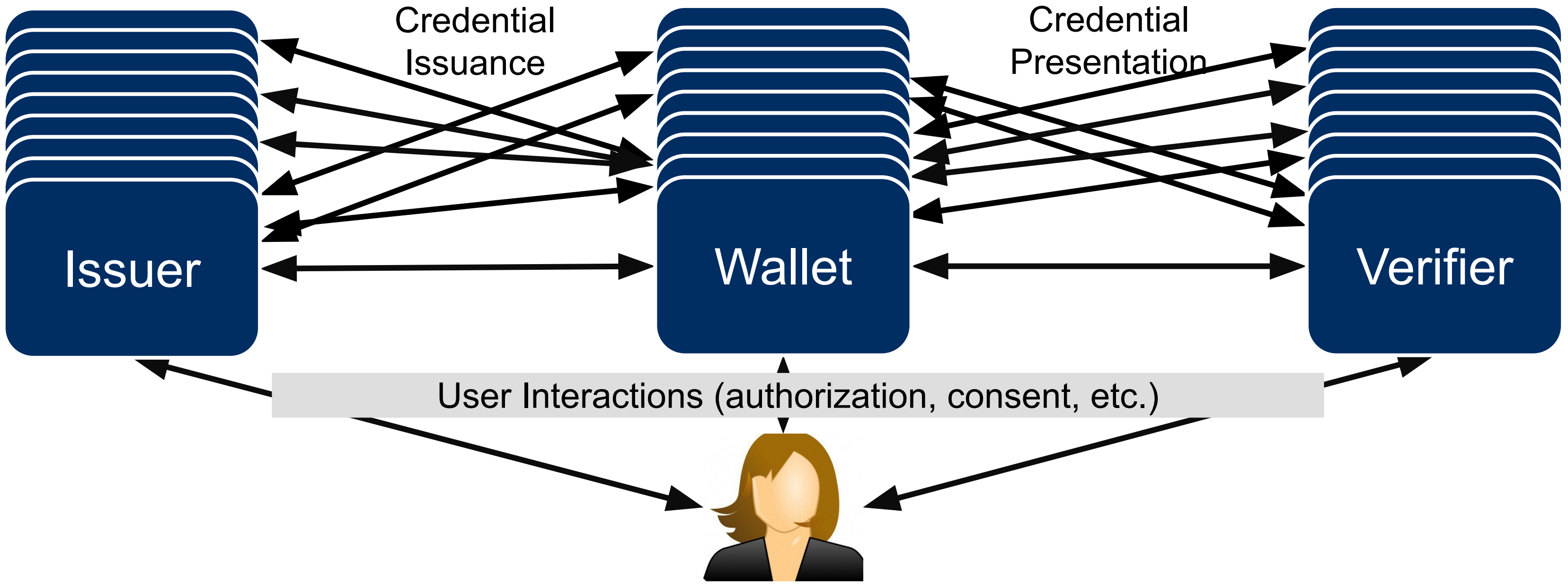
OpenID for Verifiable Credentials

The next generation of OpenID

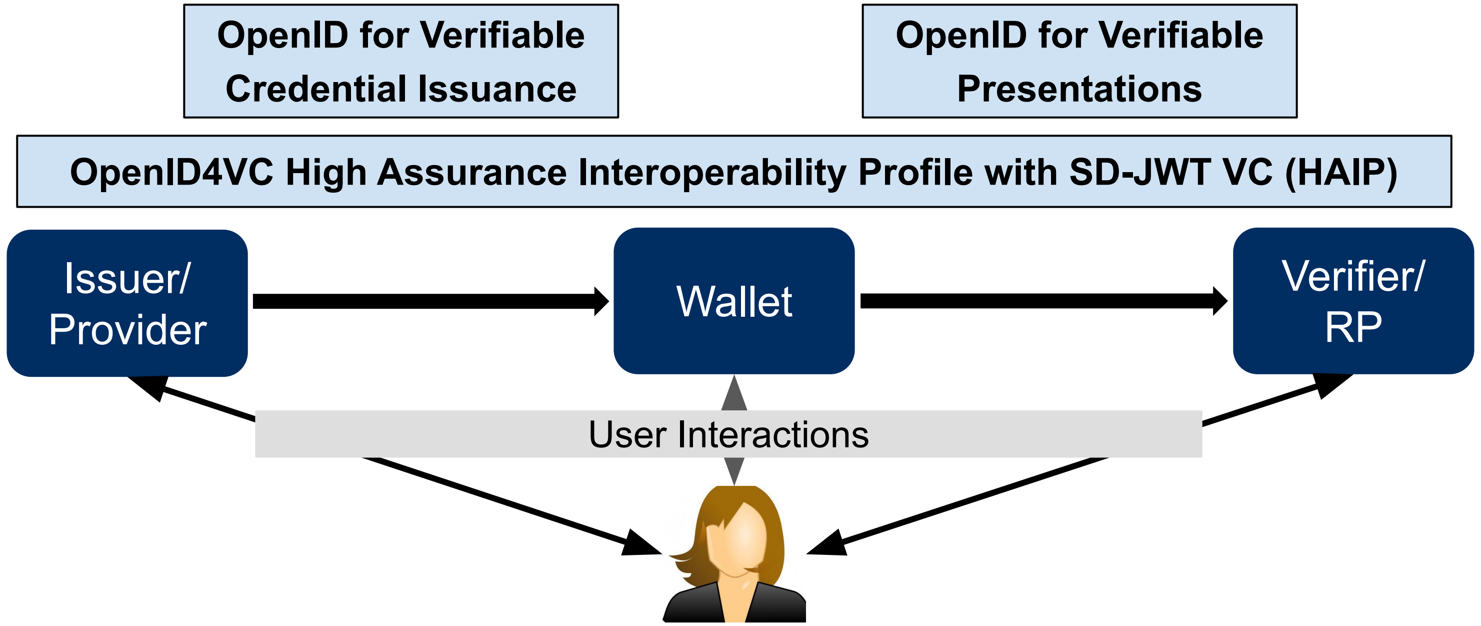
Kristina Yasuda (SPRIND), Oliver Terbu (MATTR)

Protocol Layer Interoperability is Crucial

There was a need for the interoperable protocol layer that can support all of the credential formats, key resolution mechanisms and trust frameworks.



OID4VC: OpenID for Verifiable Credentials set of protocols

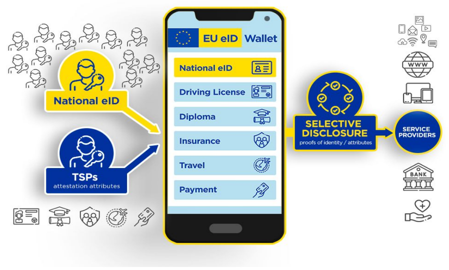

OpenID for Verifiable Credential Issuance
OpenID for Verifiable Presentations
OpenID4VC High Assurance Interoperability Profile with SD-JWT VC (HAIP)
Issuer/ Provider
Wallet
Verifier/ RP
User Interactions

OID4VC set of protocols also includes Self-Issued OpenID Provider v2 (SIOPv2) and OpenID4VP over BLE. OpenID4VC high assurance interoperability profile for mdoc is being developed in ISO

We won a prize, European Identity & Cloud Award ;-)

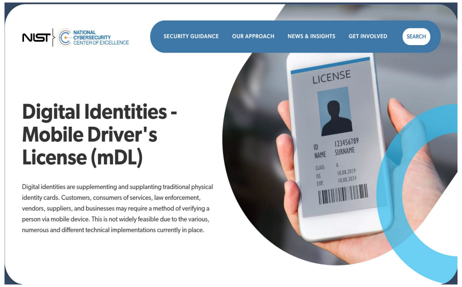
eic
European Identity and Cloud Award Winner
2024
OpenID for Verifiable Credentials / OpenID Foundation
Category:
Future Technologies and Standards
kuppingercole
european identity and cloud conference 2024

Global Adoption (selected use-cases)



The European Digital Identity Wallet[1], ARF v.1.4

mandates the usage of OpenID4VC protocols

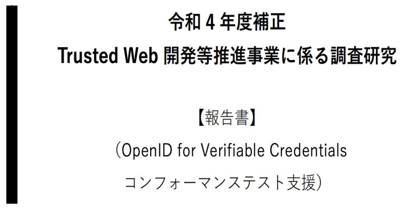


Digital Identities - Mobile Driver's License (mDL)

Digital identities are supplementing and supplanting traditional physical identity cards. Customers, consumers of services, law enforcement, vendors, suppliers, and businesses may require a method of verifying a person via mobile device. This is not widely feasible due to the various, numerous and different technical implementations currently in place.

NIST National Cybersecurity Center of Excellence[2]

is running a project implementing and testing implementations for OID4VP to present mdocs/mDL



令和 4 年度補正
Trusted Web 開発等推進事業に係る調査研究

【報告書】
（OpenID for Verifiable Credentials
コンフォーマンステスト支援）

Japanese Government's Trusted Web Project [3]

has implemented OID4VC protocols various use-cases

[1] cloudsignatureconsortium.org/new-eu-eidas-regulation-a-quantum-leap-for-electronic-identity/
[2] nccoe.nist.gov/projects/digital-identities-mdl
[3] kantei.go.jp/jp/singi/digitalmarket/trusted_web/2023seika/files/004_report_oidf_conformance_test.pdf

Open Source libraries

Walt.id

Kotlin: github.com/walt-id/waltid-ssikit

Kotlin Multiplatform: shorturl.at/XtEXw

Sphereon

Transcript: tinyurl.com/2de634na

shorturl.at/yUnkA

shorturl.at/MHW1z

Microsoft

Swift: tinyurl.com/2jejntsp

Kotlin: tinyurl.com/4bd5p3bx

Spruce

Rust: github.com/spruceid/oidc4vci-rs

Rust: tinyurl.com/rp35fsc8

EBSI

Javascript: tinyurl.com/y945s5xu

Impierce Technologies

Rust: github.com/impierce/openid4vc

Animo

Typescript: github.com/animo/paradym-wallet

Trustbloc

Go: github.com/trustbloc/vcs

github.com/trustbloc/wallet-sdk

Italian Government

Python: tinyurl.com/56ft5m34

Python: shorturl.at/Gxd2D

AltMe

Dart: github.com/TalaoDAO/AltMe

MOSIP

Kotlin/ Swift/ ReactNative: github.com/mosip/tuvali

EUDI Reference

Wallet Implementation: shorturl.at/rD7tf

OpenID4VC Conformance Tests

Implementers can use conformance tests to ensure compliance to the specification and interoperability with other implementations

Available

Tests for the Wallets for OpenID4VP profiles with SD-JWT VC and mdocs (HAIP and 18013-7). Being updated to the recent specifications changes. 10+ Wallets have already passed certification

In-progress

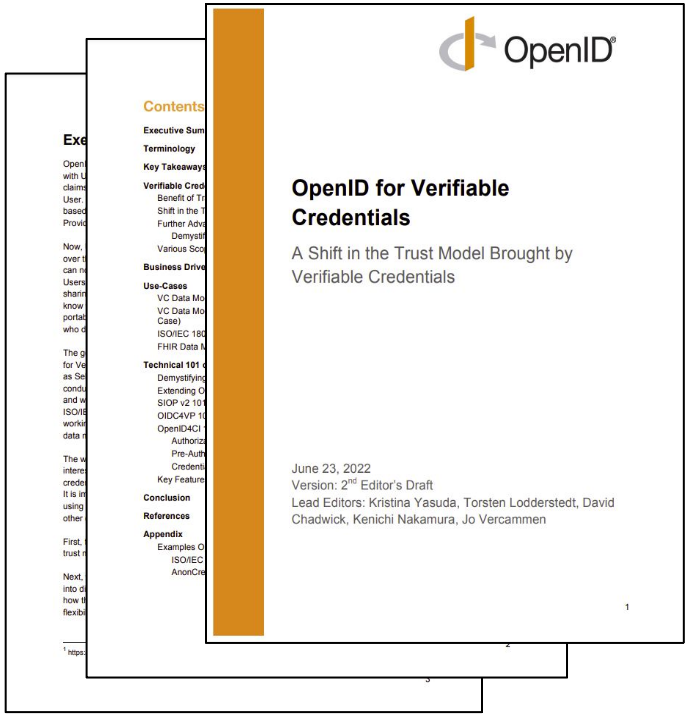
Tests for the Wallets for OpenID4VCI profiles with SD-JWT VC and mdocs (HAIP and 23220-3). Projected to complete in the next 3 months

OpenID4VC Security Analysis

„Security and Trust in OpenID for Verifiable Credentials“ document describes the trust architecture in OpenID for Verifiable Credentials specifications, outlines security considerations and requirements for the components in an ecosystem

Master Thesis **„OpenID for Verifiable Credentials: formal security analysis using the Web Infrastructure Model“** published:

Next: OpenID4VP and OpenID4VCI



OpenID

OpenID for Verifiable Credentials

A Shift in the Trust Model Brought by Verifiable Credentials

June 23, 2022
Version: 2nd Editor's Draft
Lead Editors: Kristina Yasuda, Torsten Lodderstedt, David Chadwick, Kenichi Nakamura, Jo Vercammen

1

Follow QR-Code for the "OpenID for Verifiable Credentials" whitepaper

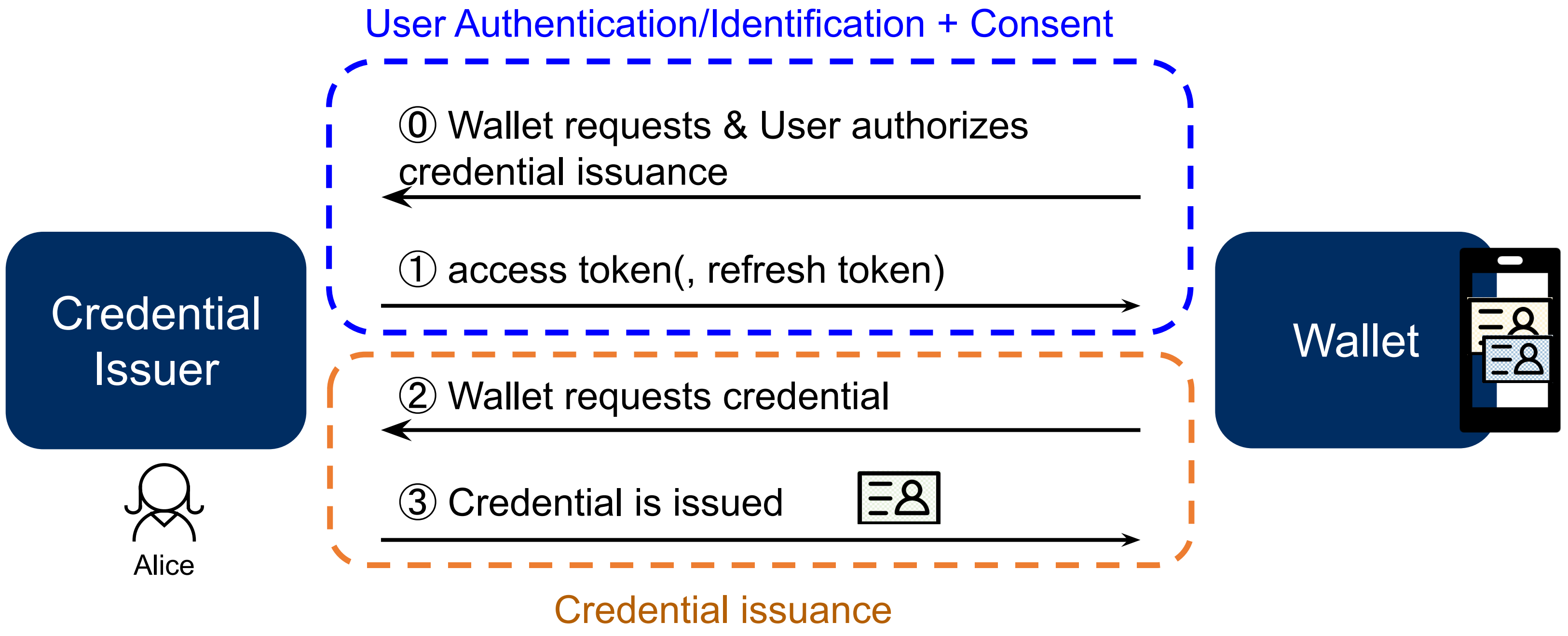
OpenID for Verifiable Credential Issuance

OpenID for Verifiable Credential Issuance: Highlights

Status: First Implementer‘s draft published on April 1st!

