Verifiable Credentials: Concepts to Practice

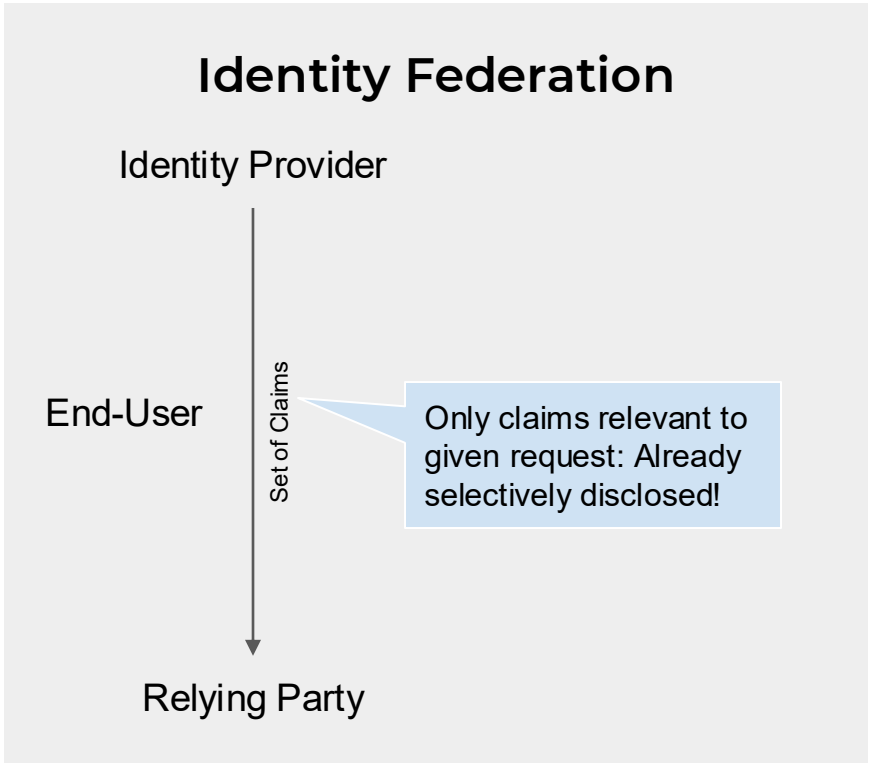
Kristina Yasuda

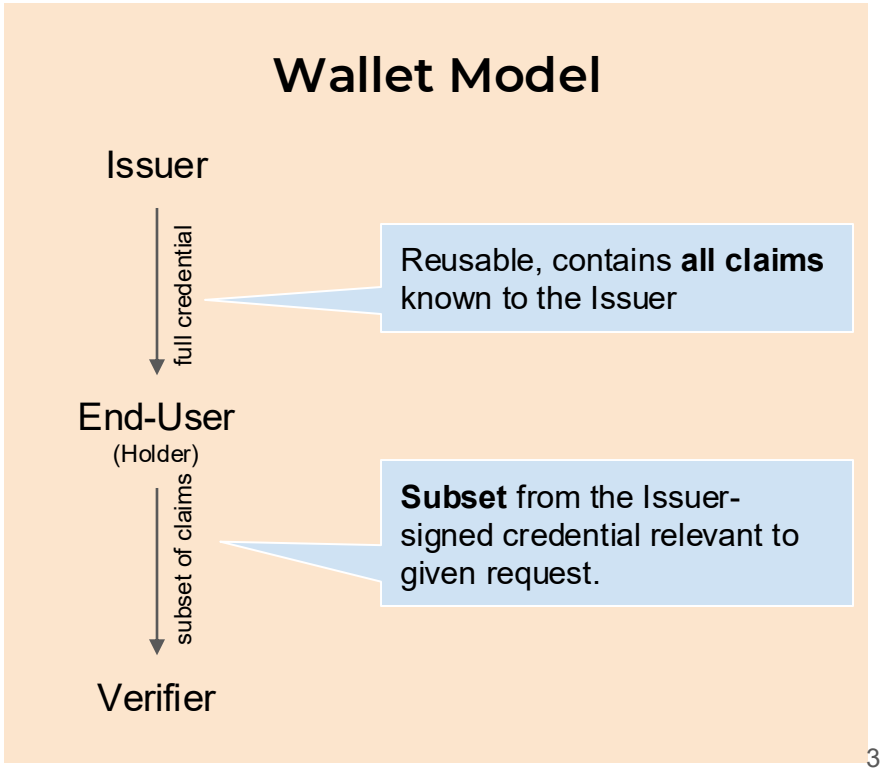
Agenda

- Context
- Technical Stack Overview
- Demo
- Credential Formats (SD-JWT VC and mdocs)
- Protocol: OpenID for Verifiable Credentials
  - Presentation: DCQL, transaction data, the Digital Credentials API
  - Issuance: Batch Issuance, key attestations
  - Interoperability Profile
- Q&A

Context

A New Model: Credential Issuance & Presentation Decoupled





Verifiable Credentials: Benefits

- **Enhanced privacy and portability for end-users** over their identity information.
- **Faster, cheaper, and more secure verification**: Digitally issued credentials reduce costs and delays associated with physical documents, while improving resistance to fraud.
- **Universal approach to handle identification, authentication, and authorization** in digital and physical space, hopefully across platforms, sectors, and borders.

Global Adoption (selected use-cases)

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

mandates the usage of OpenID4VC protocols

NIST National Cybersecurity Center of Excellence[2]

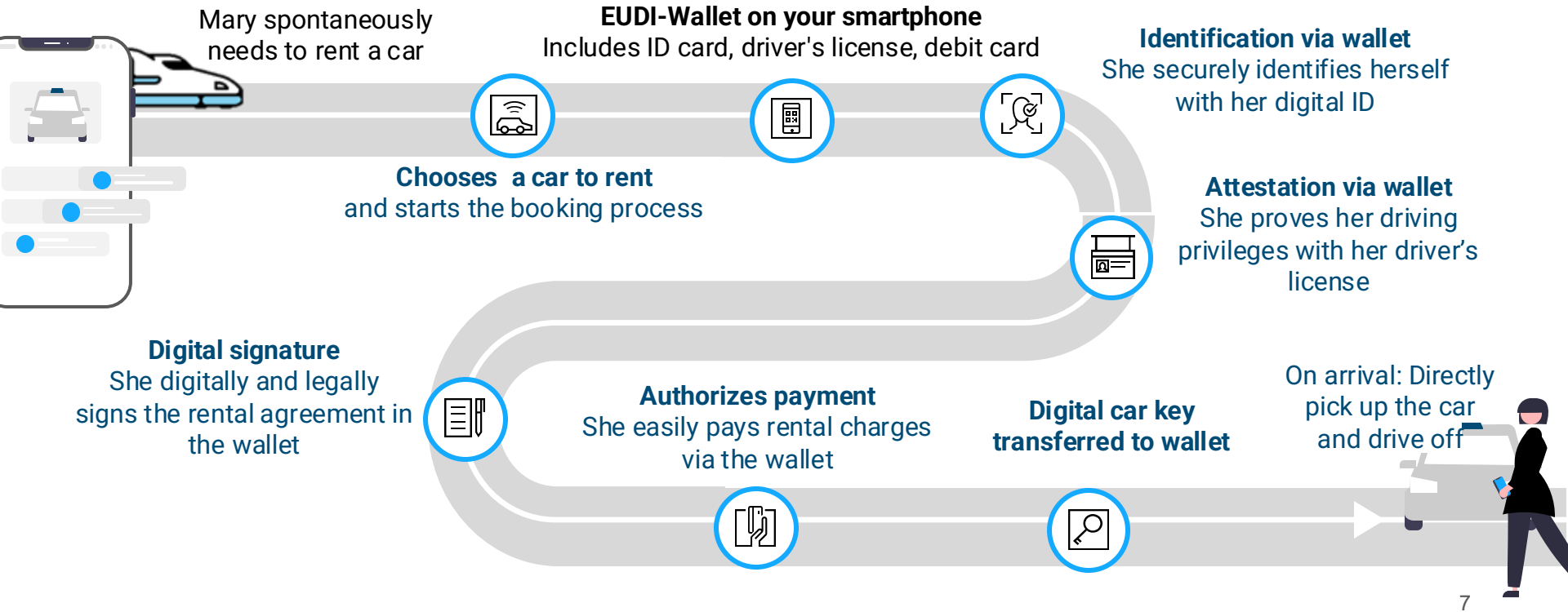
is running a project implementing and testing implementations for OID4VP to present mdocs/mDL

Japanese Government's Trusted Web Project [3]

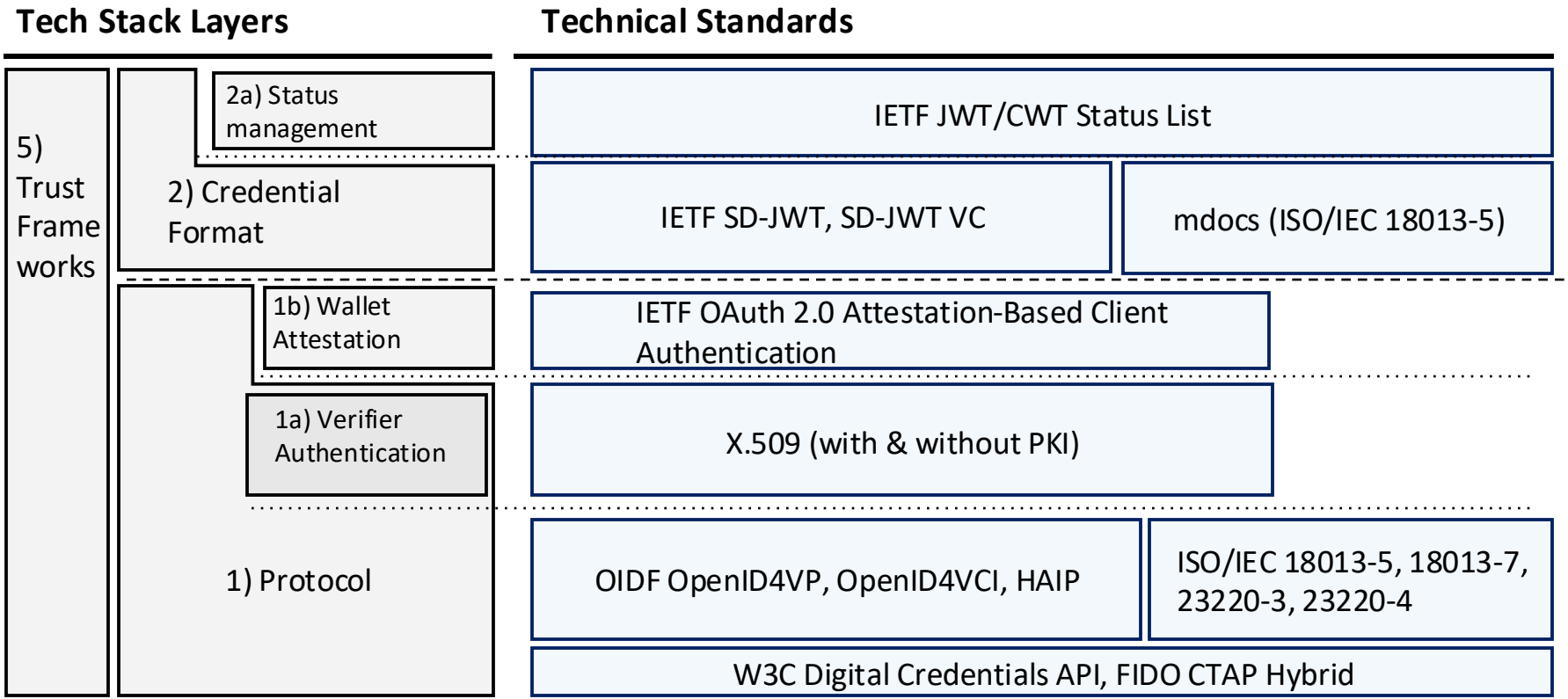
has implemented OID4VC protocols various use-cases

[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

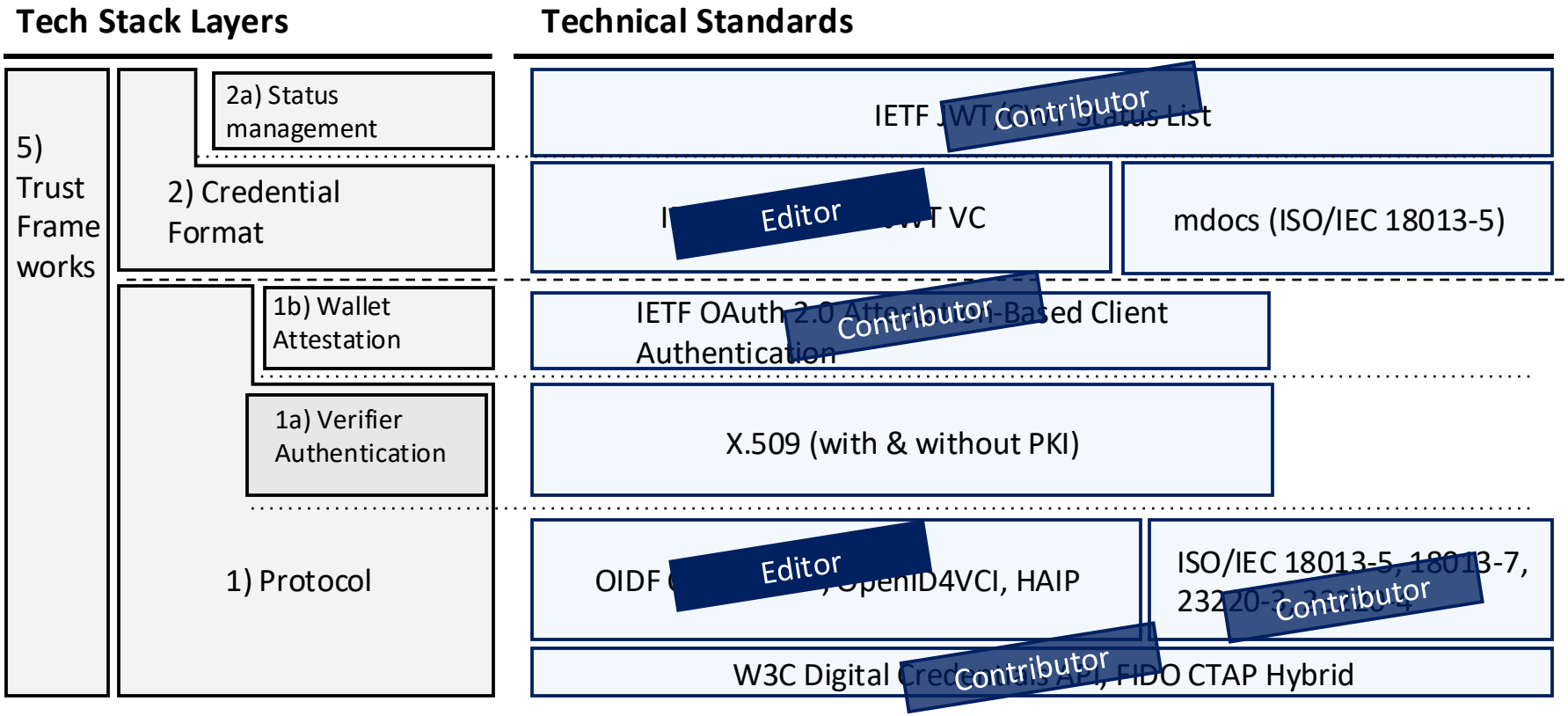
Renting a car easily with the EUDI Wallet



