SD-JWT with Commitments DRAFT

Abstract

In order to allow zero knowledge proofs on attributes within SD-JWTs, the hash algorithm definition is extended to allow the usage of commitment schemes. With commitment schemes, we can for example use simple sigma protocols to derive proofs about age, or to bind various VCs without revealing the actual properties (e.g. when using claim based binding for bridging).

1. Introduction

There are various use cases where selective disclosure alone is not enough to secure privacy, or is too restricted for arbitrary use cases. A good example is age verification, where the current state of the art is to include multiple boolean flags based on the age during issuance. This has obvious drawbacks, as for example a person whose age reaches a threshold needs a new credential (as the boolean flag is fixed in the signed JWT-part). With this draft we could use a simple sigma protocol to proof "older than" for any threshold.

When using batch issuance to reduce link-ability (as currently used signature schemes don't allow for rerandomization), there needs to be a way of refreshing batches. In the current process this is only possible in redisclosing all personal data to the issuing party. In this draft we sketch a method in Appendix C on how rerandomization of a VC can be done without revealing attributes inside the SD-JWT by adding a new blinding factor to the present commitments.

1.1. Hash Function Claim

The `_sd_alg` of the SD-JWT *MUST* be one of the list members in Appendix A. If `_sd_alg` is one of the values in that list, the SD-JWT *MUST* add a `_sd_alg_param` value containing the required parameters for the commitment scheme.

1.1.1. Commitment Scheme specification

The `_sd_alg_param` contains the following fields:

`commitment_scheme`: REQUIRED. The commitment scheme to use. In Appendix A the parameters for commitment schemes defined in this specification are given.

1.1.2. Commitment Linking

For zero knowledge proofs, a unique linking of attributes to commitments needs to be given. As such, if `_sd_alg` is one of the elements in Appendix A the top level object needs a map of attribute names to indices in the `_sd` array. This object itself can be selectively disclosable to not reveal more information than necessary:

`com_link`: OPTIONAL. A map of attribute name to index in the `_sd` object. This map needs to be present if commitments in `_sd` of this level of disclosures is needed.

1.2. Keybinding JWT

The keybinding JWT is extended with the following fields:

`zkp_proofs`: OPTIONAL. An array of `ZkpProof` structs containing zero knowledge proofs over the respective inputs.

1.2.1. ZkpProof

The `ZkpProof` struct contains the values required to verify the zero knowledge proof on the properties:

`inputs`: REQUIRED. An array of `public` and `private` inputs. Public input means a revealed value, such as a age threshold. Private inputs are for example commitments from other credentials that are not revealed but used within the zero knowledge proof
`system`: REQUIRED. Array of length of `input` variables, specifying the *linear* equation that should be statisfied.
`context`: REQUIRED. Byte-Array containing the relevant proof transcript bytes used for the Fiat-Shamir transform.
`proof`: REQUIRED. The proof value that can be verified.
`proof_type`: REQUIRED. A proof type specifying how to deserialize `proof`.

1.2.1.1. equality_proof

An equality proof is serialized as bytes (each scalar and RistrettoPoint has 32 bytes), where `com1` and `com2` are the randomized commitments from stage one of the sigma protocol:

```
struct EqualityProof {
    g: RistrettoPoint,
    h: RistrettoPoint,
    s1: Scalar,
    r1: Scalar,
    s2: Scalar,
    r2: Scalar,
    com1: RistrettoPoint,
    com2: RistrettoPoint,
}
```

1.2.1.2. Public Inputs

Public inputs are clear text values that are lifted into the scalar field of the respective scheme. Requirements and algorithms to do so are defined in the respective commitment schemes.

`public_value`: OPTIONAL. JSON value that can be lifted into the scalar field. Provided as plaintext.

1.2.1.3. Private Inputs

Private inputs are for example commitments from other credentials for which an equality should be proven. Those are already byte representations of elements of the respective group/ring of the commitment scheme.

`path`: REQUIRED. ClaimsPointer to `com_link` entry for the relevant commitment
`value`: REQUIRED. The base64-url-safe encoded commitment used in the proof.

Appendix A

A.1 List of Commitment Schemes

- ec_pedersen

A.2 ec_pedersen

A.2.1 Parameters

When using the `ec_pedersen` format the following parameters are used:

`public_params`: REQUIRED. Object containing the generator used for the value (g) and the blinding (h)
`crv`: REQUIRED. Curve used for the Pedersen commitment.

A.2.2 Hashing

To produce commitments that can be used, number types should be converted to values on the scalar field, and any other type should be converted using `hash_to_curve` algorithm. The first value of the disclosures array MUST be used as the blinding, and the attribute name MUST be added to the proof transcript.