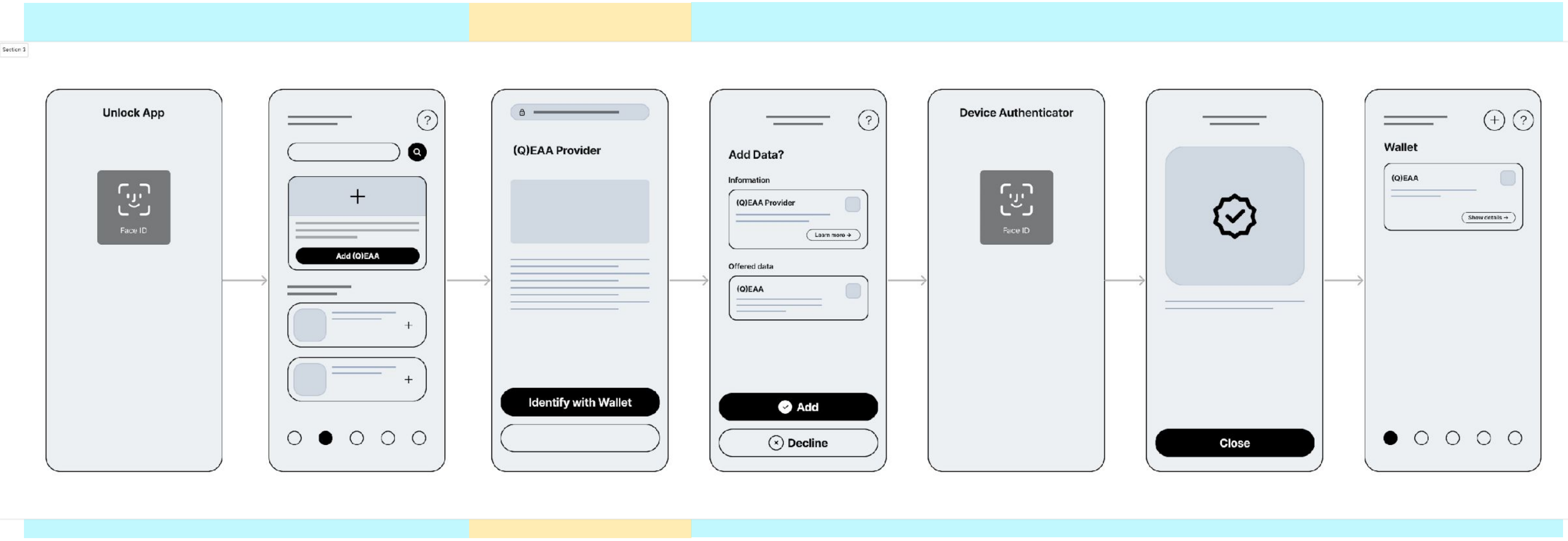
- Easy to use for developers
- Various Security levels can be supported
- Various business requirements and user-experiences can be achieved
- Various trust frameworks and credential formats can be supported

OAuth-protected API

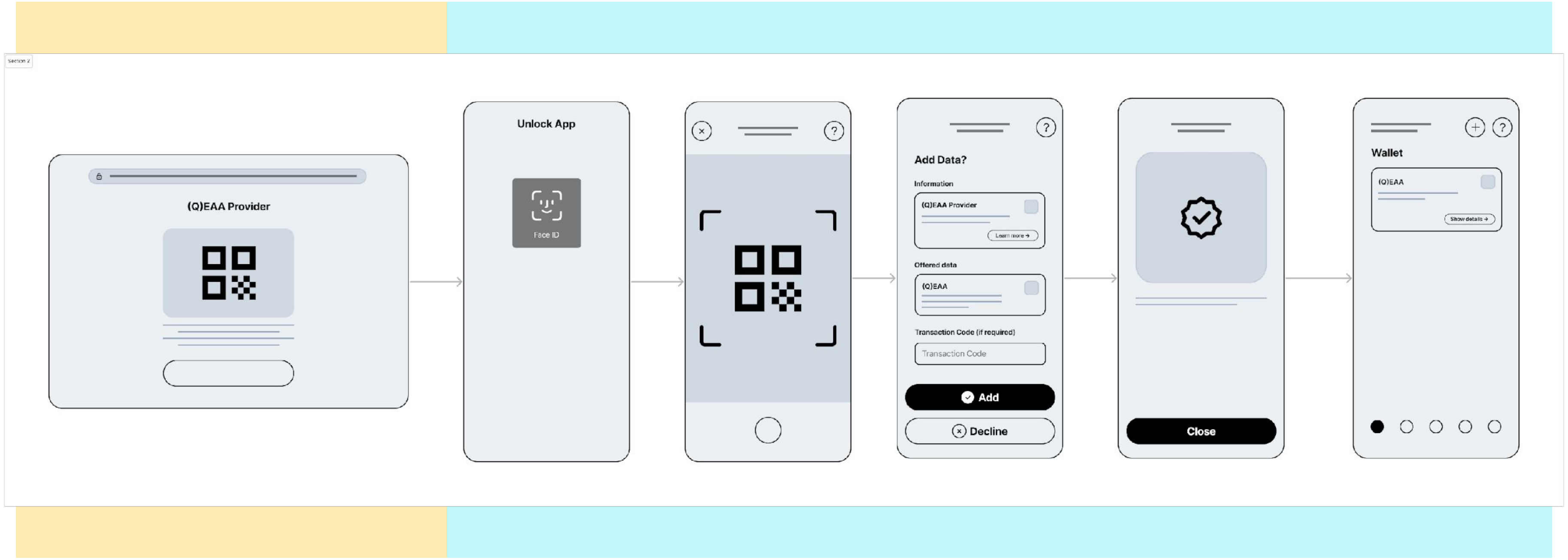

User Authentication/Identification + Consent
⓪ Wallet requests & User authorizes credential issuance
① access token(, refresh token)
Credential Issuer
Alice
Wallet
② Wallet requests credential
③ Credential is issued
Credential issuance

OpenID4VCI can be used in conjunction with any other OAuth extension RFC

Authorization Code Flow



Pre-Authorized Code Flow



OpenID4VCI: updates

OpenID4VCI: updates

Optimizing Issuance of Credential Batches

- Fulfills a requirement to issue multiple Credentials with the same claim values, but different cryptographic materials to achieve unlinkability without using ZKP
- Optimizes passing multiple proofs for each Credential Configuration in the same Credential Request

Deferred authori-zation removed

- (`authorization_pending`) option was removed
- Due to the security concerns around long-lived authorization code, and lack of implementations of the feature

Notification endpoint introduced

- Enables the Wallet to notify the Issuer about (un)successful issuance and deletion of the Credential by the user to improve UX and Credential lifecycle management
- A new Notification Endpoint hosted by the Issuer

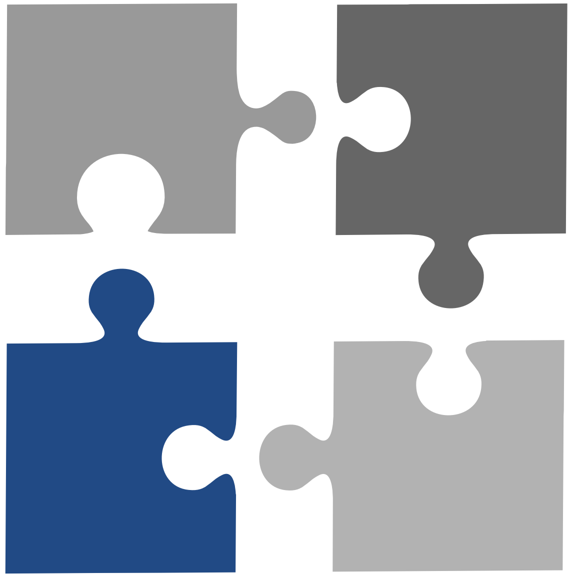
OpenID4VCI: topics under discussion

Clarify Wallet/Key attestations during Issuance

How wallet/key attestations are passed throughout the protocol steps, what are the schemas for those attestations, etc.

Optimizing number of endpoints

What are pros and cons of having separate Credential, Batch Credential and Deferred Endpoints?



Discussing adding a mechanism for the Issuer to notify the wallet about certain events

For example, process-related, risk-related, lifecycle-related events

Issuer Metadata enhancements

Enabling specifying value types for the parameters, dark-mode of display parameters, etc.

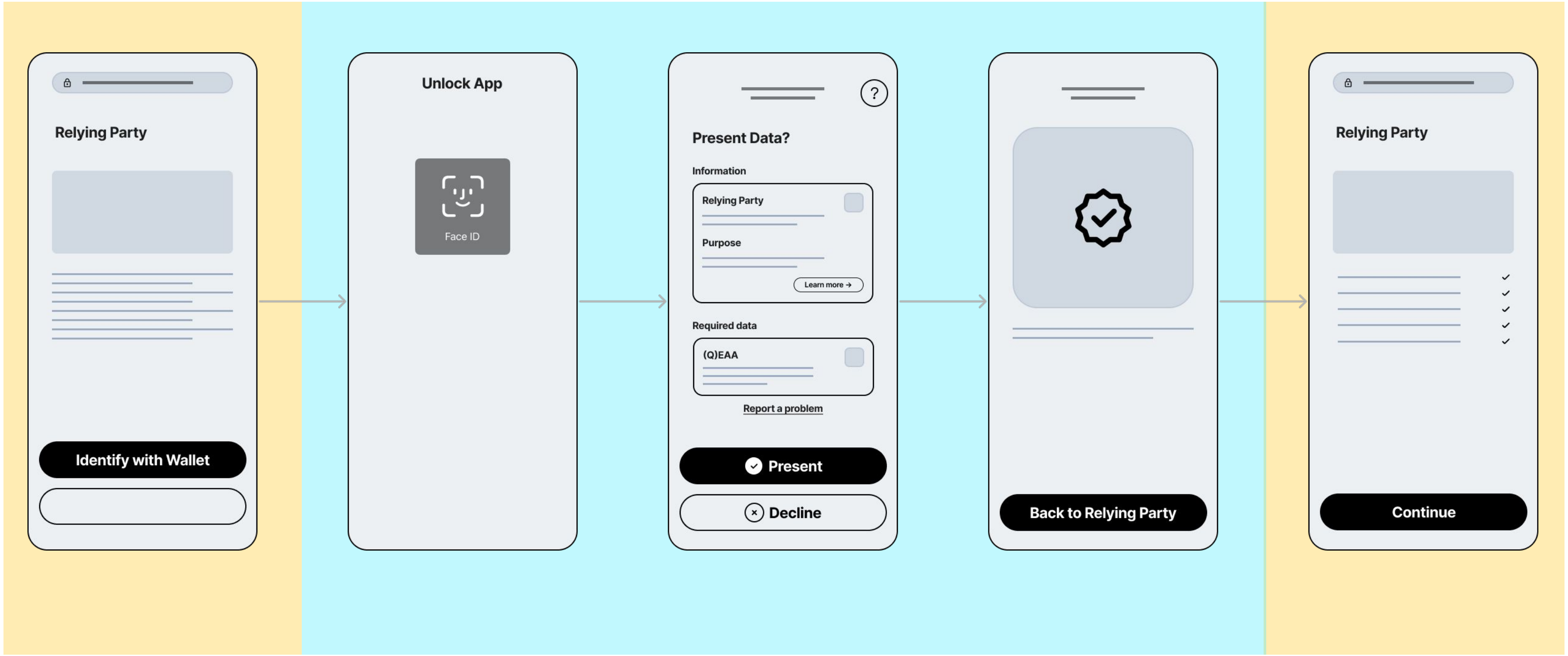
OpenID for Verifiable Presentations

OpenID for Verifiable Presentations: Highlights

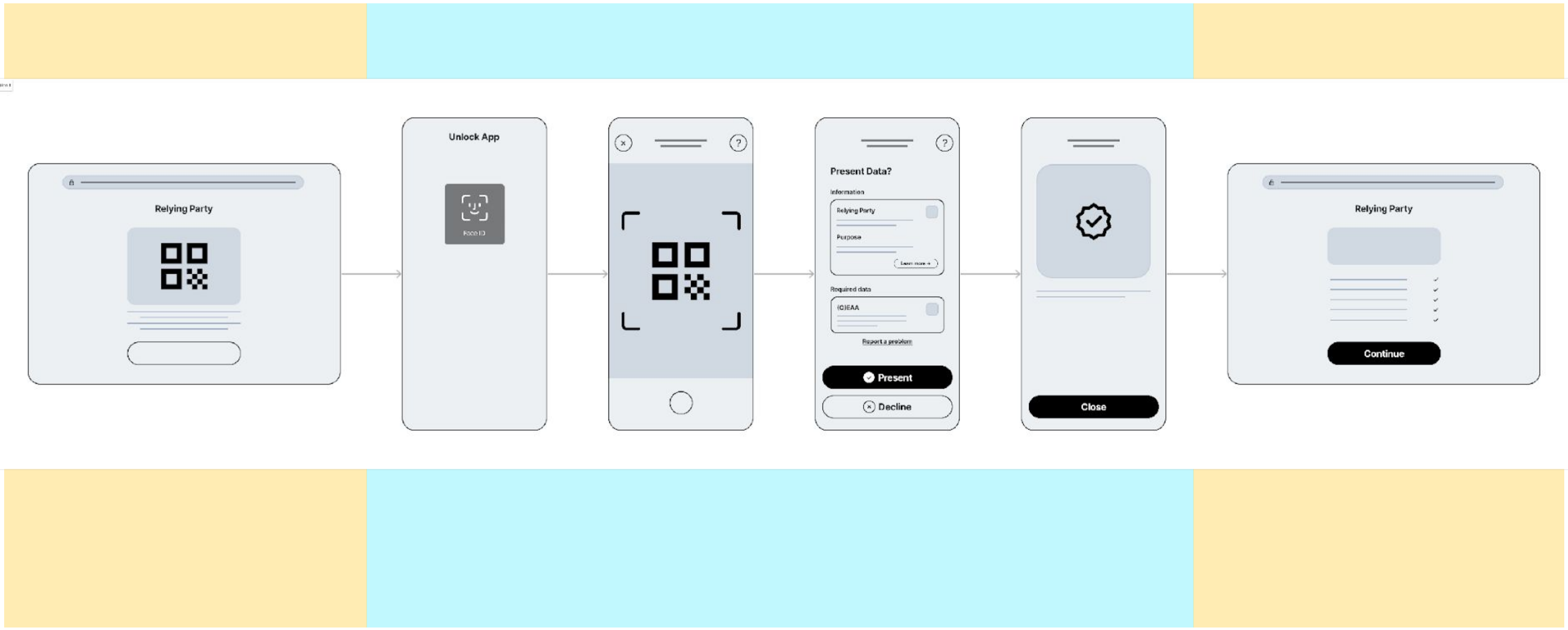
Status: 2nd Implementer‘s draft published in May 2023

- Designed for highest degree of privacy (e.g. wallet does not need a backend to store and transmit Credentials)
- Various Security levels can be supported
- Easy of use for developers
- Presentation of multiple Credentials in one response supported
- Various Wallet deployment models supported
- Various trust frameworks and credential formats can be supported

Same Device Presentation



Cross Device Presentation



OpenID4VP: updates

Profile of OpenID4VP over Digital Credentials API

- Expected benefits:
  1. Flexible and Privacy-preserving credential-based wallet selection and getting rid of custom schemes
  2. Increasing security of cross-device, cross-platform presentation of credentials;
  3. Improved UX (user getting back to the same browser);
  4. Improved security (platform-provided calling origin)
- Defining how OpenID4VP request can be passed using digital credential API being defined in W3C

request_uri_method=post

- Ability for the Wallet to negotiate its capabilities and request Verifier to include wallet provided nonce in the signed request object