Main* Standards for Data Formats and Protocols



Main* Standards for Data Formats and Protocols



Standards Bodies 101

- ★ IETF — Internet Engineering Task Force
- ★ W3C — World Wide Web Consortium
- ★ OIDF — OpenID Foundation
- ★ ISO — International Organization for Standardization
- ★ ETSI — European Telecommunications Standards Institute
- ★ FIDO — Fast Identity Online

Credential Formats

In the IETF OAuth Working Group:

- ➔ **SD-JWT** (Selective Disclosure JSON Web Token) — basic data format, encoding
  essential building block, very close to final
- ➔ **SD-JWT VC** — how to create credentials based on SD-JWT
  essential building block, work in progress

Protocols

OpenID Foundation (Digital Credentials Protocols WG):

- ➔ **OpenID for Verifiable Presentations** — based on OAuth 2.0
  essential building block, moving to first final version (1.0)
- ➔ **OpenID for Verifiable Credential Issuance** — based on OAuth 2.0
  essential building block, moving to first final version (1.0)
- ➔ **Self-Issued OpenID Provider v2 (SIOP v2)**
- ➔ **OpenID4VC High Assurance Interoperability Profile** (HAIP)

W3C (Web Incubation Community Group)

- ➔ **Digital Credential API**
  can become an essential building block for ensuring security and good UX

Other Mechanisms

IETF OAuth Working Group:

- ➔ **Status List**
  essential building block for status management
- ➔ **Attestation-Based Client Authentication**
  essential building block for trusting the Wallet

Security

IETF OAuth Working Group:

- ➔ **RFC 9449: DPoP** (Sender-constrained Access Token)
  essential building block, final, deployed & tested
- ➔ **OAuth Security BCP**
  How not to use OAuth.
- ➔ **Cross-Device Flows: Security Best Current Practice**
  Can inform decisions on cross-device flows

OpenID Foundation:

- ➔ **Security and Trust in OpenID for Verifiable Credentials**

Demo

(Presentation of a German PID using German Government’s official EUDI Wallet)

Credential Formats

SD-JWT & SD-JWT VC 101

IETF SD-JWT & SD-JWT VC standards

- Formats for
  - enabling **selective disclosure and key binding for JWS/JWT** (SD-JWT)
  - **credentials** based on that format (SD-JWT VC)
- Attributes are structured as JSON

SD-JWT

Selective Disclosure for JWTs
using a simple, salted-hash based format
— for verifiable credentials and more.

IETF Draft: https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/

Daniel Fett
Kristina Yasuda
Brian Campbell

Selective Disclosure

Issuer issued a whole set of claims:


```
{
  "iss": "https://server.example.com",
  "sub": "some-user-identifier",
  "aud": "s6BhdRkqt3",
  "given_name": "John",
  "family_name": "Doe",
  "email": "johndoe@example.com",
  "phone_number": "+1-202-555-0101",
  "address": {
    "street_address": "123 Main St",
    "locality": "Anytown",
    "region": "Anystate",
    "country": "US"
  },
  "birthdate": "1940-01-01"
}
```


✓ signed by Issuer

It's not the user who selects a subset during presentation

But **Verifier** only needs a subset in a given request:


```
{
  "iss": "https://server.example.com",
  "sub": "some-user-identifier",
  "aud": "s6BhdRkqt3",
  "given_name": "John",
  "family_name": "Doe",
  "email": "johndoe@example.com",
  "phone_number"
  "address": {
  },
```


✓ signed by Issuer

Step 1: Prepare User Data


```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },

 "given_name": "Max",
 "family_name": "Mustermann",
 "email": "mustermann@example.com",
 "address": {
   "street_address": "Musterstr. 23",
   "locality": "Berlin",
   "country": "DE"
  }
}
```

Step 2: Create Disclosures

```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },

 "given_name": "Max",          ····· ["GO0r26nO-iW50ZcAoOilFw", "given_name", "Max"]
 "family_name": "Mustermann",  ····· ["cSIbR135i0NjhsouMxrjjg", "family_name", "Mustermann"]
 "email": "mustermann@example.com", ····· ["oHDt43Vwuhpo8mzaprgCcw", "email", "mustermann@example.com"]
 "address": {
   "street_address": "Musterstr. 23", ····· ["rGc0KtY6WmflywTTKEWIEQ", "street_address", "Musterstr. 23"]
   "locality": "Berlin",       ····· ["pGQMQx-2tH2XwC_eQCFn4g", "locality", "Berlin"]
   "country": "DE"             ····· ["TI15M8G5UIxPiWNZ-VLYBA", "country", "DE"]
  }
}
```

salt claim name claim value

Step 3: Hash Disclosures & Replace Original Claims

```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },

   "_sd": [ "EW1o0egqa5mGcbytT5S-kAubcEjYEUwRkXlu2vC5l20",      ← ["GO0r26nO-iW50ZcAoOilFw", "given_name", "Max"]
            "FEx-ITHt41I8_cn0SS-hvoLneX_RGIJo_8o2xRNhfdk",      ← ["cSlbR135i0NjhsouMxrjjg", "family_name", "Mustermann"]
            "igg7H5fn2eBEMIEkE5Ckbm23QuwDJITYoKRip08dYIc" ],     ← ["oHDt43Vwuhpo8mzaprgCcw", "email", "mustermann@example.com"]
 "address": {
   "_sd": [ "gqB5kmAwyry88aHjaAeO-USX6JOMaojukKsheo38O0c",      ← ["rGc0KtY6WmflywTTKEWIEQ", "street_address", "Musterstr. 23"]
            "w8InvxsPXdKoowuVpyBMgl1b9_R2b6Xpa3OYOljgQro",      ← ["pGQMQx-2tH2XwC_eQCFn4g", "locality", "Berlin"]
            "vOnIYtcjr872fP3Wa75Ozl7c-6_MOVdlUNtwLKKxZw0" ]     ← ["TI15M8G5UIxPiWNZ-VLYBA", "country", "DE"]
 }
}
```

Step 4: Sign SD-JWT & Encode for Transport


```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZ
UkF6VmRsZnhOMjgwTmdlYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc
3N1ZXIiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJSU0EiLCAibiI6ICIwdng3YWdvZ
WJHY1FTdS4uLi4tY3NGQ3VyLWtFZ1U4YXdhcEp6S25xREtndyIsICJlIjogIkFRQUIif
X0sICJ0eXBlIjogIkl kZW50aXR5Q3JlZGVudGlhbCIsICJjcmVkZW50aWFsU3ViamVjd
CI6IHsiX3NkIjogWyJFVzFvMGVncWE1bUdjYnl0VDVTLWtBdWJjRWpZRVV3UmtYbHUyd
kM1bDIwIiwgIkZFeC1JVEh0NDFJOF9jbjBTUy1odm9MbmVYX1JHbEpvXzhvMnhSTmhmZ
GsiLCAiUXhKVi0yVjFIOG1jbHRSNnZWQzRtM3JlVTVhTkg5d2RKejJVZG1Sb0kxRSIsI
CJhdFVuMVRZd1JBbDRHUTdQZUV0WGFNdzJmNHVJVGlKclg0ODV3TTh2NjdFIiwgImZUT
XczdmtrRUx3TDFYTnVZSzhlN3pCS0NIdV91aWY2MFNsRzFweVhJVVEiLCAiaWdnN0g1Z
m4yZUJFTUlFa0U1Q2tibTlzUXV3REpsVFlvS1JpcDA4ZFlJYyIsICJ0cFV0bDcwaHBVX
3hucnZaaTBHaEdvUlIxam10MXpZZ3Z2NUlZMEF4N0tjIl0sICJhZGRyZXNzIjogeyJfc
2QiOiBbImdxQjVrbUF3eXJ5ODhhSGphQWVPLVVTWDZKT01hb2p1a0tzaGVvMzhPMGMiL
CAidk9ubFl0Y2pyODcyZlAzV2E3NU96bDdjLTZfTU9WZElVTnR3TEtLeFp3MCIsICJ3Q
EludnhzUFhkS29vd3VWcHICTWdsMWI5X1IyYjZYcGEzT1lPSWpnUXJvll19fSwgImlhd
CI6IDE1MTYyMzkwMjIsICJleHAiOiAxNTE2MjQ3MDIyLCAic2RfZGlnZXN0X2RlcmI2Y
XRpb25fYWxnIjogInNoYS0yNTYifQ.1UHEPtLLUXOT51jH3gg-3C-ZidWzsB9Un-VxmM
VdQtTbLLhwDTB6HJtt15p43yCXTzdpiZxtDI6fr07Tp0Dy_Umg3Q5_FxFj4WHnsVuVzu
ASU8cFlGPi6xgH9D3w1G2hqepBS8DyQ5bA_p5kN_tKJVoP1xWhcQujRJ8kkEKQsRia4F
hrBldl8f41wgu_ipPqh1Ix4BVI7GJCIZNx94nWPT7JUFkI6Y6JkahLf3S6gB0MxtmLAe
Y0qkuz8VeOZNfl_CDog55kVTkArorfoL6D6TEjl__-w6YyU0PnIRJXJ0wrYfoyhNl8LK
AP38QYMpdR7z_rsvHpQHzFAPTmevnHDg
```

Step 5: Base64url-encode Disclosures for Transport

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZ
UkF6VmRsZnhOMjgwTmdlYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc
3N1ZXIiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJSU0EiLCAibiI6ICIwdng3YWdvZ
WJHY1FTdS4uLi4tY3NGQ3VyLWtFZ1U4YXdhcEp6S25xREtndyIsICJlIjogIkFRQUlif
X0sICJ0eXBlIjogIkJlkZW50aXR5Q3JlZGVudGlhbCIsICJjcmVkZW50aWFsU3ViamVjd
CI6IHsiX3NkIjogWyJFVzFvMGVncWE1bUdjYnl0VDVTLWtBdWJjRWpZRVV3UmtYbHUyd
kM1bDIwIiwgIkZFeC1JVEh0NDFJOF9jbjBTUy1odm9MbmVYX1JHbEpvXzhvMnhSTmhmZ
GsiLCAiUXhKVi0yVjFIOG1jbHRSNnZWQzRtM3JlVTVhTkg5d2RKejJVZG1Sb0kxRSIsI
CJhdFVuMVRZd1JBbDRHUTdQZUV0WGFNdzJmNHVJVGlKclg0ODV3TTh2NjdFIiwgImZUT
XczdmtrRUx3TDFYTnVZSzhlN3pCS0NIdV91aWY2MFNsRzFweVhJVVEiLCAiaWdnN0g1Z
m4yZUJFTUlFa0U1Q2tibTlzUXV3REpsVFlvS1JpcDA4ZFlJYyIsICJ0cFV0bDcwaHBVX
3hucnZaaTBHaEdvUIIxam10MXpZZ3Z2NUIZMEF4N0tjII0sICJhZGRyZXNzIjogeyJfc
2QiOiBbImdxQjVrbUF3eXJ5ODhhSGphQWVPLVVTWDZKT01hb2p1a0tzaGVvMzhPMGMiL
CAidk9ubFI0Y2pyODcyZlAzV2E3NU96bDdjLTZfTU9WZElVTnR3TEtLeFp3MCIsICJ3O
EludnhzUFhkS29vd3VWcHICTWdsMWI5X1IyYjZYcGEzT1lPSWpnUXJvll19fSwgImlhd
CI6IDE1MTYyMzkwMjIsICJleHAiOiAxNTE2MjQ3MDIyLCAic2RfZGlnZXN0X2RlcmI2Y
XRpb25fYWxnIjogInNoYS0yNTYifQ.1UHEPtLLUXOT51jH3gg-3C-ZidWzsB9Un-VxmM
VdQtTbLLhwDTB6HJtt15p43yCXTzdpiZxtDI6fr07Tp0Dy_Umg3Q5_FxFj4WHnsVuVzu
ASU8cFlGPi6xgH9D3w1G2hqepBS8DyQ5bA_p5kN_tKJVoP1xWhcQujRJ8kkEKQsRia4F
hrBldl8f41wgu_ipPqh1Ix4BVI7GJCIZNx94nWPT7JUFkI6Y6JkahLf3S6gB0MxtmLAe
Y0qkuz8VeOZNfl_CDog55kVTkArorfoL6D6TEjI__-w6YyU0PnIRJXJ0wrYfoyhNl8LK
AP38QYMpdR7z_rsvHpQHzFAPTmevnHDg
```

