OID4VP profile for the W3C Digital Credentials API

Introduction

This document is a proposal for an OpenID 4 VP profile for the Browser API, so OpenID 4 VP can be passed through the Browser API between Wallet and Verifier. The profile is based on OID4VP, Draft 20. Any changes that will be made to OID4VP in the future will be leveraged by this profile.

The profile is intended to be an easy migration path for OpenID 4 VP implementations to the Browser API. This proposal is based on the ideas proposed and discussed in the issue https://github.com/WICG/digital-identities/issues/80 at WICG.

There are the following differences to the current interaction model of OpenID 4 VP:

- The selection of the wallet in the WICG's Digital Credentials API model is based on the credentials being requested for presentation, whereas the current model is based on the selection of wallets based on custom schemes used as aliases for groups of wallets with the same capabilities.
- The WICG's Digital Credentials API will allow a secure cross device, and even cross-platform, presentation of credentials.
- No need for redirects between the parties
- The web platform provides the calling origin (or the app package if calling from an native app) that can be used as additional data point by the Wallet

The W3C Digital Credentials API (https://wicg.github.io/digital-identities/) aims at providing a secure and privacy-preserving way to invoke wallets from within a Web Browser. It intends to replace custom schemes for invoking wallets. For example, a wallet is today invoked using a URL like this “openid4vp://”. Information about the Android prototype is available at https://github.com/WICG/digital-identities/wiki/HOWTO:-Try-the-Prototype-API-in-Chrome-Android#verifier-api.

The Layering and message flow is shown in https://github.com/WICG/digital-identities/blob/main/resources/DigitalCredentialsAPI-Layering-v20240301.pdf

Browser API allows web sites to request credential presentations. The Browser (and mobile OS) will assist the user to select a wallet maintaining a matching credential and dispatches the credential request to the respective wallet. This wallet will conduct the user experience required to release the credential and prepare a response that is handed back to the browser/platform, which in turn provides it back to the web site.

Responsibilities of the different parties involved

The following split of responsibilities is underlying the proposal:

- Verifier
    - prepares and signs presentation requests
    - receives and processes responses
- Browser:
    - provides API to web sites as an entry point
    - Accepts requests and enriches those with the calling origin
    - Delivers responses to web sites
- OS
    - responsible for privacy-preserving wallet selection
    - responsible for type checking the request is well-formed and valid
    - passes the Verifier's presentation request through to the selected wallet
    - enriches Verifier request with web platform data (such as the web app's origin)
    - allows requests to be passed between different devices
- Wallet
    - processes the Verifier's request
    - authenticates the Verifier in the context of the respective trust framework (eIDAS, …) and the technology used in the respective ecosystem (e.g. x509 Certificates, JWTs, OpenID Federation, …)
    - asks user for consent to share credential data with Verifier
    - creates presentation response and passes it to the Browser

Requirements

RP Authentication

Wallets need to be able to authenticate and authorize Verifiers using trust infrastructures other than the web's trust infrastructure. This is to allow implementation of applications compatible with regulatory requirements. As one example, the eIDAS obliques wallets to authenticate and identify relying parties (in this document designated as "verifiers").

For the convenience of the reader, here is the respective text from the eIDAS regulation.

"Article 5a
European Digital Identity Wallets shall, in particular:
(a) support common protocols and interfaces:
…
(vii) for authenticating and identifying relying parties by implementing authentication mechanisms in accordance with

Article 5b;

…
(c) ensure that the relying parties can be authenticated and identified by implementing authentication mechanisms in accordance with Article 5b;

