Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-selective-disclosure-jwt-05
Published: 30 June 2023
Intended Status: Standards Track
Expires: 1 January 2024
Authors: D. Fett K. Yasuda B. Campbell
yes.com Microsoft Ping Identity

Selective Disclosure for JWTs (SD-JWT)

Abstract

This specification defines a mechanism for selective disclosure of individual elements of a JSON object used as the payload of a JSON Web Signature (JWS) structure. It encompasses various applications, including but not limited to the selective disclosure of JSON Web Token (JWT) claims.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/oauth-wg/oauth-selective-disclosure-jwt.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 January 2024.

Copyright Notice

Table of Contents

1. Introduction

This document specifies conventions for creating JSON Web Signature (JWS) [RFC7515] structures with JSON [RFC8259] objects as the payload while supporting selective disclosure of individual elements of that JSON. Because JSON Web Token (JWT) [RFC7519] is a very prevalent application of JWS with a JSON payload, the selective disclosure of JWT claims receives primary treatment herein. However, that does not preclude the mechanism's applicability to other or more general applications of JWS with JSON payloads.

The JSON-based representation of claims in a signed JWT is secured against modification using JWS digital signatures. A consumer of a signed JWT that has checked the signature can safely assume that the contents of the token have not been modified. However, anyone receiving an unencrypted JWT can read all the claims. Likewise, anyone with the decryption key receiving encrypted JWT can also read all the claims.

One of the common use cases of a signed JWT is representing a user's identity. As long as the signed JWT is one-time use, it typically only contains those claims the user has consented to disclose to a specific Verifier. However, there is an increasing number of use cases where a signed JWT is created once and then used a number of times by the user (the "Holder" of the JWT). In such use cases, the signed JWT needs to contain the superset of all claims the user of the signed JWT might want to disclose to Verifiers at some point. The ability to selectively disclose a subset of these claims depending on the Verifier becomes crucial to ensure minimum disclosure and prevent Verifiers from obtaining claims irrelevant for the transaction at hand. SD-JWTs defined in this document enable such selective disclosure of JWT claims.

One example of a multi-use JWT is a verifiable credential, an Issuer-signed credential that contains the claims about a subject, and whose authenticity can be cryptographically verified.

Similar to the JWT specification on which it builds, this document is a product of the Web Authorization Protocol (oauth) working group. However, while both JWT and SD-JWT have potential OAuth 2.0 applications, their utility and application is certainly not constrained to OAuth 2.0. JWT was developed as a general-purpose token format and has seen widespread usage in a variety of applications. SD-JWT is a selective disclosure mechanism for JWT and is similarly intended to be general-purpose specification.

While JWTs with claims describing natural persons are a common use case, the mechanisms defined in this document can be used for other use cases as well.

In an SD-JWT, claims can be hidden, but cryptographically protected against undetected modification. "Claims" here refers to both object properties (key-value pairs) as well as array elements. When issuing the SD-JWT to the Holder, the Issuer includes the cleartext counterparts of all hidden claims, the so-called Disclosures, outside the signed part of the SD-JWT.

The Holder decides which claims to disclose to a particular Verifier and includes the respective Disclosures in the SD-JWT to that Verifier. The Verifier has to verify that all disclosed claim values

were part of the original Issuer-signed JWT. The Verifier will not, however, learn any claim values not disclosed in the Disclosures.

This document also specifies an optional mechanism for Key Binding, which is the concept of binding an SD-JWT to a Holder's public key and requiring that the Holder prove possession of the corresponding private key when presenting the SD-JWT. The strength of the binding is conditional upon the trust in the protection of the private key of the key pair an SD-JWT is bound to.

SD-JWT can be used with any JSON-based representation of claims, including JSON-LD.

This specification aims to be easy to implement and to leverage established and widely used data formats and cryptographic algorithms wherever possible.

1.1. Feature Summary

* This specification defines
  - a format for the payload of an Issuer-signed JWT containing selectively disclosable claims that include object properties (key-value pairs), array elements, and nested data structures built from these,
  - a format for data associated with the JWT that enables selectively disclosing those claims,
  - facilities for binding the JWT to a key and associated data to prove possession thereof, and
  - a format, extending the JWS Compact Serialization, for the combined transport of the JWT and associated data that is suitable for both issuance and presentation.
* An alternate format utilizing the JWS JSON Serialization is also specified.
* This specification enables combining selectively disclosable claims with clear-text claims that are always disclosed.
* For selectively disclosable claims that are object properties, both the key and value are always blinded.

1.2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

base64url denotes the URL-safe base64 encoding without padding defined in Section 2 of [RFC7515].

2. Terms and Definitions

Selective disclosure: Process of a Holder disclosing to a Verifier a subset of claims contained in a claim set issued by an Issuer.

Selectively Disclosable JWT (SD-JWT): A composite structure, consisting of an Issuer-signed JWT (JWS, [RFC7515]), Disclosures, and optionally a Key Binding JWT that supports selective disclosure as defined in this document. It can contain both regular claims and digests of selectively-disclosable claims.

Disclosure: A combination of a salt, a cleartext claim name (present when the claim is a key-value pair and absent when the claim is an array element), and a cleartext claim value, all of which are used to calculate a digest for the respective claim.

Key Binding: Ability of the Holder to prove legitimate possession of an SD-JWT by proving control over the same private key during the issuance and presentation. An SD-JWT with Key Binding contains a public key, or a reference to a public key, that matches to the private key controlled by the Holder.

Issuer: An entity that creates SD-JWTs.

Holder: An entity that received SD-JWTs from the Issuer and has control over them.

Verifier: An entity that requests, checks, and extracts the claims from an SD-JWT with its respective Disclosures.

3. Flow Diagram

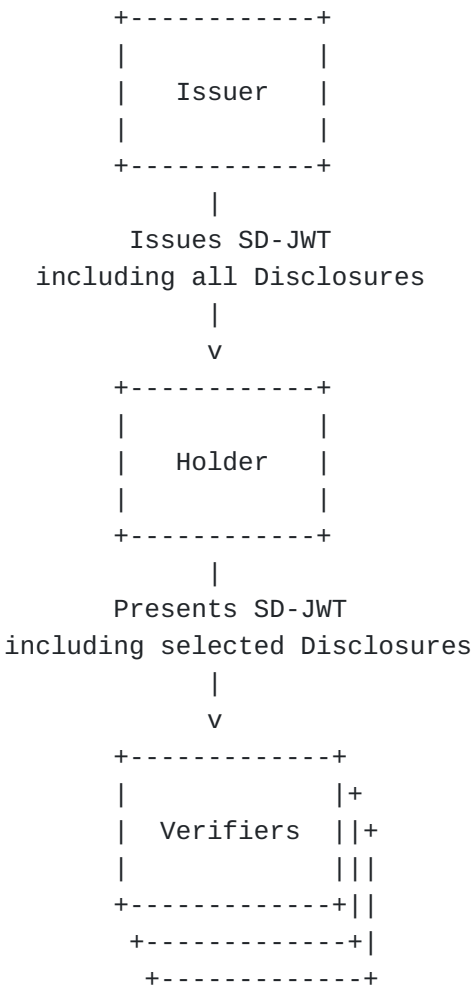

Figure 1: SD-JWT Issuance and Presentation Flow

4. Concepts

This section describes SD-JWTs with their respective Disclosures and Key Binding at a conceptual level, abstracting from the data formats described in Section 5.

4.1. SD-JWT and Disclosures

An SD-JWT, at its core, is a digitally signed JSON document containing digests over the selectively disclosable claims with the Disclosures outside the document. Disclosures can be omitted without breaking the signature, and modifying them can be detected. Selectively disclosable claims can be individual object properties (key-value pairs) or array elements.

Each digest value ensures the integrity of, and maps to, the respective Disclosure. Digest values are calculated using a hash function over the Disclosures, each of which contains a random salt, the claim name (only when the claim is an object property), and the

claim value. The Disclosures are sent to the Holder as part of the SD-JWT in the format defined in Section 5.11.

An SD-JWT MAY also contain clear-text claims that are always disclosed to the Verifier.

4.2. Disclosing to a Verifier

To disclose to a Verifier a subset of the SD-JWT claim values, a Holder sends only the Disclosures of those selectively released claims to the Verifier as part of the SD-JWT.

4.3. Optional Key Binding

Key Binding is an optional feature. When Key Binding is required by the use-case, the SD-JWT MUST contain information about the key material controlled by the Holder.

Note: How the public key is included in SD-JWT is out of scope of this document. It can be passed by value or by reference.