OpenID4VP: topics under discussion

Query Language

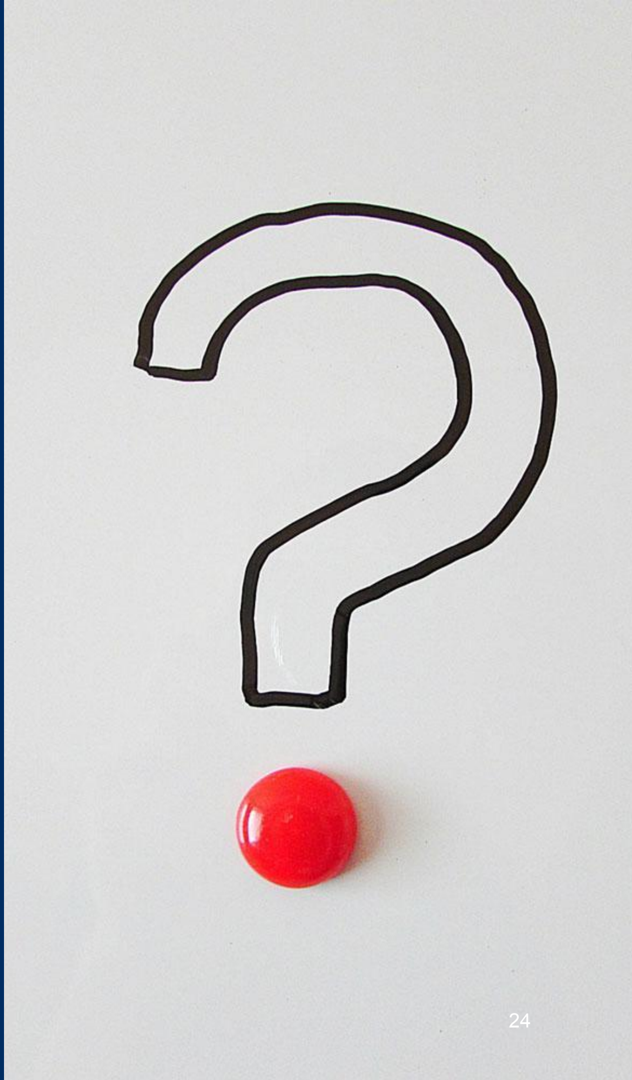
- Simplifying how the Verifier communicates to the Wallet requirements about the Credentials and claims being requested

Transaction Data

- Designing an explicit mechanism how presentation of a particular Credential can be bound to a transaction specific data (dynamic linking)
- Flagship use-cases: payments confirmation, QES authorization

Any questions?

Mail:
kristina.yasuda@sprind.org,
oliver.terbu@mattr.global



Backup

Verifiable Credentials

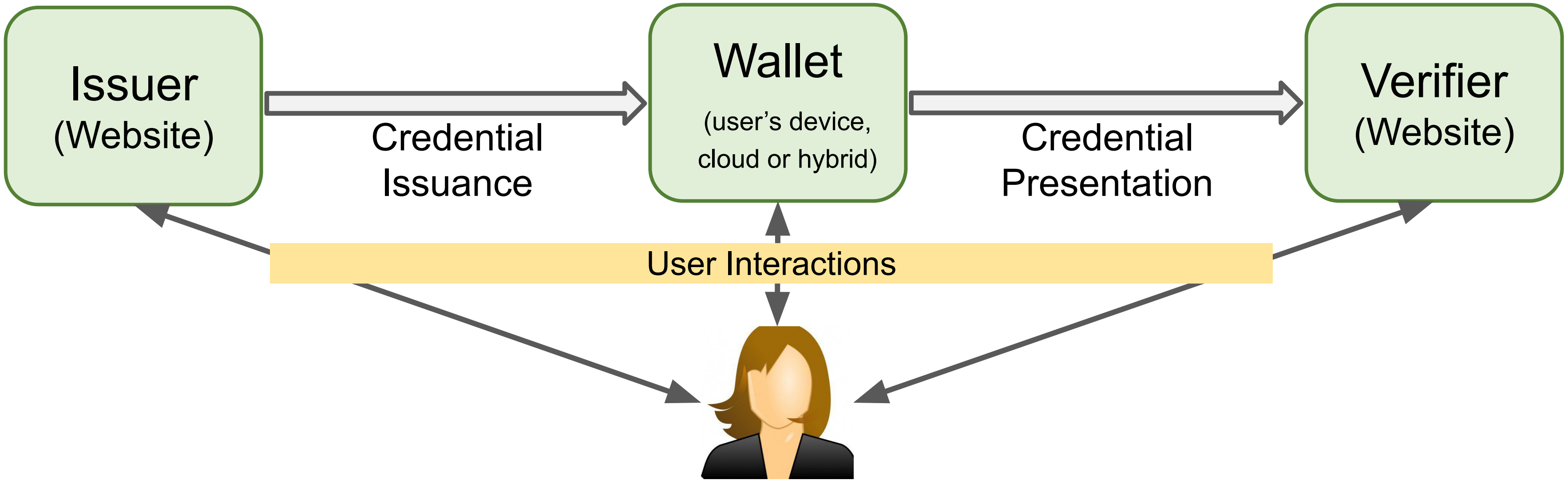
Verifiable Credentials

- A verifiable credential (VC) is a set of tamper-evident claims and metadata about real life achievements, qualifications, or attributes that includes a cryptographic proof created by the issuer of the credential.
- Examples of verifiable credentials is anything that is currently issued and shared on paper form is a candidate for a verifiable credential + more
    - Driving licence
    - Health card
    - Personal identity card
    - Product passport
    - …

What is Decentralized Identity?

- The User presenting the Identity data directly to the Verifier from the Wallet
  - <> In the federated model where Identity data is sent directly from the IdP to the Verifier

- Usually expressed with the flow below:



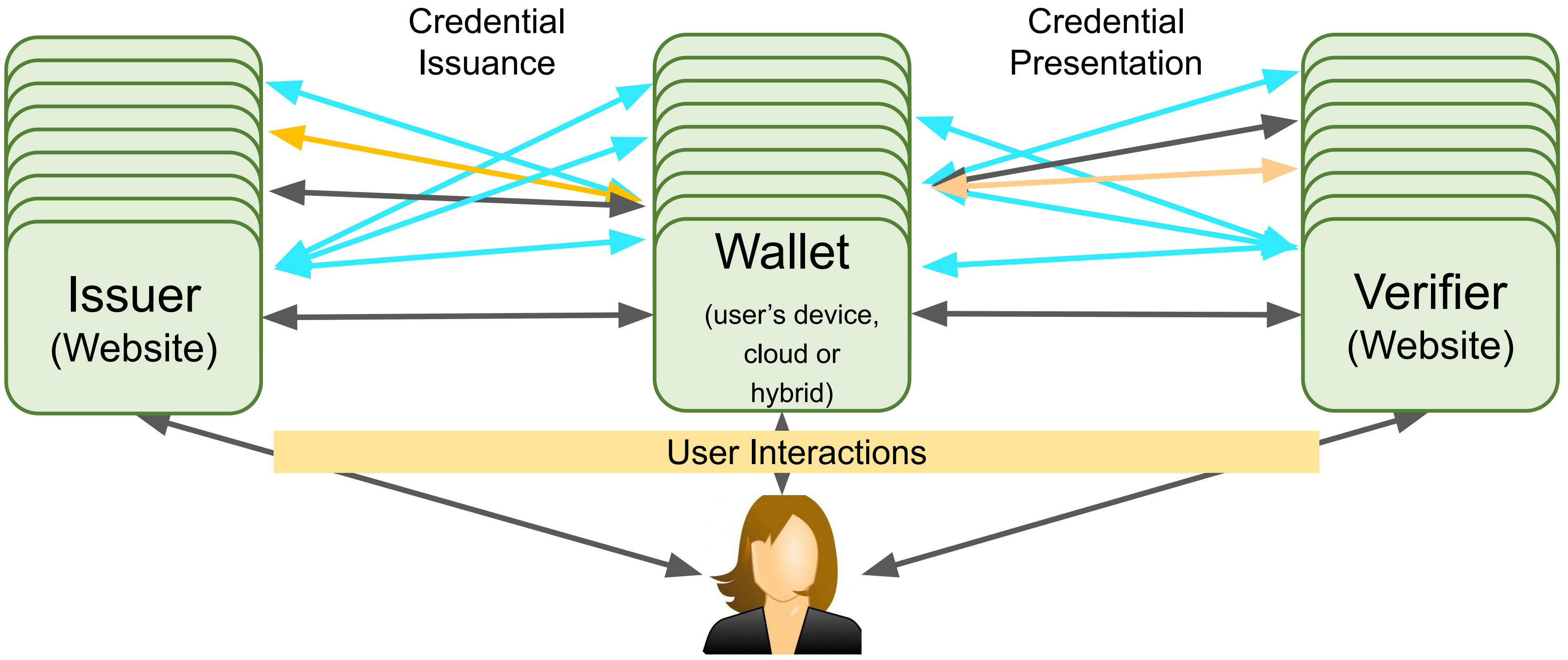
Verifiable Credentials: Benefits

- End-Users gain more privacy, and portability over their identity information.
- Cheaper, faster, and more secure identity verification, when transforming physical credentials into digital ones.
- Universal approach to handle identification, authentication, and authorization in digital and physical space.

OpenID for Verifiable Credentials

Why Protocol Layer Interoperability is Crucial.

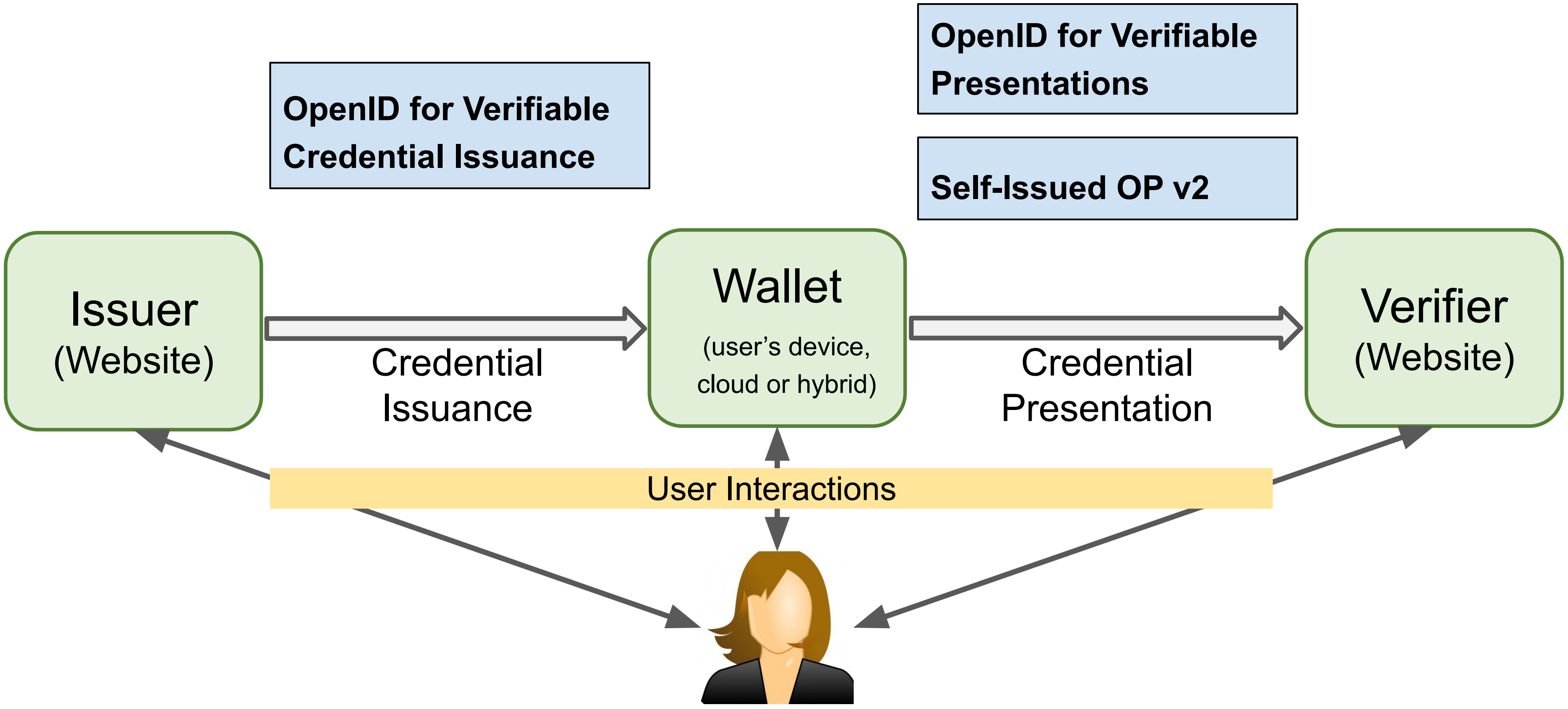
One entity needs to talk to the large the number of entities, to increase the value of "Decentralized Identity".



Problems we identified and how we solved them

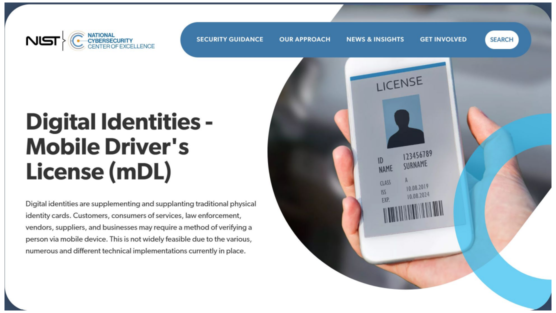
Problem		Solution
A lot of entirely new Protocols. (Hard to get security right, steep learning curve)	⇒	Building upon currently widely used protocols: OAuth 2.0 and OpenID Connect. (Secure, already understood)
No clear winner among Credential Formats	⇒	Designing a protocol agnostic to the Credential Formats.
No one way to do key management.	⇒	Designing a protocol agnostic to the key management mechanism.
Participating entities cannot typically establish trust upfront, using traditional mechanisms.	⇒	Flexibility in Trust Management. Third Party Trust.

...so here comes OpenID for Verifiable Credentials (OID4VC)!



Adoption (selected use-cases)

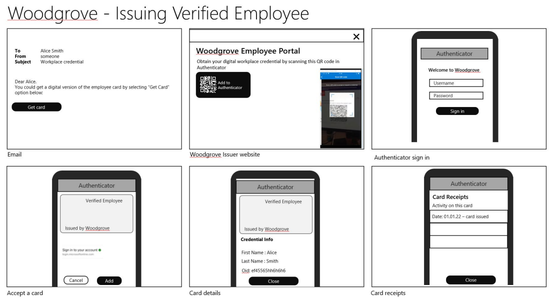
National eID
TSPs
attestation attributes
EU eID Wallet
National eID
Driving License
Diploma
Insurance
Travel
Payment
SELECTIVE DISCLOSURE
proofs of identity / attributes
SERVICE PROVIDERS
WWW
BANK


NIST | NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

SECURITY GUIDANCE OUR APPROACH NEWS & INSIGHTS GET INVOLVED SEARCH

Digital Identities - Mobile Driver's License (mDL)

Digital identities are supplementing and supplanting traditional physical identity cards. Customers, consumers of services, law enforcement, vendors, suppliers, and businesses may require a method of verifying a person via mobile device. This is not widely feasible due to the various, numerous and different technical implementations currently in place.