Appendix B

An non normative example of a SD-JWT proofing equality of two claims accross two SD-JWTs revealing no properties:

SD-JWT 1

eyJhbGciOiJFUzI1NiJ9.eyJfc2QiOlsibG1UNEd5LWY2VGl6aFQ5QU96OFZQd2tMZ2JF
eTYzYklDS1pLc1k2U0ZIayIsIlRLTlI1ejlLQThROVc1U0s4RU1WSnJvOW9pZHJQdWJEW
mZYRkxMTDJjUUEiXSwiX3NkX2FsZyI6ImVjX3BlZGVyc2VuIiwiX3NkX2FsZ19wYXJhbS
I6eyJjb21taXRtZW50X3NjaGVtZSI6eyJwdWJsaWNfcGFyYW1zIjp7ImciOiJVbEFOSDB
CZWxoRFl5OVVjZlFWbUdyUndtOV9vdnZEejNhemUzT1hRNHpRIiwiaCI6ImpocFBqOGJh
NTVBc2VyMGQ2V1I3OTBZOGgyNVRFTG95SU9WUW93a3FIa1EifSwiY3J2IjoiZWQyNTUxO
SJ9fSwiY29tX2xpbmsiOnsidGVzdCI6MCwiZG9iIjoxfSwiaXNzIjoic2FtcGxlX2lzc3
VlciIsImlhdCI6MTc2ODIzMTM5MiwibmJmIjoxNzY4MjMxMDkyLCJleHAiOjE3NjgyMzE
3NTJ9.qoSek61vtcUTdgOYoMuf8q8VSL_nds_zycRHL1PxBanqxHRYXedNUzSGU8RKePR
CtyIq_cEL2cixN_HZNK5Q9Q~eyJ0eXAiOiJrYitqd3QiLCJhbGciOiJFUzI1NiJ9.eyJu
b25jZSI6Im5vbmNlIiwiYXVkIjoicHJvb2ZlciIsInNkX2hhc2giOiJXYVdZV0tSVVVkd
S1yTU9UTk42eDBZYk9VZi1ZRGpPR1V6SExMMDhYWEE4IiwicHJvb2ZzIjpbeyJpbnB1dH
MiOlt7IlByaXZhdGUiOnsicGF0aCI6ImRvYiIsInZhbHVlIjoiVEtOUjV6OUtBOFE5VzV
TSzhFTVZKcm85b2lkclB1YkRaZlhGTExMMmNRQSJ9fSx7IlByaXZhdGUiOnsicGF0aCI6
ImRvYiIsInZhbHVlIjoieU5ISWV1dm9FRWVsVFlTQ1c5Qjg3MTRFSTdRcDBmZzdkbm12M
TZnN29sVSJ9fV0sInN5c3RlbSI6WzEsLTFdLCJjb250ZXh0IjoiWkc5aVVsQU5IMEJlbG
hEWXk5VWNmUVZtR3JSd205X292dkR6M2F6ZTNPWFE0elNPR2stUHh0cm5rQ3g2dlIzcFp
IdjNSanlIYmxNUXVqSWc1VkNqQ1NvZVJBIiwicHJvb2YiOiJReHM3MUllTkJyQVUtSkhj
Q09RbjloVk9lUXlsQzJ5VGg1NHl6dlhuR1FEU1dlS19xQ1RLRnFIUlVGd1FvcmRlaTFzd
VVuZU4tT2E0MkhrdS05cDNERU1iTzlTSGpRYXdGUGlSM0Fqa0pfWVZUbmtNcFF0c2s0ZW
VNczcxNXhrQUw3TTVJYUpQclRzSnZKMmJVSnB3dFhib0RYbnNOWkplVmJ4ZTB5OFhqQWE
0UEd6UzNGMWI3WGg1aUtZczhQVzl4S1V6LXExMVZGem1HanhQdklXNExsQ1FwWExhSThv
QXVYVjBhVGFVLTRYNm5wcVR4YWRWMFR4eVhzVUN4MWhIVWxBTkgwQmVsaERZeTlVY2ZRV
m1HclJ3bTlfb3Z2RHozYXplM09YUTR6U09Hay1QeHRybmtDeDZ2UjNwWkh2M1JqeUhibE
1RdWpJZzVWQ2pDU29lUkEiLCJwcm9vZl90eXBlIjoiZXF1YWxpdHkifV0sImlhdCI6MTc
2ODIzMTM5MiwibmJmIjoxNzY4MjMxMDkyLCJleHAiOjE3NjgyMzE2OTJ9.tZfSCpZnqPc
AhYx_a5cwkUIgwC3QNbc4d_FpBt0EiL4W4o-
N1LADDmXPQLO3irptEGCKNN-8Kjp7QUiW
QtYaDA