The Holder can then create a signed document, the Key Binding JWT as defined in Section 5.10, using its private key. This document contains some data provided by the Verifier such as a nonce to ensure the freshness of the signature, and audience to indicate the intended audience for the document.

The Key Binding JWT can be included as part of the SD-JWT and sent to the Verifier as described in Section 5.11.

Note that there may be other ways to send a Key Binding JWT to the Verifier or for the Holder to prove possession of the key material included in an SD-JWT. In these cases, inclusion of the Key Binding JWT in the SD-JWT is not required.

4.4. Verification

At a high level, the Verifier

* receives the SD-JWT from the Holder and verifies its signature using the Issuer's public key,

* verifies the Key Binding JWT, if Key Binding is required by the Verifier's policy, using the public key included in the SD-JWT,

* calculates the digests over the Holder-Selected Disclosures and verifies that each digest is contained in the SD-JWT.

The detailed algorithm is described in Section 6.3.

5. Data Formats

This section defines data formats for SD-JWT including the Issuer-signed JWT content, Disclosures, and Key Binding JWT.

5.1. SD-JWT Payload

An SD-JWT has a JWT component that MUST be signed using the Issuer's private key. It MUST use a JWS asymmetric digital signature algorithm. It MUST NOT use none or an identifier for a symmetric algorithm (MAC).

The payload of an SD-JWT is a JSON object according to the following rules:

1. The payload MAY contain the _sd_alg key described in Section 5.8.
2. The payload MAY contain one or more digests of Disclosures to enable selective disclosure of the respective claims, created and formatted as described below.
3. The payload MAY contain one or more decoy digests to obscure the actual number of claims in the SD-JWT, created and formatted as described in Section 5.6.
4. The payload MAY contain one or more non-selectively disclosable claims.
5. The payload MAY also contain a Holder's public key or a reference thereto, as well as further claims such as iss, iat, etc. as defined or required by the application using SD-JWTs.
6. The payload MUST NOT contain the reserved claims _sd or ... except for the purpose of transporting digests as described below.
7. The same digest value MUST NOT appear more than once in the SD-JWT.

Applications of SD-JWT SHOULD be explicitly typed using the typ header parameter. See Section 9.10 for more details.

It is the Issuer who decides which claims are selectively disclosable and which are not. However, claims controlling the validity of the SD-JWT, such as iss, exp, or nbf are usually included in plaintext. End-User claims MAY be included as plaintext as well, e.g., if hiding the particular claims from the Verifier is not required in the intended use case.

Claims that are not selectively disclosable are included in the SD-JWT in plaintext just as they would be in any other JSON structure.

5.2. Creating Disclosures

Disclosures are created differently depending on whether a claim is an object property (key-value pair) or an array element.

- For a claim that is an object property, the Issuer creates a Disclosure as described in Section 5.2.1.

- For a claim that is an array element, the Issuer creates a Disclosure as described in Section 5.2.2.

5.2.1. Disclosures for Object Properties

For each claim that is an object property and that is to be made selectively disclosable, the Issuer MUST create a Disclosure as follows:

- Create an array of three elements in this order:

  1. A salt value. MUST be a string. See Section 9.3 and Section 9.4 for security considerations. It is RECOMMENDED to base64url-encode minimum 128 bits of cryptographically secure pseudorandom data, producing a string. The salt value MUST be unique for each claim that is to be selectively disclosed. The Issuer MUST NOT disclose the salt value to any party other than the Holder.

  2. The claim name, or key, as it would be used in a regular JWT body. The value MUST be a string.

  3. The claim value, as it would be used in a regular JWT body. The value MAY be of any type that is allowed in JSON, including numbers, strings, booleans, arrays, and objects.

- JSON-encode the array, producing an UTF-8 string.

- base64url-encode the byte representation of the UTF-8 string, producing a US-ASCII [RFC0020] string. This string is the Disclosure.

The order is decided based on the readability considerations: salts would have a constant length within the SD-JWT, claim names would be around the same length all the time, and claim values would vary in size, potentially being large objects.

The following example illustrates the steps described above.

The array is created as follows:

```
["_26bc4LT-ac6q2KI6cBW5es", "family_name", "Möbius"]
```

The resulting Disclosure would be:
WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsICJmYW1pbHlfbmFtZSIsICJNw7ZiaXVz
Il0

Note that variations in whitespace, encoding of Unicode characters, ordering of object properties, etc., are allowed in the JSON representation and no canonicalization needs be performed before base64url-encoding. For example, the following strings are all valid and encode the same claim value "Möbius":

- A different way to encode the unicode umlaut:
  WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsICJmYW1pbHlfbmFtZSIsICJNX
  HUwMGY2Yml1cyJd

- No white space:
  WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsImZhbWlseV9uYW1lIiwiTcO2Y
  ml1cyJd

- Newline characters between elements:
  WwoiXzI2YmM0TFQtYWM2cTJLSTZjQlc1ZXMiLAoiZmFtaWx5X25hbWUiLAoiT
  cO2Yml1cyIKXQ

See Appendix B for some further considerations on the Disclosure format approach.

5.2.2. Disclosures for Array Elements

For each claim that is an array element and that is to be made selectively disclosable, the Issuer MUST create a Disclosure as follows:

- The array MUST contain two elements in this order:

  1. The salt value as described in Section 5.2.1.

  2. The array element that is to be hidden. This value MAY be of any type that is allowed in JSON, including numbers, strings, booleans, arrays, and objects.

The Disclosure string is created by JSON-encoding this array and base64url-encoding the byte representation of the resulting string as described in Section 5.2.1. The same considerations regarding variations in the result of the JSON encoding apply.

For example, a Disclosure for the second element of the nationalities array in the following claim set:

```
{
  "nationalities": ["DE", "FR"]
}
```

could be created by first creating the following array:

```
["lklxF5jMYlGTPUovMNIvCA", "FR"]
```

The resulting Disclosure would be:
WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIkZSIl0

5.3. Hashing Disclosures

For embedding the Disclosures in the SD-JWT, the Disclosures are hashed using the hash algorithm specified in the _sd_alg claim described in Section 5.8. The resulting digest is then included in the SD-JWT payload instead of the original claim value, as described next.

The digest MUST be taken over the US-ASCII bytes of the base64url-encoded Disclosure. This follows the convention in JWS [RFC7515] and JWE [RFC7516]. The bytes of the digest MUST then be base64url-encoded.

It is important to note that:

- The input to the hash function MUST be the base64url-encoded Disclosure, not the bytes encoded by the base64url string.
- The bytes of the output of the hash function MUST be base64url-encoded, and are not the bytes making up the (often used) hex representation of the bytes of the digest.

For example, the SHA-256 digest of the Disclosure WyI2cU1RdlJMNWhhaiIsICJmYW1pbHlfbmFtZSIsICJNw7ZiaXVzIl0 would be uutlBuYeMDyjLLTpf6Jxi7yNkEF35jdyWMn9U7b_RYY.

The SHA-256 digest of the Disclosure WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIkZSIl0 would be w0I8EKcdCtUPkGCNUrfwVp2xEgNjtoIDlOxc9-PlOhs.

5.4. Embedding Disclosure Digests in SD-JWTs

For selectively disclosable claims, the digests of the Disclosures are embedded into the SD-JWT instead of the claims themselves. The precise way of embedding depends on whether a claim is an object property (key-value pair) or an array element.

- For a claim that is an object property, the Issuer embeds a Disclosure as described in Section 5.4.1.

*For a claim that is an array element, the Issuer creates a Disclosure as described in Section 5.4.2.

5.4.1. Object Properties

Digests of Disclosures for object properties are added to an array under the new key _sd in the object. The _sd key MUST refer to an array of strings, each string being a digest of a Disclosure or a decoy digest as described in Section 5.6.

The array MAY be empty in case the Issuer decided not to selectively disclose any of the claims at that level. However, it is RECOMMENDED to omit the _sd key in this case to save space.

The Issuer MUST hide the original order of the claims in the array. To ensure this, it is RECOMMENDED to shuffle the array of hashes, e.g., by sorting it alphanumerically or randomly, after potentially adding decoy digests as described in Section 5.6. The precise method does not matter as long as it does not depend on the original order of elements.

For example, using the digest of the object property Disclosure created above, the Issuer could create the following SD-JWT payload to make family_name selectively disclosable:

```
{
  "given_name": "Alice",
  "_sd": ["uutlBuYeMDyjLLTpf6Jxi7yNkEF35jdyWMn9U7b_RYY"]
}
```

5.4.2. Array Elements

Digests of Disclosures for array elements are added to the array in the same position as the original claim value in the array. For each digest, an object of the form {"...": "<digest>"} is added to the array. The key MUST always be the string ... (three dots). The value MUST be the digest of the Disclosure created as described in Section 5.3. There MUST NOT be any other keys in the object.