Woodgrove - Issuing Verified Employee

To Alice Smith
From someone
Subject Workplace credential

Dear Alice.
You could get a digital version of the employee card by selecting "Get Card" option below:

Get card

Email

Woodgrove Employee Portal
Obtain your digital workplace credential by scanning this QR code in Authenticator
Add to Authenticator

Woodgrove Issuer website

Authenticator
Welcome to Woodgrove
Username
Password
Sign in

Authenticator sign in

Authenticator
Verified Employee
Issued by Woodgrove
Sign in to your account
Cancel Add

Accept a card

Authenticator
Verified Employee
Issued by Woodgrove
Credential Info
First Name : Alice
Last Name : Smith
Oid: ef45565hh6h6h6
Close

Card details

Authenticator
Card Receipts
Activity on this card
Date: 01.01.22 – card issued
Close

Card receipts

The European Digital Identity Wallet[1]**, ARF v.1.3:** "the EUDI Wallet Solution MUST support OpenID4VCI as an Issuance protocol."

NIST National Cybersecurity Center of Excellence[2] is running a project implementing and testing implementations for OID4VP to present mdocs/mDL.

DIF JWT VC Issuance / Presentation Profile [3] [4] uses OID4VC protocols for the enterprise identity use-cases: fraud prevention in B2B, B2E scenarios.

[1] https://cloudsignatureconsortium.org/new-eu-eidas-regulation-a-quantum-leap-for-electronic-identity/ [2] https://www.nccoe.nist.gov/projects/digital-identities-mdl
[3] https://identity.foundation/jwt-vc-issuance-profile/ [4] https://identity.foundation/jwt-vc-presentation-profile/

Open Source libraries

1. Walt.id
   - https://github.com/walt-id/waltid-ssikit (Kotlin)
   - https://github.com/walt-id/waltid-openid4vc (Kotlin Multiplatform)
2. Sphereon
   - https://github.com/Sphereon-Opensource/SIOP-OpenID4VP (Typescript)
   - https://github.com/Sphereon-Opensource/OpenID4VCI-client (Typescript)
   - https://github.com/Sphereon-Opensource/ssi-sdk (Typescript)
3. Microsoft
   - https://github.com/microsoft/VerifiableCredential-SDK-Android (Kotlin)
   - https://github.com/microsoft/VerifiableCredential-SDK-iOS (Swift)
4. Spruce
   - https://github.com/spruceid/oidc4vci-rs (Rust)
   - https://github.com/spruceid/oidc4vci-issuer (Rust)
5. EBSI
   - https://api-pilot.ebsi.eu/docs/libraries (Javascript)
6. Impierce Technologies
   - https://github.com/impierce/openid4vc (Rust)
7. Animo
   - https://github.com/animo/paradym-wallet (Typescript)
8. Trustbloc
   - https://github.com/trustbloc/vcs (Go)
   - https://github.com/trustbloc/wallet-sdk (Go)
9. Italian Government
   - https://github.com/italia/eudi-wallet-it-python (Python)
   - https://github.com/italia/eudi-wallet-it-pid-provider/tree/ v.1.1.1 (Python)
10. AltMe
    - https://github.com/TalaoDAO/AltMe (Dart)
11. MOSIP
    - https://github.com/mosip/tuvali (Kotlin/Swift/ReactNative)
12. EUDI Reference Wallet Implementation
    - https://github.com/eu-digital-identity-wallet/.github/blob/ma n/profile/reference-implementation.md

OpenID Foundation Certification for OID4VC specs

- A light-weight, low-cost, self-certification program to serve members, drive adoption and promote high-quality implementations (since 2015~)
- 2,400+ total certifications to date!
- Benefits (there are more!)
  - Testers get direct support from the OIDF certification team
  - Internationally recognized, award winning
  - Updated as the specification evolves
- Current progress
  - Started development for OpenID for Verifiable Presentations. initial focus is on testing wallets.
  - OpenID for Verifiable Credential Issuance planned
- Things to know
  - Strictly tests protocol specification conformance and does not test what happens inside the wallet
  - Can be integrated in continuous development and deployment processes
  - Tests are open source

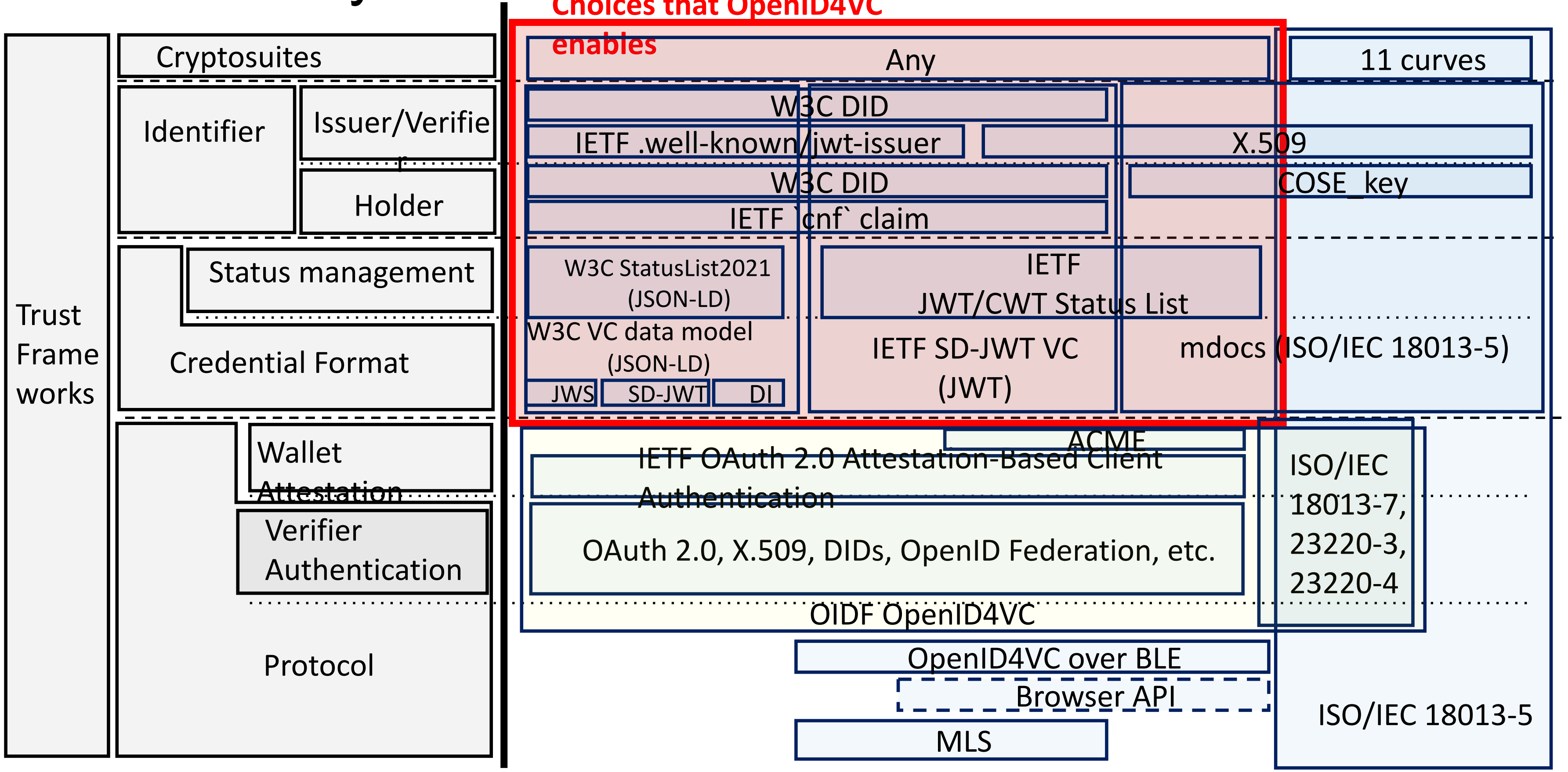
OID4VC Formal Security Analysis

- “Security and Trust in OpenID for Verifiable Credentials”
  - Describes the trust architecture in OpenID for Verifiable Credentials, outlines security considerations and requirements for the components in an ecosystem.
- Results of the formal security analysis of OpenID for VC protocols were also presented at the OAuth Security Workshop in August: “Protocols are secure under the assumptions made”.

Let us tell you more about the protocol

“OpenID for Verifiable Credentials” whitepaper

The World of “verifiable credentials”, in which OID4VCs allows variety of choices in the VC Tech Stack

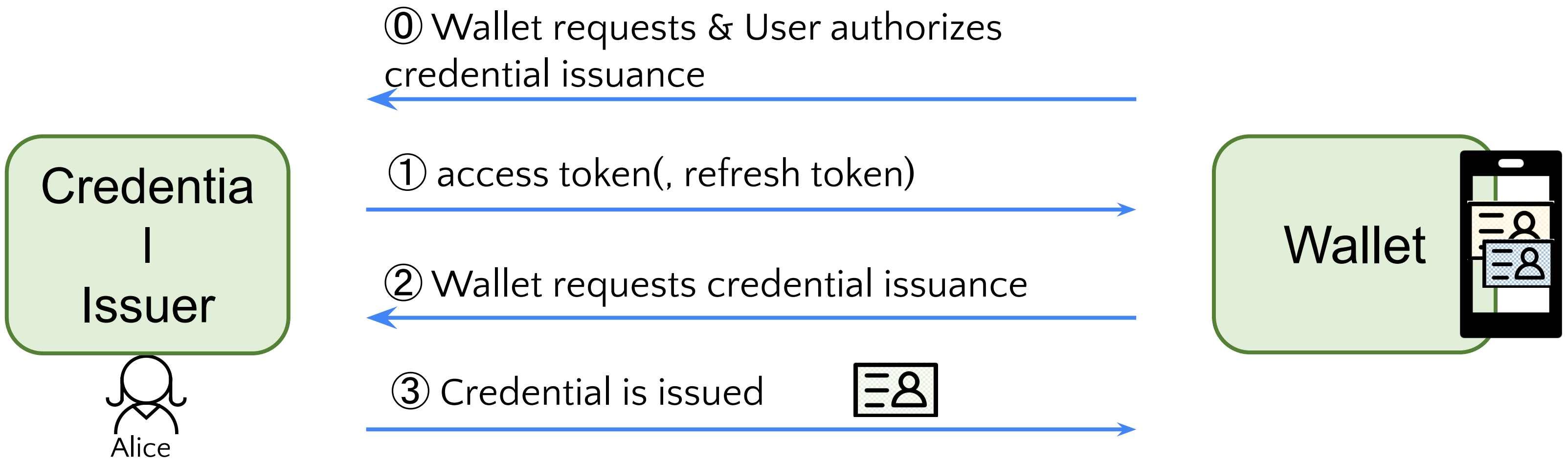


OpenID for Verifiable Credential Issuance

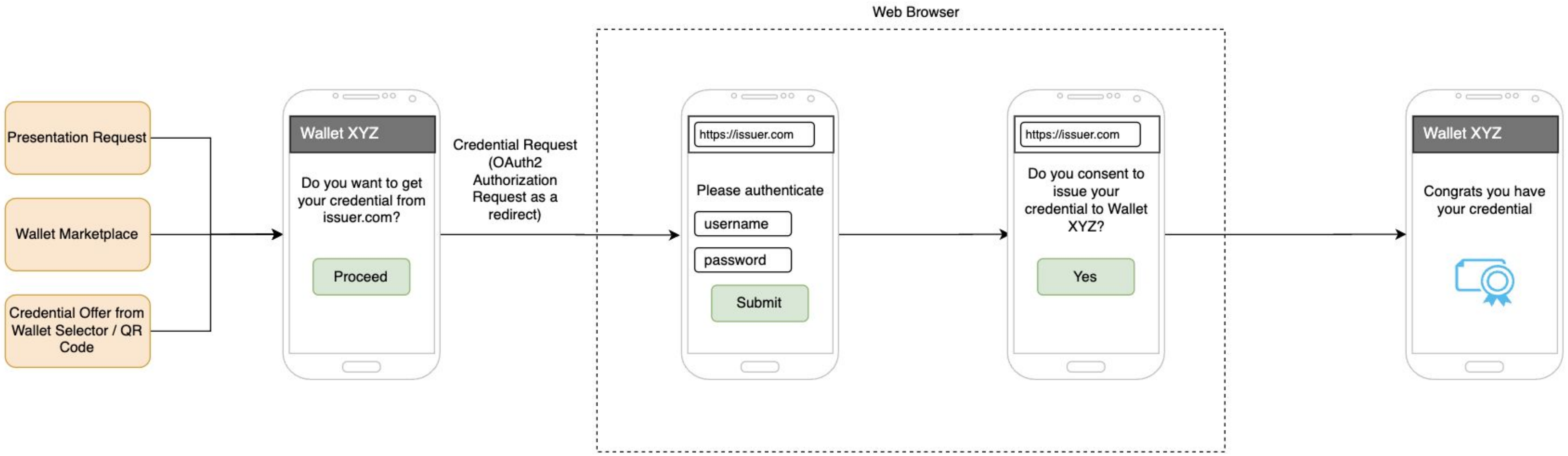
OpenID for Verifiable Credential Issuance (Highlights)

- First Implementer's draft published on April 1st!
- It's an OAuth-protected API
  - Leverages existing OAuth features and implementations
  - Easy of use for developers
- Supports various Security levels (including high security with hardware bound keys)
- Various business requirements supported (ex. remote and in-person provisioning)
- Different user-experiences can be achieved (multiple ways to initiate the flow)
- Issuer can check Wallet's capabilities & Wallet can discover Issuer metadata
- New Notification Endpoint - Wallet notifying the Issuer of un/successful issuance
- Open Batch Credential Endpoint request - response being updated

Protocol Flow

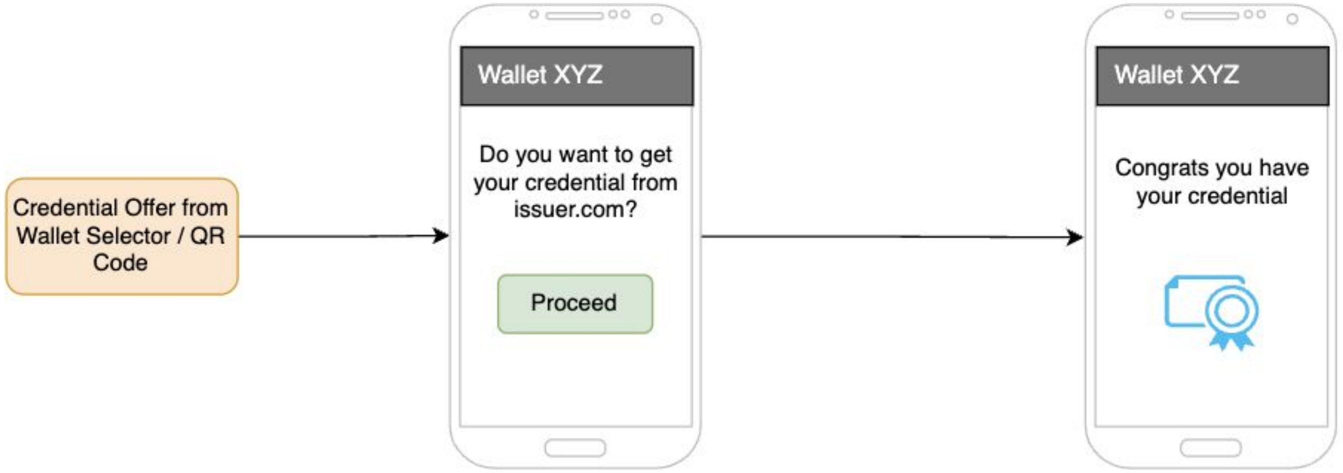

⓪ Wallet requests & User authorizes credential issuance
① access token(, refresh token)
② Wallet requests credential issuance
③ Credential is issued
Credentia
l
Issuer
Alice
Wallet