…

Article 5b
European Digital Identity Wallet-Relying Parties
1. Where a relying party intends to rely upon European Digital Identity Wallets for the provision of public or private services by means of digital interaction, the relying party shall register in the Member State where it is established.
2. The registration process shall be cost-effective and proportionate-to-risk. The relying party shall provide at least:
(a) the information necessary to authenticate to European Digital Identity Wallets, which as a minimum includes:
(i) the Member State in which the relying party is established; and
(ii) the name of the relying party and, where applicable, its registration number as stated in an official record together with identification data of that official record;
(b) the contact details of the relying party;
(c) the intended use of European Digital Identity Wallets, including a indication of the data to be requested by the relying party from users.
3. Relying parties shall not request users to provide any data other than that indicated pursuant to paragraph 2, point (c).
Paragraphs 1 and 2 shall be without prejudice to Union or national law that is applicable to the provision of specific services.
5. Member States shall make the information referred to in paragraph 2 publicly available online in electronically signed or sealed form suitable for automated processing.
6. Relying parties registered in accordance with this Article shall inform Member States without delay about any changes to the information provided in the registration pursuant to paragraph 2.
7. Member States shall provide a common mechanism for allowing the identification and authentication of relying parties, as referred to in Article 5a(5), point (c).
8. Where relying parties intend to rely upon European Digital Identity Wallets, they shall identify themselves to the user.
…
10. Intermediaries acting on behalf of relying parties shall be deemed to be relying parties and shall not store data about the content of the transaction."

In any case, the governance is with the EU Member States. Fulfilling these requirements requires member states to set up and maintain a dedicated trust infrastructure for managing Relying Parties and the respective credentials and information. Currently, different technical solutions are being discussed. Use of x.509 Certificate (probably in conjunction with ETSi trust lists) or OpenID Federation are being considered.

Replay Prevention of Requests

The design must prevent replay of authenticated requests.

Proposal

This document suggests how OID4VP can be used with the W3C Digital Credentials API.

The design works with the currently suggested WICG Digital Credentials API design while also allowing to leverage advanced security features of OID4VP to comply with the regulatory requirements cited above.

There are two modes in this proposal that are determined by the request parameter values:

1. Default mode: origin provided by the browser API is the main security feature
2. Request URI POST mode: request is signed and wallet can establish trust using OID4VP mechanisms and an external trust management

Default Mode

The following shows how a Verifier would use OID4VP in conjunction with the W3C Digital Credentials API.

```
const credential = await navigator.identity.get({
   digital: {
       providers: [{
           protocol: "urn:openid.net:oid4vp",
           request:  JSON.stringify({
                       "client_id": "client.example.org",
                       "client_id_scheme": "web-origin",
                       "response_type": "vp_token",
                       "nonce": "n-0S6_WzA2Mj",
                       "client_metadata": {...},
                       "presentation_definition": {...}
                     })
       }]
   }
});
```

The protocol identifier is "`urn:openid.net:oid4vp`".

The “request” parameter resembles the standard OpenID4VP Authorization Request. In this proposal, the request parameters are claims of a JSON object instead of URI query parameters. All request parameters defined for OpenID4VP can be used, except “redirect_uri”. If “redirect_uri” is used, the wallet shall ignore it.

Note: this proposal introduces a new client id scheme value "web-origin". This scheme is used to indicate the Verifier is authenticated based on the Origin as asserted by the platform.

After the wallet was selected, the Wallet resolves a promise from the platform containing:

- The value of the “protocol” parameter above.
- The value of the “request” parameter above.
- “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)

In the default mode, the Wallet uses the calling origin to identify the Verifier.

The wallet then selects available credentials and conducts all necessary user interactions, including gathering consent to disclose credentials to the verifier.

The wallet then prepares the response according to (in this case) SD-JWT VC. The returned credential includes the value of the `nonce` request parameter. The response is not encrypted.

The wallet passes the response back to the platform, which in turn passes it back to the Verifier.

The the respective Verifier code (according to Digital Credentials Section 8):

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

The “response” claim contains a OpenID4VP response, consisting of a `vp_token` and a `presentation_submission`. The latter allows the RP to determine, which of the requested credentials (if the request had alternatives), was provided by the wallet.

The calling origin is always authenticated by the web platform, so even if an attacker replays a request captured someplace else, the actual origin will be the attacker’s origin.

Request URI (POST method)

If the Verifier needs to authenticate using an external trust framework, like the one described above for the EUDIW, the Verifier needs to sign the request using a key associated with that trust framework, e.g. through an OpenID Federation Entity Statement.

In order to prevent replay of such signed requests, this design utilizes the OID4VP feature Request URI (POST method) to let the Verifier create a signed request, which includes a Wallet-provided nonce.