For example, using the digest of the array element Disclosure created above, the Issuer could create the following SD-JWT payload to make the second element of the nationalities array selectively disclosable:

```
{
  "nationalities":
    ["DE", {"...": "w0I8EKcdCtUPkGCNUrfwVp2xEgNjtoIDlOxc9-PlOhs"}]
}
```

As described in Section 6.3, Verifiers ignore all selectively disclosable array elements for which they did not receive a Disclosure. In the example above, the verification process would output an array with only one element unless a matching Disclosure for the second element is received.

5.5. Example 1: SD-JWT

In this example, a simple SD-JWT is demonstrated.

The Issuer is using the following input claim set:

```
{
  "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": "1940-01-01",
  "updated_at": 1570000000,
  "nationalities": [
    "US",
    "DE"
  ]
}
```

The Issuer in this case made the following decisions:

- The nationalities array is always visible, but its contents are selectively disclosable.
- The sub element and essential verification data (iss, iat, cnf, etc.) are always visible.
- All other End-User claims are selectively disclosable.
- For address, the Issuer is using a flat structure, i.e., all of the claims in the address claim can only be disclosed in full. Other options are discussed in Section 5.7.

The following payload is used for the SD-JWT:

```
{
  "_sd": [
    "CrQe7S5kqBAHt-nMYXgc6bdt2SH5aTY1sU_M-PgkjPI",
    "JzYjH4svliH0R3PyEMfeZu6Jt69u5qehZo7F7EPYlSE",
    "PorFbpKuVu6xymJagvkFsFXAbRoc2JGlAUA2BA4o7cI",
    "TGf4oLbgwd5JQaHyKVQZU9UdGE0w5rtDsrZzfUaomLo",
    "XQ_3kPKt1XyX7KANkqVR6yZ2Va5NrPIvPYbyMvRKBMM",
    "XzFrzwscM6Gn6CJDc6vVK8BkMnfG8vOSKfpPIZdAfdE",
    "gbOsI4Edq2x2Kw-w5wPEzakob9hV1cRD0ATN3oQL9JM",
    "jsu9yVulwQQlhFlM_3JlzMaSFzglhQG0DpfayQwLUK4"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "user_42",
  "nationalities": [
    {
      "...": "pFndjkZ_VCzmyTa6UjlZo3dh-ko8aIKQc9DlGzhaVYo"
    },
    {
      "...": "7Cf6JkPudry3lcbwHgeZ8khAv1U1OSlerP0VkBJrWZ0"
    }
  ],
  "_sd_alg": "sha-256",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```


The following Disclosures are created by the Issuer:

Claim given_name:

*SHA-256 Hash: jsu9yVulwQQlhFlM_3JlzMaSFzglhQG0DpfayQwLUK4

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9o
biJd

*Contents: ["2GLC42sKQveCfGfryNRN9w", "given_name", "John"]

Claim family_name:

*SHA-256 Hash: TGf4oLbgwd5JQaHyKVQZU9UdGE0w5rtDsrZzfUaomLo

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRv
ZSJd

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "family_name", "Doe"]

Claim email:

*SHA-256 Hash: JzYjH4svliH0R3PyEMfeZu6Jt69u5qehZo7F7EPYlSE

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImVtYWlsIiwgImpvaG5kb2VA
ZXhhbXBsZS5jb20iXQ

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "email",
"johndoe@example.com"]

Claim phone_number:

*SHA-256 Hash: PorFbpKuVu6xymJagvkFsFXAbRoc2JGlAUA2BA4o7cI

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgInBob25lX251bWJlciIsICIr
MS0yMDItNTU1LTAxMDEiXQ

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "phone_number",
"+1-202-555-0101"]

Claim phone_number_verified:

*SHA-256 Hash: XQ_3kPKt1XyX7KANkqVR6yZ2Va5NrPIvPYbyMvRKBMM

*Disclosure:
WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bWJlcl92ZXJp
ZmllZCIsIHRydWVd

*Contents: ["Qg_O64zqAxe412a108iroA", "phone_number_verified",
true]

Claim address:

*SHA-256 Hash: XzFrzwscM6Gn6CJDc6vVK8BkMnfG8vOSKfpPIZdAfdE

*Disclosure:
WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVl
dF9hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRv
d24iLCAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0

*Contents: ["AJx-095VPrpTtN4QMOqROA", "address",
{"street_address":

"123 Main St", "locality": "Anytown", "region": "Anystate",
"country": "US"}]

Claim birthdate:

*SHA-256 Hash: gbOsI4Edq2x2Kw-w5wPEzakob9hV1cRD0ATN3oQL9JM

*Disclosure:
WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImJpcnRoZGF0ZSIsICIxOTQw
LTAxLTAxIl0

*Contents: ["Pc33JM2LchcU_lHggv_ufQ", "birthdate", "1940-01-01"]

Claim updated_at:

*SHA-256 Hash: CrQe7S5kqBAHt-nMYXgc6bdt2SH5aTY1sU_M-PgkjPI

*Disclosure:
WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInVwZGF0ZWRfYXQiLCAxNTcw
MDAwMDAwXQ

*Contents: ["G02NSrQfjFXQ7Io09syajA", "updated_at", 1570000000]

Array Entry:

*SHA-256 Hash: pFndjkZ_VCzmyTa6UjlZo3dh-ko8aIKQc9DlGzhaVYo

*Disclosure:
WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIlVTIl0

*Contents: ["lklxF5jMYlGTPUovMNIvCA", "US"]

Array Entry:

*SHA-256 Hash: 7Cf6JkPudry3lcbwHgeZ8khAv1U1OSlerP0VkBJrWZ0

*Disclosure:
WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgIkRFIl0

*Contents: ["nPuoQnkRFq3BIeAm7AnXFA", "DE"]

The payload is then signed by the Issuer to create a JWT like the following:

eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkd
DJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZakg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5d
TVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBLdVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHb
EFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc3Jae
mZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tBTmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS
0JNTSIsICJYekZyendzY002R242Q0pEYzZ2Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFI
iwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFrb2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAia
nN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpnbGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzI
jogImh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlhdCI6IDE2ODMwMDAwMDAsI
CJleHAiOiAxODgzMDAwMDAwLCAic3ViIjogInVzZXJfNDIiLCAibmF0aW9uYWxpdGllc
yI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15VGE2VWpsWm8zZGgta284YUlLUWM5RGxHe
mhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlc
lAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siO
iB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IR
jRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIV
ldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlGMkhaUSJ9fX0.kmx687kUBiIDvKWgo2Dub
-TpdCCRLZwtD7TOj4RoLsUbtFBI8sMrtH2BejXtm_P6fOAjKAVc_7LRNJFgm3PJhg

5.6. Decoy Digests

An Issuer MAY add additional digests to the SD-JWT payload that are not associated with any claim. The purpose of such "decoy" digests is to make it more difficult for an attacker to see the original number of claims contained in the SD-JWT. Decoy digests MAY be added both to the _sd array for objects as well as in arrays.

It is RECOMMENDED to create the decoy digests by hashing over a cryptographically secure random number. The bytes of the digest MUST then be base64url-encoded as above. The same digest function as for the Disclosures MUST be used.

For decoy digests, no Disclosure is sent to the Holder, i.e., the Holder will see digests that do not correspond to any Disclosure. See Section 10.3 for additional privacy considerations.

To ensure readability and replicability, the examples in this specification do not contain decoy digests unless explicitly stated. For an example with decoy digests, see Appendix A.1.

5.7. Nested Data in SD-JWTs

Being JSON, an object in an SD-JWT payload MAY contain key-value pairs where the value is another object or objects MAY be elements in arrays. In SD-JWT, the Issuer decides for each claim individually, on each level of the JSON, whether the claim should be selectively disclosable or not. This choice can be made on each level independent from whether keys higher in the hierarchy are selectively disclosable.

From this it follows that the _sd key containing digests MAY appear multiple times in an SD-JWT, and likewise, there MAY be multiple arrays within the hierarchy with each having selectively disclosable elements. Digests of selectively disclosable claims MAY even appear within other Disclosures.

The following examples illustrate some of the options an Issuer has. It is up to the Issuer to decide which option to use, depending on, for example, the expected use cases for the SD-JWT, requirements for privacy, size considerations, or ecosystem requirements. For more examples with nested structures, see Appendix A.1 and Appendix A.2.

The following input claim set is used as an example throughout this section:

```
{
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "address": {
    "street_address": "Schulstr. 12",
    "locality": "Schulpforta",
    "region": "Sachsen-Anhalt",
    "country": "DE"
  }
}
```

Important: Throughout the examples in this document, line breaks had to be added to JSON strings and base64-encoded strings (as shown in the next example) to adhere to the 72 character limit for lines in RFCs and for readability. JSON does not allow line breaks in strings.

5.7.1. Option 1: Flat SD-JWT

The Issuer can decide to treat the address claim as a block that can either be disclosed completely or not at all. The following example shows that in this case, the entire address claim is treated as an object in the Disclosure.

