

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:

TODO: Explain exactly how this works for deeply nested structures

`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

TODO: Can we have ZKPs without key binding? If so how?

The keybinding JWT is extended with the following fields:

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

1.2.1. ZkProof

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

TODO: should we allow random linear equations, and if so how does this impact the serialization mechanism.

`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 satisfied.

`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 {
    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` or array 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. For nested objects the JCS (<https://www.rfc-editor.org/rfc/rfc8785.html>) **MUST** be used. The first value of the disclosures array **MUST** be used as the blinding, and the attribute name **MUST** be added to the proof transcript.

TODO: Is there a better way to handle non numerical values? E.g. lexicographical representation of strings to allow ordering?

TODO: Do we need to consider too big numerical values?

TODO: How are floating point numbers encoded? Fixed precision and then using integers ("multiply out" the decimals?)

Appendix B

This Appendix goes over a fictitious example of presenting a Diploma, which is not bound to a specific user hold key material, together with an identity statement, which is device (aka user) bound. Normally, to have confidence that both were issued to the same person, we would need to disclose various properties and show equality between them (as the by default used sha hash does not allow for ZKP).

Here we present both credentials, but only disclose the subject and the grade of the diploma, while providing a zero knowledge proof of equality between multiple different attributes.

B.1 Test Vectors

B.1.1 Identity Card

B.1.1.1 Issuer Verification Key

```
{  
    "kty": "EC",  
    "crv": "P-256",  
    "x": "N3g_o1SqMpLQMqSdyGbG9nK01fWUBBny-h-xxlFQlFk",  
    "y": "5P7RM7ul-bCrTfcnkWrBGKgwd0fG7RFwhNay1kvLu8"  
}
```

B.1.1.2 Holder Private Key

```
{  
    "kty": "EC",
```

```

    "crv": "P-256",
    "x": "0q1azTI-nfXRjxeoxqwktbSayE7Sd-2S0kp2MdCXdJM",
    "y": "Xz4eqZG6bq017tCtqsx97KiJzgLzbSQv0zQJBFasXKE",
    "d": "dbUsHxvpAb_D26ok8T2D5EugxWfUu0faPU9ueYhc56k"
}

```

B.1.1.3 Identity Card Claims

```

{
  "sub": "user_42",
  "given_name": "John",
  "family_name": "Doe",
  "email": "johndoe@example.com",
  "phone_number": "+1-202-555-0101",
  "phone_number_verified": true,
  "address": {
    "street_address": "123 Main St",
    "locality": "Anytown",
    "region": "Anystate",
    "country": "US"
  },
  "birthdate": 19400101,
  "updated_at": 1570000000,
  "nationalities": [
    "US",
    "DE"
  ]
}