```
~WyJHTzByMjZuTy1pVzUwWmNBb09pbEZ3IiwgImdpdmVuX25hbWUiLCAiTWF4II0
~WyJjU2xiUjEzNWkwTmpoc291TXhyampnIiwgImZhbWlseV9uYW1lIiwgIk11c3Rlcm1hbm4iX
Q
~WyJvSER0NDNWd3VocG84bXphcHJnQ2N3IiwgImVtYWlsIiwgIm11c3Rlcm1hbm5AZXhhbXB
sZS5jb20iXQ
~WyJyR2MwS3RZNldtZmx5d1RUS0VXSUVRIiwgInN0cmVIdF9hZGRyZXNzIiwgIk11c3RlcnN0c
i4gMjMiXQ
~WyJwR1FNUXgtMnRlMlh3Q19lUUNGbjRnIiwgImxvY2FsaXR5IiwgIkJlcmxpbiJd
~WyJUSTE1TThHNVVJeFBpV05aLVZMWUJBIiwgImNvdW50cnkiLCAiREUiXQ
```

→ Done!

Design Principles

	SD-JWT
Complexity	Selective disclosure, as simple as possible
Algorithms	Standard cryptography: JWS Signature + Hash function
Format	JWT & JSON
Security	Security-by-design Easy to understand & verify Hardware binding possible Cryptographic agility
Availability	Widely-available JWT libraries can be leveraged Already five independent implementations
Use Cases	Universal (beyond identity use cases)

Issuer

Issuance

JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Disclosures
salt + claim name + claim value

End-User
(Holder)

Presentation

Verifier

Issuer

Issuance

JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Disclosures
salt + claim name + claim value

End-User
(Holder)

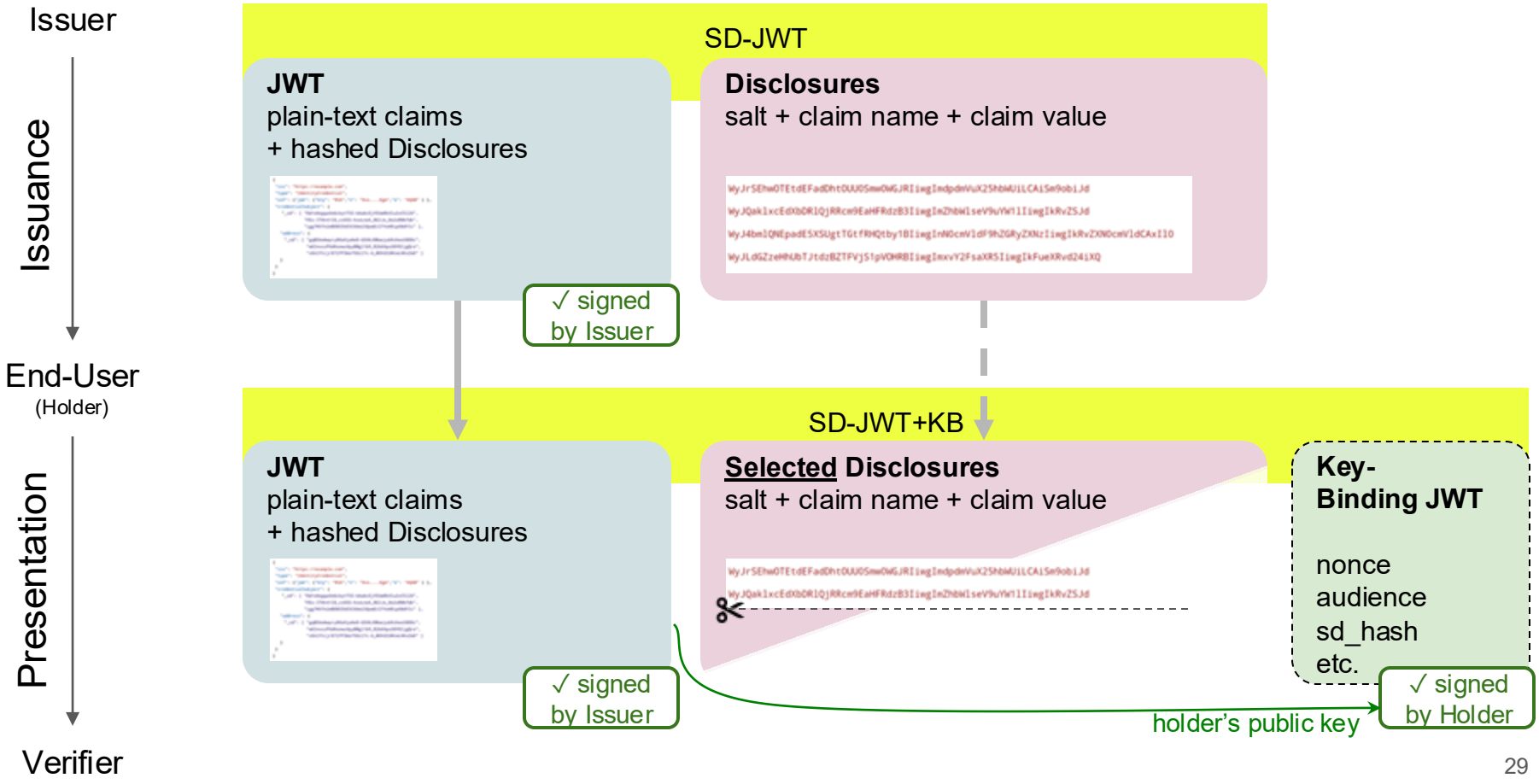
Presentation

JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Selected Disclosures
salt + claim name + claim value

Verifier


Issuer
Issuance
End-User
(Holder)
Presentation
Verifier
SD-JWT
JWT
plain-text claims
+ hashed Disclosures
Disclosures
salt + claim name + claim value
✓ signed
by Issuer
SD-JWT+KB
JWT
plain-text claims
+ hashed Disclosures
Selected Disclosures
salt + claim name + claim value
Key-
Binding JWT
nonce
audience
sd_hash
etc.
✓ signed
by Issuer
holder's public key
✓ signed
by Holder

Any Element may be Selectively Disclosable

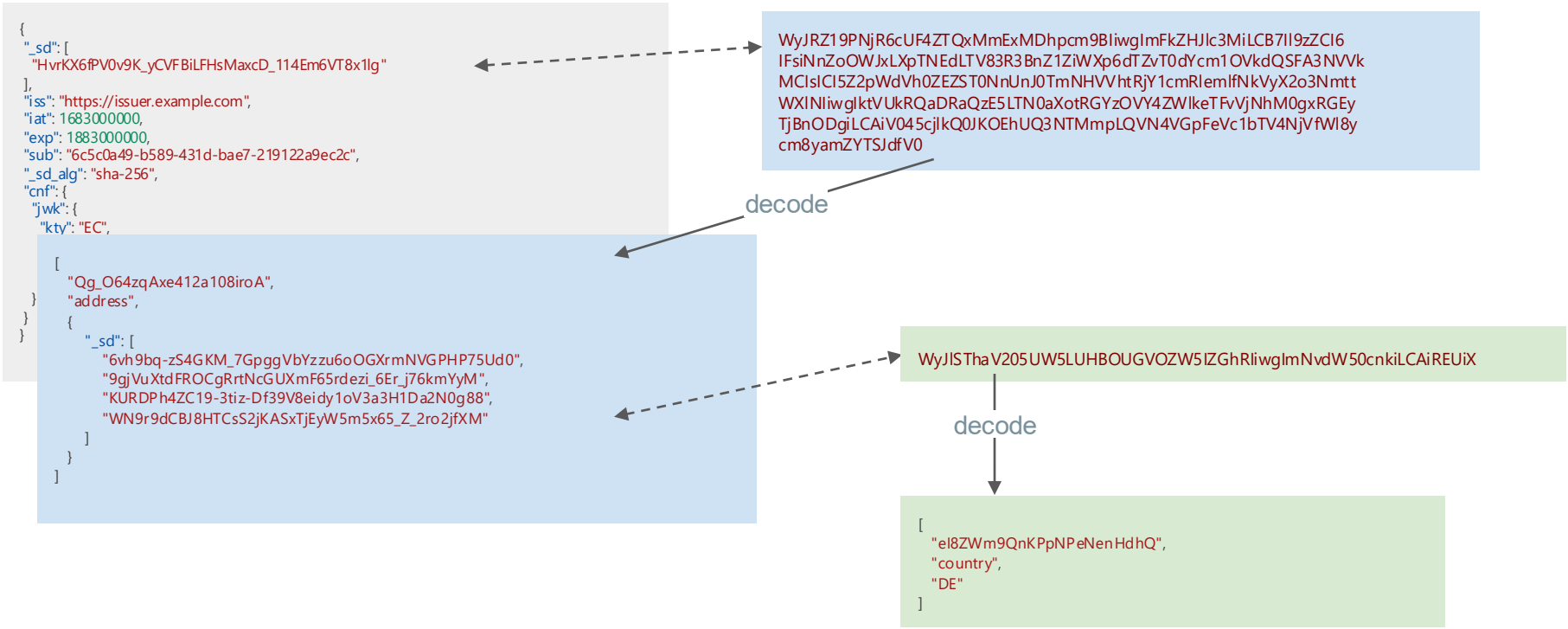
in sub-structures


```
{
 "iss": "https://issuer.example.com",
 "iat": 1683000000,
 "exp": 1883000000,
 "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
 "address": {
   "_sd": [
    "6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0",
    "9gjVuXtdFROCgRtNcGUXmF65rdezi_6Er_j76kmYyM",
    "KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88"
   ],
   "country": "DE"
 },
 "_sd_alg": "sha-256"
}
```


array elements


```
{
 "iss": "https://issuer.example.com",
 "iat": 1683000000,
 "exp": 1883000000,
 "sub": "user_42",
 "nationalities": [
   {
     "...": "pFndjkZ_VCzmyTa6UjlZo3dh-ko8aIKQc9DlGzhaVYo"
   },
   {
     "...": "7Cf6JkPudry3lcbwHgeZ8khAv1U1OSlerP0VkBJrWZ0"
   }
 ],
 "_sd_alg": "sha-256",
 "cnf": {...}
}
```

Recursive Selective Disclosure for Fine-Grained Release



Security Considerations (I)

Signature verification: Verifiers could verify the signature inadequately/partially and accept tampered credentials

Mitigating measures:

- Simple processing model, specified in detail in the standard
- Established algorithms enable the use of existing implementations

Manipulation of disclosures: If the hashes of the disclosures are not checked by the verifier, manipulated plaintext values could be accepted.

Mitigating measures:

- Design: Generally no assignment to the document possible without hash calculation
- Processing model specified in detail

Security Considerations (II)

Missing check of key binding: Verifiers could accept credentials without key binding

Mitigating measures:

- Different formats with/without key binding
- Differentiation in terminology
- Detailed discussion in the standard

Privacy Considerations

RP-RP unlinkability: Several presentations of the same credential can be traced back to the same person (due to the same hash values).

Mitigating measures:

- Single use: Credentials are always issued in groups - same data, different salt values. Each individual credential is then only used once.

- Solution in the making: ZKP (zero-knowledge proofs)

SD-JWT VC

SD-JWT VC

Credentials based on SD-JWT VC
using an extensible data model

IETF Draft: https://datatracker.ietf.org/doc/draft-ietf-oauth-sd-jwt-vc/

Daniel Fett
Oliver Terbu
Brian Campbell

Defined Claims

- **iss** — The Issuer of the Verifiable Credential. The value of iss MUST be a URI.
- **nbf** — The time before which the Verifiable Credential MUST NOT be accepted before validating.
- **exp** — The expiry time of the Verifiable Credential after which the Verifiable Credential is no longer valid.
- **cnf** — Contains the confirmation method identifying the proof of possession key. For proof of cryptographic Key Binding, the Key Binding JWT in the presentation of the SD-JWT MUST be signed by the key identified in this claim.
- **vct — The type of the Verifiable Credential, e.g., https://credentials.example.com/identity_credential.**
- **status** — The information on how to read the status of the Verifiable Credential.
- **sub** — The identifier of the Subject of the Verifiable Credential. The Issuer MAY use it to provide the Subject identifier known by the Issuer. There is no requirement for a binding to exist between sub and cnf claims.
- **iat** — The time of issuance of the Verifiable Credential. See [RFC7519] for more information.

mdoc 101

mdoc/MSO basics

- Defined in the ISO/IEC 18013-5 (https://www.iso.org/standard/69084.html)
  - focuses on mobile driving licence scenarios but could be used in other use-cases,
  - - Includes a selective disclosure mechanism based on the salted hash values
- Expressed in CBOR
  - because NFC/BLE
  - Not originally defined as a “credential format”

MSO (issuer-signed object) structure

```
MobileSecurityObject = {
    "digestAlgorithm" : tstr, ; Message digest algorithm used
    "valueDigests"    : ValueDigests, ; Array of digests of all data elements
    "deviceKey"       : DeviceKey, ; Device key in COSE_Key as defined in RFC 8152
    "docType"         : tstr, ; DocType as used in Documents
    "validityInfo"    : validity of the MSO and its signature
}
```

Blinds claim name by using “digestID”

mdoc response (presentation) structure