```
{
  "_sd": [
    "fOBUSQvo46yQO-wRwXBcGqvnbKIueISEL961_Sjd4do"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "_sd_alg": "sha-256"
}
```

The Issuer would create the following Disclosure:

Claim address:

*SHA-256 Hash: fOBUSQvo46yQO-wRwXBcGqvnbKIueISEL961_Sjd4do

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImFkZHJlc3MiLCB7InN0cmVl
dF9hZGRyZXNzIjogIlNjaHVsc3RyLiAxMiIsICJsb2NhbGl0eSI6ICJTY2h1
bHBmb3J0YSIsICJyZWdpb24iOiAiU2FjaHNlbi1BbmhhbHQiLCAiY291bnRy
eSI6ICJERSJ9XQ

*Contents: ["2GLC42sKQveCfGfryNRN9w", "address",
{"street_address":
"Schulstr. 12", "locality": "Schulpforta", "region":
"Sachsen-Anhalt", "country": "DE"}]

5.7.2. Option 2: Structured SD-JWT

The Issuer may instead decide to make the address claim contents selectively disclosable individually:

```
{
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "address": {
    "_sd": [
      "6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0",
      "9gjVuXtdFROCgRrtNcGUXmF65rdezi_6Er_j76kmYyM",
      "KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88",
      "WN9r9dCBJ8HTCsS2jKASxTjEyW5m5x65_Z_2ro2jfXM"
    ]
  },
  "_sd_alg": "sha-256"
}
```

In this case, the Issuer would use the following data in the Disclosures for the address sub-claims:

Claim street_address:

*SHA-256 Hash: 9gjVuXtdFROCgRrtNcGUXmF65rdezi_6Er_j76kmYyM

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN0cmVldF9hZGRyZXNzIiwg
IlNjaHVsc3RyLiAxMiJd

*Contents: ["2GLC42sKQveCfGfryNRN9w", "street_address", "Schulstr.
12"]

Claim locality:

*SHA-256 Hash: 6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImxvY2FsaXR5IiwgIlNjaHVs
cGZvcnRhIl0

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "locality", "Schulpforta"]

Claim region:

*SHA-256 Hash: KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInJlZ2lvbiIsICJTYWNoc2Vu
LUFuaGFsdCJd

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "region", "Sachsen-Anhalt"]

Claim country:

*SHA-256 Hash: WN9r9dCBJ8HTCsS2jKASxTjEyW5m5x65_Z_2ro2jfXM

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImNvdW50cnkiLCAiREUiXQ

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "country", "DE"]

The Issuer may also make one sub-claim of address non-selectively disclosable and hide only the other sub-claims:

```
{
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "address": {
    "_sd": [
      "6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0",
      "9gjVuXtdFROCgRrtNcGUXmF65rdezi_6Er_j76kmYyM",
      "KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88"
    ],
    "country": "DE"
  },
  "_sd_alg": "sha-256"
}
```

There would be no Disclosure for country in this case.

5.7.3. Option 3: SD-JWT with Recursive Disclosures

The Issuer may also decide to make the address claim contents selectively disclosable recursively, i.e., the address claim is made selectively disclosable as well as its sub-claims:

```
{
  "_sd": [
    "HvrKX6fPV0v9K_yCVFBiLFHsMaxcD_114Em6VT8x1lg"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "_sd_alg": "sha-256"
}
```

The Issuer creates Disclosures first for the sub-claims and then includes their digests in the Disclosure for the address claim:

Claim street_address:

*SHA-256 Hash: 9gjVuXtdFROCgRrtNcGUXmF65rdezi_6Er_j76kmYyM

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN0cmVldF9hZGRyZXNzIiwg
IlNjaHVsc3RyLiAxMiJd

*Contents: ["2GLC42sKQveCfGfryNRN9w", "street_address", "Schulstr.
12"]

Claim locality:

*SHA-256 Hash: 6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImxvY2FsaXR5IiwgIlNjaHVs
cGZvcnRhIl0

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "locality", "Schulpforta"]

Claim region:

*SHA-256 Hash: KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInJlZ2lvbiIsICJTYWNoc2Vu
LUFuaGFsdCJd

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "region", "Sachsen-Anhalt"]

Claim country:

*SHA-256 Hash: WN9r9dCBJ8HTCsS2jKASxTjEyW5m5x65_Z_2ro2jfXM

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImNvdW50cnkiLCAiREUiXQ

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "country", "DE"]

Claim address:

*SHA-256 Hash: HvrKX6fPV0v9K_yCVFBiLFHsMaxcD_114Em6VT8x1lg

*Disclosure:
WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImFkZHJlc3MiLCB7Il9zZCI6
IFsiNnZoOWJxLXpTNEdLTV83R3BnZ1ZiWXp6dTZvT0dYcm1OVkdQSFA3NVVk
MCIsICI5Z2pWdVh0ZEZST0NnUnJ0TmNHVVhtRjY1cmRlemlfNkVyX2o3Nmtt
WXlNIiwgIktVUkRQaDRaQzE5LTN0aXotRGYzOVY4ZWlkeTFvVjNhM0gxRGEy
TjBnODgiLCAiV045cjlkQ0JKOEhUQ3NTMmpLQVN4VGpFeVc1bTV4NjVfWl8y
cm8yamZYTSJdfV0

*Contents: ["Qg_O64zqAxe412a108iroA", "address", {"_sd":
["6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0",
"9gjVuXtdFROCgRrtNcGUXmF65rdezi_6Er_j76kmYyM",
"KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88",
"WN9r9dCBJ8HTCsS2jKASxTjEyW5m5x65_Z_2ro2jfXM"]}]

5.8. Hash Function Claim

The claim _sd_alg indicates the hash algorithm used by the Issuer to generate the digests as described in Section 5.2. When used, this claim MUST appear at the top level of the SD-JWT payload. It MUST NOT be used in any object nested within the payload. If the _sd_alg claim is not present at the top level, a default value of sha-256 MUST be used.

The hash algorithm identifier MUST be a hash algorithm value from the "Hash Name String" column in the IANA "Named Information Hash Algorithm" registry [IANA.Hash.Algorithms] or a value defined in another specification and/or profile of this specification.

To promote interoperability, implementations MUST support the sha-256 hash algorithm.

See Section 9 for requirements regarding entropy of the salt, minimum length of the salt, and choice of a hash algorithm.

5.9. Holder Public Key Claim

If the Issuer wants to enable Key Binding, it includes a public key associated with the Holder, or a reference thereto.

It is out of the scope of this document to describe how the Holder key pair is established. For example, the Holder MAY provide a key pair to the Issuer, the Issuer MAY create the key pair for the Holder, or Holder and Issuer MAY use pre-established key material.

Note: The examples in this document use the cnf claim defined in [RFC7800] to include the raw public key by value in SD-JWT.

5.10. Key Binding JWT

This section defines the contents of the Key Binding JWT, which the Holder MAY include in the SD-JWT to prove the Key Binding to the Verifier.

The JWT MUST contain the following elements:

*in the JOSE header,

-typ: REQUIRED. MUST be kb+jwt, which explicitly types the Key Binding JWT as recommended in Section 3.11 of [RFC8725].

-alg: REQUIRED. A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry. MUST NOT be none or an identifier for a symmetric algorithm (MAC).

*in the JWT body,

-iat: REQUIRED. The value of this claim MUST be the time at which the Key Binding JWT was issued using the syntax defined in [RFC7519].

-aud: REQUIRED. The intended receiver of the Key Binding JWT. How the value is represented is up to the protocol used and out of scope of this specification.

-nonce: REQUIRED. Ensures the freshness of the signature. The value type of this claim MUST be a string. How this value is obtained is up to the protocol used and out of scope of this specification.

To validate the signature on the Key Binding JWT, the Verifier MUST use the key material in the SD-JWT. If it is not clear from the SD-JWT, the Key Binding JWT MUST specify which key material the

Verifier needs to use to validate the Key Binding JWT signature using JOSE header parameters such as kid and x5c.

Below is a non-normative example of a Key Binding JWT header:

```
{
  "alg": "ES256",
  "typ": "kb+jwt"
}
```

Below is a non-normative example of a Key Binding JWT payload:

```
{
  "nonce": "1234567890",
  "aud": "https://example.com/verifier",
  "iat": 1688160483
}
```

Below is a non-normative example of a Key Binding JWT produced by signing a payload in the example above:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZSI6ICIxMjM0NTY
3ODkwIiwgImF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29tL3ZlcmlmaWVyIiwgImlhdCI
6IDE2ODgxNjA0ODN9.tKnLymr8fQfupOgvMgBK3GCEIDEzhgta4MgnxYm9fWGMkqrz2R
5PSkv0I-AXKXtIF6bdZRbjL-t43vC87jVoZQ
```

Whether to require Key Binding is up to the Verifier's policy, based on the set of trust requirements such as trust frameworks it belongs to.