Authorization Code Flow

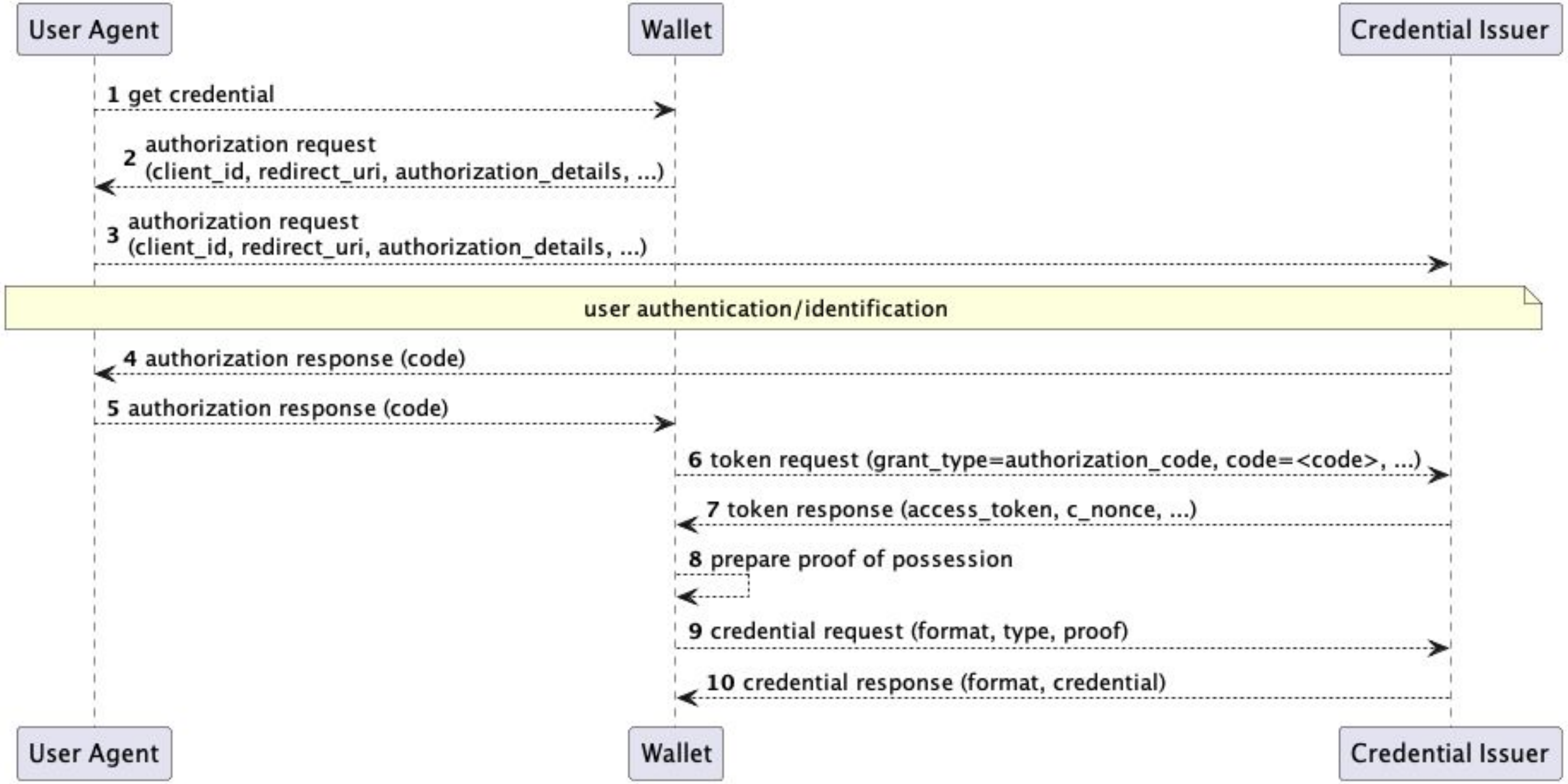


Web Browser
Presentation Request
Wallet Marketplace
Credential Offer from Wallet Selector / QR Code
Wallet XYZ
Do you want to get your credential from issuer.com?
Proceed
Credential Request (OAuth2 Authorization Request as a redirect)
https://issuer.com
Please authenticate
username
password
Submit
https://issuer.com
Do you consent to issue your credential to Wallet XYZ?
Yes
Wallet XYZ
Congrats you have your credential

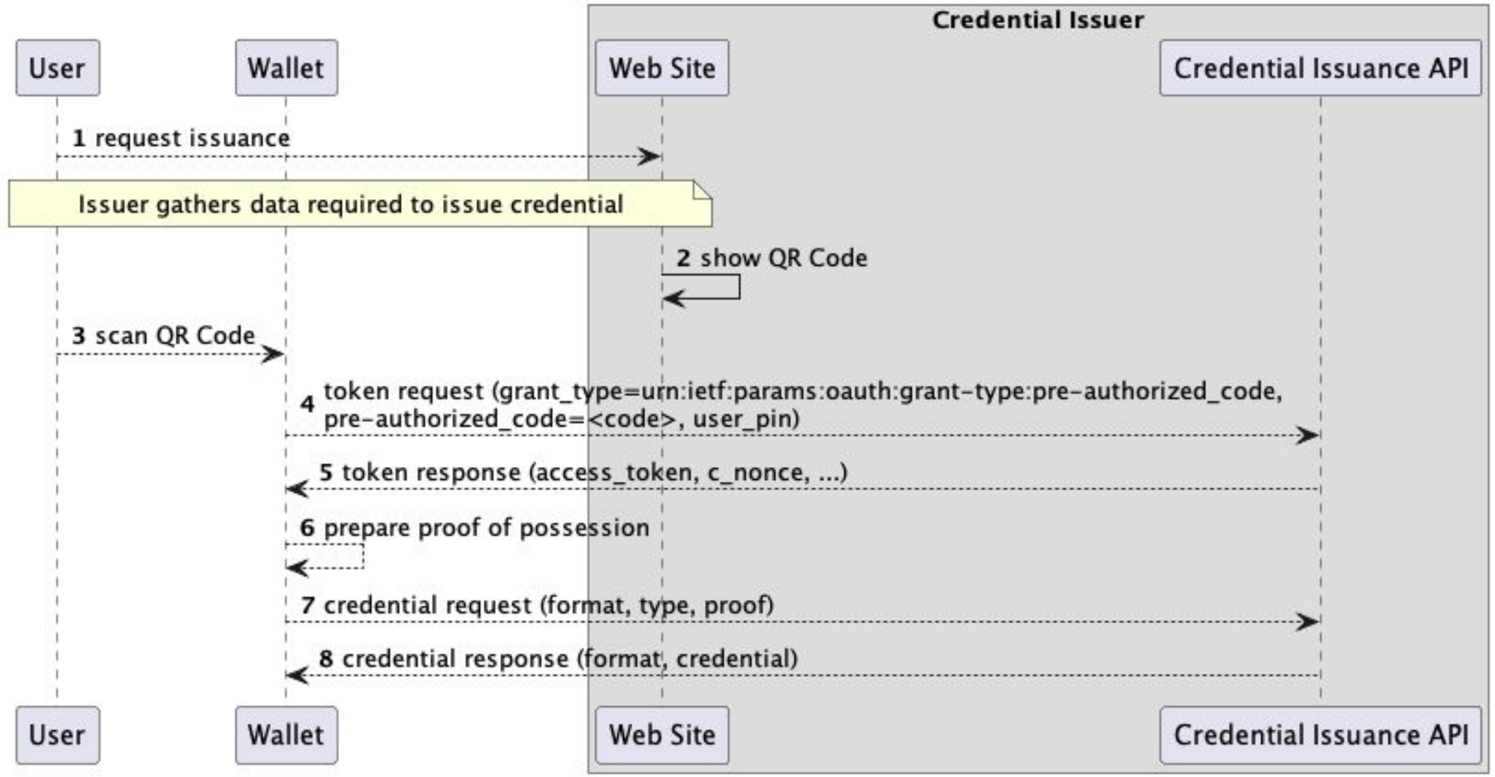
Pre-Authorized Code Flow


Credential Offer from Wallet Selector / QR Code
Wallet XYZ
Do you want to get your credential from issuer.com?
Proceed
Wallet XYZ
Congrats you have your credential

Authorization Code Flow (Overview)


User Agent
Wallet
Credential Issuer
1 get credential
2 authorization request (client_id, redirect_uri, authorization_details, ...)
3 authorization request (client_id, redirect_uri, authorization_details, ...)
user authentication/identification
4 authorization response (code)
5 authorization response (code)
6 token request (grant_type=authorization_code, code=<code>, ...)
7 token response (access_token, c_nonce, ...)
8 prepare proof of possession
9 credential request (format, type, proof)
10 credential response (format, credential)
User Agent
Wallet
Credential Issuer

Pre-Authorized Code Flow (Overview)


Credential Issuer
User
Wallet
Web Site
Credential Issuance API
1 request issuance
Issuer gathers data required to issue credential
2 show QR Code
3 scan QR Code
4 token request (grant_type=urn:ietf:params:oauth:grant-type:pre-authorized_code, pre-authorized_code=<code>, user_pin)
5 token response (access_token, c_nonce, ...)
6 prepare proof of possession
7 credential request (format, type, proof)
8 credential response (format, credential)
User
Wallet
Web Site
Credential Issuance API

Credential Issuer metadata (1/2)

```
{
    "credential_issuer": "https://credential-issuer.example.com",
    "authorization_servers": [ "https://server.example.com" ],
    "credential_endpoint": "https://credential-issuer.example.com",
    "batch_credential_endpoint":
"https://credential-issuer.example.com/batch_credential",
    "deferred_credential_endpoint":
"https://credential-issuer.example.com/deferred_credential",
    "credential_response_encryption": {
        "alg_values_supported" : [
            "ECDH-ES"
        ],
        "enc_values_supported" : [
            "A128GCM"
        ],
        "encryption_required": false
    },
```

```
"display": [
    {
        "name": "Example University",
        "locale": "en-US"
    },
    {
        "name": "Example Université",
        "locale": "fr-FR"
    }
],
```

Credential Issuer metadata (2/2)

```
{....
    "credential_configurations_supported": {
        "UniversityDegreeCredential": {
            "format": "jwt_vc_json",
            "scope": "UniversityDegree",
            "cryptographic_binding_methods_supported": [
                "did:example"
            ],
            "credential_signing_alg_values_supported": [
                "ES256"
            ],
            "credential_definition":{
                "type": [
                    "VerifiableCredential",
                    "UniversityDegreeCredential"
                ],
                "credentialSubject": {
                    "given_name": {
                        "display": [
                            {
                                "name": "Given Name",
                                "locale": "en-US"
                            }
```

```
            "proof_types_supported": {
                "jwt": {
                    "proof_signing_alg_values_supported": [
                        "ES256"
                    ]
                }
            },
            "display": [
                {
                    "name": "University Credential",
                    "locale": "en-US",
                    "logo": {
                        "url":
"https://university.example.edu/public/logo.png",
                        "alt_text": "a square logo of a university"
                    },
                    "background_color": "#12107c",
                    "text_color": "#FFFFFF"
                }
            ]
        }
    }
}
```

Credential Offer

```
openid-credential-offer://?credential_offer_uri=https%3A%2F%2Fserver%2Ee
xample%2Ecom%2Fcredential-offer.json
```

```
{
   "credential_issuer": "https://credential-issuer.example.com",
   "credential_configuration_ids": [
      "UniversityDegreeCredential",
      "org.iso.18013.5.1.mDL"
   ],
   "grants": {
      "urn:ietf:params:oauth:grant-type:pre-authorized_code": {
         "pre-authorized_code": "oaKazRN8I0IbtZ0C7JuMn5",
         "tx_code": {
            "length": 4,
            "input_mode": "numeric",
            "description": "Please provide the one-time code that was
sent via e-mail"
         }
      }
   }
}
```

Example: Credential Request - Authorization Details

```
GET /authorize?
  response_type=code
  &client_id=s6BhdRkqt3
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
&authorization_details=%5B%7B%22type%22%3A%20%22openid_credential%22%2C%20%
22credential_configuration_id%22%3A%20%22UniversityDegreeCredential%22%7D%5
D
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb

Host: server.example.com
```

Example: Credential Request - Scopes

```
GET /authorize?
  response_type=code
  &scope=UniversityDegreeCredential
  &resource=https%3A%2F%2Fcredential-issuer.example.com
  &client_id=s6BhdRkqt3
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
Host: server.example.com
```

```
Response

HTTP/1.1 302 Found
Location: https://Wallet.example.org/cb?
    code=SplxlOBeZQQYbYS6WxSbIA
```

Example: Token Request (authorized code)

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
&redirect_uri=https%3A%2F%2FWallet.example.org%2Fcb
```

Example: Token Request (pre-authorized code)

```
POST /token HTTP/1.1
Host: credential-issuer.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn:ietf:params:oauth:grant-type:pre-authorized_code
&pre-authorized_code=SplxlOBeZQQYbYS6WxSbIA
&tx_code=493536
```

Example: Token Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

  {
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6Ikp..sHQ",
    "token_type": "bearer",
    "expires_in": 86400,
    "c_nonce": "tZignsnFbp",
    "c_nonce_expires_in": 86400,
    "authorization_details": [
      {
        "type": "openid_credential",
        "credential_configuration_id": "UniversityDegreeCredential",
        "credential_identifiers": [ "CivilEngineeringDegree-2023",
"ElectricalEngineeringDegree-2023" ]
      }
    ]
  }
```

Example: Credential Issuance (format/type)

Request

```
POST /credential HTTP/1.1
Host: credential-issuer.example.com
Content-Type: application/json
Authorization: BEARER czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
   "format":"vc+sd-jwt",
   "vct":"Identity",
   "proof":{
      "proof_type":"jwt",
"jwt":"eyJhbGciOiJFUzI1NiIsInR5cCI6Im9wZW5pZDR2Y2ktcHJvb2Yrand0Iiw
...
jhe0xQmfIBCQz20xVjaM91ODdIt5JX_ztrcq4nkglH907Ofbugg"
   }
}
```

Response

```
HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store
  Pragma: no-cache

{
  "credential" : "eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIl
   ...
   gImVtYWlsIiwgInRlc3RAZXhhbXBsZS5jb20iXQ"
}
```

Example: Credential Issuance (identifier)

Request

```
POST /credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: BEARER czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
   "credential_identifier": "CivilEngineeringDegree-2023",
   "proof": {
      "proof_type": "jwt",
      "jwt":
"eyJ0eXAiOiJvcGVuaWQ0dmNpLXByb29mK2p3dCIsImFsZyI6IkVTMjU2IiwiandrI
jp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiblVXQW9BdjNYWml0aDhFN2k
xOU9kYXhPTFlGT3dNLVoyRXVNMDJUaXJUNCIsInkiOiJIc2tIVThCalVpMVU5WHFpN
1N3bWo4Z3dBS18weGtjRGpFV183MVNvc0VZIn19.eyJhdWQiOiJodHRwczovL2NyZW
```