```
IssuerSignedItem = {
    "digestID" : uint, ; Digest ID for issuer data authentication
    "random" : bstr, ; Random value for issuer data authentication
    "elementIdentifier" : DataElementIdentifier, ; Data element identifier
    "elementValue" : DataElementValue ; Data element value
}
```

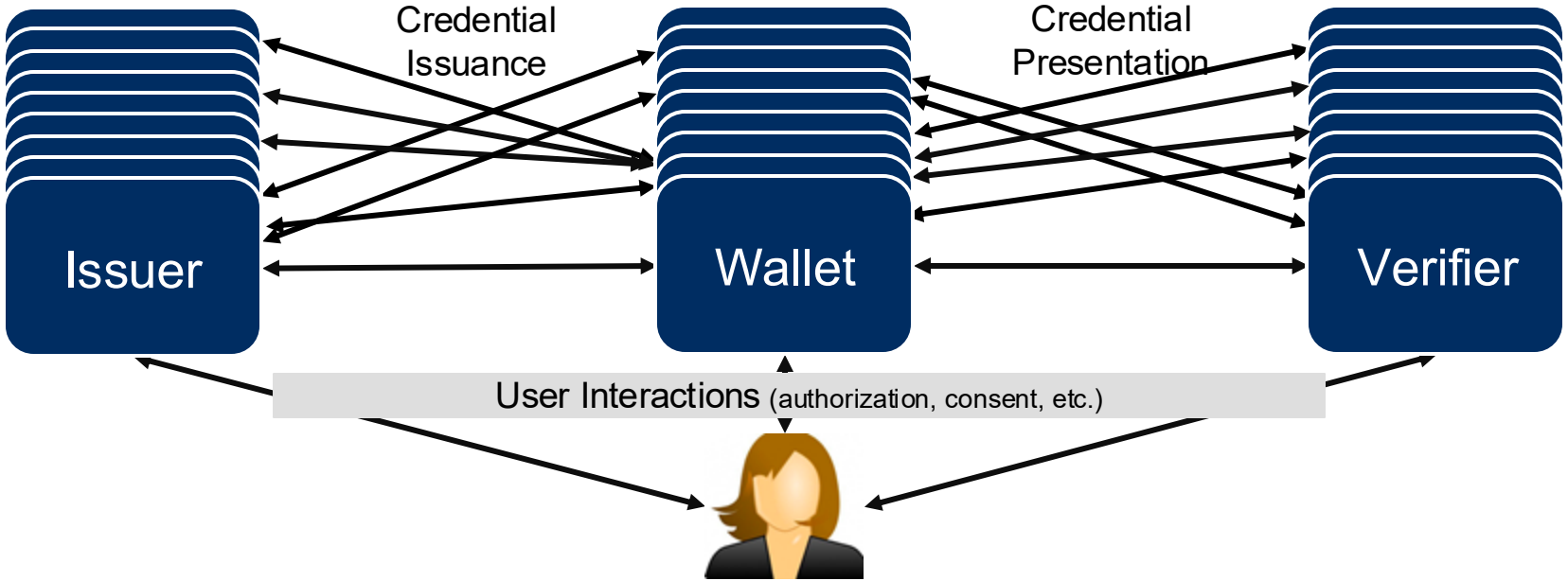
How to send this mapping of direstID, random (salt), element name and claim value during issuance is not defined.

mdocs: other facts

- predicates: `age_over_NN` claim
- unlinkability: issue the same copy of the credential with different User public key that can be used per verifier (to prevent RP-RP' unlinkability)
- refresh: can be only the issuer's signature over hashes, or the entire "mdoc"

Protocol Layer Interoperability is Crucial

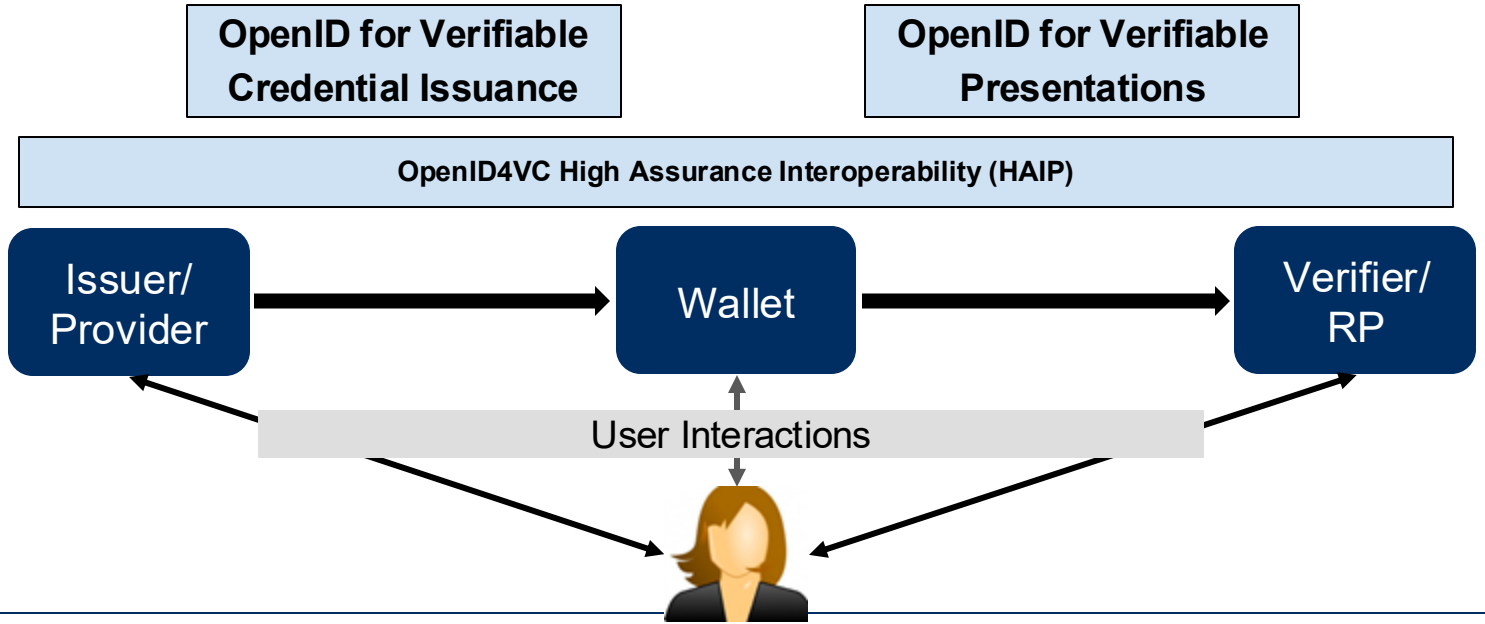
There was a need for the interoperable protocol layer that can support all of the credential formats, key resolution mechanisms and trust frameworks.



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.

OID4VC: OpenID for Verifiable Credentials set of protocols



OID4VC set of protocols also includes Self-Issued OpenID Provider v2 (SIOPv2) and OpenID4VP over BLE.

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

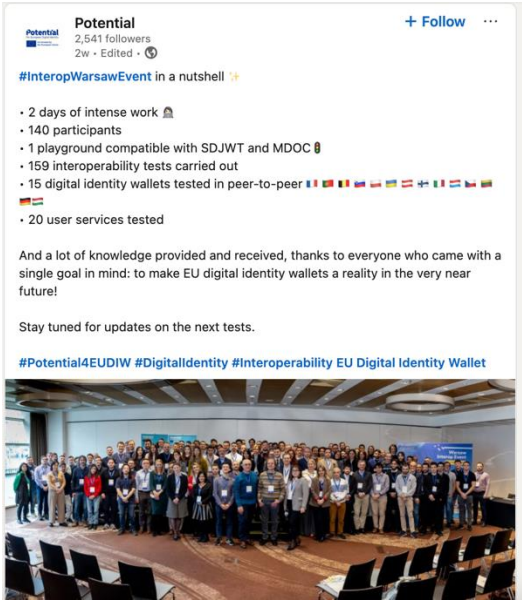
Wallet Implementation: shorturl.at/rD7tf

OpenID4VC 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:

Interoperability Events (selected)



LSP POTENTIAL

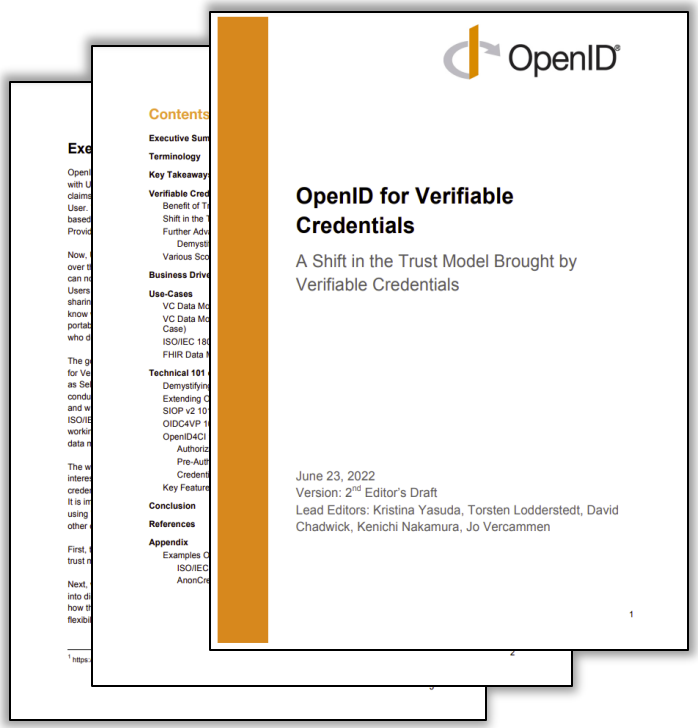
NIST National Cybersecurity Center of Excellence[2]

- 18013-7 Annex B with vanilla OpenID4VP with mdocs
- HAIP OpenID4VP over the DC API with mdocs

ISO/IEC SC17 WG10 Interoperability events (mDL)

- 18013-7 Annex B with vanilla OpenID4VP with mdocs
- HAIP OpenID4VP over the DC API with mdocs

Next: OpenID4VP and OpenID4VCI



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

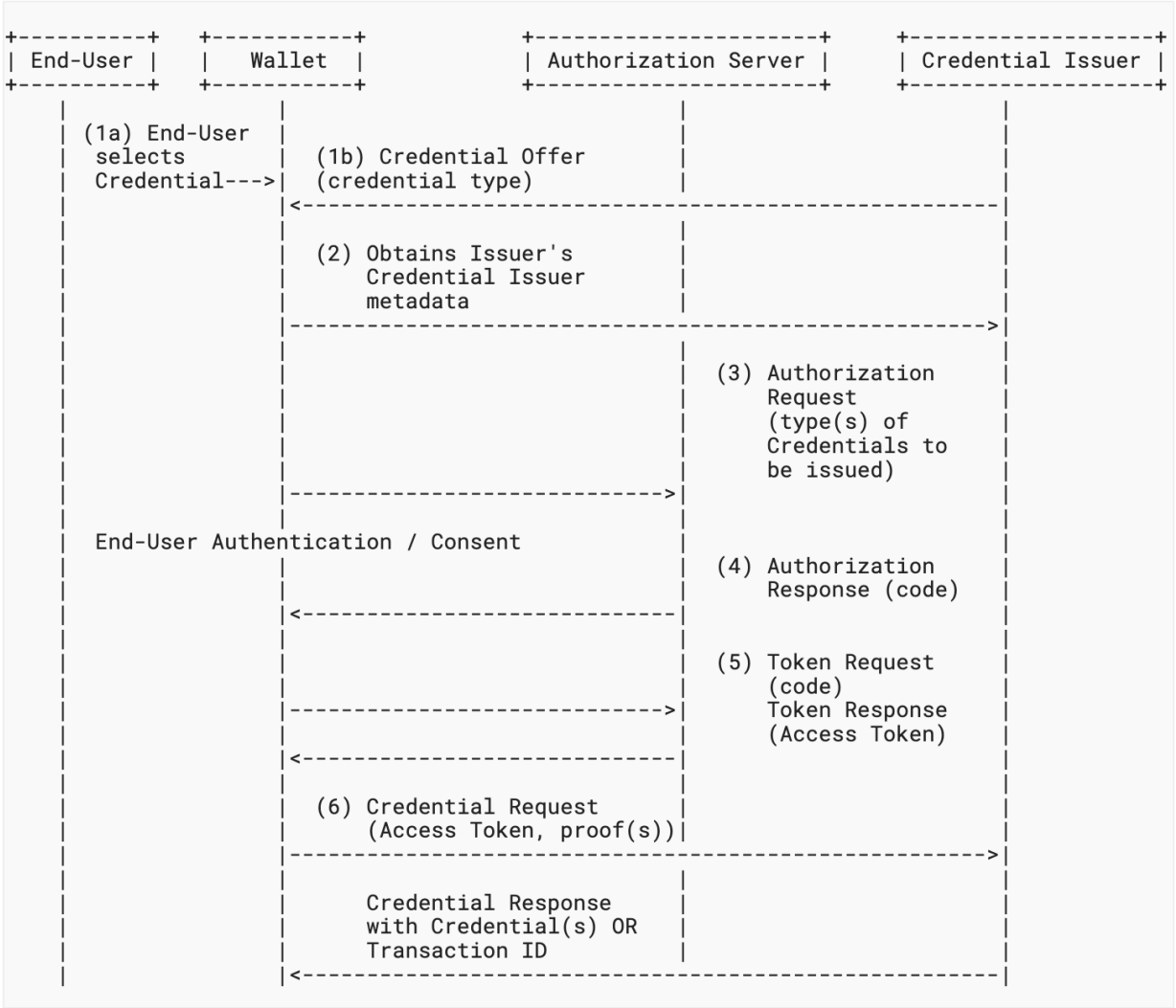
OpenID for Verifiable Presentations

OpenID for Verifiable Presentations: Highlights

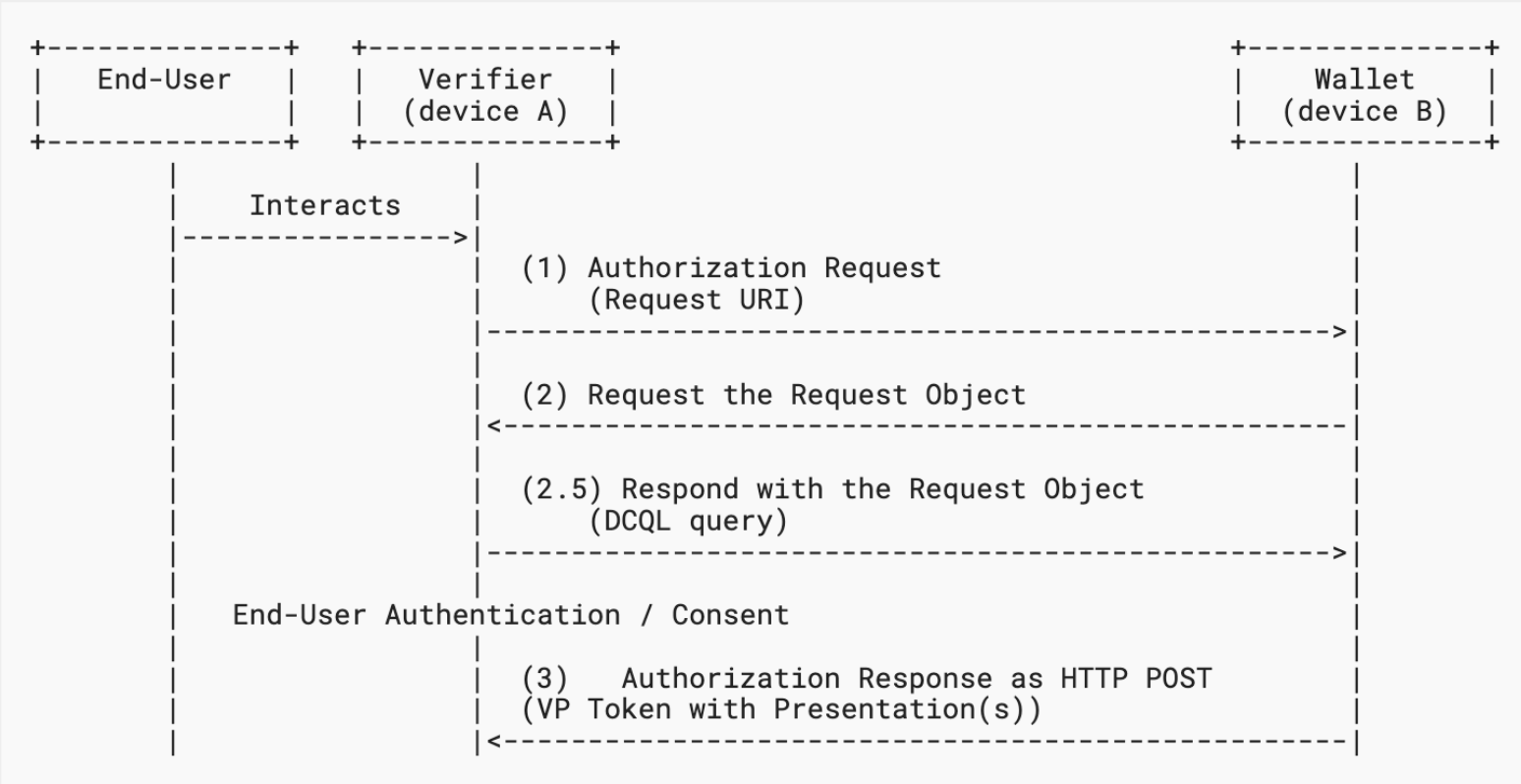
Voting for Final 1.0 starting in few weeks

- 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-de


End-User
Wallet
Authorization Server
Credential Issuer
(1a) End-User selects Credential
(1b) Credential Offer (credential type)
(2) Obtains Issuer's Credential Issuer metadata
(3) Authorization Request (type(s) of Credentials to be issued)
End-User Authentication / Consent
(4) Authorization Response (code)
(5) Token Request (code)
Token Response (Access Token)
(6) Credential Request (Access Token, proof(s))
Credential Response with Credential(s) OR Transaction ID

Cross-device flow



Presentation Request

The following is a non-normative example of an Authorization Request with a Request Object as value:

```
GET /authorize?
  client_id=redirect_uri%3Ahttps%3A%2F%2Fclient.example.org%2Fcb
  &request=eyJrd...
```

Where the contents of the `request` query parameter consist of a base64url-encoded and signed (in the example with RS256 algorithm) Request Object. The decoded payload is:

```
{
  "iss": "redirect_uri:https://client.example.org/cb",
  "aud": "https://self-issued.me/v2",
  "response_type": "vp_token",
  "client_id": "redirect_uri:https://client.example.org/cb",
  "redirect_uri": "https//client.example.org/cb",
  "dcql_query": {
    "credentials": [
      {
        "id": "some_identity_credential",
        "format": "dc+sd-jwt",
        "meta": {
          "vct_values": [ "https://credentials.example.com/identity_credential" ]
        },
        "claims": [
            {"path": ["last_name"]},
            {"path": ["first_name"]}
        ]
      }
    ]
  },
  "nonce": "n-0S6_WzA2Mj"
}
```

DCQL Query

The following is a non-normative example of a DCQL query that requests a Credential of the format `dc+sd-jwt` with a type value of `https://credentials.example.com/identity_credential` and the claims `last_name`, `first_name`, and `address.street_address`:

```
{
  "credentials": [
    {
      "id": "my_credential",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": [ "https://credentials.example.com/identity_credential" ]
      },
      "claims": [
          {"path": ["last_name"]},
          {"path": ["first_name"]},
          {"path": ["address", "street_address"]}
      ]
    }
  ]
}
```

Presentation Response

The following is a non-normative example of an Authorization Response when the Response Type value in the Authorization Request was `vp_token`:

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

8.1.1. Examples

The following is a non-normative example of the contents of a VP Token containing a single Verifiable Presentation in the SD-JWT VC format after a request using DCQL like the one shown in Section 7.4 (shortened for brevity):