```

B.1.1.4 Identity Card SD-Jwt

eyJhbGciOiJFUzI1NiIsImtpZCI6IjEyMyJ9.eyJzdWIiOiJ1c2VyXzQyIiwidXBkYXR1ZF9hdCI6MTU3MDAwMDAwMCwiY25mIjp7Imp3ayI6eyJrdHkiOijFQyIsImNydiI6IlAtMjU2IiwieCI6Ik9xMWF6VEktbmZYUmp4ZW94cXdrdGJTYX1FN1NkLTJTMGtwMk1kQ1hkSk0iLCJ5IjoiWHo0ZXFaRzZicTAxN3RDdHFzeDk3S21KemdMemJTUXZPe1FKQkZhc1hLRSJ9fSwiX3NkIjpbImpsLXh6a0ZlQUhjeHloTkFwR1dHQkx1ZHmamhabWtpcTRrank1VEpQRKUiLCJHRFV1OWtmUWxISVV5bXh0aEdQZEhLeXBQWUUwY3ZURmFuekcxZHduUkJJIwiWkVRT0NZMEhYNC0yZzhvU2tSN11fv204NnFVVU9sZDNBOFFT1k4U0FUbyIsImVOd0F4WnQ3UHhVMVQxbmhPYlhTbkRMU2s3MXk3Y2pfakNHcD1YSDZTUTQilCI0Z0tkSGZCc1FXUwdZRzdDT1hWMU16SThBeDlrOE5XQ0xnUzBYSFFFRWt3IiwiZG9VZFRMM1pJZ25YOD1acUlzLWk4MWNiRnZtOGJrSERjUU9heTZ6dFJnYyIsImtPNTZjY3V1aHF1clpYdGpNeE1QNWNwZDFac2V1RVZKOUY0N25l0HBUSGMiLCJkc1hNTVVsRFZwMTbjdkdLeXAtbDc1R1VJRkvjUGhwZ0hXN1VmVE1Nem04I10sI19zZF9hbGdfcGFyYW0iOnsiY29tbW10bWVudF9zY2h1bWUiOnsicHVibGljX3BhcmFtcyI6eyJnIjoiNUFsOUJCRTdOU2pacXhRSktrQjF4bnZCM2t4R1FHdm1fbHBpNW1uQU1WVSIIsImgioiJqa3h1dk50S1Y0X2FRS1pxSUZYU11allpSk92VFZSOW5hb1NPdWVmalFJIn0sImNydiI6ImVkJU1MTkifX0sI19zZF9hbGciOij1Y19wZWRLcnNlbiIsImNvbV9saW5rIjp7ImdpdmVuX25hbWUi0jAsImZhbW1seV9uYW11IjoxLCJ1bWFpbCI6MiwighvbmVfbnVtYmVyIjozLCJwaG9uZV9udW1izXJfdmVyaWZpZWQi0jQsImFkZHJlc3Mi0jUsImJpcnRoZGF0ZSI6NiwbmF0aW9uYWxpdl1cyI6N319.GkA6zkXjYPhjtppnBGEKAvrHLUVUbR4JnkPI4qe4h1oKCxh6RfgRZAFLjYU3BmMM2do30oz_SXQi18wIr7Ulcw~WyJPV21TX05sTEFCRVVLRFhZX2ZDWm1HRU5DSkliOVQ0b3M4eDVxTUpRc2drIiwiZ212ZW5fbmFtZSIIsIkpvag4iXQ~WyJyVXo3emJFZGVYR1Q3VGZqRzRnRk9DWU1u0VAyZ3p2eHBGSUVuQjRhaXdZIiwiZmFtaWx5X25hbWUiLCJEb2UiXQ~WyJzUj1NX0JCYzFyVzB0Mi1ZUDdJcE5BQXRiXz1EMHN3aH1MUC1LZ1cwQXdviwiZw1haWwiLCJqb2huZG91

QGV4YW1wbGUuY29tI10~WyJYdVB0UjEzSVhXSERLNURKMkV3cVQ5ZGNqWmRBTv9JYm9zWmxZekJTS3dBIiwicGhvbmVfbnVtYmVyIiwiKzEtMjAyLTU1NS0wMTAxI10~WyIxTFk0ZTJWS21kUHF0M25oYk1kemhzY3RoRDZrWXQ1djhvZjRQeXlnMmdRIiwicGhvbmVfbnVtYmVyX3Z1cm1maWVkIix0cnV1XQ~WyJHQ0pOSExTdzRrU2JoaUR3ZEVPQjV1S01uU3kxeFRVb01TT3NtZzdHdFIwiYWRkcmVzcyIseyJzdHJ1ZXRFYWRkcmVzcyI6IjEyMyBNYWl1uIFN0IiwibG9jYWxpdHkiOiJBbn10b3duIiwicmVnaW9uIjoiQW55c3RhGUILCJjb3VudHJ5IjoiVVMifV0~WyJxNVpCM29Fa01ib2NBT0tneT1DOWJUaW4wN0ZULV1BanNJdHh1TmRGZ1E4IiwiYmlydGhkYXR1IiwxOTQwMDEwMV0~WyJXZDZVS29GeVZFSngxRmZGZGNjWnJCdDYwLXYtbVBOR1RFV3VrM3U3U0EwIiwibmF0aW9uYWxpdGllcyIsWyJVUyIsIkRFI11d~

B.1.2 Diploma

B.1.2.1 Issuer Verification Key

```
{  
  "kty": "EC",  
  "crv": "P-256",  
  "x": "NMV5Bdl85g3g_55y-n34fQHFe46VffI4T_R9AavAFVU",  
  "y": "PYgzY-ufz3iszlhvjaleh7_s11izVMbZ-Pis7QgoKIE"  
}
```