Response

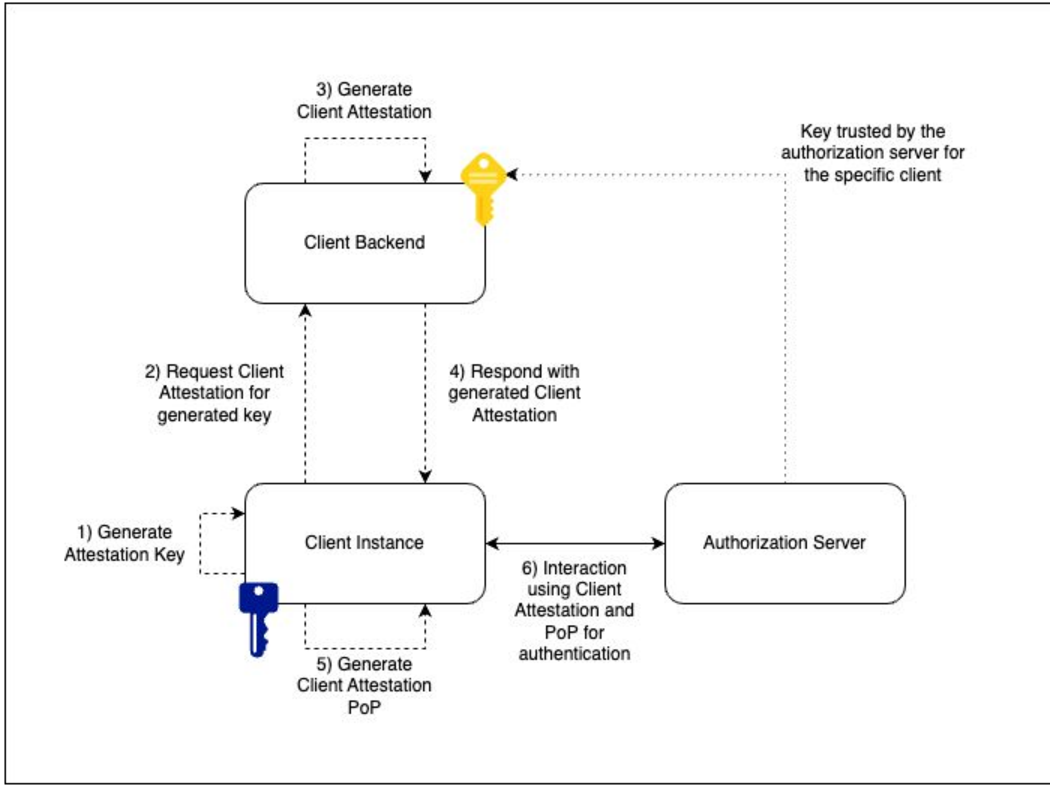
```
HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store
  Pragma: no-cache

{
  "credential" : "eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIl
   ...
   gImVtYWlsIiwgInRlc3RAZXhhbXBsZS5jb20iXQ"
}
```

Example: Issued Credential

```
{
 "iss": "https://credential-issuer.example.com",
 "iat": 1516239022,
 "exp": 1516247022,
 "vct": "https://credentials.example.com/identity_credential",
 "_sd": [
   "UiuRGkTW7e_5UQauGeQRQdF8u3WYevS4Fs0IuB_DgYM",
   "tmPlXq0MID-oRXbUNHyoVZrc9Qkm8cwJTohVyOVlUgQ",
   "vTz0JI103v4k4pKIloT83Yzi33L1SdZlWBPmsfJBefk"
 ],
 "_sd_alg": "sha-256",
 "cnf": {
   "jwk": {
     "kty": "EC",
     "crv": "P-256",
     "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
     "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
   }
 }
}
```

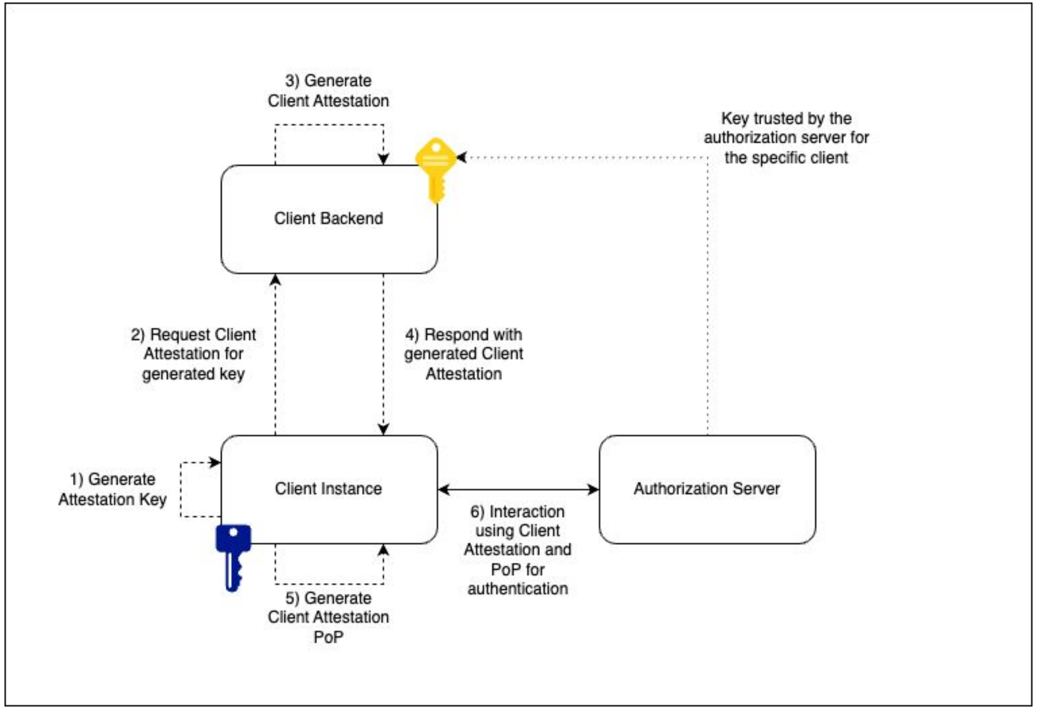
Wallet Attestation Architecture

- Differentiate Client and Client Instance
- Client Backend attests a Client Instance
- Client backend may perform any number of security checks before issuing a key-bound attestation JWT to the client instance, however, steps 2 and 4 are out of scope
    - Mechanisms of authentication
    - Issuance process
- Trust mechanism for the Client Backend public key is out of scope


3) Generate Client Attestation
Key trusted by the authorization server for the specific client
Client Backend
2) Request Client Attestation for generated key
4) Respond with generated Client Attestation
1) Generate Attestation Key
Client Instance
Authorization Server
6) Interaction using Client Attestation and PoP for authentication
5) Generate Client Attestation PoP

Wallet Attestation Architecture

- Proof of possession enabled client authentication method
- Can be used to authenticate the key used to bind to an access token via DPoP
- Direct mode of authentication between the client instance and the authorization server rather than a backend for front end pattern
- Avoids the client instance from having to register with the AS via DCR

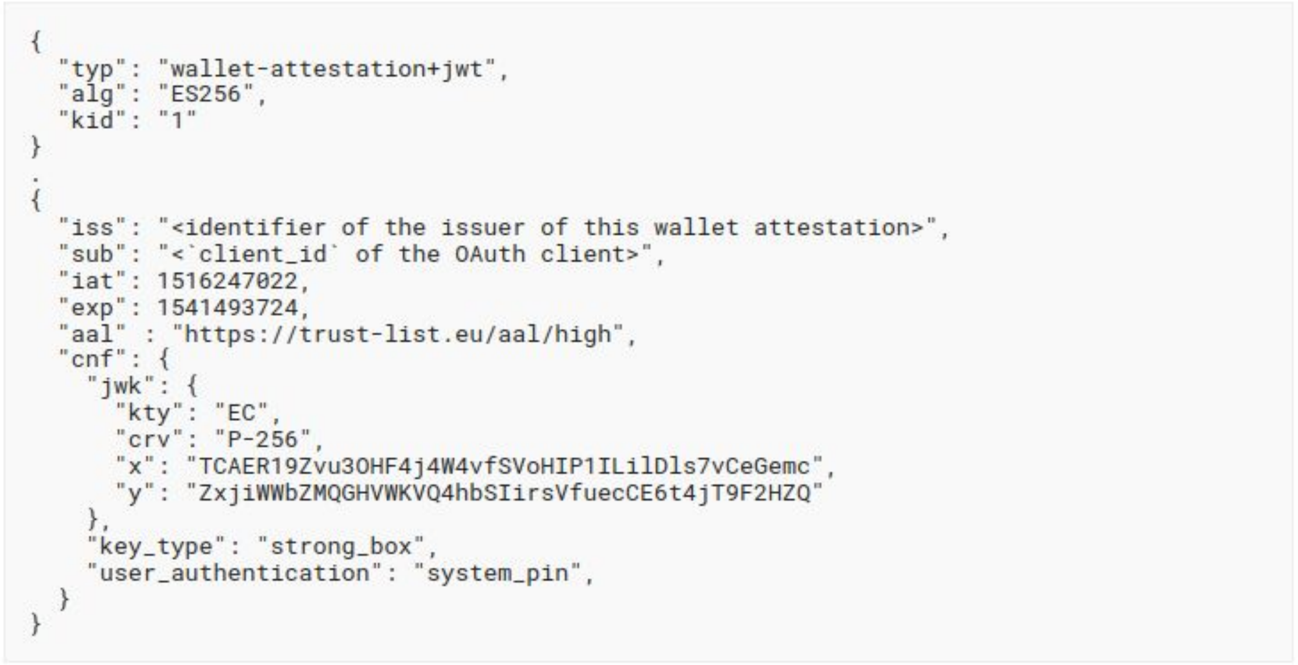

3) Generate Client Attestation
Key trusted by the authorization server for the specific client
Client Backend
2) Request Client Attestation for generated key
4) Respond with generated Client Attestation
1) Generate Attestation Key
Client Instance
Authorization Server
6) Interaction using Client Attestation and PoP for authentication
5) Generate Client Attestation PoP

