Current Work and Future Trends in Selective Disclosure

Thursday, May 11, 2023

Agenda

Mike Jones – **Introductory remarks**

Daniel Fett – **SD-JWT**

Kristina Yasuda – **ISO mdoc**

Tobias Looker – **Zero-Knowledge Proofs and BBS**

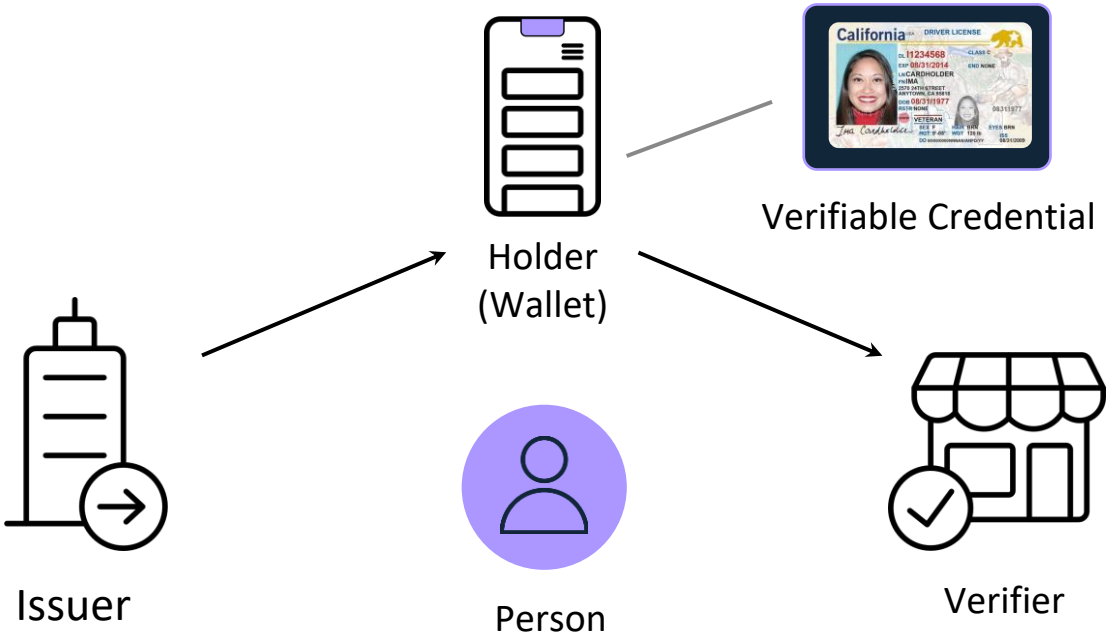
David Waite – **JSON Web Proofs and JOSE**

All – **Closing Remarks and Discussion**

Selective Disclosure

- A lot of foundational work happening in Selective Disclosure right now
- Enables you to have a token with many claims and only release the claims necessary to the interaction
  - For instance, disclose your birthdate but not your home address
- Selective Disclosure enables Minimal Disclosure
- Sometimes uses Zero Knowledge Proofs (ZKPs) but not always necessary
- Real deployments under way
  - For instance, ISO Mobile Driving Licenses use Selective Disclosure

Issuer / Holder / Verifier Model



SD-JWT

draft-ietf-oauth-selective-disclosure-jwt-03

‘Simple’ is a feature.

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 six independent implementations
Use Cases	Universal (beyond identity use cases)

SD-JWT in 5 Simple Steps

Step 1: Prepare User Data


```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },
 "credentialSubject": {
   "given_name": "Max",
   "family_name": "Mustermann",
   "email": "mustermann@example.com",
   "address": {
     "street_address": "Musterstr. 23",
     "locality": "Berlin",
     "country": "DE"
   }
 }
}
```

SD-JWT in 5 Simple Steps

Step 2: Create *Disclosures*

```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },
 "credentialSubject": {
   "given_name": "Max",          ["GO0r26nO-iW50ZcAoOilFw", "given_name", "Max"]
   "family_name": "Mustermann",  ["cSlbR135i0NjhsouMxrjjg", "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

SD-JWT in 5 Simple Steps

Step 3: Hash Disclosures & Replace Original Claims