B.1.2.2 Diploma Claims

```
{  
  "sub": "user_42",  
  "given_name": "John",  
  "family_name": "Doe",  
  "subject": "Computer Science",  
  "final_grade": 4.8  
}
```

B.1.2.3 Diploma SD-Jwt

eyJhbGciOiJFUzI1NiIsImtpZCI6IjEyMyJ9.eyJzdWIiOiJ1c2VyXzQyIiwiX3NkIjpBImhwLUtzX3o4RjB4UV80TUFybEpMbjiZ1VVdkTzaXZwT3c1Y3dWb3pIQ1kiLCJKQ0pUdXozTHN1eUd5WHBsYUhMRVprZVVNMEwzNV93T3NLR2xFaWJxb2x3IiwiTmkyVENVTVwMTQ1cWFsYW0tVmVPQWd0YmdtdDQtTF9fc3ZBRU45SGN4QSIiIm5KLVBIUldmbmU5Sk9PeDBZNnNiQjNkUnlfR0pmVnprV3FXcWd6Z1VfQ0EiXSwiX3NkX2FsZ19wYXJhbSI6eyJjb21taXRtZW50X3NjaGVtZSI6eyJjcnyiOijZDI1NTE5IiwiHVibGljX3BhcmFtcyI6eyJoIjoiwmd2SVJPNHZpU1JVMghCWFh2WEM2c21faWhsQmhoTVAxTW9VZFJIb3VIayIsImciOiIzUGVPSmwzZ1RHQTd4SC1jV0ZwaENURERtdDcwUDZOR1d0Y295aVVoQWxBIn19fSwiX3NkX2FsZyI6ImVjX3B1ZGVyc2VuIiwiY29tX2xpbsmsiOnsiZ212Zw5fbmFtZSI6MCwiZmFtaWx5X25hbWUiOjEsInN1Ymp1Y3QiOjIsImZpbmFsX2dyYWR1IjofX0.gDs0zxTqgJKMuBuJv1UvSC8R_tshugj3-HpwV062H2wrvnos1kDwMtUknCSnNPLEQdVloMp13sSZ0hZVbcJBZQ~WyJYTpmTXdjUWw0czZMNkdZYVFFM1lHM1ZMZTg0VXFVdG1QTEtanNpU1EwIiwiZ212Zw5fbmFtZSIiKpvag4iXQ~WyJEM190c1NtNzYxQzc0RDhyQWNHaExhb1dVb3JaUFpkbTBRVmJNbEp2QndzIiwiZmFtaWx5X25hbWUiLCJEb2UiXQ~WyJ2MXl1akU5bDloYmxsR11nY3hPaU1WS19jcUZfUGYyR0dHTUhPvkxaZGc0IiwiC3ViamVjdCIiKnvbXB1dGVyIFNjaWVuY2UiXQ~WyIwWW9tW1dPRmU1V0o0OGF1Z2FZMjR3M1R1UGHha2Z5NGt5V2x6aENSNXdBIiwiZmluYWxfZ3JhZGUiLDQuOF0~