Example - Wallet Attestation

```
{
    "alg": "ES256",
    "kid": "11"
}
.
{
  "iss": "https://client.example.com",
  "sub": "https://client.example.com",
  "nbf": 1300815780,
  "exp": 1300819380,
  … //other claims, e.g. key type, user authentication, LoA
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
      "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```

Key used to verify the Client Attestation PoP

Example of Wallet Attestation from HAIP



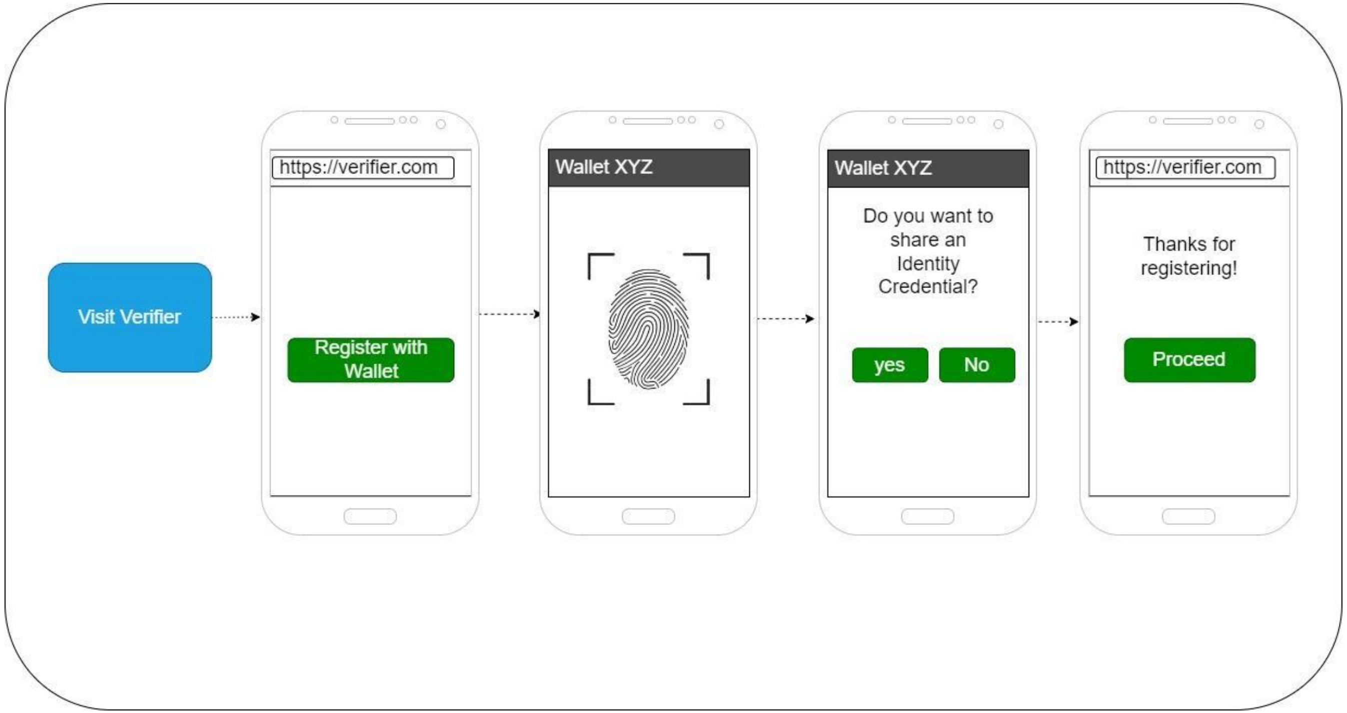
```
{
  "typ": "wallet-attestation+jwt",
  "alg": "ES256",
  "kid": "1"
}
.
{
  "iss": "<identifier of the issuer of this wallet attestation>",
  "sub": "<`client_id` of the OAuth client>",
  "iat": 1516247022,
  "exp": 1541493724,
  "aal" : "https://trust-list.eu/aal/high",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    },
    "key_type": "strong_box",
    "user_authentication": "system_pin",
  }
}
```

OpenID for Verifiable Presentations

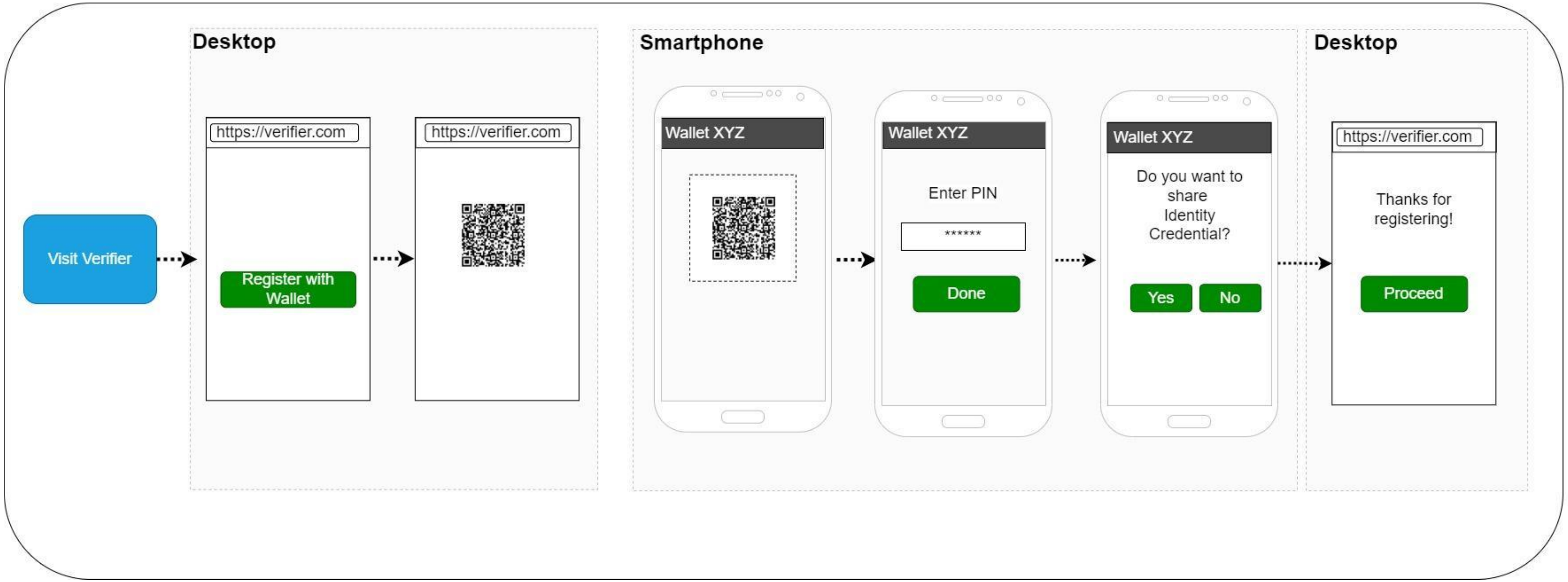
OpenID for Verifiable Presentations (Highlights)

- 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

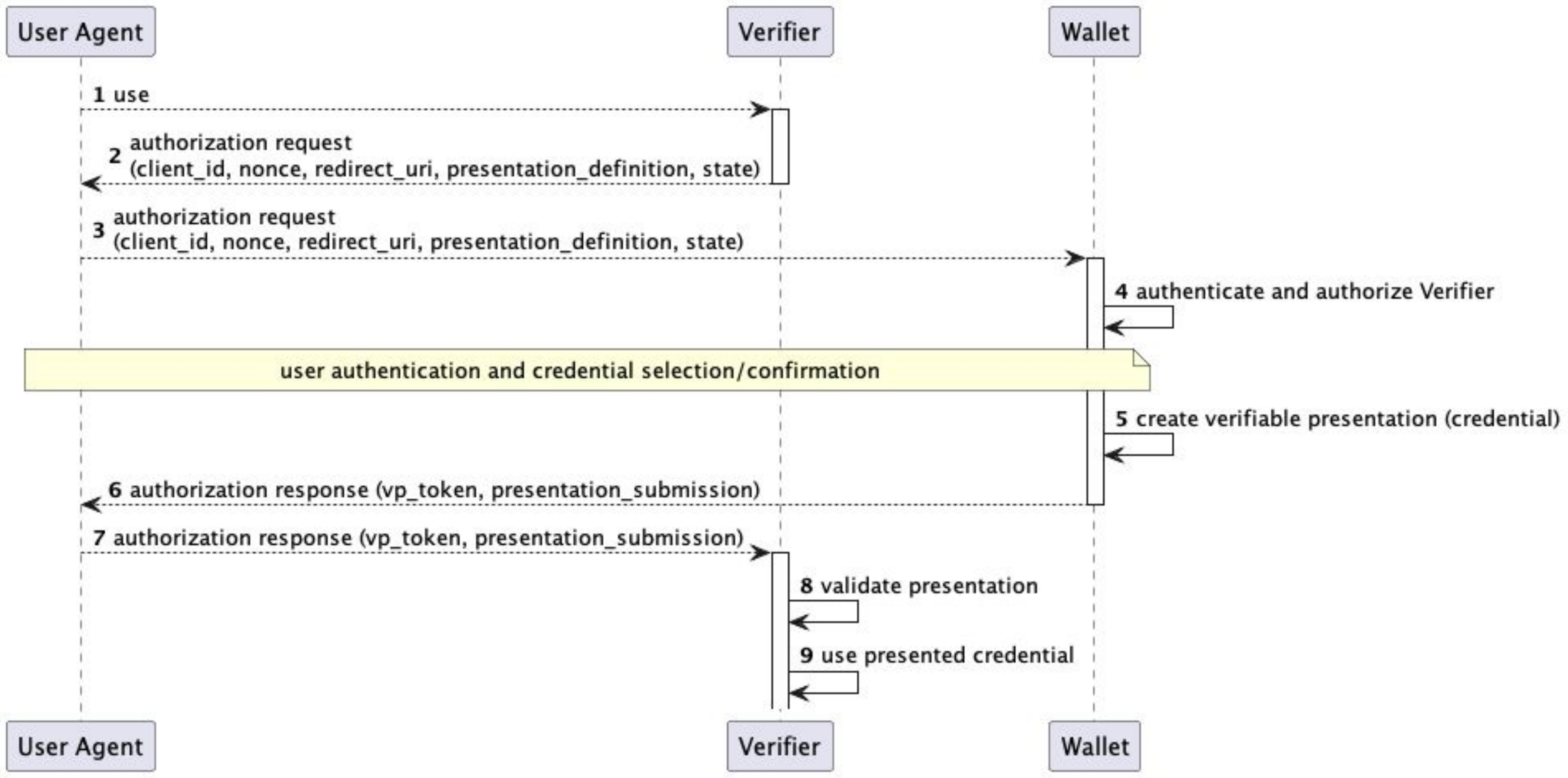
Same Device Presentation



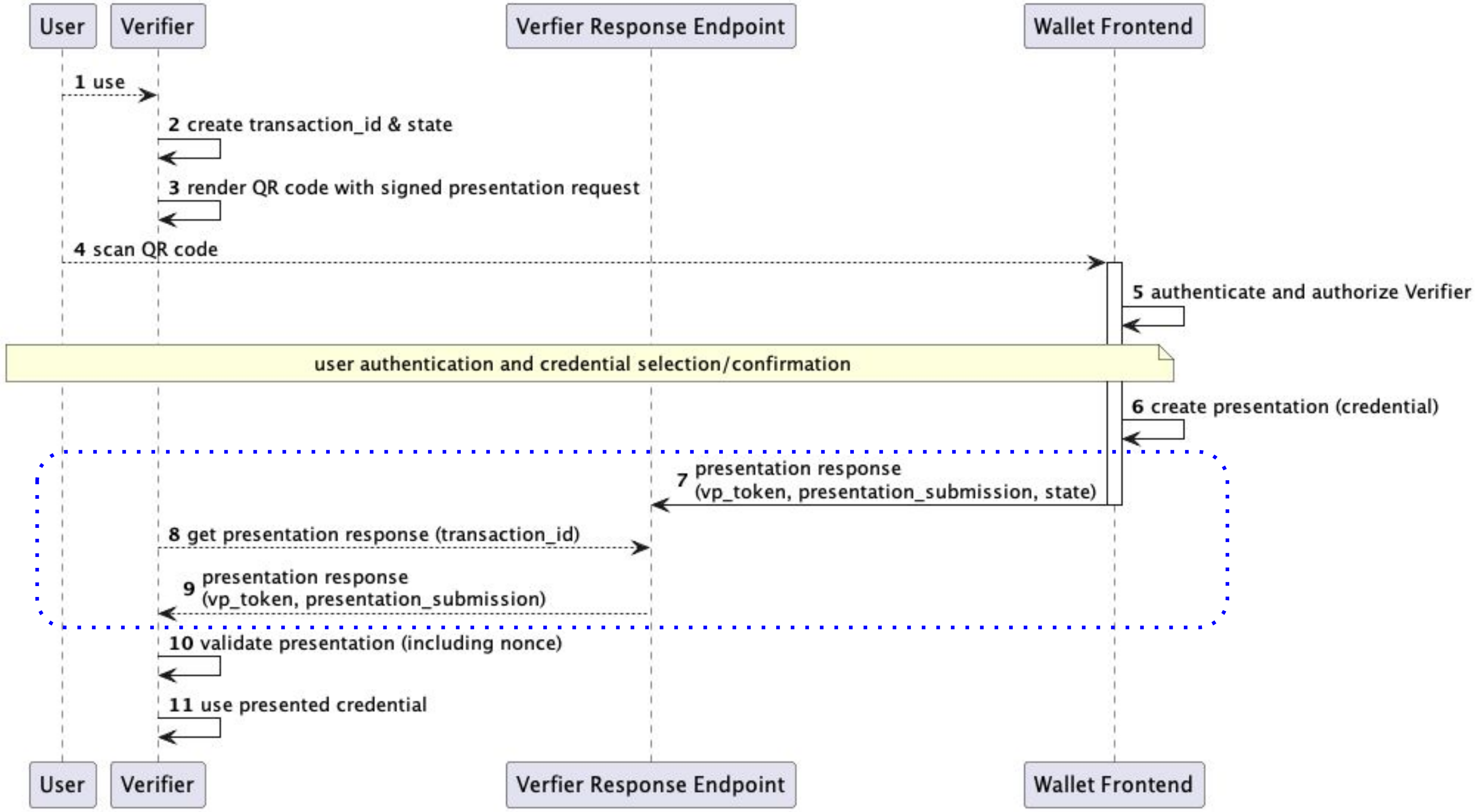
Cross Device Presentation



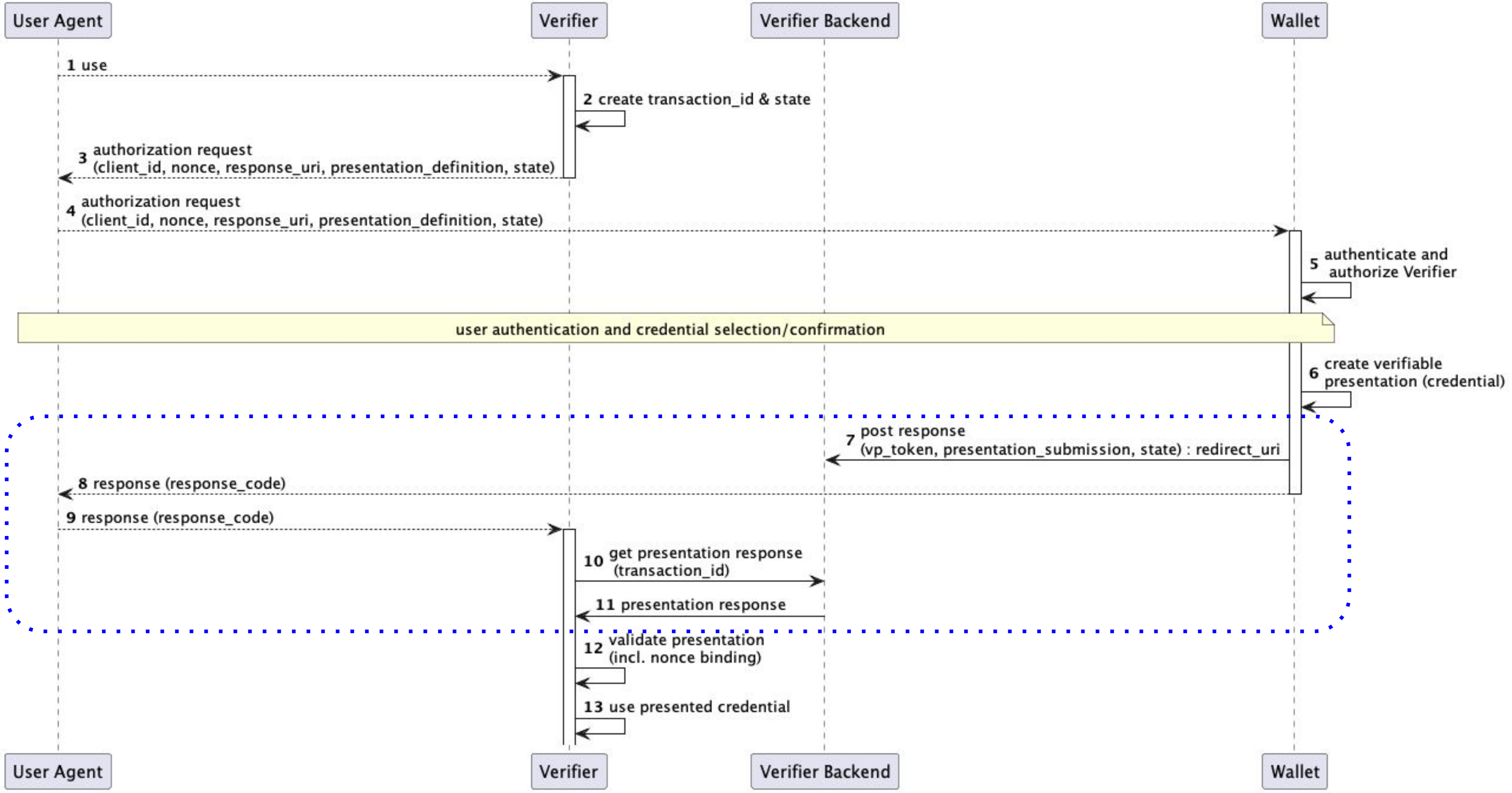
Same Device (Overview)


User Agent
Verifier
Wallet
1 use
2 authorization request (client_id, nonce, redirect_uri, presentation_definition, state)
3 authorization request (client_id, nonce, redirect_uri, presentation_definition, state)
4 authenticate and authorize Verifier
user authentication and credential selection/confirmation
5 create verifiable presentation (credential)
6 authorization response (vp_token, presentation_submission)
7 authorization response (vp_token, presentation_submission)
8 validate presentation
9 use presented credential
User Agent
Verifier
Wallet

Cross-Device Flow (VP Token sent via HTTP POST)


User
Verifier
Verfier Response Endpoint
Wallet Frontend
1 use
2 create transaction_id & state
3 render QR code with signed presentation request
4 scan QR code
5 authenticate and authorize Verifier
user authentication and credential selection/confirmation
6 create presentation (credential)
7 presentation response (vp_token, presentation_submission, state)
8 get presentation response (transaction_id)
9 presentation response (vp_token, presentation_submission)
10 validate presentation (including nonce)
11 use presented credential
User
Verifier
Verfier Response Endpoint
Wallet Frontend

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


User Agent
Verifier
Verifier Backend
Wallet
1 use
2 create transaction_id & state
3 authorization request (client_id, nonce, response_uri, presentation_definition, state)
4 authorization request (client_id, nonce, response_uri, presentation_definition, state)
5 authenticate and authorize Verifier
user authentication and credential selection/confirmation
6 create verifiable presentation (credential)
7 post response (vp_token, presentation_submission, state) : redirect_uri
8 response (response_code)
9 response (response_code)
10 get presentation response (transaction_id)
11 presentation response
12 validate presentation (incl. nonce binding)
13 use presented credential
User Agent
Verifier
Verifier Backend
Wallet

Presentation Request

```
GET /authorize?
  response_type=vp_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &presentation_definition=...
  &nonce=n-0S6_WzA2Mj HTTP/1.1
 Host: wallet.example.com
```

presentation_definition

```
{
    "id":"mDL-sample-req",
    "input_descriptors":[
        {
            "id":"org.iso.18013.5.1.mDL",
            "format":{
                "mso_mdoc":{
                    "alg":[
                        "EdDSA",
                        "ES256"
                    ]
                },
            },
            "constraints":{
                "limit_disclosure":"required",
                "fields":[
                    {
                        "path":[
                          "$['org.iso.18013.5.1']['family_name']"
                        ],
                        "intent_to_retain":false
                    }
                    {
                        "path":[
                   "$['org.iso.18013.5.1']['driving_privileges']"
                        ],
                        "intent_to_retain":false
                    }
                ]
            }
        }
    ]
```

Presentation Response

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb#
    presentation_submission=...
    &vp_token=...
```

presentation_submission

```
{
    "definition id": "mDL-sample-req",
    "id": "org.iso.18013.5.1.mDL",
    "descriptor_map": [
        {
            "id": "mDL",
            "format": "mso_mdoc",
            "path": "$"
        }
    ]
}
```

vp_token

```
{
    "status": 0,
    "version": "1.0",
    "documents": [
        {
            "docType": "org.iso.18013.5.1.mDL",
            "deviceSigned": {
                "deviceAuth": {
                    "deviceMac": [
                        << {1: 5} >>,
                        {},
                        null, h'A574C64F18902BFE18B742F17C581218F88EA279A
                    ]
                },
                "nameSpaces": 24(h'A0')
            },
            "issuerSigned": {
                "issuerAuth": [
                  << {1: -7} >>,
                    {
                        33:
h'30820215308201BCA003020102021404AD06A30C1A6DC6E93BE0E2E8F78DCAFA7907C23
5040613025A453059301306072A8648CE3D020106082A8648CE3D030107034200047C5545
E2000E9C46618C02202C1F778AD252285ED05D9B55469F1CB78D773671F30FE7AB8153719
                    },
                    <<
                      24(<<
                         {
                           "docType": "org.iso.18013.5.1.mDL",
                           "version": "1.0",
```

New request_uri method POST

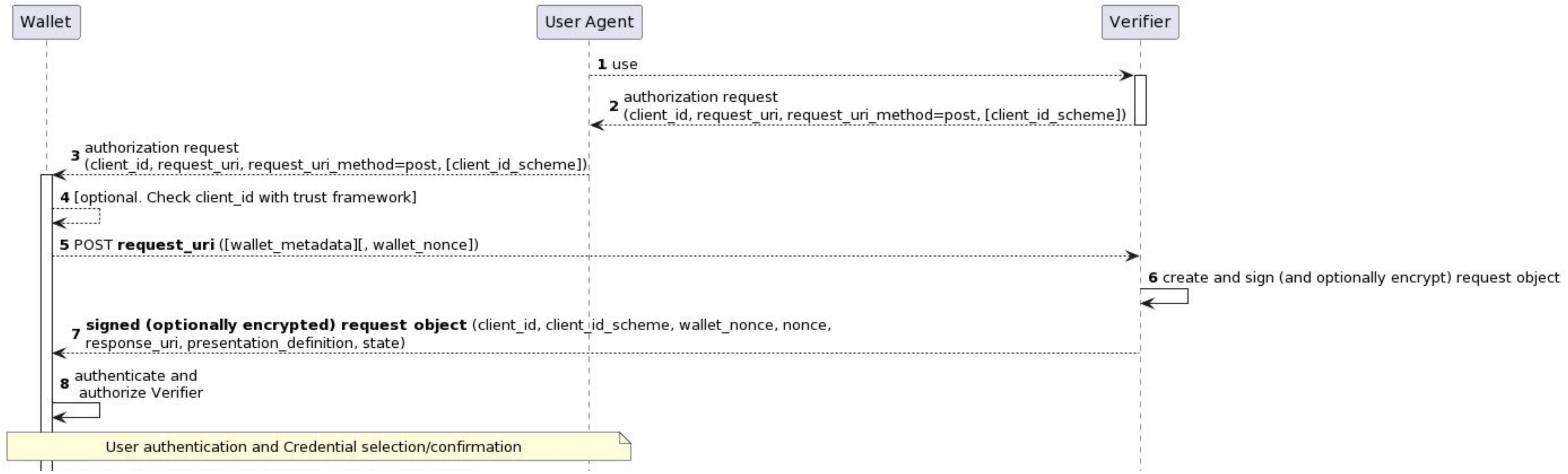
- A new mechanism that allows the Wallet to provide to the Verifier details about its technical capabilities. This enables the Verifier to generate a request that matches the technical capabilities of that Wallet.
- New request_uri_method Authorization Request parameter is introduced. When the value of request_uri_method is `post`, the Wallet can make an HTTP POST request to the Verifier's request_uri endpoint with information about its capabilities
- When request_uri_method is absent or has the value of `get`, or the Wallet does not support new POST method, the Wallet continues with JWT-Secured Authorization Request (JAR) [RFC9101].