Other ways of proving Key Binding MAY be used when supported by the Verifier, e.g., when the presented SD-JWT without a Key Binding JWT is itself embedded in a signed JWT. See Section 7 for details.

5.11. SD-JWT Structure

An SD-JWT is composed of the following:

- *the Issuer-signed JWT
- *The Disclosures
- *optionally a Key Binding JWT

The serialized format for the SD-JWT is the concatenation of each part delineated with a single tilde ('~') character as follows:

```
<JWT>~<Disclosure 1>~<Disclosure 2>~...~<Disclosure N>~<optional KB-JWT>
```

The order of the tilde separated values MUST be the Issuer-signed JWT, followed by any Disclosures, and lastly the optional Key Binding JWT. In the case that there is no Key Binding JWT, the last element MUST be an empty string and the last separating tilde character MUST NOT be omitted.

The Disclosures are linked to the SD-JWT payload through the digest values included therein.

When issued to a Holder, the Issuer includes all the relevant Disclosures in the SD-JWT.

For presentation to a Verifier, the Holder sends the SD-JWT including only its selected set of the Disclosures to the Verifier.

The Holder MAY send any subset of the Disclosures to the Verifier, i.e., none, multiple, or all Disclosures. For data that the Holder does not want to reveal to the Verifier, the Holder MUST NOT send Disclosures or reveal the salt values in any other way.

A Holder MUST NOT send a Disclosure that was not included in the SD-JWT or send a Disclosure more than once.

For Example 1, a non-normative example of an issued SD-JWT might look as follows (with Line breaks for formatting only):

```
eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkd
DJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZakg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5d
TVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBLdVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHb
EFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc3Jae
mZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tBTmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS
0JNTSIsICJYekZyendzY002R242Q0pEYzZ2Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFI
iwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFrb2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAia
nN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpnbGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzI
jogImh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlhdCI6IDE2ODMwMDAwMDAsI
CJleHAiOiAxODgzMDAwMDAwLCAic3ViIjogInVzZXJfNDIiLCAibmF0aW9uYWxpdGllc
yI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15VGE2VWpsWm8zZGgta284YUlLUWM5RGxHe
mhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlc
lAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siO
iB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IR
jRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIV
ldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlGMkhaUSJ9fX0.kmx687kUBiIDvKWgo2Dub
-TpdCCRLZwtD7TOj4RoLsUbtFBI8sMrtH2BejXtm_P6fOAjKAVc_7LRNJFgm3PJhg~Wy
IyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9obiJd~WyJlb
HVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRvZSJd~WyI2SWo3
dE0tYTVpVlBHYm9TNXRtdlZBIiwgImVtYWlsIiwgImpvaG5kb2VAZXhhbXBsZS5jb20i
XQ~WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgInBob25lX251bWJlciIsICIrMS0yM
DItNTU1LTAxMDEiXQ~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bW
Jlcl92ZXJpZmllZCIsIHRydWVd~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZ
HJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5I
jogIkFueXRvd24iLCAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMif
V0~WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImJpcnRoZGF0ZSIsICIxOTQwLTAxL
TAxIl0~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInVwZGF0ZWRfYXQiLCAxNTcwM
DAwMDAwXQ~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIlVTIl0~WyJuUHVvUW5rUk
ZxM0JJZUFtN0FuWEZBIiwgIkRFIl0~
```


The following non-normative example shows an associated SD-JWT Presentation as it would be sent from the Holder to the Verifier. The claims given_name, family_name, and address are disclosed and the Key Binding JWT is included as the last element.

eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkd
DJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZakg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5d
TVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBLdVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHb
EFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc3Jae
mZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tBTmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS
0JNTSIsICJYekZyendzY002R242Q0pEYzZ2Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFI
iwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFrb2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAia
nN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpnbGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzI
jogImh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlhdCI6IDE2ODMwMDAwMDAsI
CJleHAiOiAxODgzMDAwMDAwLCAic3ViIjogInVzZXJfNDIiLCAibmF0aW9uYWxpdGllc
yI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15VGE2VWpsWm8zZGgta284YUlLUWM5RGxHe
mhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlc
lAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siO
iB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IR
jRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIV
ldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlGMkhaUSJ9fX0.kmx687kUBiIDvKWgo2Dub
-TpdCCRLZwtD7TOj4RoLsUbtFBI8sMrtH2BejXtm_P6fOAjKAVc_7LRNJFgm3PJhg~Wy
JlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRvZSJd~WyJBS
ngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzI
jogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iLCAicmVnaW9uIjogI
kFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5ST
jl3IiwgImdpdmVuX25hbWUiLCAiSm9obiJd~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNB
IiwgIlVTIl0~eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZSI
6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29tL3ZlcmlmaWV
yIiwgImlhdCI6IDE2ODgxNjA0ODN9.tKnLymr8fQfupOgvMgBK3GCEIDEzhgta4MgnxY
m9fWGMkqrz2R5PSkv0I-AXKXtIF6bdZRbjL-t43vC87jVoZQ

6. Verification and Processing

6.1. Verification of the SD-JWT

Upon receiving an SD-JWT, a Holder or a Verifier MUST ensure that

- the Issuer-signed JWT is valid, i.e., it is signed by the Issuer and the signature is valid, and
- all Disclosures are correct, i.e., their digests are referenced in the Issuer-signed JWT.

The Holder or the Verifier MUST perform the following (or equivalent) steps when receiving an SD-JWT:

1. Separate the SD-JWT into the Issuer-signed JWT, the Disclosures (if any), and the Key Binding JWT (if present).
2. Validate the Issuer-signed JWT:
    1. Ensure that a signing algorithm was used that was deemed secure for the application. Refer to [RFC8725], Sections

3.1 and 3.2 for details. The none algorithm MUST NOT be accepted.

2. Validate the signature over the Issuer-signed JWT.

3. Validate the Issuer and that the signing key belongs to this Issuer.

4. Check that the Issuer-signed JWT is valid using nbf, iat, and exp claims, if provided in the SD-JWT, and not selectively disclosed.

5. Check that the _sd_alg claim value is understood and the hash algorithm is deemed secure.

3. Process the Disclosures and embedded digests in the issuser-signed JWT as follows:

    1. For each Disclosure provided:

        1. Calculate the digest over the base64url-encoded string as described in Section 5.3.

    2. (*) Identify all embedded digests in the Issuer-signed JWT as follows:

        1. Find all objects having an _sd key that refers to an array of strings.

        2. Find all array elements that are objects with one key, that key being ... and referring to a string.

    3. (**) For each embedded digest found in the previous step:

        1. Compare the value with the digests calculated previously and find the matching Disclosure. If no such Disclosure can be found, the digest MUST be ignored.

        2. If the digest was found in an object's _sd key:

            1. If the respective Disclosure is not a JSON-encoded array of three elements, the SD-JWT MUST be rejected.

            2. Insert, at the level of the _sd key, a new claim using the claim name and claim value from the Disclosure.

3. If the claim name already exists at the same level, the SD-JWT MUST be rejected.

4. Recursively process the value using the steps described in (*) and (**).

3. If the digest was found in an array element:

1. If the respective Disclosure is not a JSON-encoded array of two elements, the SD-JWT MUST be rejected.

2. Replace the array element with the claim value from the Disclosure.

3. Recursively process the value using the steps described in (*) and (**).

4. If any digests were found more than once in the previous step, the SD-JWT MUST be rejected.

5. Remove all array elements for which the digest was not found in the previous step.

6. Remove all _sd keys and their contents from the Issuer-signed JWT payload.

7. Remove the claim _sd_alg from the SD-JWT payload.

If any step fails, the SD-JWT is not valid and processing MUST be aborted.