B.1.2.4 Presentations

B.1.2.4.1 Identity with ZKPs

B.1.2.4.1.1 Encoded VP-Token including ZK-Proofs

eyJhbGciOiJFUzI1NiIsImtpZCI6IjEyMyJ9.eyJzdWIiOiJ1c2VyXzQyIiwidXBkYXR1ZF9hdCI6MTU3MDAwMDAwMCwiY25mIjp7Imp3ayI6eyJrdHkiOijFQyIsImNydiI6IlAtMjU2IiwieCI6Ik9xMWF6VEktbmZYUmp4ZW94cXdrdGJTYX1FN1NkLTJTMGtwMk1kQ1hkSk0iLCJ5IjoiWHoZXFaRzZicTAxN3RDdHFzeDk3S21KemdMemJTUXZPe1FKQkZhc1hLRSJ9fSwiX3NkIjpbImpsLXh6a0Z1QUhjeHloTkFwR1dHQkx1ZHmamhabWtpcTRrank1VEpQRkUiLCJHRFV1OWtmUWxISVV5bXhOaEdQZEhLeXBQWUUwY3ZURmFuekcxZHduUkJJIwiWkVRT0NZMEhYNC0yZzhvU2tSN11fV204NnFVVU9sZDNBOFFT1k4U0FUbyIsImVOd0F4WnQ3UHhVMVQxbmhPY1hTbkRMU2s3MXk3Y2pfakNHcD1YSDZTUTQilCI0Z0tkSGZCclFXUwdZRzdDT1hWMU16SThBeDlrOE5XQ0xnUzBYSFFFRWt3IiwiZG9VZFRM1pJZ25YODlacUlzLwk4MWNiRnZtOGJrSERjUU9heTZ6dFJnYyIsImtPNTzjY3V1aHF1c1pYdGpNeE1QNWNwZDFac2V1RVZKOY0N25l0HBUSGMiLCJkc1hNTVsRFZwMTBjdkdLeXAtbDc1R1VJRkVjUGhwZ0hXN1VmVE1Nem04I10sI19zZF9hbGdfcGFyYW0iOnsiY29tbW10bWVudF9zY2h1bWUiOnsicHVibGljX3BhcmFtcyI6eyJnIjoiNUFsOUJCRTdOU2pacXhRSktrQjF4bnZCM2t4R1FHdm1fbHBpNW1uQU1WVSIisImgioiJqa3h1dk50S1Y0X2FRS1pxSUZSYU11allpSk92VFZSOW5hb1NPdWVmalfJIn0sImNydiI6ImVkmjU1MTkifX0sI19zZF9hbGciOiJ1Y19wZWRLcnNlbiIsImNvbV9saW5rIjp7ImdpdmVuX25hbWUi0jAsImZhbW1seV9uYW11IjoxLCJ1bWFpbCI6MiwicGhvbmVfbnVtYmVyIjozLCJwaG9uZV9udW1iZXjfmdVyaWZpZWQi0jQsImFkZHJlc3Mi0jUsImJpcnRoZGF0ZSI6NiwbmF0aW9uYWxpdlcyI6N319.GkA6zkXjYPhjtppnBGEKAvrHLUVUbR4JnkPI4qe4h1oKCxh6RfgRZAFLjYU3BmMM2do30oz_SXQi18wIr7Ulcw~eyJ0eXAiOiJrYitqd3QjilCJhbGciOiJFUzI1NiJ9.eyJub25jZSI6IjFWMTNRRXJQYjhqR1NpSU5kR2RnWUV10WU2bHhVU09nIiwiWF0IjoxNzY4NDc2NjU3LCJzZF9oYXNoIjoiQXBZd2FGN1M4Vhk1VWFLa2h1cXFvYzdMTn1QU11JaT13R1VNZ2hGRDlnOCIsInprX3Byb29mcyI6W3siaW5wdXRzIjpbeYJQcm12YXR1Ijp7InBhdGgiOlsiY29tX2xpbsiLCJnaXZ1b19uYW11I10sInZhbHV1IjoiamwteHprRmVBSGN4eWh0QXBHV0dCTGVkeGZqaFpta21xNGZqeTVUS1BGRSJ9fSx7I1ByaXZhdGUIOnsicGF0aCI6WyJjb21fbGluayIsImdpdmVuX25hbWUiXSwidmFsdWUi0iJocC1Lc1960EYweFFfNE1BcmxKTG4yM2dVVXZLWW12cE93NWN3Vm96SENZIn19Xswic31zdGVtIjpbeMSwtMV0sImNvbnR1eHQiOiJaMmwyW1c1ZmJtRnRaZVFKZ1FRUk96VW8yYXNVQ1NwQWRjWjd3ZDVNUmtCcJV2NWFZdVpwd0NGVmpreGV2TnRKVjRfYVFLWnFJR1JhSXVqWW1KT3ZUV1I5bmFvU091ZWZqUUxjOTQ0bVhXQk1ZRHZFzjV4WVdtRUpNTU9hM3ZRX28wVmExeWpLS1NFQ1VHWUx5RVR1TDRrVVZOSVFWMTcx d3VySXY0b1pRWV1URD1US0ZIVVI2TGg1QVFJRCIsInByb29mIjoid0RMQ1YyX2NaeTBqsSVFrS1hfVVdFeE5oS1JwN2JWd2s1VXdhNS1KGFBM1E3U31HRk4yMk9iTzJfa01Mb jNHMk04V2dCeEk3QkhkdzlN0p1d2xRQjhBeXdWZHYzR2N0SX1FSkNWxzFGaE1UWVNvYwUyMWN KT1ZNR3VmaVYyZ05VM2xadWdrenBTMmd1N0VX2c2VjdxZk5QT3JHc21iWjZudmdPdFc5 bmd6eVpZTmFTS3RmeDJWazJELUg2Y11DQjIyY1I5YXNqSjhaZ0RCdy1TWWhKUTZOSmFGNjRhamgwRDdmZHtN1Jfb1BfcGZJeDFqRTYxQWtnY1BBdzlkVCIsInByb29mX3R5cGUioiJ1cXVhbGl0eV9wcm9vZiJ9LhsiaW5wdXRzIjpbeYJQcm12YXR1Ijp7InBhdGgiOlsiY29tX2xpbsiLCJmYW1pbH1fbmFtZSJdLCJ2YWx1ZSI6Ik dEVXU5a2ZRbEhJVX1teE5oR1BkSEt5cFBZRTBjdlRGYW56RzFkd25SQkkifX0seyJQcm12YXR1Ijp7InBhdGgiOlsiY29tX2xpbsiLCJmYW1pbH1fbmFtZSJdLCJ2YWx1ZSI6Ik pDS1R1ejNMc2V5R31YcGxhSExFWmt1VU0wTDM1X3dPc0tHbEVpYnFvbHcifX1dLCJzeXN0ZW0i0lsxLC0xXswiY29udGV4dCI6IlptRnRhV3g1WDI1aGXWGtDWDBFRVRzMu tObXJGQWtxUUhYR2U4SGVURVpBYS1iLVdtTG1hY0FoV1k1TVhyemJTVmVQMmtDbWFpQ1VXaUxvMk1pVHIwMVVmWjJxRWpybm40MEMzU