```
{
 "iss": "https://example.com",
 "type": "IdentityCredential",
 "cnf": {"jwk": {"kty": "RSA","n": "0vx....Kgw","e": "AQAB" } },
 "credentialSubject": {
     "_sd": [ "EW1o0egqa5mGcbytT5S-kAubcEjYEUwRkXlu2vC5l20",        ← ["GO0r26nO-iW50ZcAoOilFw", "given_name", "Max"]
           "FEx-ITHt41I8_cn0SS-hvoLneX_RGlJo_8o2xRNhfdk",         ← ["cSlbR135i0NjhsouMxrjjg", "family_name", "Mustermann"]
           "igg7H5fn2eBEMIEkE5Ckbm23QuwDJlTYoKRip08dYIc" ],       ← ["oHDt43Vwuhpo8mzaprgCcw", "email", "mustermann@example.com"]
   "address": {
     "_sd": [ "gqB5kmAwyry88aHjaAeO-USX6JOMaojukKsheo38O0c",       ← ["rGc0KtY6WmflywTTKEWIEQ", "street_address", "Musterstr. 23"]
           "w8InvxsPXdKoowuVpyBMgl1b9_R2b6Xpa3OYOIjgQro",         ← ["pGQMQx-2tH2XwC_eQCFn4g", "locality", "Berlin"]
           "vOnlYtcjr872fP3Wa75Ozl7c-6_MOVdlUNtwLKKxZw0" ]       ← ["TI15M8G5UIxPiWNZ-VLYBA", "country", "DE"]
  }
 }
}
```

SD-JWT in 5 Simple Steps

Step 4: Sign SD-JWT & Encode for Transport

```
{
  "iss": "https://example.com",
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZ
UkF6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc
3N1ZXIiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJSU0EiLCAibiI6ICIwdng3YWdvZ
WJHY1FTdS4uLi4tY3NGQ3VyLWtFZ1U4YXdhcEp6S25xREtndyIsICJlIjogIkFRQUIif
X0sICJ0eXBlIjogIkltZW50aXR5Q3JlZGVudGlhbCIsICJjcmVkZW50aWFsU3ViamVjd
CI6IHsiX3NkIjogWyJFVzFvMGVncWE1bUdjYnl0VDVTLWtBdWJjRWpZRVV3UmtYbHUyd
kM1bDIwIiwgIkZFeC1JVEh0NDFJOF9jbjBTUy1odm9MbmVYX1JHbEpvXzhvMnhSTmhmZ
GsiLCAiUXhKVi0yVjFlOG1jbHRSNnZWQzRtM3JlVTVhTkg5d2RKejJVZG1Sb0kxRSIsI
CJhdFVuMVRZd1JBbDRHUTdQZUV0WGFNdzJmNHVJVGlKclg0ODV3TTh2NjdFIiwgImZUT
XczdmtrRUx3TDFYTnVZSzhIN3pCS0NIdV91aWY2MFNsRzFweVhJVVEiLCAiaWdnN0g1Z
m4yZUJFTUlFa0U1Q2tibTIzUXV3REpsVFlvS1JpcDA4ZFlJYyIsICJ0cFV0bDcwaHBVX
3hucnZaaTBHaEdvUlIxam10MXpZZ3Z2NUlZMEF4N0tjIl0sICJhZGRyZXNzIjogeyJfc
2QiOiBbImdxQjVrbUF3eXJ5ODhhSGphQWVPLVVTWDZKT01hb2p1a0tzaGVvMzhPMGMiL
CAidk9ubFI0Y2pyODcyZlAzV2E3NU96bDdjLTZfTU9WZElVTnR3TEtLeFp3MCIsICJ3O
EludnhzUFhkS29vd3VWcHlCTWdsMWI5X1IyYjZYcGEzT1lPSWpnUXJvIl19fSwgImlhd
CI6IDE1MTYyMzkwMjIsICJleHAiOiAxNTE2MjQ3MDIyLCAic2RfZGlnZXN0X2RlcmI2Y
XRpb25fYWxnIjogInNoYS0yNTYifQ.1UHEPtLLUXOT51jH3gg-3C-ZidWzsB9Un-VxmM
VdQtTbLLhwDTB6HJtt15p43yCXTzdpiZxtDI6fr07Tp0Dy_Umg3Q5_FxFj4WHnsVuVzu
ASU8cFlGPi6xgH9D3w1G2hqepBS8DyQ5bA_p5kN_tKJVoP1xWhcQujRJ8kkEKQsRia4F
hrBldl8f41wgu_ipPqh1Ix4BVI7GJClZNx94nWPT7JUFkI6Y6JkahLf3S6gB0MxtmLAe
Y0qkuz8VeOZNfl_CDog55kVTkArorfoL6D6TEjI__-w6YyU0PnIRJXJ0wrYfoyhNl8LK
AP38QYMpdR7z_rsvHpQHzFAPTmevnHDg
  }
}
```

SD-JWT in 5 Simple Steps

Step 5: Base64url-encode Disclosures for Transport