Example: Authorization Request with request_uri_method POST

Request

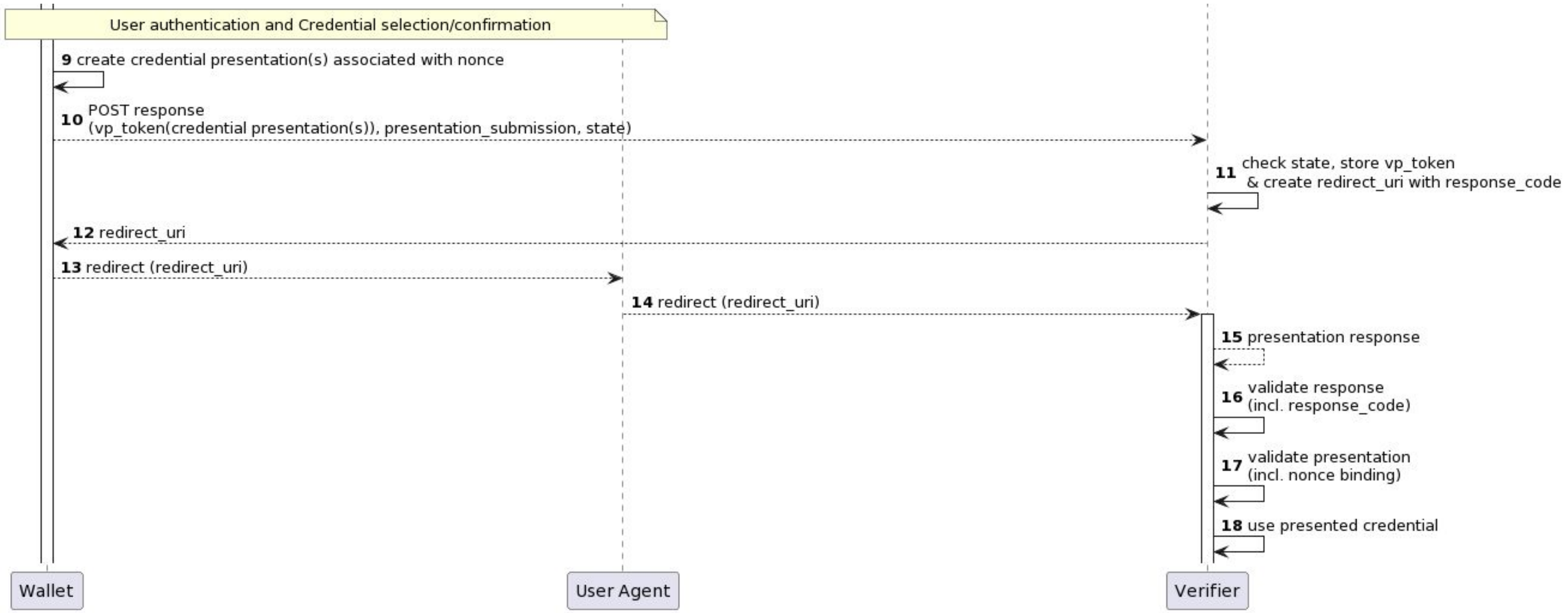
```
  GET /authorize?
    client_id=client.example.org
    &client_id_scheme=x509_san_dns
    &client_metadata=...

&request_uri=https%3A%2F%2Fclient.example.org%2Frequest%2Fvapof4ql
2i7m41m68uep
    &request_uri_method=post
```

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

request_uri_method = post (1/2)

Same Device (Request URI POST + Direct POST + redirect)


User authentication and Credential selection/confirmation
9 create credential presentation(s) associated with nonce
10 POST response (vp_token(credential presentation(s)), presentation_submission, state)
11 check state, store vp_token & create redirect_uri with response_code
12 redirect_uri
13 redirect (redirect_uri)
14 redirect (redirect_uri)
15 presentation response
16 validate response (incl. response_code)
17 validate presentation (incl. nonce binding)
18 use presented credential
Wallet
User Agent
Verifier

OpenID4VP over Browser API

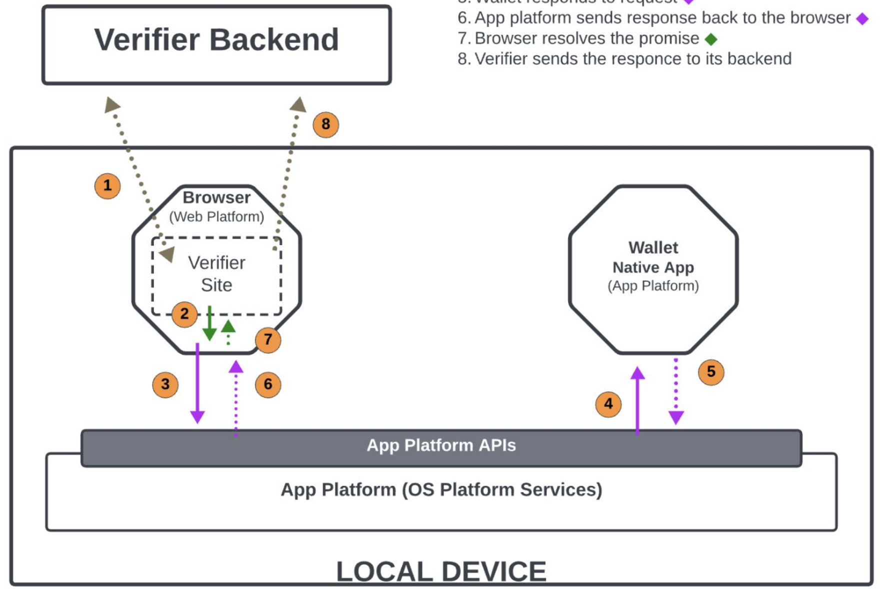
Why?

- Getting rid of custom schemes in favor of a flexible and privacy preserving model for Wallet selection based on 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 native app) that can be used as additional data point by the Wallet

Browser API Overview

SCENARIO
same-device
web-based verifier
native app wallet

1. Verifier site loaded in browser, request initiated
2. Web platform API request initiated ◆
3. Browser processes request and routes to the app platform ◆
4. App platform processes request and routes to wallet ◆
5. Wallet responds to request ◆
6. App platform sends response back to the browser ◆
7. Browser resolves the promise ◆
8. Verifier sends the responce to its backend



standardized API (W3C) 🟩	platform-specific function API 🟪	protocol-specific 🟫
standardized API (Other) 🟦	platform-specific web translation API 🟥	

OpenID4VP over Browser API proposal: unsigned request

```
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": {...}
          })
     }]
   }
});
```

this is an OID4VP request

new client id scheme

Standard OID4VP Request

The Wallet receives

- 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)

Note: At the minimum, the Wallet gets the calling origin to identify the Verifier.

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.

When external trust establishment mechanism is needed

- Request is signed, using external trust establishment mechanisms
  - Wallet validates the signature
  - Wallet needs to be able to establish trust in the verifier (e.g. know the root cert, etc.)
- How replay is prevented:
  - Verifiers signs 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.)

OpenID4VP over Browser API proposal: signed request

```
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",
            "response_type": "vp_token",
            "nonce": "n-0S6_WzA2Mj",
            "client_metadata": {...},
            "presentation_definition":"...",
               })
         }]
      }
});
```

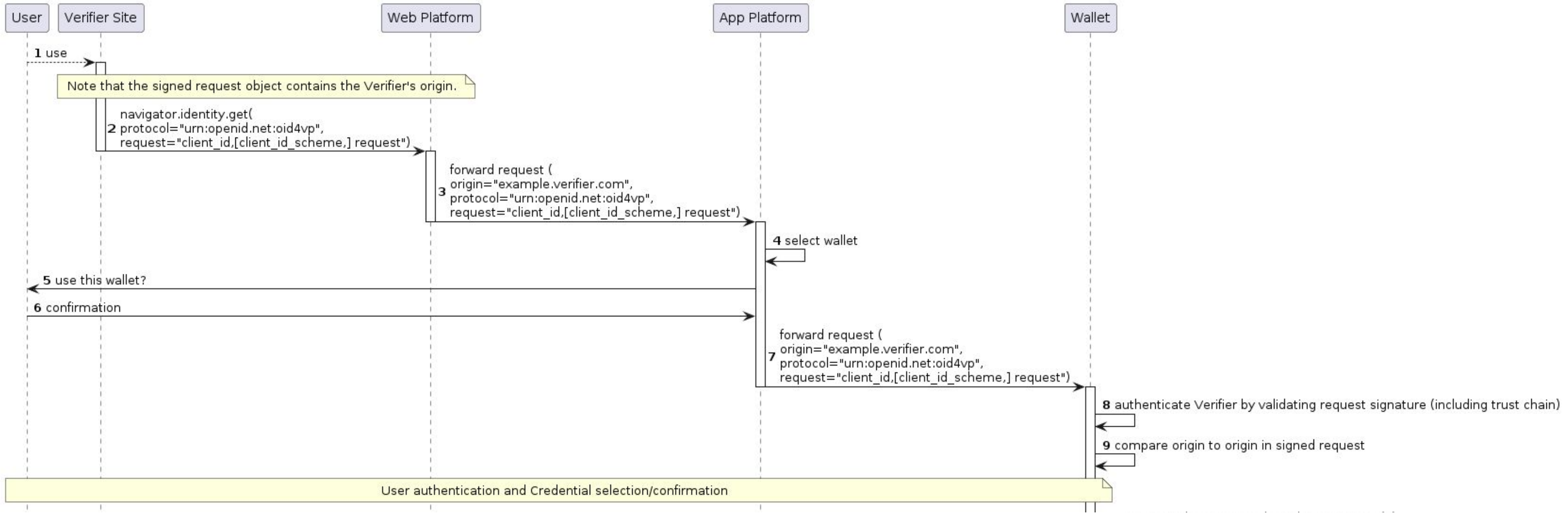
Standard OID4VP Request

this is an OID4VP request

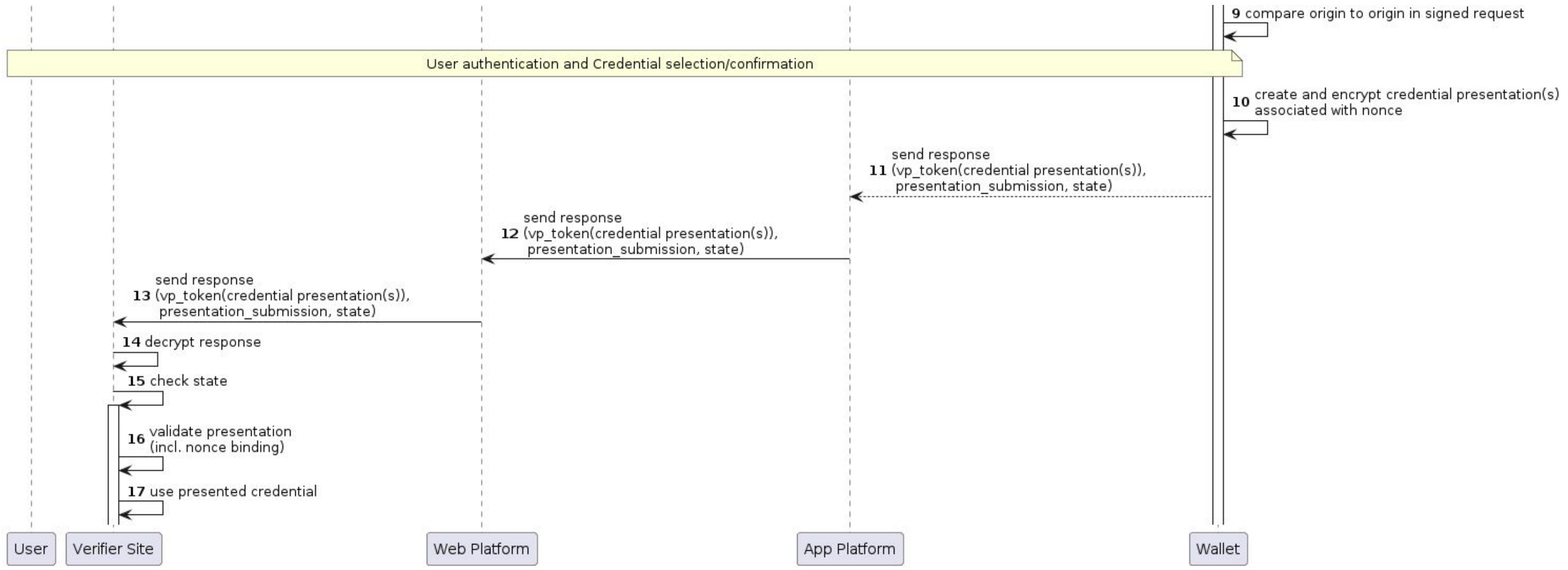
contains Verifier’s public key used to encrypt the response

Array. can contain multiple requests.

Request URI (signed request)


User
Verifier Site
Web Platform
App Platform
Wallet
1 use
Note that the signed request object contains the Verifier's origin.
2 navigator.identity.get(
protocol="urn:openid.net:oid4vp",
request="client_id,[client_id_scheme,] request")
3 forward request (
origin="example.verifier.com",
protocol="urn:openid.net:oid4vp",
request="client_id,[client_id_scheme,] request")
4 select wallet
5 use this wallet?
6 confirmation
7 forward request (
origin="example.verifier.com",
protocol="urn:openid.net:oid4vp",
request="client_id,[client_id_scheme,] request")
8 authenticate Verifier by validating request signature (including trust chain)
9 compare origin to origin in signed request
User authentication and Credential selection/confirmation

Request URI (signed request)


9 compare origin to origin in signed request
User authentication and Credential selection/confirmation
10 create and encrypt credential presentation(s) associated with nonce
send response 11 (vp_token(credential presentation(s)), presentation_submission, state)
send response 12 (vp_token(credential presentation(s)), presentation_submission, state)
send response 13 (vp_token(credential presentation(s)), presentation_submission, state)
14 decrypt response
15 check state
16 validate presentation (incl. nonce binding)
17 use presented credential
User
Verifier Site
Web Platform
App Platform
Wallet

Response

- The wallet prepares the vp_token and presentation_submission
- The wallet 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.