GVPSmwXZ1RHQTd4SC1jV0ZwaENURERTdDcwUDZORld0Y295aVVoQWxCbUM4aEU3aS1KRkZUU0VGZGU5Y0xxeUwtS0dVR0dFd19VeWhSMUV1aTR1UUVDQXciLCJwcm9vZiI6IlpDWGZ3VGhRTTBtTnk1Zk0tX05uV09yZGIxdkFPcWJ5eG44NDNrcjIx0Z1eE8tNETzYUtEYUVQNDJOSGZSSnNrTU9mU1h1aFp5ZTk0R0FBX1FVSkFHUWwzOEU0VUROSmpjdVh6UHZ6WjFqcTNXOWJ3RHFTOHNaX09ONUs5dGNCaFVXS1hVdjFDcUt6UC1BNEMyVEI1S1FFaVJWZHxRF1Ja1NZQWRhdWdnS015VVhwbE5sSWRxcVRjN2030E0yd1dVaWdBbWZ1eTV1YnVVUTFQNKRUKdKMTZoV114emdXRXA2SnhzTHpTSHRiZ18yZzlpLTJ6MF1maXV2SWhkZ0kiLCJwcm9vZ190eXB1IjoizXF1YWxpdH1fcHJvb2YifV19.uIYkpRJttXimLtWssxpnyTmPpg73QUUpDJnw72CJoeRyikJYJaa6Dw5pQ0xNE5yPUbnGZ80DFWG36yg7TCEg

B.1.2.4.1.2 Decoded and reconstructed JSON-Payload

All SD-Jwt related claims have been removed for readability.

```
{
  "updated_at": 1570000000,
  "cnf": {
    "jwk": {
      "x": "0q1azTI-nfXRjxeoxqwktbSayE7Sd-2S0kp2MdCXdJM",
      "crv": "P-256",
      "y": "Xz4eqZG6bq017tCtqsx97KiJzgLzbSQv0zQJBfasXKE",
      "kty": "EC"
    }
  },
  "sub": "user_42",
}
```