```
{
  "iss": "https://example.com",
```

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZ
UkF6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc
3N1ZXIiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJSU0EiLCAibiI6ICIwdng3YWdvZ
WJHY1FTdS4uLi4tY3NGQ3VyLWtFZ1U4YXdhcEp6S25xREtndyIsICJlIjogIkFRQUIif
X0sICJ0eXBlIjogIkJsZW50aXR5Q3JlZGVudGlhbCIsICJjcmVkZW50aWFsU3ViamVjd
CI6IHsiX3NkIjogWyJFVzFvMGVncWE1bUdjYnl0VDVTLWtBdWJjRWpZRVV3UmtYbHUyd
kM1bDIwliwgIkZFeC1JVEh0NDFJOF9jbjBTUy1odm9MbmVYX1JHbEpvXzhvMnhSTmhmZ
GsiLCAiUXhKVi0yVjFIOG1jbHRSNnZWQzRtM3JlVTVhTkg5d2RKejJVZG1Sb0kxRSIsI
CJhdFVuMVRZd1JBbDRHUTdQZUV0WGFNdzJmNHVJVGlKclg0ODV3TTh2NjdFIiwgImZUT
XczdmtrRUx3TDFYTnVZSzhIN3pCS0NIdV91aWY2MFNsRzFweVhJVVEiLCAiaWdnN0g1Z
m4yZUJFTUlFa0U1Q2tibTIzUXV3REpsVFlvS1JpcDA4ZFlJYyIsICJ0cFV0bDcwaHBVX
3hucnZaaTBHaEdvUlIxam10MXpZZ3Z2NUlZMEF4N0tjIl0sICJhZGRyZXNzIjogeyJfc
2QiOiBbImdxQjVrbUF3eXJ5ODhhSGphQWVPLVVTWDZKT01hb2p1a0tzaGVvMzhPMGMiL
CAidk9ubFl0Y2pyODcyZlAzV2E3NU96bDdjLTZfTU9WZElVTnR3TEtLeFp3MCIsICJ3O
EludnhzUFhkS29vd3VWcHlCTWdsMWI5X1lyYjZYcGEzT1lPSWpnUXJvIl19fSwgImlhd
CI6IDE1MTYyMzkwMjIsICJleHAiOiAxNTE2MjQ3MDIyLCAic2RfZGlnZXN0X2RlcmI2Y
XRpb25fYWxnIjogInNoYS0yNTYifQ.1UHEPtLLUXOT51jH3gg-3C-ZidWzsB9Un-VxmM
VdQtTbLLhwDTB6HJtt15p43yCXTzdpiZxtDI6fr07Tp0Dy_Umg3Q5_FxFj4WHnsVuVzu
ASU8cFlGPi6xgH9D3w1G2hqepBS8DyQ5bA_p5kN_tKJVoP1xWhcQujRJ8kkEKQsRia4F
hrBldl8f41wgu_ipPqh1Ix4BVI7GJClZNx94nWPT7JUFkI6Y6JkahLf3S6gB0MxtmLAe
Y0qkuz8VeOZNfl_CDog55kVTkArorfoL6D6TEjI__-w6YyU0PnIRJXJ0wrYfoyhNl8LK
AP38QYMpdR7z_rsvHpQHzFAPTmevnHDg
```

```
  }
}
```

```
~WyJHTzByMjZuTy1pVzUwWmNBb09pbEZ3IiwgImdpdmVuX25hbWUiLCAiTWF4Il0
~WyJjU2xiUjEzNWkwTmpoc291TXhyampnIiwgImZhbWlseV9uYW1lIiwgIk11c3Rlcm1hb
m4iXQ
~WyJvSER0NDNWd3VocG84bXphcHJnQ2N3IiwgImVtYWlsIiwgIm11c3Rlcm1hbm5AZXh
hbXBsZS5jb20iXQ
~WyJyR2MwS3RZNldtZmx5d1RUS0VXSUVRIiwgInN0cmVldF9hZGRyZXNzIiwgIk11c3Rlc
nN0ci4gMjMiXQ
~WyJwR1FNUXgtMnRlMlh3Q19lUUNGbjRnIiwgImxvY2FsaXR5IiwgIkJlcmxpbiJd
~WyJUSTE1TThHNVVJeFBpV05aLVZMWUJBIiwgImNvdW50cnkiLCAiREUiXQ
```