The Verifier initiates the interaction.

```
const credential = await navigator.identity.get({
   digital: {
       providers: [{
           protocol: "urn:openid.net:oid4vp",
           request: JSON.stringify({
                       "client_id":"https//client.example.org",
                       "client_id_scheme":"entity_id",
                       "presentation_definition":"...",
```

```
                        "request_uri":"https://client.example.org/request",
                        "request_uri_method":"post"})
        }]
    }
});
```

This example includes the parameters “request_uri”, which asks the Wallet to obtain a Request Object from the location passed in the parameter value. The parameter “request_uri_method” with the value “post” asks the Wallet to do so with a HTTPS POST request. This allows the Wallet to send parameters with the request, most notably a nonce. The verifier will then create a request object, which MUST include this nonce.

Note: this draft does not propose use of the request_uri_method “get” as it does not offer advantages over the default mode as described above.

The request also contains a “presentation_definition” parameter even though the same parameter will be contained in the request object created in a later step. This is to facilitate the wallet selection process.

After the platform has selected the wallet, the wallet sends a HTTP POST request to the “request_uri” with its capabilities. This request includes a “wallet_nonce”.

The Verifier creates a request object and returns it to the wallet.

This response is a signed request object (a JWT), which includes the “wallet_nonce” and may also contain additional data as required under the external trust framework to validate the request.

Here is an example of a request object:

```
{
 "client_id": "client.example.org",
 "client_id_scheme": "entity_id",
 "response_type": "vp_token",
 "nonce": "n-0S6_WzA2Mj",
 "wallet_nonce": "...",
 "client_metadata": {
   "vp_formats": {
     "vc+sd-jwt": {
       "sd-jwt_alg_values": [
         "PS256"
       ],
       "kb-jwt_alg_values": [
         "PS256",
       ]
     }
   },
   "jwks": {
     "keys": [
```

```
        {
          "kty": "EC",
          "crv": "P-256",
          "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
          "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
          "use": "enc",
          "kid": "1"
        }
      ]
    }
  }
 "presentation_definition": {...},
 "state": "eyJhb...6-sVA
}
```

The wallet then decodes and processes the request. It especially checks whether the `wallet_nonce` with the correct value is present.
Note that this request also requests the Wallet to encrypt the response by passing an encryption key.

The wallet then selects available credentials and conducts all necessary user interactions.

The wallet passes the encrypted response back to the platform, which in turn passes it back to the Verifier.