```
{
  "my_credential": ["eyJhbGci...QMA"]
}
```

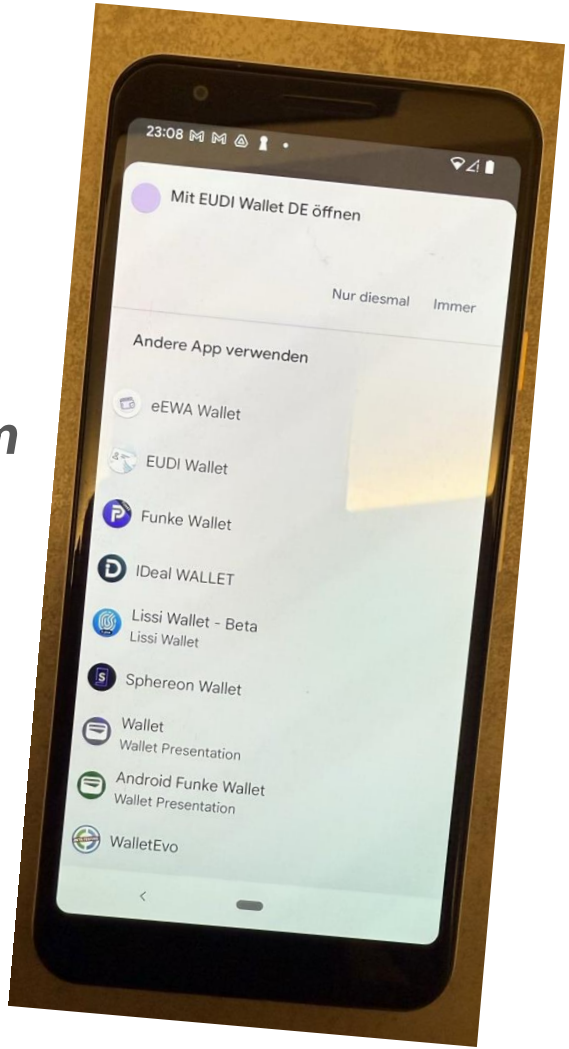
Digital Credentials API

- Background
- Demo
- Components
- The Digital Credentials API
- Cross-Device Presentation
- Issuance
- Q&A

Background

The problem

*digital credential presentation on the web currently relies on primitives such as **custom schemes** and **QR codes** which have **poor security properties** and an even **worse user experience***



What is a custom URI scheme?

A custom identifier that an app can register with an operating system with the goal of being invoked from other contexts, such as other apps or from the web.

In many cases, these identifiers are not globally unique, and may be shared.

CUSTOM SCHEMES IN THE WILD

`mdoc://`

`openid4vp://`

`eudi-wallet://`

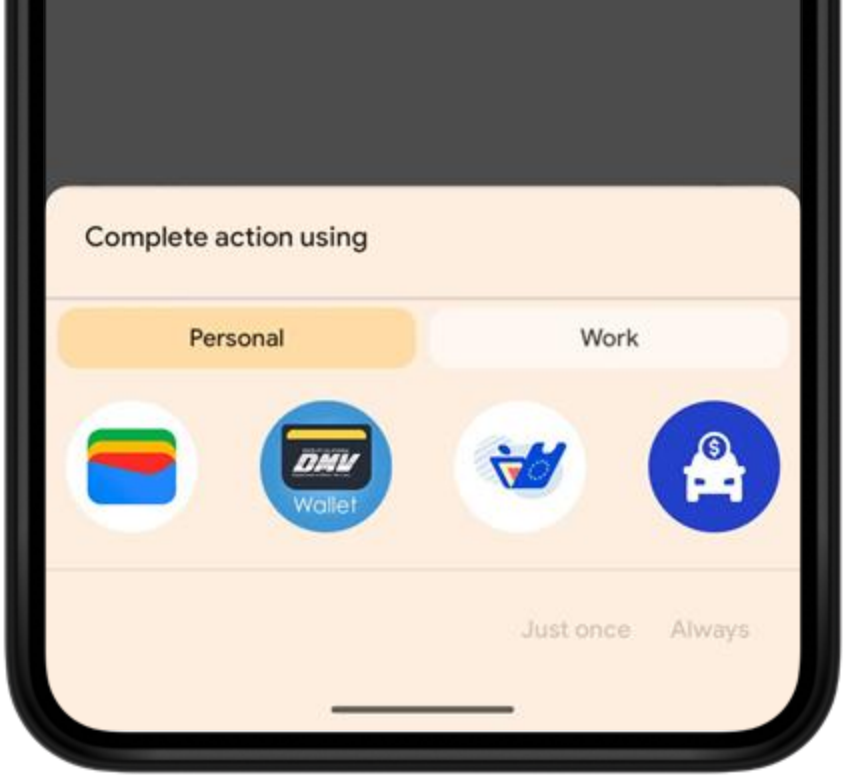
`eudi-openid4vp://`

`mdoc-openid4vp://`

`openid-credential-offer://`

Issues w/ custom schemes

- invocation from insecure contexts
- on-device phishing via app selection
- no requestor origin / identity
- not standardized & not guaranteed
- context switch during app launch
- no graceful fallback for errors



poor UX for credential selection

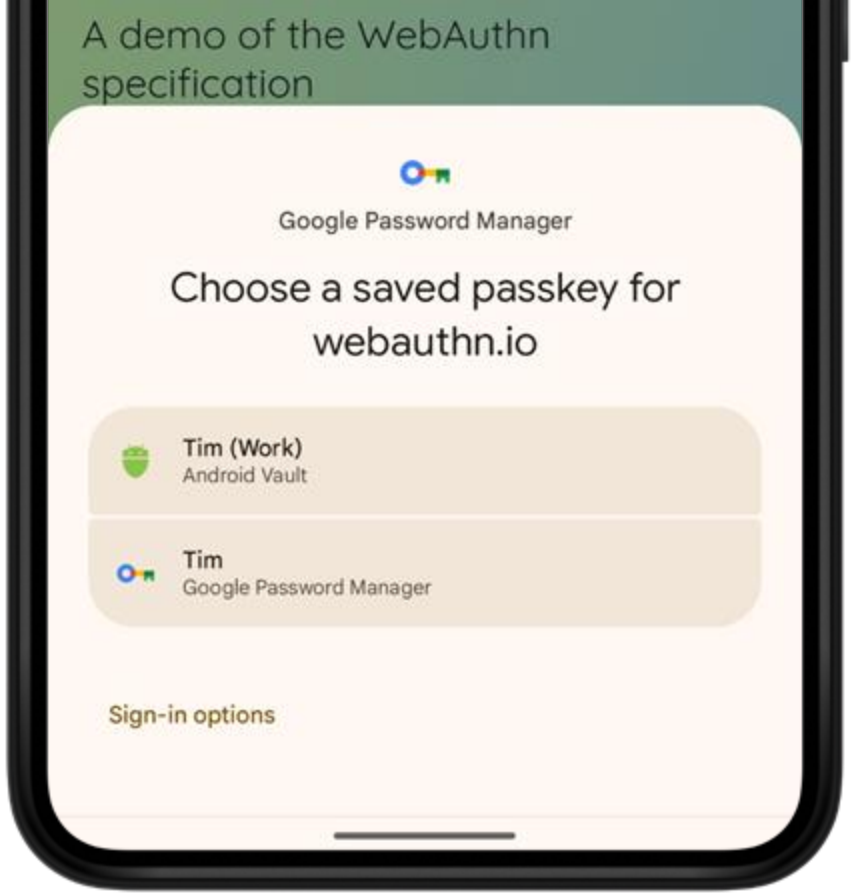
(*users don't understand wallet selection*)

Learnings from passkeys

users think about **accounts** and **credentials**, not **authenticators**

caller context is key

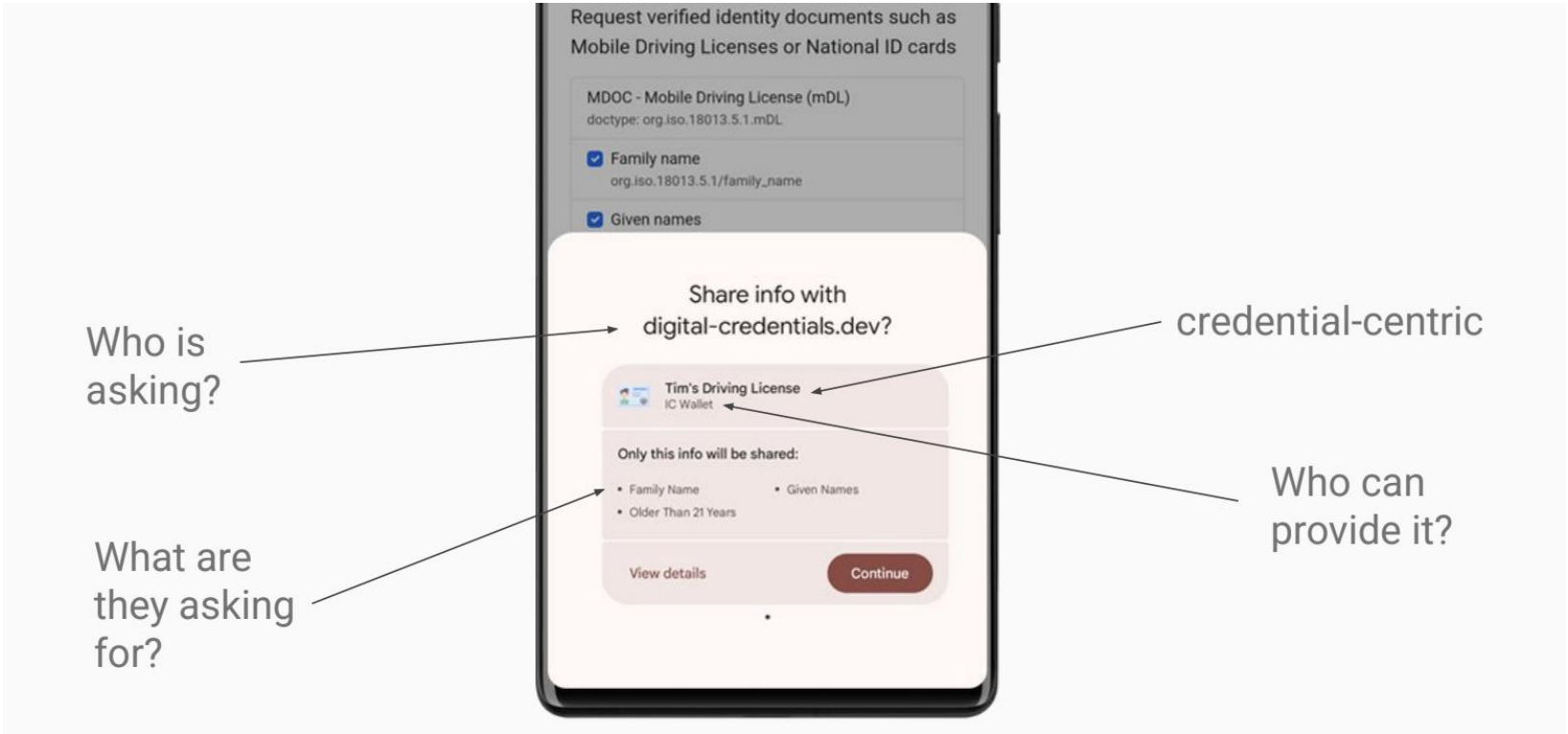
cross-device authentication needs to be **secure**, **easy**, and **resistant to phishing**



Design Principles

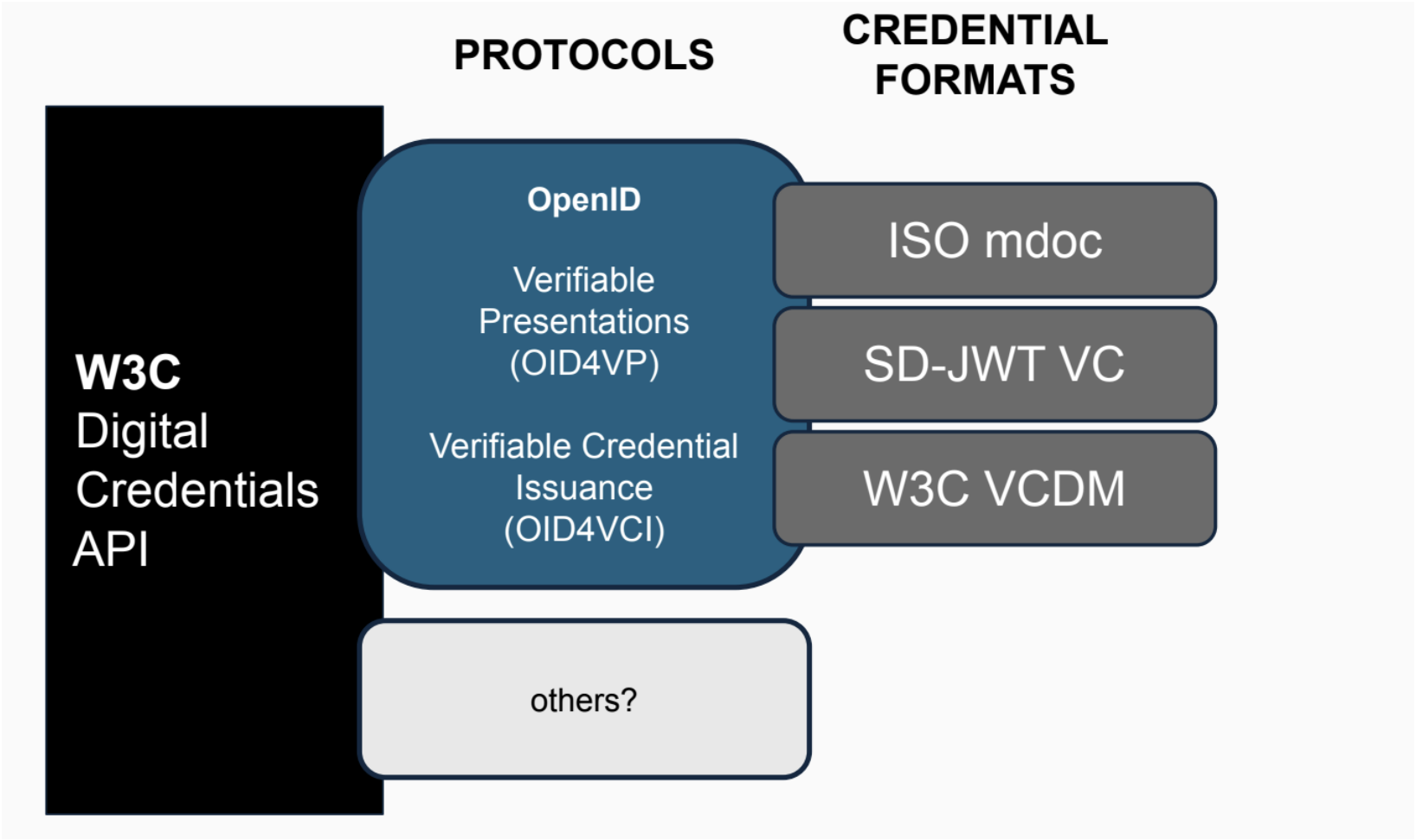
- Separate the act of requesting from the specific protocol, allowing flexibility in both the protocol and credential formats. This way, the pace of changes in browsers won't hinder progress or block new developments.
- Require request transparency, enabling user-agent inspection for risk analysis
- Assume response opacity (encrypted responses), enabling verifiers and holders to control where potentially sensitive PII is exposed
- Prevent website from silently querying for the availability of digital credentials and communicating with credential providers without explicit user consent

From the User perspective



Demo Later

Layering


PROTOCOLS
CREDENTIAL FORMATS
W3C Digital Credentials API
OpenID
Verifiable Presentations (OID4VP)
Verifiable Credential Issuance (OID4VCI)
ISO mdoc
SD-JWT VC
W3C VCDM
others?

Roles and Responsibilities

Browser (web platform)	**OS Platform** (app platform)	**Credential Provider** (app/wallet)
<<<<<< Permission >>>>>>		Holder consent
API surface	Credential selector (presentation)	Holder verification
Basic request validation	Provider selector (issuance)	Presentation & Issuance Protocols (verifier / RP authentication, selective disclosure, signing, encryption)
Secure context validation	Cross-device transport	
Interaction with OS platform	Native app requests	Key management

Components: Same Device

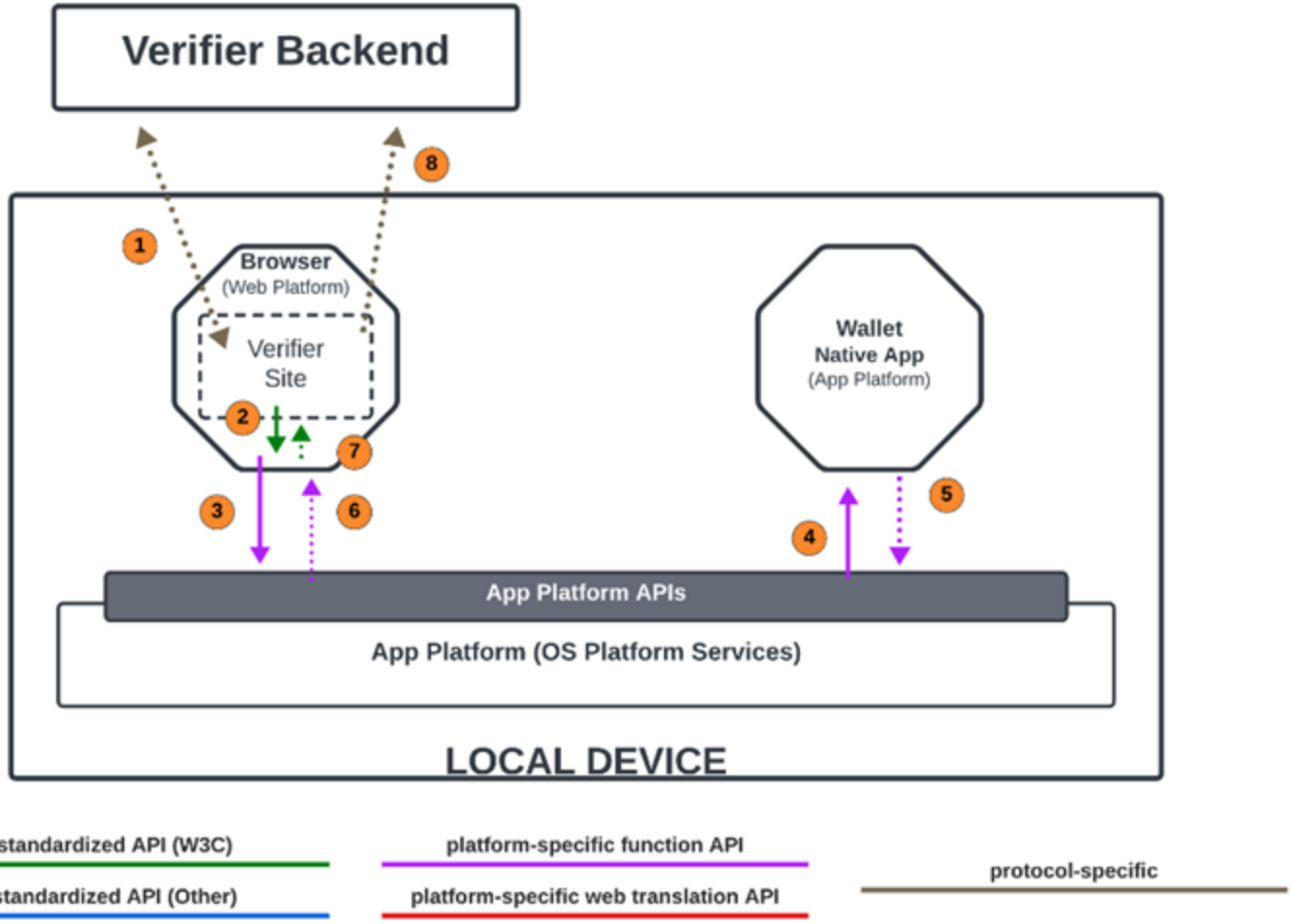
Verifier: website or native app

Client: web browser or app instance

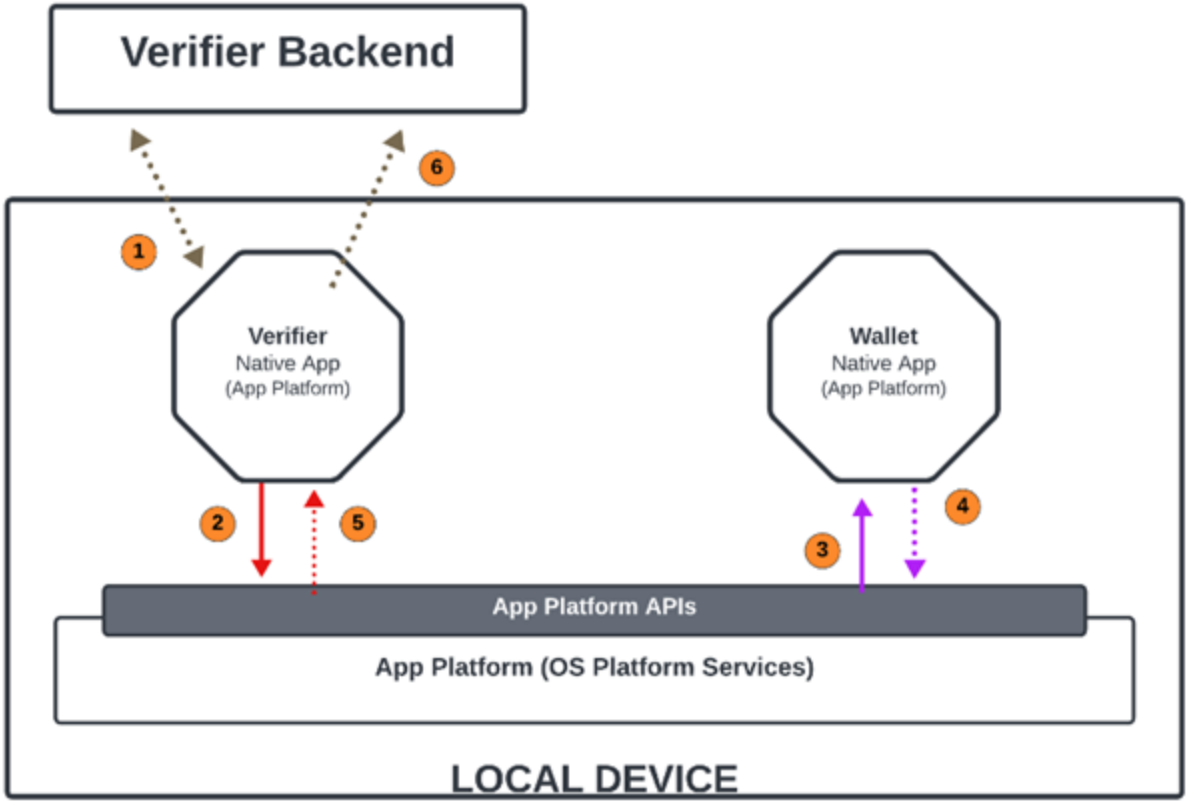
App Platform: underlying OS

Identity Wallet: native app

Layers: Same Device (Web Verifier)



Layers: Same Device (App Verifier)



standardized API (W3C)

standardized API (Other)

platform-specific function API

platform-specific web translation API

protocol-specific

Components: Cross-Device

Verifier: website or native app

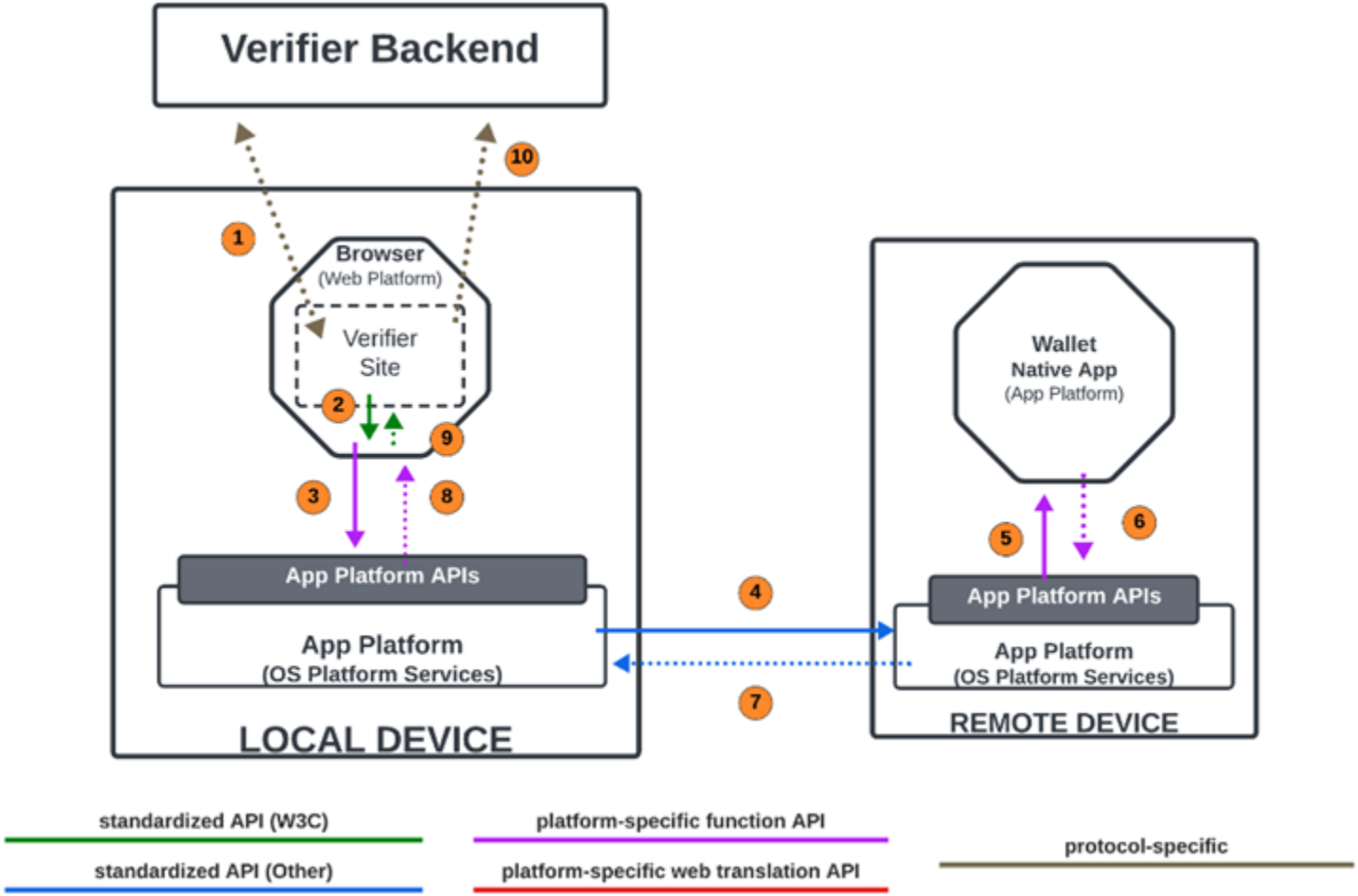
Local Client: web browser or app instance

Local App Platform: underlying OS on calling device

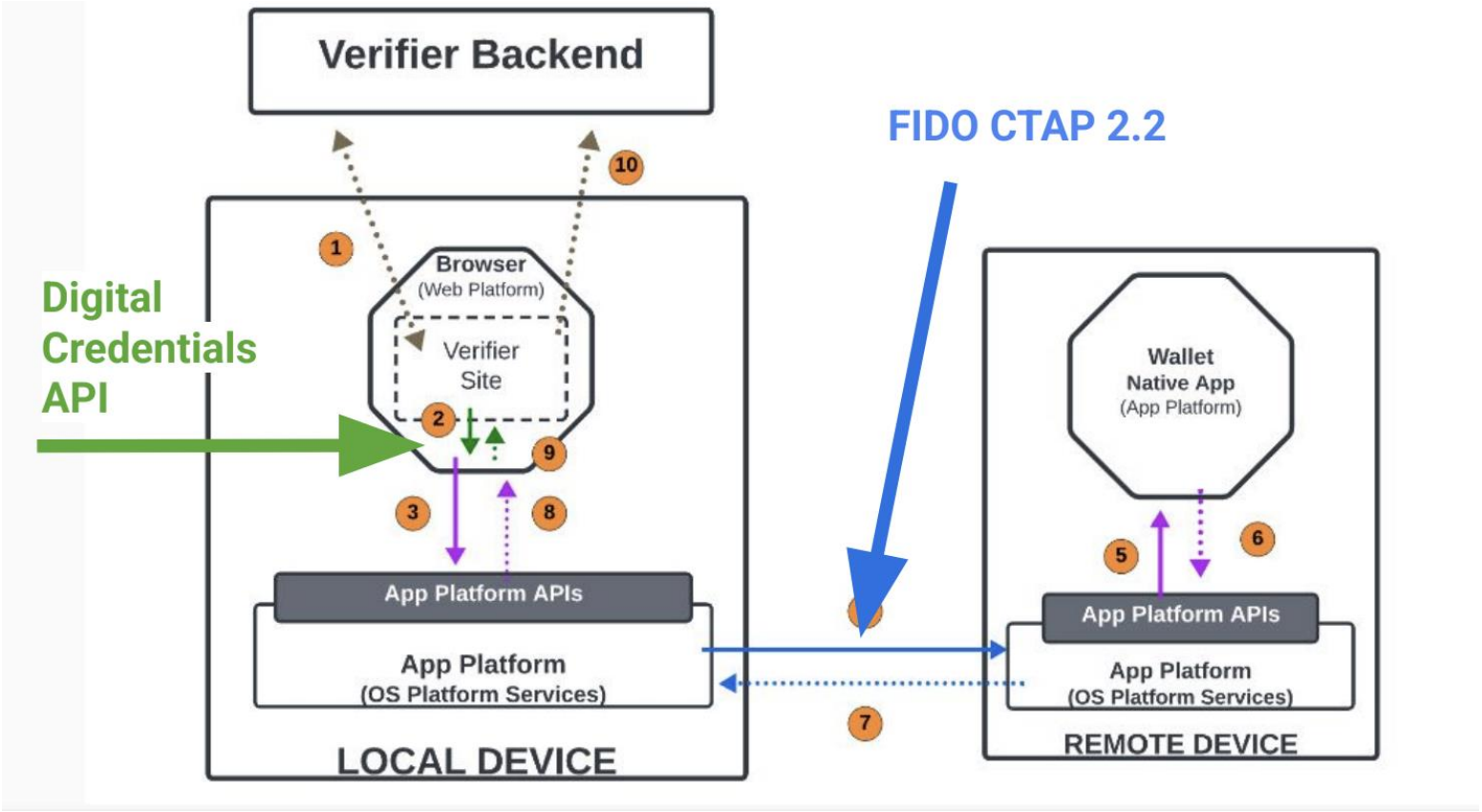
Remote App Platform: underlying OS on remote device

Remote Identity Wallet: native app on remote device

Layers: Cross-Device (Web Verifier)



Layers: Cross-Device (App Verifier)



The API