SD-JWT 2

eyJhbGciOiJFUzI1NiJ9.eyJfc2QiOlsiQkp2WFQ3OVJnZGtmMGlnalFKTEpEd2lKdUd1
WGdUSnpnbHVyTGxVU0RCOCIsInlOSElldXZvRUVlbFRZU0NXOUI4NzE0RUk3UXAwZmc3Z
G5tdjE2ZzdvbFUiXSwiX3NkX2FsZyI6ImVjX3BlZGVyc2VuIiwiX3NkX2FsZ19wYXJhbS
I6eyJjb21taXRtZW50X3NjaGVtZSI6eyJwdWJsaWNfcGFyYW1zIjp7ImciOiJVbEFOSDB
CZWxoRFl5OVVjZlFWbUdyUndtOV9vdnZEejNhemUzT1hRNHpRIiwiaCI6ImpocFBqOGJh

```
NTVBc2VyMGQ2V1I3OTBZOGgyNVRFTG95SU9WUW93a3FIa1EifSwiY3J2IjoiZWQyNTUxO
SJ9fSwiY29tX2xpbmsiOnsidGVzdCI6MCwiZG9iIjoxfSwiaXNzIjoic2FtcGxlX2lzc3
VlciIsImlhdCI6MTc2ODIzMTM5MiwibmJmIjoxNzY4MjMxMDkyLCJleHAiOjE3NjgyMzE
3NTJ9.7bV60ky3nH6zpsNcgjgRq4zYnPWBB1GdSgLBuZu_Mjzj2PWH6sonizimvB1VuEC
ZGs6eUYicpr_tiIj33GUaNQ~eyJ0eXAiOiJrYitqd3QiLCJhbGciOiJFUzI1NiJ9.eyJu
b25jZSI6Im5vbmNlIiwiYXVkIjoicHJvb2ZlciIsInNkX2hhc2giOiJzUHNKTTF6MVlDX
0otVmUxb2VyVG5vT0dvbFQ1aHZzQTJzMDZuQzEtckFnIiwicHJvb2ZzIjpbeyJpbnB1dH
MiOlt7IlByaXZhdGUiOnsicGF0aCI6ImRvYiIsInZhbHVlIjoiVEtOUjV6OUtBOFE5VzV
TSzhFTVZKcm85b2lkclB1YkRaZlhGTExMMmNRQSJ9fSx7IlByaXZhdGUiOnsicGF0aCI6
ImRvYiIsInZhbHVlIjoieU5ISWV1dm9FRWVsVFlTQ1c5Qjg3MTRFSTdRcDBmZzdkbm12M
TZnN29sVSJ9fV0sInN5c3RlbSI6WzEsLTFdLCJjb250ZXh0IjoiWkc5aVVsQU5IMEJlbG
hEWXk5VWNmUVZtR3JSd205X292dkR6M2F6ZTNPWFE0elNPR2stUHh0cm5rQ3g2dlIzcFp
IdjNSanlIYmxNUXVqSWc1VkNqQ1NvZVJBIiwicHJvb2YiOiJReHM3MUllTkJyQVUtSkhj
Q09RbjloVk9lUXlsQzJ5VGg1NHl6dlhuR1FEU1dlS19xQ1RLRnFIUlVGd1FvcmRlaTFzd
VVuZU4tT2E0MkhrdS05cDNERU1iTzlTSGpRYXdGUGlSM0Fqa0pfWVZUbmtNcFF0c2s0ZW
VNczcxNXhrQUw3TTVJYUpQclRzSnZKMmJVSnB3dFhib0RYbnNOWkplVmJ4ZTB5OFhqQWE
0UEd6UzNGMWI3WGg1aUtZczhQVzl4S1V6LXExMVZGem1HanhQdklXNExsQ1FwWExhSThv
QXVYVjBhVGFVLTRYNm5wcVR4YWRWMFR4eVhzVUN4MWhIVWxBTkgwQmVsaERZeTlVY2ZRV
m1HclJ3bTlfb3Z2RHozYXplM09YUTR6U09Hay1QeHRybmtDeDZ2UjNwWkh2M1JqeUhibE
1RdWpJZzVWQ2pDU29lUkEiLCJwcm9vZl90eXBlIjoiZXF1YWxpdHkifV0sImlhdCI6MTc
2ODIzMTM5MiwibmJmIjoxNzY4MjMxMDkyLCJleHAiOjE3NjgyMzE2OTJ9.22MHj9iGfw_
ik0Ckn6ykD9IAJ55xdq6kgptxTJJcvSW35zp7Y_O9eSBKb04IbkQKZdbi53ylW7PJiLNJ
z3LRcA
```

The proof is generated over the `dob` property contained in both SD-JWTs.

Appendix C

C.1 Refresh of batch issued credentials in OID4VCI context

Instead of using a refresh token to refresh the batches of a credential, the issuer could allow issuing of new credentials after presentation of a valid original credential. Instead of reissuing the credential, the issuer would just take all commitments in the `sd` object and add a random blinding factor to the respective commitments (using the generator defined in `commitment_scheme`). The returned SD-JWT's disclosures would then only contain a delta to the blinding factor, the wallet could use to calculate the actual blinding (by adding it to the blinding factor of the credential used in the request). Furthermore, using request and response encryption as defined in Section 10, OpenID for VCI, the wallet could use a TOR like routing to hide its network trail to the issuer.