B.1.2.4.2 Diploma

B.1.2.4.2.1 Encoded VP-Token

eyJhbGciOiJFUzI1NiIsImtpZCI6IjEyMyJ9.eyJzdWIiOiJ1c2VyXzQyIiwiX3NkIjpBImhwLutzX3o4RjB4UV80TUFybEpMbjIzZ1VVdktaXZwT3c1Y3dWb3pIQ1kiLCJKQ0pUdXozTHN1eUd5WHBsYUhMRVprZVNMewzNV93T3NLR2xFaWJxb2x3IiwiTmkyVENVTVVwMTQ1cWFsYW0tVmVPQWd0YmdtdDQtTF9fc3ZBRU45SGN4QSIisIm5KLVBI1lmbmU5Sk9PeDBZNnNiQjNkUn1fR0pmVnprV3FXcWd6Z1VfQ0EiXSwiX3NkX2FsZ19wYXJhbSI6eyJjb21taXRtZW50X3NjaGVtZSI6eyJjcnyiOiJlZDI1NTE5IiwigHVibGljX3BhcmFtcyI6eyJoIjoIWmd2SVJPNHZpU1JVMGhCWfh2WEM2c21faWhsQmhoTVAxTW9VZFJIb3VIayIsImciOiIzUGVPSmwXZ1RHQTd4SC1jV0ZwaENURERTdDcwUDZORld0Y295aVVoQWxBIn19fSwiX3NkX2FsZyI6ImVjX3B1ZGVyc2VuIiwiY29tX2xpbmsiOnsiZ212Zw5fbmFtZSI6MCwiZmFtaWx5X25hbWUiOjEsInN1YmplY3QiOjIsImZpbmFsX2dyYWR1IjofX0.gDs0zxTqgJKMuBuJv1UvSC8R_tshugj3-HpwV062H2wrvnos1kDwMtUknCSnNPLeQdVloMp13sSZ0hZVbcJBZQ~WyIwWW9tWldPRmU1V0o0OGF1Z2FZMjR3M1R1UGHhaZ2Z5NGt5V2x6aENSNXdBIiwiZmluYWxfZ3JhZGUiLDQuOF0~WyJ2MX11akU5bDloYmxsR1lnY3hPaU1WS19jcUZfUGYyR0dHTUhpkxaZGc0Iiwig3ViamVjdCIsIkNvbXB1dGVyIFNjaWVuY2UiXQ~

B.1.2.4.2.2 Decoded and reconstructed JSON payload

All SD-Jwt related claims have been removed for readability.

```
{
  "final_grade": 4.8,
  "subject": "Computer Science",
  "sub": "user_42",
}
```

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.

Appendix D

D.1 Sample Code for JSON canonicalization

```
pub fn canonicalize_object(v: &heidi_util_rust::value::Value) -> String {
    let Some(obj) = v.as_object() else {
        return String::new();
    };
    let mut keys = obj.keys().collect::<Vec<_>>();
    keys.sort();
    let mut output_string = String::new();
    output_string.push_str("{");
    for key in keys {
        output_string.push_str(r#"##");
        output_string.push_str(key);
        output_string.push_str(r#"##");
        output_string.push_str(":");
        output_string.push_str(&stringify_value(obj.get(key).unwrap()));
        output_string.push_str(",");
    }
    if output_string.contains(",") {
        output_string = (&output_string[..output_string.len() - 1]).to_string();
    }
    output_string.push_str("}");
    output_string
}

fn canonicalize_array(v: &Value) -> String {
    let mut output_string = String::new();
    output_string.push_str("[");
    for item in v.as_array().unwrap() {
        output_string.push_str(&stringify_value(item));
        output_string.push_str(",");
    }
    if output_string.contains(",") {
        output_string = (&output_string[..output_string.len() - 1]).to_string();
    }
    output_string.push_str("]");
    output_string
}
fn canonicalize_primitive(v: &Value) -> String {
    let serde_json_value: serde_json::Value = v.into();
    serde_json::to_string(&serde_json_value)
```

```
.unwrap()
.trim()
.to_string()
}
pub fn stringify_value(value: &heidi_util_rust::value::Value) -> String {
    match value {
        Value::Array(_) => canonicalize_array(value),
        Value::Object(_) => canonicalize_object(value),
        _ => canonicalize_primitive(value),
    }
}
```