→ Done!

Issuer

Issuance

SD-JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Disclosures
salt + claim name + claim value

End-User
(Holder)

Presentation

Verifier

Issuer

Issuance

SD-JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Disclosures
salt + claim name + claim value

```
WyJrSEhwOTEtdEFadDhtOUUOSmwOWGJRIiwgImdpdmVuX25hbWUiLCAiSm9obiJd
WyJQaklxcEdXbDRlQjRRcm9EaHFRdzB3IiwgImZhbWlseV9uYW1lIiwgIkRvZSJd
WyJ4bmlQNEpadE5XSUgtTGtfRHQtby1BIiwgInN0cmVldF9hZGRyZXNzIiwgIkRvZXN0cmVldCAxIl0
WyJLdGZzeHhUbTJtdzBZTFVjS1pVOHRBIiwgImxvY2FsaXR5IiwgIkFueXRvd24iXQ
```

Presentation

SD-JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

<u>Selected</u> Disclosures
salt + claim name + claim value

```
WyJrSEhwOTEtdEFadDhtOUUOSmwOWGJRIiwgImdpdmVuX25hbWUiLCAiSm9obiJd
WyJQaklxcEdXbDRlQjRRcm9EaHFRdzB3IiwgImZhbWlseV9uYW1lIiwgIkRvZSJd
```

End-User
(Holder)

Verifier

Issuer

Issuance

SD-JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

Disclosures
salt + claim name + claim value

```
WyJrSEhwOTEtdEFadDhtOUUOSmwOWGJRIiwgImdpdmVuX25hbWUiLCAiSm9obiJd
WyJQaklxcEdXbDRlQjRRcm9EaHFRdzB3IiwgImZhbWlseV9uYW1lIiwgIkRvZSJd
WyJ4bmlQNEpadE5XSUgtTGtfRHQtby1BIiwgInN0cmVldF9hZGRyZXNzIiwgIkRvZXN0cmVldCAxIl0
WyJLdGZzeHhUbTJtdzBZTFVjS1pVOHRBIiwgImxvY2FsaXR5IiwgIkFueXRvd24iXQ
```

SD-JWT
plain-text claims
+ hashed Disclosures

✓ signed by Issuer

<u>Selected</u> Disclosures
salt + claim name + claim value

```
WyJrSEhwOTEtdEFadDhtOUUOSmwOWGJRIiwgImdpdmVuX25hbWUiLCAiSm9obiJd
WyJQaklxcEdXbDRlQjRRcm9EaHFRdzB3IiwgImZhbWlseV9uYW1lIiwgIkRvZSJd
```

Holder-Binding JWT

nonce
audience
etc.