It is up to the Holder how to maintain the mapping between the Disclosures and the plaintext claim values to be able to display them to the End-User when needed.

6.2. Processing by the Holder

If a Key Binding JWT is received by a Holder, the SD-JWT SHOULD be rejected.

For presentation to a Verifier, the Holder MUST perform the following (or equivalent) steps:

1. Decide which Disclosures to release to the Verifier, obtaining proper End-User consent if necessary.

2. If Key Binding is required, create a Key Binding JWT.

3. Assemble the SD-JWT for Presentation, including the Issuer-signed JWT, the selected Disclosures and, if applicable, the Key Binding JWT.

4. Send the Presentation to the Verifier.

6.3. Verification by the Verifier

Upon receiving a Presentation, in addition to the checks outlined in Section 6.1, Verifiers MUST ensure that

* if Key Binding is required, the Key Binding JWT is signed by the Holder and valid.

To this end, Verifiers MUST follow the following steps (or equivalent):

1. Determine if Key Binding is to be checked according to the Verifier's policy for the use case at hand. This decision MUST NOT be based on whether a Key Binding JWT is provided by the Holder or not. Refer to Section 9.6 for details.

2. Process the SD-JWT as defined in Section 6.1.

3. If Key Binding is required:

    1. If Key Binding is provided by means not defined in this specification, verify the Key Binding according to the method used.

    2. Otherwise, verify the Key Binding JWT as follows:

        1. If a Key Binding JWT is not provided, the Verifier MUST reject the Presentation.

        2. Determine the public key for the Holder from the SD-JWT.

        3. Ensure that a signing algorithm was used that was deemed secure for the application. Refer to [RFC8725], Sections 3.1 and 3.2 for details. The none algorithm MUST NOT be accepted.

        4. Validate the signature over the Key Binding JWT.

        5. Check that the typ of the Key Binding JWT is kb+jwt.

        6. Check that the creation time of the Key Binding JWT, as determined by the iat claim, is within an acceptable window.

7. Determine that the Key Binding JWT is bound to the current transaction and was created for this Verifier (replay protection) by validating nonce and aud claims.

8. Check that the Key Binding JWT is valid in all other respects, per [RFC7519] and [RFC8725].

If any step fails, the Presentation is not valid and processing MUST be aborted.

Otherwise, the processed SD-JWT payload can be passed to the application to be used for the intended purpose.

7. Enveloping SD-JWTs

In some applications or transport protocols, it is desirable to put an SD-JWT into an outer JWT container. For example, an implementation may envelope multiple credentials and presentations, independent of their format, in a JWT to enable application-layer encryption during transport.

For such use cases, the SD-JWT SHOULD be transported as a single string. Key Binding MAY be achieved by signing the envelope JWT instead of including a separate Key Binding JWT in the SD-JWT.

The following non-normative example shows an SD-JWT Presentation enveloped in a JWT payload:

```
{
  "aud": "https://verifier.example.com",
  "iat": 1580000000,
  "nonce": "iRnRdKuu1AtLM4ltc16by2XF0accSeutUescRw6BWC14",
  "_sd_jwt": "eyJhbGci...emhlaUJhZzBZ~eyJhb...dYALCGg~"
}
```

Here, the SD-JWT is shown as the value of an _sd_jwt claim where eyJhbGci...emhlaUJhZzBZ represents the Issuer-signed JWT and eyJhb...dYALCGg represents a Disclosure. The SD-JWT does not contain a Key Binding JWT as the outer container can be signed instead.

Other specifications or profiles of this specification may define alternative formats for transporting an SD-JWT that envelope multiple such objects into one object, and provides Key Binding using means other than the Key Binding JWT.

8. JWS JSON Serialization

This section describes an optional alternate format for SD-JWT using the JWS JSON Serialization from [RFC7515].

For both the General and Flattened JSON Serialization, the SD-JWT is represented as a JSON object according to Section 7.2 of [RFC7515]. The disclosures (both for issuance and presentation) are included in the serialized JWS using the key disclosures at the top-level of the JSON object (the same level as the payload member). The value of the disclosures member is an array of strings where each element is an individual Disclosure as described in Section 5.2. The Issuer includes a Disclosure for each selectively disclosable claim of the SD-JWT payload, whereas the Holder includes only the Disclosures selected for the given presentation. Additionally, for presentation with a Key Binding, the Holder adds the key kb_jwt at the top-level of the serialized JWS with a string value containing the Key Binding JWT as described in Section 5.10.

Verification of the JWS JSON serialized SD-JWT follows the same rules defined in Section 4.4, except that the SD-JWT does not need to be split into component parts, but disclosures and (if applicable) a Key Binding JWT can be found in the respective members of the JSON object.

Using a payload similar to that from Example 1, the following is a non-normative example of a JWS JSON serialized SD-JWT from an Issuer with all the respective Disclosures.

```
{
  "payload": "eyJfc2QiOiBbIjRIQm42YUlZM1d0dUdHV1R4LXFVajZjZGs2V0JwWn
    lnbHRkRmF2UGE3TFkiLCAiOHNtMVFDZjAyMXBObkhBQ0k1c1A0bTRLWmd5Tk9PQV
    ljVGo5SE5hQzF3WSIsICJTRE43OU5McEFuSFBta3JkZVlkRWE4OVhaZHNrME04RE
    tZU1FPVTJaeFFjIiwgIlh6RnJ6d3NjTTZHbjZDSkRjNnZWSzhCa01uZkc4dk9TS2
    ZwUElaZEFmZEUiLCAiZ2JPc0k0RWRxMngyS3ctdzV3UEV6YWtvYjloVjFjUkQwQV
    ROM29RTDlKTSIsICJqTUNYVnotLTliOHgzN1ljb0RmWFFpbnp3MXdaY2NjZkZSQk
    NGR3FkRzJvIiwgIm9LSTFHZDJmd041V3d2amxGa29oaWRHdmltLTMxT3VsUjNxMG
    hyRE8wNzgiXSwgImlzcyI6ICJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlciIsIC
    JpYXQiOiAxNjgzMDAwMDAwLCAiZXhwIjogMTg4MzAwMDAwMCwgIl9zZF9hbGciOi
    Aic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydiI6IC
    JQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTGlsRG
    xzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVmZ1ZW
    NDRTZ0NGpUOUYySFpRIn19fQ",
  "protected": "eyJhbGciOiAiRVMyNTYifQ",
  "signature": "qNNvkravlssjHS8TSnj5lAFc5on6MjG0peMt8Zjh1Yefxn0DxkcV
    OU9r7t1VNehJISOFL7NuJ5V27DVbNJBLoA",
  "disclosures": [
    "WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN1YiIsICJqb2huX2RvZV80MiJ
      d",
    "WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiSm9obiJ
      d",
    "WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIkRvZSJ
      d",
    "WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImVtYWlsIiwgImpvaG5kb2VAZXh
      hbXBsZS5jb20iXQ",
    "WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bWJlciIsICIrMS0
      yMDItNTU1LTAxMDEiXQ",
    "WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9
      hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iL
      CAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0",
    "WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImJpcnRoZGF0ZSIsICIxOTQwLTA
      xLTAxIl0"
  ]
}
```


Below is a non-normative example of a presentation of the JWS JSON serialized SD-JWT, where the Holder includes a Key Binding JWT and has selected to disclose given_name, family_name, and address.

```
{
  "payload": "eyJfc2QiOiBbIjRIQm42YUlZM1d0dUdHV1R4LXFVajZjZGs2V0JwWn
    lnbHRkRmF2UGE3TFkiLCAiOHNtMVFDZjAyMXBObkhBQ0k1c1A0bTRLWmd5Tk9PQV
    ljVGo5SE5hQzF3WSIsICJTRE43OU5McEFuSFBta3JkZVlkRWE4OVhaZHNrME04RE
    tZU1FPVTJaeFFjIiwgIlh6RnJ6d3NjTTZHbjZDSkRjNnZWSzhCa01uZkc4dk9TS2
    ZwUElaZEFmZEUiLCAiZ2JPc0k0RWRxMngyS3ctdzV3UEV6YWtvYjloVjFjUkQwQV
    ROM29RTDlKTSIsICJqTUNYVnotLTliOHgzN1ljb0RmWFFpbnp3MXdaY2NjZkZSQk
    NGR3FkRzJvIiwgIm9LSTFHZDJmd041V3d2amxGa29oaWRHdmltLTMxT3VsUjNxMG
    hyRE8wNzgiXSwgImlzcyI6ICJodHRwczovL2V4YW1wbGUuY29tL2lzc3VlciIsIC
    JpYXQiOiAxNjgzMDAwMDAwLCAiZXhwIjogMTg4MzAwMDAwMCwgIl9zZF9hbGciOi
    Aic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydiI6IC
    JQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTGlsRG
    xzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVmZ1ZW
    NDRTZ0NGpUOUYySFpRIn19fQ",
  "protected": "eyJhbGciOiAiRVMyNTYifQ",
  "signature": "qNNvkravlssjHS8TSnj5lAFc5on6MjG0peMt8Zjh1Yefxn0DxkcV
    OU9r7t1VNehJISOFL7NuJ5V27DVbNJBLoA",
  "disclosures": [
    "WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIkRvZSJ
      d",
    "WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9
      hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iL
      CAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0",
    "WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiSm9obiJ
      d"
  ],
  "kb_jwt": "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZS
    I6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29tL3Zlcm
    lmaWVyIiwgImlhdCI6IDE2ODgxNjA0ODN9.cTpy2xDWGdV6WwdWSrlRXGNb3aD9m
    sbf83XGLrmdTdOooo6y5ICEfbFzDstp5wOBuaLww-k6xv2BPqyzWDxRAQ"
}
```