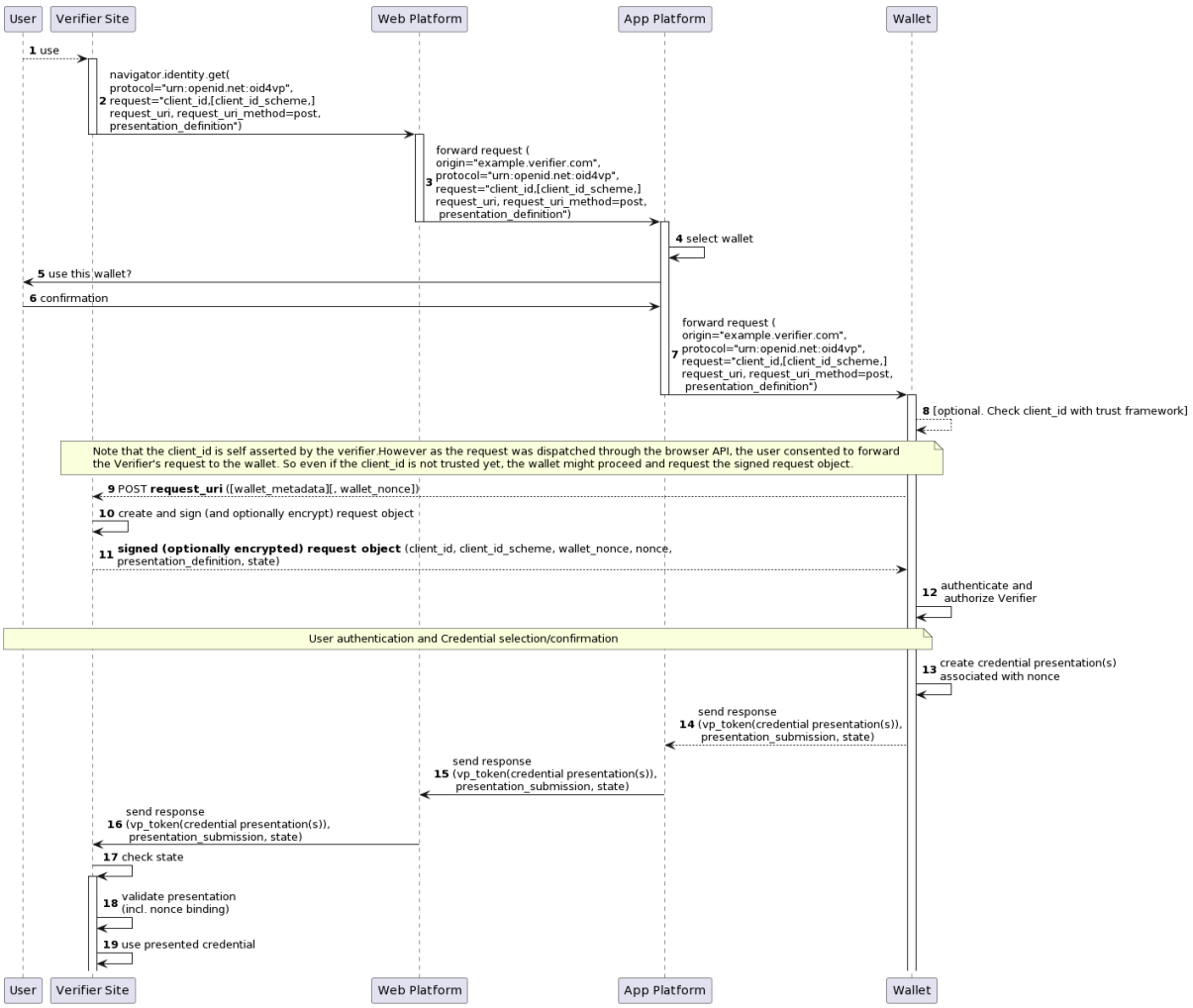
```
const { data } = credential;
const response = new URLSearchParams(data)

// Decrypts and checks response as specified in Section 6.3 of OID4VP
// extracts `vp_token` and `presentation_submission`
```

Sequence Diagram

The following sequence diagram visualizes the overall messages/data flow

https://hackmd.io/@hisgarden/S1TvFVFkA/edit


User
Verifier Site
Web Platform
App Platform
Wallet
1 use
navigator.identity.get(
protocol="urn:openid.net:oid4vp",
2 request="client_id,[client_id_scheme,]
request_uri, request_uri_method=post,
presentation_definition")
forward request (
origin="example.verifier.com",
3 protocol="urn:openid.net:oid4vp",
request="client_id,[client_id_scheme,]
request_uri, request_uri_method=post,
presentation_definition")
4 select wallet
5 use this wallet?
6 confirmation
forward request (
origin="example.verifier.com",
7 protocol="urn:openid.net:oid4vp",
request="client_id,[client_id_scheme,]
request_uri, request_uri_method=post,
presentation_definition")
8 [optional. Check client_id with trust framework]
Note that the client_id is self asserted by the verifier.However as the request was dispatched through the browser API, the user consented to forward the Verifier's request to the wallet. So even if the client_id is not trusted yet, the wallet might proceed and request the signed request object.
9 POST request_uri ([wallet_metadata][, wallet_nonce])
10 create and sign (and optionally encrypt) request object
11 signed (optionally encrypted) request object (client_id, client_id_scheme, wallet_nonce, nonce, presentation_definition, state)
12 authenticate and authorize Verifier
User authentication and Credential selection/confirmation
13 create credential presentation(s) associated with nonce
send response
14 (vp_token(credential presentation(s)), presentation_submission, state)
send response
15 (vp_token(credential presentation(s)), presentation_submission, state)
send response
16 (vp_token(credential presentation(s)), presentation_submission, state)
17 check state
18 validate presentation (incl. nonce binding)
19 use presented credential
User
Verifier Site
Web Platform
App Platform
Wallet