✓ signed by Holder

holder's public key

Presentation

Verifier

End-User
(Holder)

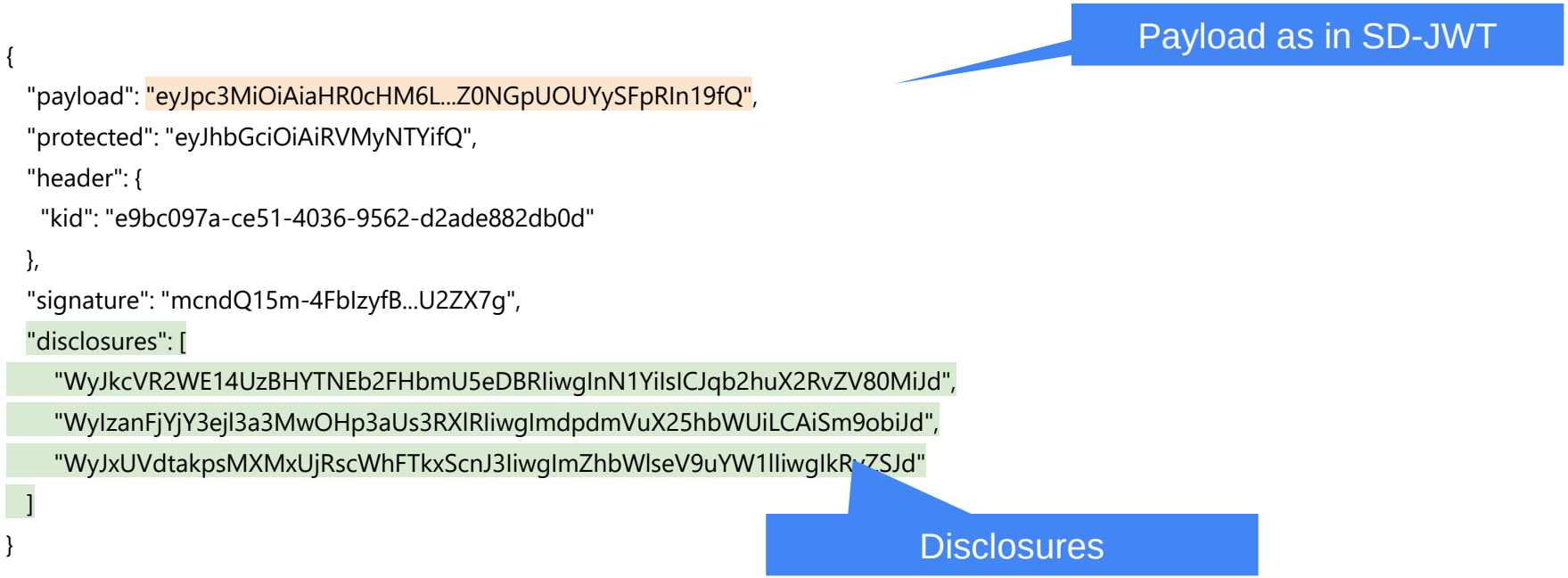
Verification

- Verify SD-JWT signature
- Hash over disclosed Disclosures
- Find hash digests in SD-JWT
- Replace disclosed claims in SD-JWT
- Check holder binding, if required.

Verification requires hash check!

Done!

SD-JWT with JWS using JSON serialization (proposal)



Compatibility

- Can be used with any JSON-based data format
  - JSON-LD
  - W3C-VC Data Model
  - OpenID Connect for Identity Assurance (OIDC4IA)
- Flexibility regarding holder binding
  - External signature
  - Key distribution
- Makes no assumptions on the transport protocol
  - E.g., OIDC4VC

Available, Testable, Auditable

All examples in specification generated via reference implementation:
oauthstuff/draft-selective-disclosure-jwt (Python)

tooling might be separated into another GH repo in the future

```
### Produce SD-JWT
sdjwt = SDJWT(
  user_claims,
  issuer,
  ISSUER_KEY,
  HOLDER_KEY,
  iat,
  exp,
)
```

Independent open-source implementations:

- Kotlin: IDunion/SD-JWT-Kotlin
- Rust: kushaldas/sd_jwt
- TypeScript: christianpaquin/sd-jwt
- TypeScript: chike0905/sd-jwt-ts
- Typescript: OR13/vc-sd-jwt **NEW**
- Java: authlete/sd-jwt **NEW**

IETF OAuth WG Draft

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

Daniel Fett
Authlete

Kristina Yasuda
Microsoft

Brian Campbell
Ping

mdoc

with one small caveat...