9. Security Considerations

Security considerations in this section help achieve the following properties:

Selective Disclosure: An adversary in the role of the Verifier cannot obtain information from an SD-JWT about any claim name or claim value that was not explicitly disclosed by the Holder unless that information can be derived from other disclosed claims or sources other than the presented SD-JWT.

Integrity: A malicious Holder cannot modify names or values of selectively disclosable claims without detection by the Verifier.

Additionally, as described in Section 9.6, the application of Key Binding can ensure that the presenter of an SD-JWT credential is the legitimate Holder of the credential.

9.1. Mandatory Signing of the Issuer-signed JWT

The Issuer-signed JWT MUST be signed by the Issuer to protect integrity of the issued claims. An attacker can modify or add claims if this JWT is not signed (e.g., change the "email" attribute to take over the victim's account or add an attribute indicating a fake academic qualification).

The Verifier MUST always check the signature of the Issuer-signed JWT to ensure that it has not been tampered with since the issuance. The Issuer-signed JWT MUST be rejected if the signature cannot be verified.

The security of the Issuer-signed JWT depends on the security of the signature algorithm. Any of the JSON Web Signature and Encryption Algorithms registered in [IANA.JWS.Algorithms] can be used, including post-quantum algorithms, when they are ready.

9.2. Manipulation of Disclosures

Holders can manipulate the Disclosures by changing the values of the claims before sending them to the Verifier. The Verifier MUST check the Disclosures to ensure that the values of the claims are correct, i.e., the digests of the Disclosures are actually present in the signed SD-JWT.

A naive Verifier that extracts all claim values from the Disclosures (without checking the hashes) and inserts them into the SD-JWT payload is vulnerable to this attack. However, in a structured SD-JWT, without comparing the digests of the Disclosures, such an implementation could not determine the correct place in a nested object where a claim needs to be inserted. Therefore, the naive implementation would not only be insecure, but also incorrect.

The steps described in Section 6.3 ensure that the Verifier checks the Disclosures correctly.

9.3. Entropy of the salt

The security model that conceals the plaintext claims relies on the fact that salts not revealed to an attacker cannot be learned or guessed by the attacker, even if other salts have been revealed. It is vitally important to adhere to this principle. As such, each salt MUST be created in such a manner that it is cryptographically random, long enough, and has high entropy that it is not practical for the attacker to guess. A new salt MUST be chosen for each claim independently from other salts.

9.4. Minimum length of the salt

The RECOMMENDED minimum length of the randomly-generated portion of the salt is 128 bits.

The Issuer MUST ensure that a new salt value is chosen for each claim, including when the same claim name occurs at different places in the structure of the SD-JWT. This can be seen in Example 3 in the Appendix, where multiple claims with the name type appear, but each of them has a different salt.

9.5. Choice of a Hash Algorithm

For the security of this scheme, the hash algorithm is required to be preimage resistant and second-preimage resistant, i.e., it is infeasible to calculate the salt and claim value that result in a particular digest, and, for any salt and claim value pair, it is infeasible to find a different salt and claim value pair that result in the same digest, respectively.

Hash algorithms that do not meet the aforementioned requirements MUST NOT be used. Inclusion in the "Named Information Hash Algorithm" registry [IANA.Hash.Algorithms] alone does not indicate a hash algorithm's suitability for use in SD-JWT (it contains several heavily truncated digests, such as sha-256-32 and sha-256-64, which are unfit for security applications).

Furthermore, the hash algorithms MD2, MD4, MD5, and SHA-1 revealed fundamental weaknesses and they MUST NOT be used.

9.6. Key Binding

Key Binding aims to ensure that the presenter of an SD-JWT credential is actually the legitimate Holder of the credential. An SD-JWT with Key Binding contains a public key, or a reference to a public key, that corresponds to a private key possessed by the Holder. The Verifier requires that the Holder prove possession of that private key when presenting the SD-JWT credential.

Without Key Binding, a Verifier only gets the proof that the credential was issued by a particular Issuer, but the credential itself can be replayed by anyone who gets access to it. This means that, for example, after a credential was leaked to an attacker, the attacker can present the credential to any verifier that does not require a binding. But also a malicious Verifier to which the Holder presented the credential can present the credential to another Verifier if that other Verifier does not require Key Binding.

Verifiers MUST decide whether Key Binding is required for a particular use case before verifying a credential. This decision can

be informed by various factors including, but not limited to the following: business requirements, the use case, the type of binding between a Holder and its credential that is required for a use case, the sensitivity of the use case, the expected properties of a credential, the type and contents of other credentials expected to be presented at the same time, etc.

It is important that a Verifier does not make its security policy decisions based on data that can be influenced by an attacker or that can be misinterpreted. For this reason, when deciding whether Key Binding is required or not, Verifiers MUST NOT take into account

- whether a Key Binding JWT is present or not, as an attacker can remove the Key Binding JWT from any Presentation and present it to the Verifier, or

- whether Key Binding data is present in the SD-JWT or not, as the Issuer might have added the key to the SD-JWT in a format/claim that is not recognized by the Verifier.

If a Verifier has decided that Key Binding is required for a particular use case and the Key Binding is not present, does not fulfill the requirements (e.g., on the signing algorithm), or no recognized Key Binding data is present in the SD-JWT, the Verifier will reject the presentation, as described in Section 6.3.

9.6.1. Key Binding JWT

Issuer provided integrity protection of the SD-JWT payload and Disclosures is achieved by the signature on the Issuer-signed JWT that covers the SD-JWT payload including the digest values of the Disclosures as described in Section 9.1 and Section 9.2, respectively. The Key Binding JWT, defined in Section 5.10, serves exclusively as a mechanism for the Holder to demonstrate possession of the private key corresponding to the public key in the SD-JWT payload. As such, the signature on the Key Binding JWT does not cover other parts of the SD-JWT. In cases where it's desirable for the Holder's signature to convey more than a proof-of-possession, such as signing over the selected Disclosures to prove those were the Disclosures selected, the SD-JWT to be presented can be embedded in another JWT (as described in Enveloping SD-JWTs) or otherwise signed by the Holder via the application protocol delivering it.

9.7. Blinding Claim Names

SD-JWT ensures that names of claims that are selectively disclosable are always blinded. This prevents an attacker from learning the names of the disclosable claims. However, the names of the claims that are not disclosable are not blinded. This includes the keys of objects that themselves are not blinded, but contain disclosable

claims. This limitation needs to be taken into account by Issuers when creating the structure of the SD-JWT.

9.8. Issuer Signature Key Distribution and Rotation

This specification does not define how signature verification keys of Issuers are distributed to Verifiers. However, it is RECOMMENDED that Issuers publish their keys in a way that allows for efficient and secure key rotation and revocation, for example, by publishing keys at a predefined location using the JSON Web Key Set (JWKS) format [RFC7517]. Verifiers need to ensure that they are not using expired or revoked keys for signature verification using reasonable and appropriate means for the given key-distribution method.

9.9. Forwarding Credentials

When Key Binding is not enforced, any entity in possession of an SD-JWT Presentation can forward the contents to third parties. When doing so, that entity may remove Disclosures such that the receiver learns only a subset of the claims contained in the original SD-JWT.

For example, a device manufacturer might produce an SD-JWT containing information about upstream and downstream supply chain contributors. Each supply chain party can verify only the claims that were selectively disclosed to them by an upstream party, and they can choose to further reduce the disclosed claims when presenting to a downstream party.