```
let cred = await
  navigator.credentials.get({
    signal: controller.signal,
    digital: {
     requests: [{
     protocol: "openid4vp-v1-unsigned",
     data: { ...request }
   }]
  }
});
```

Issuance

- In scope now!

Get Involved

Prototype with Android and Chrome!

Instructions:

Short link: tcslides.link/dc-androidprotoype

Full link

OpenID for Verifiable Credential Issuance

OpenID for Verifiable Credential Issuance: Highlights

Status: WG Last Call expected to start this week

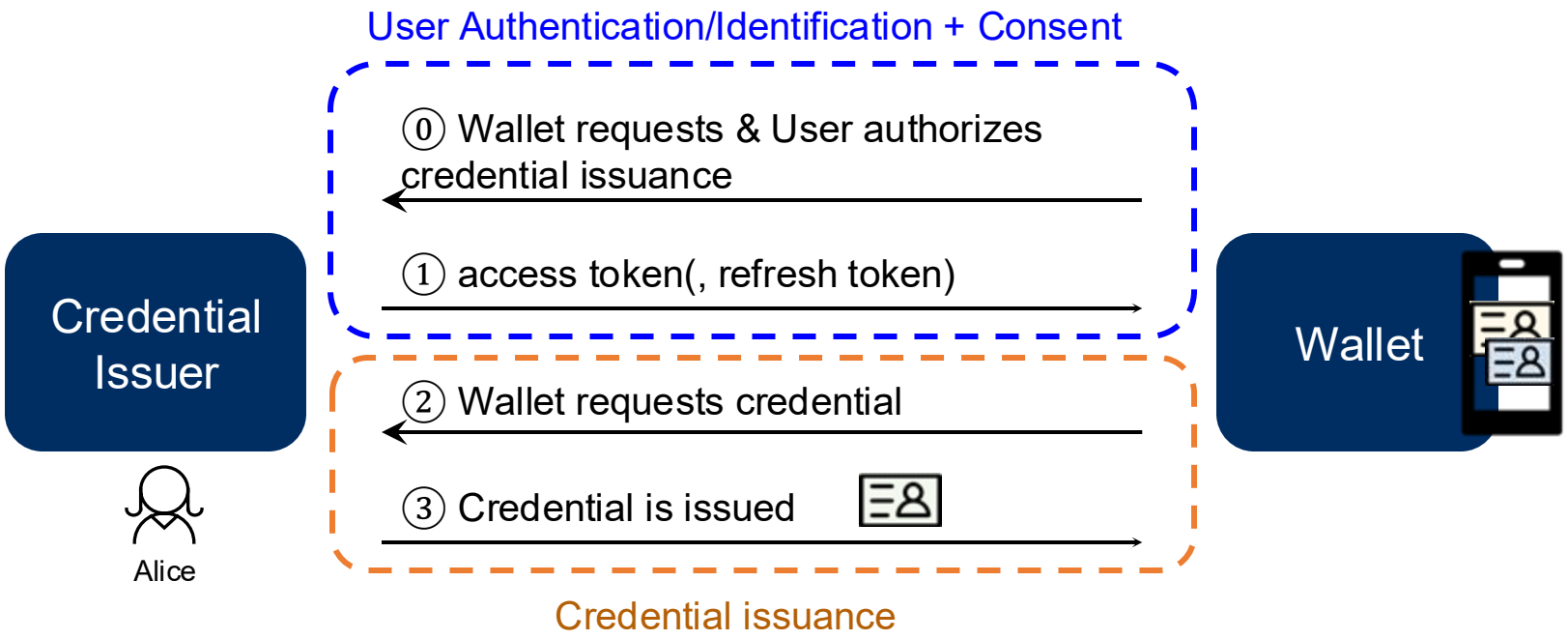
Easy to use for developers

Various Security levels can be supported

Various business requirements and user-experiences can be achieved

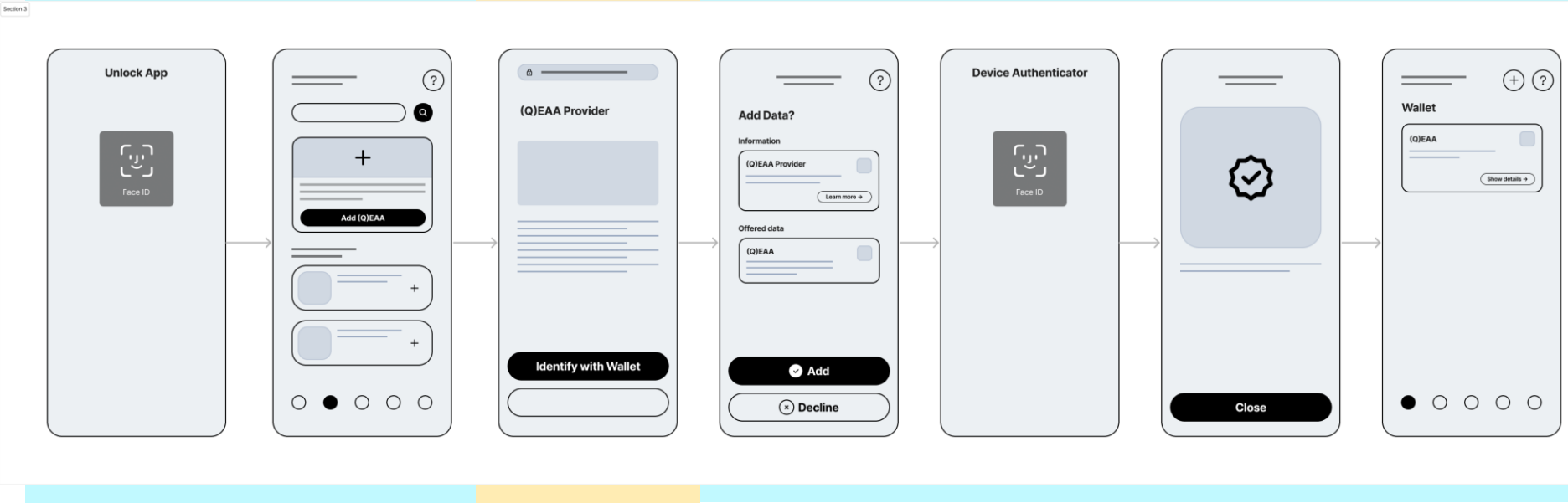
Various trust frameworks and credential formats can be supported

OAuth-protected API

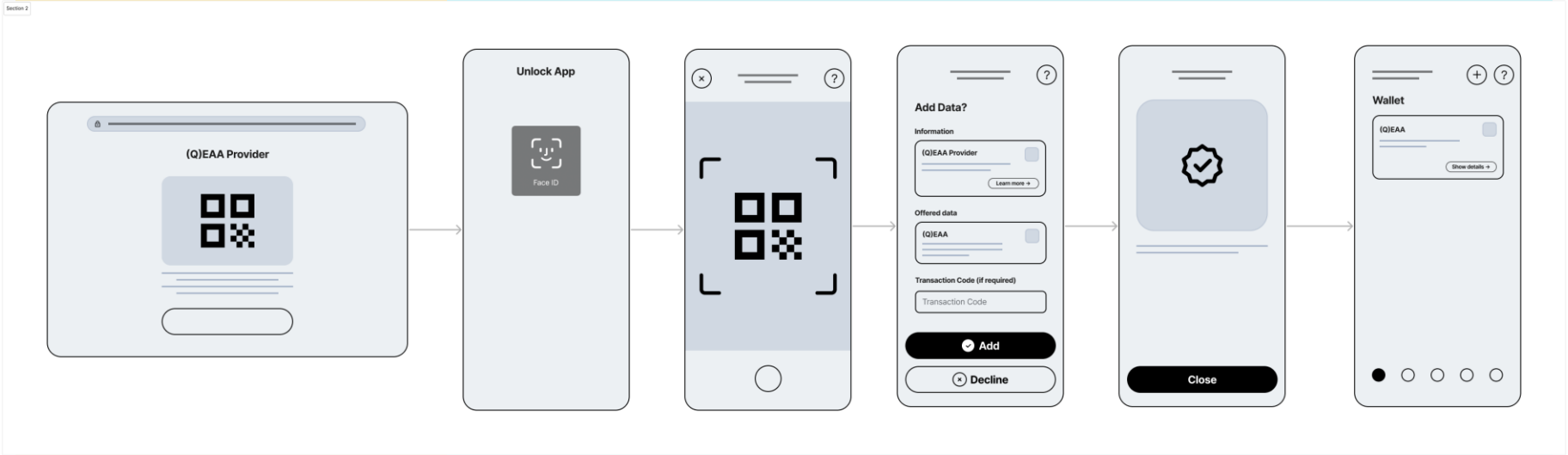


OpenID4VCI can be used in conjunction with any other OAuth extension RFC

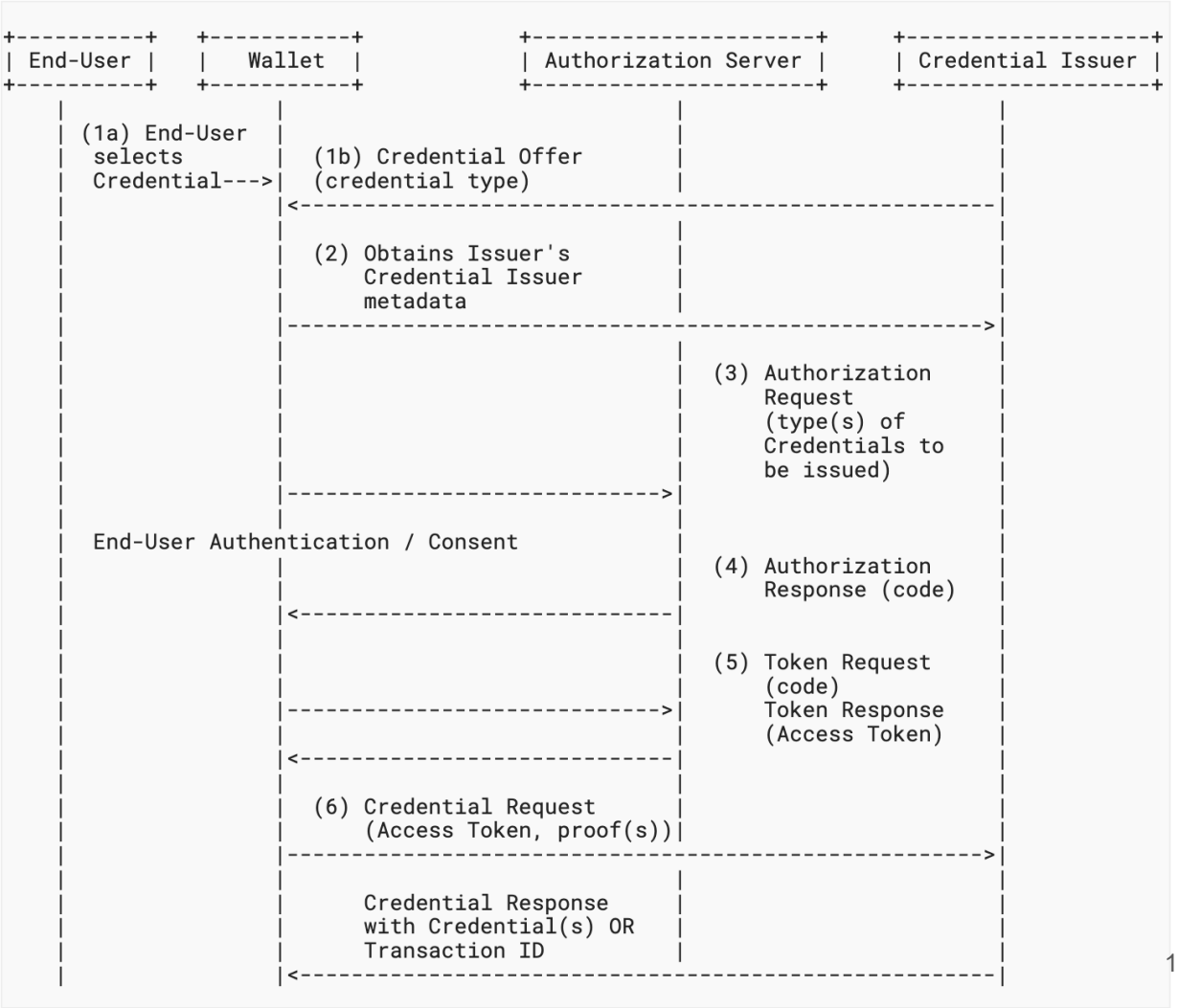
Authorization Code Flow



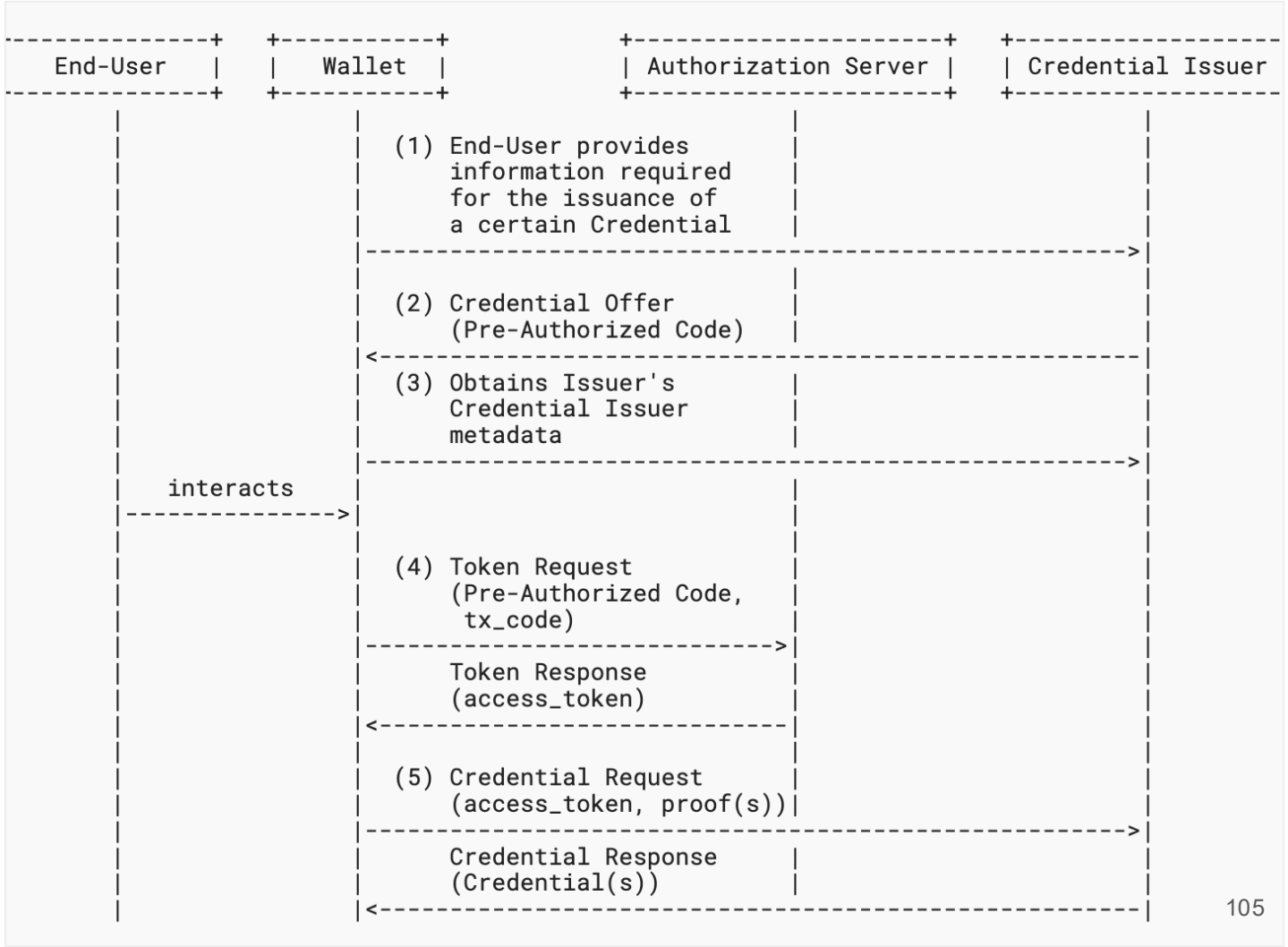
Pre-Authorized Code Flow



Authorization code flow



Pre-Authorized Code flow



Credential Offer

When the Credential Offer is displayed as a QR code, it would usually contain the Credential Offer by reference due to the size limitations of the QR codes. Below is a non-normative example:

```
openid-credential-offer://?
  credential_offer_uri=https%3A%2F%2Fserver%2Eexample%2Ecom%2Fcredential-offer
  %2FGkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

Below is a non-normative example of a response from the Credential Issuer that contains a Credential Offer Object used to encourage the Wallet to start an Authorization Code Flow:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "credential_issuer": "https://credential-issuer.example.com",
  "credential_configuration_ids": [
    "UniversityDegreeCredential"
  ],
  "grants": {
    "authorization_code": {
      "issuer_state": "eyJhbGciOiJSU0Et...FYUaBy"
    }
  }
}
```

Authorization Request

Below is a non-normative example of an Authorization Request provided by the Wallet to the Authorization Server using the scope `UniversityDegreeCredential` and in response to an HTTP 302 redirect (with line breaks within values for display purposes only):

```
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
```

Authorization Response

Below is a non-normative example of a successful Authorization Response:

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

- PAR (Pushed Authorization Request can be used too)

Token Request

Below is a non-normative example of a Token Request in a Pre-Authorized Code Flow (without Client Authentication):

```
POST /token HTTP/1.1
Host: server.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
&authorization_details=%5B%7B%22type%22%3A%20%22openid_credential%22%2C%20%22
    credential_configuration_id%22%3A%20%22UniversityDegreeCredential%22%7D%5D
```

- `authorization_details` or `scope` parameter

Token Response

Below is a non-normative example of a Token Response when the `authorization_details` parameter was used to request issuance of a certain Credential type:

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

{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6Ikp..sHQ",
  "token_type": "Bearer",
  "expires_in": 86400,
  "authorization_details": [
    {
      "type": "openid_credential",
      "credential_configuration_id": "UniversityDegreeCredential",
      "credential_identifiers": [ "CivilEngineeringDegree-2023", "ElectricalEngineeringDe
    }
  ]
}
```

Credential Request

Below is a non-normative example of a Credential Request for a Credential in [ISO.18013-5] format using the Credential configuration identifier and a key proof type `jwt`:

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

{
  "credential_configuration_id": "org.iso.18013.5.1.mDL",
  "proofs": {
    "jwt": [
      "eyJraWQiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEva2V5cy8xIiwiYWxnIj
    ]
  }
}
```

Credential Response

Below is a non-normative example of a Credential Response in an immediate issuance flow for multiple Credential instances in JWT VC format (JSON encoded) with an additional `notification_id` parameter:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "credentials": [
    {
      "credential": "LUpixVCWJk0eOt4CXQe1NXK....WZwmhmn9OQp6YxX0a2L"
    },
    {
      "credential": "YXNkZnNhZGZkamZqZGFza23....29tZTIzMjMyMzIzMjMy"
    }
  ],
  "notification_id": "3fwe98js"
}
```

- There is also a deferred issuance endpoint

HAIP
High Assurance
Interoperability Profile

HAIP was restructured

- **Not be limited to SD-JWT VC, mdoc added**
- **To be a collection of 4 profiles that can be used independently:**

1. **Issuance of IETF SD-JWT VC using OpenID4VCI;**
2. **Presentation of IETF SD-JWT VC using OpenID4VP;**
3. **Presentation of IETF SD-JWT VC using OpenID4VP over W3C Digital Credentials API;**
4. **Presentation of ISO mdocs using OpenID4VP over W3C Digital Credentials API;**
5. [coming] Presentation of ISO mdocs using OpenID4VP;
6. [coming] Issuance of ISO mdocs using OpenID4VCI

OpenID Certification Program Overview

- A light-weight, low-cost, self-certification program to serve members, drive adoption and promote high-quality implementations
  - Identity Providers launched in early 2015
  - Relying Parties launched in late 2016
  - FAPI profiles launched in 2019
- Each certification makes it easier for those that follow and helps make subsequent deployments more trustworthy, interoperable and secure
- All certified implementations are openly listed at https://openid.net/developers/certified/

The process

For example to test a wallet for verifiable presentations:

1. Wallet provider runs the tests either locally or on our cloud server
2. The tests check if the wallet responds correctly to both positive and negative tests
3. Any failures that require fixing are surfaced to the tester
4. The logs & statement of compliance are submitted to OIDF
5. Certification fee is paid
6. OIDF publishes results after checking logs/etc are correct
7. “OpenID Certified” mark can now be used by certified entity

Testing OpenID VC specifications

- Fairly good tests OpenID for Verifiable Presentations
  - Tests wallets & verifiers, supports ID2 and ID3 of spec,
  - 12+ wallets/verifiers tested & passed
  - Continuing to build out the tests
- OpenID for Verifiable Credential Issuance
  - Initial alpha tests available
  - Targeting ID3 (Dec 2024) of the specification
  - Access to a compliant issuer would really help us

Conformance tests support this process

- OpenID for VCs Test Suite now includes:
- **Issuers** using OpenID for Verifiable Credential Issuance + HAIP
  - Implementers Draft 2 (alpha)
- **Wallets** using OpenID for Verifiable Credential Issuance + HAIP
  - Implementers Draft 2 (alpha)
- **Wallets** using OpenID for Verifiable Presentations + HAIP
  - Implementers Draft 2
  - Implementers Draft 3 + draft 24
  - Implementers Draft 3 + draft 24 + Browser DC API
- **Verifiers** using OpenID for Verifiable Presentations + HAIP
  - Implementers Draft 2
  - Implementers Draft 3 + draft 24

OpenID For Verifiable Presentations – Current Status

- Testing latest Implementer's Draft ID2 & ID3
  https://openid.net/specs/openid-4-verifiable-presentations-1_0-ID2.html
- https://openid.net/specs/openid-4-verifiable-presentations-1_0-24.html (tests for -28 expected during June)
- response_type=vp_token
- client_id_scheme redirect_uri or x509_san_dns
- Direct Post or Direct Post JWT (encrypted response)
- Cross device or same device
- Traditional (custom url scheme) or (for testing wallets) W3C DC API
- request_uri, request object by value or plain request
- SD-JWT with SD-JWT VC, HAIP or ISO mDL
- presentation_definition or DCQL

OpenID For Verifiable Presentations – Roadmap

- Further validation in current test
  - E.g. SD-JWT signature not yet checked
- More client_id_scheme
- More negative tests

Suggestions welcome

Please tell me what features you are using

OpenID For Verifiable Issuance – Current Status

- Testing latest Implementer's Draft ID3
- Focusing on testing current HAIP draft
  - Some non-HAIP features work
- Alpha tests for both wallets & issuers

Demo

(Conformance testing Google’s Wallet that supports DC API)

Q & A