mdoc/MSO basics

- Defined in the ISO/IEC 18013-5 (https://www.iso.org/standard/69084.html)
  - focuses on mobile driving licence scenarios but can be used in other use-cases, too, in theory
- Includes a selective disclosure mechanism based on the salted hash values
- Expressed in CBOR
  - because NFC/BLE, “be happy it’s not ASN.1”
- mdoc is defined as “document or application that resides on a mobile device or requires a mobile device as part of the process to gain access to the mdoc”.
  - **Not originally defined as a “credential format”.**
- Mobile Security Object (MSO) is the issuer-signed object, contains digests

MSO (mobile security 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)

```
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
}
```

- Issuance is entirely out of scope.
  - How to send this mapping of direstID, random (salt), claim name and claim value during issuance is not defined.
  - there is also DeviceSigned. again how the issuer communicates IssuerSigned vs DeviceSigned is not defined.

mdocs: other facts

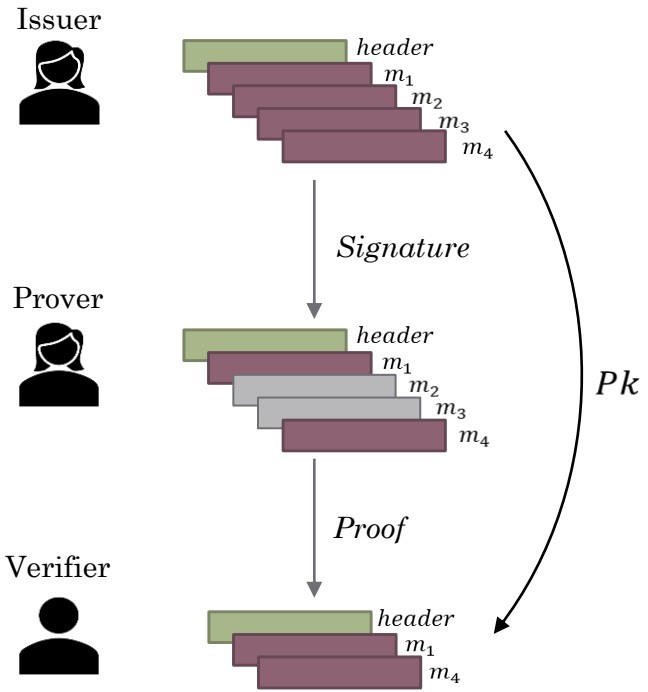
- predicates: `age_over_NN` claim
- unlinkability: issue the same copy of the credential with different User public key that can be used per verifier (to prevent RP-RP’ unlinkability)
- refresh: can be only the issuer’s signature over hashes, or the entire “mdoc”

Zero-Knowledge Proofs and BBS

Overview of ZKPs

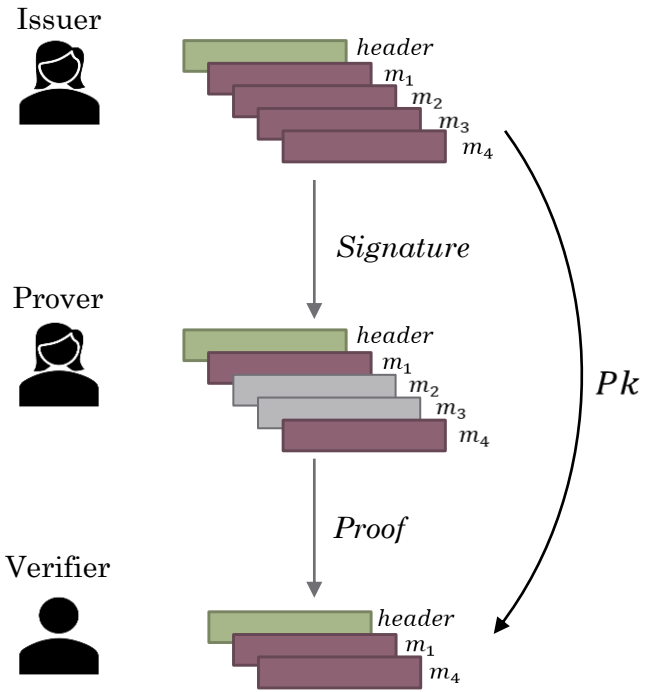
- ZKPs refer to a family of cryptographic algorithms and techniques which allow a proving party to prover a given statement is true without revealing any additional information.
- ZKPs have a variety of possible applications including with verifiable credentials.
- The BBS Signature scheme is one such algorithm that meets these properties.

