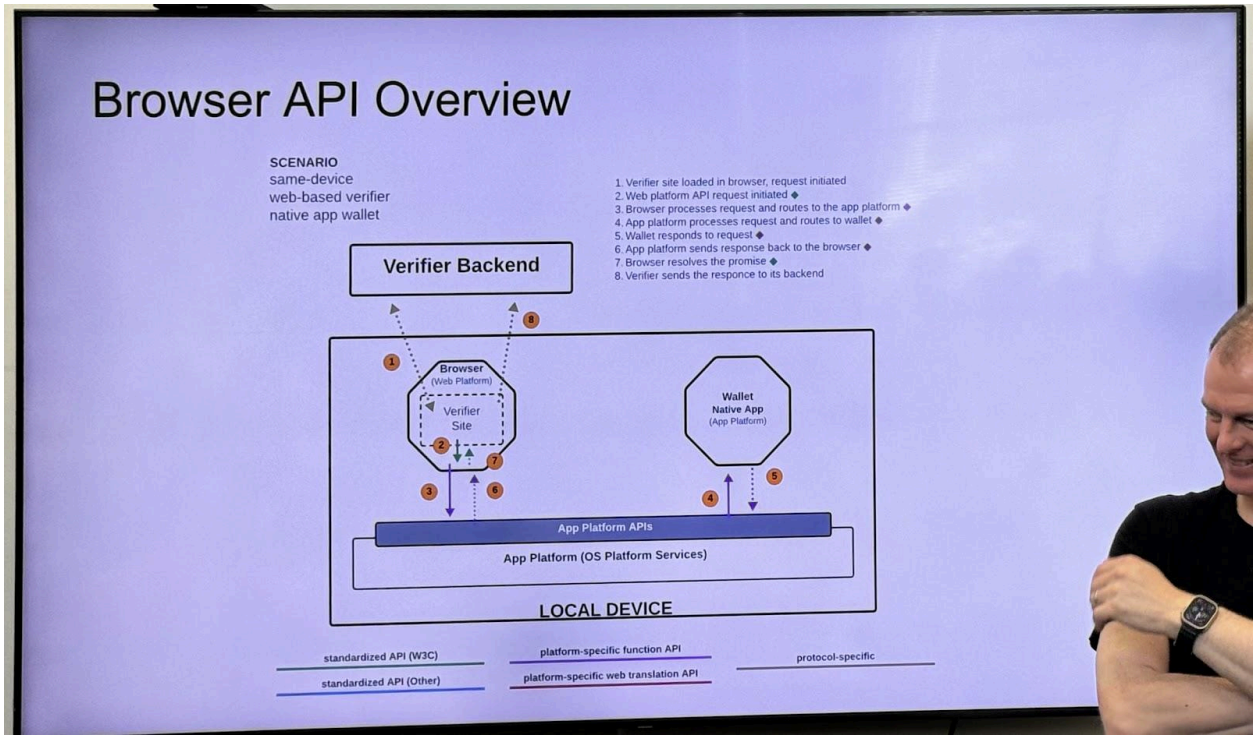
```
const { data } = response;
const response = new URLSearchParams(data);
```

- The Verifier performs standard OID4VP processing.

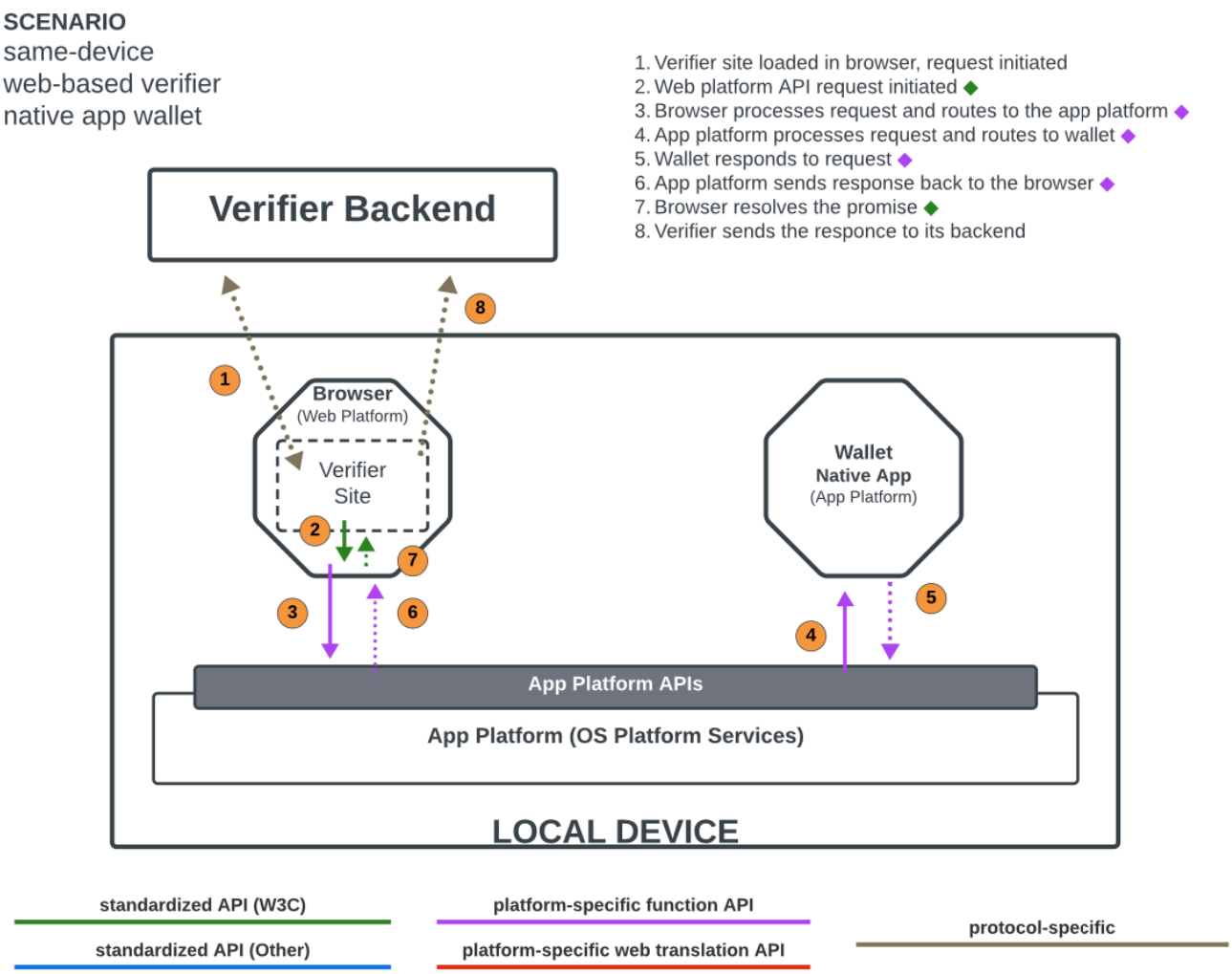
Same Device (VP Token sent via HTTP POST + redirect)



OpenID4VP over Browser API



Browser API Overview



source:

https://github.com/WICG/digital-identities/blob/main/resources/DigitalCredentialsAPI-Layering-v20240301.pdf