How does BBS work?



- The Signer can sign multiple messages and a header with a constant size signature.
- The prover can generate a (randomized) proof for a subset of the signed messages.
- The verifier can validate that proof on those messages and header with the issuers public key.
- The header must always be disclosed by the Prover (intended to contain things like the algorithm identifier).

Some deeper details on BBS



- Based on pairing based cryptography
- Leverages curves like BLS 12-381
- Scheme is currently a work item of the IRTF CFRG
- Multiple independent interoperable implementations
  - MATTR pairing_crypto
  - https://github.com/Wind4Greg/grotto-bbs-signatures
  - https://github.com/dyne/zenroom
  - https://github.com/christianpaquin/bbs-signature
  - https://github.com/hyperledger/aries-bbssignatures-rs

Note - There are several more implementations that haven't aligned to the latest draft

Key Properties of interest from ZKPs for Verifiable Credentials

- Selective Disclosure: The ability to sign multiple messages/payloads and enable an intermediary (holder/prover) to selectively reveal messages from the set, while proving integrity back to the issuer.

- Unlinkable Proofs: Ability to generate proofs that are unlinkable from a cryptographic perspective. *A property that is impossible to achieve with existing digital signature schemes.*

- Private Holder Binding: Ability to bind a credential/signature to a key pair managed by the holder/verifier in a manner such that the public key isn't revealed during proof presentation to a verifier. *A property that is impossible to achieve with existing digital signature schemes.*

JSON Web Proofs and JOSE

What is JOSE?

JOSE is an abbreviation for JSON Object Signing and Encryption

It is an IETF working group which has defined representations of various security systems as JSON

- **Digital Signatures**
- **Encryption**
- **Message Authentication Codes**
- **Cryptographic Key representations**

The content being signed/encrypted *does not need* to be JSON, but often is.

Some Places that JOSE is Leveraged

JOSE aids applications in defining interoperable data protections, such as:

- **Cross-domain single sign** (profiled under OpenID Connect and FAPI)
- Supporting automation of retrieving/renewing **TLS certificates** (as ACMEv2)
- **Signaling** a security event happened, such as email **account compromise** (OpenID RISC/SSF)
- Allowing **VOIP systems** to interface across networks (SIP/STIR)
- Representing **identity credentials** about a person or other entity (W3C Verifiable Credentials)

Why are Identity Credentials Different?

- Identity Credentials often have **active participation** by a user agent
- They may hold significantly **more sensitive and identifying information**
- They may be used multiple times over an **extended lifetime**, creating new **risks of correlation**

This user agent (e.g. *wallet*) is an important stakeholder in the security system.

It needs additional capabilities and controls to limit the information being shared

JSON Web Proofs

A new work item in the *reanimated* JOSE Working Group

Goal to support newer cryptographic techniques for controlling information sharing, and supply features such as:

- Selective Disclosure
- Unlinkability
- Pseudonymity
- Computed answers (predicates)

Some of these may be achievable using existing techniques, while others may require new technologies like zero-knowledge proofs or even verifiable compute

JSON Web Proof work

- New containers representing information facets as individual payloads
- Issued/presented forms, analogous to credentials and presentations

JSON Proof Algorithms

- Describe how existing algorithms (and emerging ones like BBS) can be used
- What capabilities they provide for limiting information disclosure
- How to represent cryptographic material using JSON Web Keys

JSON Proof Tokens

- A token format comparable to JWTs for representing claims built on top

Conclusion

- JSON Web Proofs are meant to aid in privacy-critical use cases
- Target needs of future credential adoption
  - long-lived credentials
  - rich records like medical/educational transcripts
- Early draft stage,
  welcome comments and assistance
- Some early prototypes,
  further implementations welcomed

Draft Specifications

Closing Remarks and Discussion