In some scenarios this behavior could be desirable, but if it is not, Issuers need to support and Verifiers need to enforce Key Binding.

9.10. Explicit Typing

Section 3.11 of [RFC8725] describes the use of explicit typing to prevent confusion attacks in which one kind of JWT is mistaken for another. SD-JWTs are also potentially vulnerable to such confusion attacks, so it is RECOMMENDED to specify an explicit type by including the typ header parameter when the SD-JWT is issued, and for Verifiers to check this value.

When explicit typing is employed for an SD-JWT, it is RECOMMENDED that a media type name of the format "application/example+sd-jwt" be used, where "example" is replaced by the identifier for the specific kind of SD-JWT. The definition of typ in Section 4.1.9 of [RFC7515] recommends that the "application/" prefix be omitted, so "example+sd-jwt" would be the value of the typ header parameter.

10. Privacy Considerations

The privacy principles of [ISO.29100] should be adhered to.

10.1. Storage of Signed User Data

Wherever End-User data is stored, it represents a potential target for an attacker. This target can be of particularly high value when the data is signed by a trusted authority like an official national identity service. For example, in OpenID Connect, signed ID Tokens can be stored by Relying Parties. In the case of SD-JWT, Holders have to store SD-JWTs, and Issuers and Verifiers may decide to do so as well.

Not surprisingly, a leak of such data risks revealing private data of End-Users to third parties. Signed End-User data, the authenticity of which can be easily verified by third parties, further exacerbates the risk. As discussed in Section 9.6, leaked SD-JWTs may also allow attackers to impersonate Holders unless Key Binding is enforced and the attacker does not have access to the Holder's cryptographic keys. Altogether, leaked SD-JWT credentials may have a high monetary value on black markets.

Due to these risks, systems implementing SD-JWT SHOULD be designed to minimize the amount of data that is stored. All involved parties SHOULD store SD-JWTs only for as long as needed, including in log files.

Issuers SHOULD NOT store SD-JWTs after issuance.

Holders SHOULD store SD-JWTs only in encrypted form, and, wherever possible, use hardware-backed encryption in particular for the private Key Binding key. Decentralized storage of data, e.g., on End-User devices, SHOULD be preferred for End-User credentials over centralized storage. Expired SD-JWTs SHOULD be deleted as soon as possible.

Verifiers SHOULD NOT store SD-JWTs after verification. It may be sufficient to store the result of the verification and any End-User data that is needed for the application.

If reliable and secure key rotation and revocation is ensured according to Section 9.8, Issuers may opt to publish expired or revoked private signing keys (after a grace period that ensures that the keys are not cached any longer at any Verifier). This reduces the value of any leaked credentials as the signatures on them can no longer be trusted to originate from the Issuer.

10.2. Confidentiality during Transport

If the SD-JWT is transmitted over an insecure channel during issuance or presentation, an adversary may be able to intercept and read the End-User's personal data or correlate the information with previous uses of the same SD-JWT.

Usually, transport protocols for issuance and presentation of credentials are designed to protect the confidentiality of the transmitted data, for example, by requiring the use of TLS.

This specification therefore considers the confidentiality of the data to be provided by the transport protocol and does not specify any encryption mechanism.

Implementers MUST ensure that the transport protocol provides confidentiality if the privacy of End-User data or correlation attacks by passive observers are a concern.

To encrypt the SD-JWT when transmitted over an insecure channel, implementers MAY use JSON Web Encryption (JWE) [RFC7516] by nesting the SD-JWT as the plaintext payload of a JWE. Especially, when an SD-JWT is transmitted via a URL and information may be stored/cached in the browser or end up in web server logs, the SD-JWT SHOULD be encrypted using JWE.

10.3. Decoy Digests

The use of decoy digests is RECOMMENDED when the number of claims (or the existence of particular claims) can be a side-channel disclosing information about otherwise undisclosed claims. In particular, if a claim in an SD-JWT is present only if a certain condition is met (e.g., a membership number is only contained if the End-User is a member of a group), the Issuer SHOULD add decoy digests when the condition is not met.

Decoy digests increase the size of the SD-JWT. The number of decoy digests (or whether to use them at all) is a trade-off between the size of the SD-JWT and the privacy of the End-User's data.

10.4. Unlinkability

Colluding Issuer/Verifier or Verifier/Verifier pairs could link issuance/presentation or two presentation sessions to the same user on the basis of unique values encoded in the SD-JWT (Issuer signature, salts, digests, etc.).

To prevent these types of linkability, various methods, including but not limited to the following ones can be used:

* Use advanced cryptographic schemes, outside the scope of this specification.

* Issue a batch of SD-JWTs to the Holder to enable the Holder to use a unique SD-JWT per Verifier. This only helps with Verifier/Verifier unlinkability.

10.5. Issuer Identifier

An Issuer issuing only one type of SD-JWT might have privacy implications, because if the Holder has an SD-JWT issued by that Issuer, its type and claim names can be determined.

For example, if the National Cancer Institute only issued SD-JWTs with cancer registry information, it is possible to deduce that the Holder owning its SD-JWT is a cancer patient.

Moreover, the issuer identifier alone may reveal information about the user.

For example, when a military organization or a drug rehabilitation center issues a vaccine credential, verifiers can deduce that the holder is a military member or may have a substance use disorder.

To mitigate this issue, a group of issuers may elect to use a common Issuer identifier. A group signature scheme outside the scope of this specification may also be used, instead of an individual signature.

11. Acknowledgements

We would like to thank Alen Horvat, Arjan Geluk, Christian Bormann, Christian Paquin, David Bakker, David Waite, Fabian Hauck, Filip Skokan, Giuseppe De Marco, John Mattsson, Justin Richer, Kushal Das, Matthew Miller, Mike Jones, Mike Prorock, Nat Sakimura, Oliver Terbu, Orie Steele, Paul Bastian, Pieter Kasselman, Ryosuke Abe, Shawn Butterfield, Tobias Looker, Takahiko Kawasaki, Torsten Lodderstedt, Vittorio Bertocci, and Yaron Sheffer for their contributions (some of which substantial) to this draft and to the initial set of implementations.

The work on this draft was started at OAuth Security Workshop 2022 in Trondheim, Norway.

12. IANA Considerations

TBD

12.1. Media Type Registration

This section requests registration of the "application/sd-jwt" media type [RFC2046] in the "Media Types" registry [IANA.MediaTypes] in the manner described in [RFC6838].

To indicate that the content is an SD-JWT:

- *Type name: application
- *Subtype name: sd-jwt
- *Required parameters: n/a
- *Optional parameters: n/a
- *Encoding considerations: binary; application/sd-jwt values are a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') or tilde ('~') characters.
- *Security considerations: See the Security Considerations section of [[this specification]], [RFC7519], and [RFC8725].
- *Interoperability considerations: n/a
- *Published specification: [[this specification]]
- *Applications that use this media type: TBD
- *Fragment identifier considerations: n/a
- *Additional information: Magic number(s): n/a File extension(s): n/a Macintosh file type code(s): n/a
- *Person & email address to contact for further information: Daniel Fett, mail@danielfett.de
- *Intended usage: COMMON
- *Restrictions on usage: none
- *Author: Daniel Fett, mail@danielfett.de
- *Change Controller: IESG
- *Provisional registration? No

To indicate that the content is a Key Binding JWT:

- *Type name: application

*Subtype name: kb+jwt

*Required parameters: n/a

*Optional parameters: n/a

*Encoding considerations: binary; A Key Binding JWT is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.

*Security considerations: See the Security Considerations section of [[this specification]], [RFC7519], and [RFC8725].

*Interoperability considerations: n/a

*Published specification: [[this specification]]

*Applications that use this media type: TBD

*Fragment identifier considerations: n/a

*Additional information: Magic number(s): n/a File extension(s): n/a Macintosh file type code(s): n/a

*Person & email address to contact for further information: Daniel Fett, mail@danielfett.de

*Intended usage: COMMON

*Restrictions on usage: none

*Author: Daniel Fett, mail@danielfett.de

*Change Controller: IESG

*Provisional registration? No

12.2. Structured Syntax Suffix Registration

This section requests registration of the "+sd-jwt" structured syntax suffix in the "Structured Syntax Suffix" registry [IANA.StructuredSuffix] in the manner described in [RFC6838], which can be used to indicate that the media type is encoded as an SD-JWT.

*Name: SD-JWT

*+suffix: +sd-jwt

*References: [[this specification]]

*Encoding considerations: binary; SD-JWT values are a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') or tilde ('~') characters.

*Interoperability considerations: n/a

*Fragment identifier considerations: n/a

*Security considerations: See the Security Considerations section of [[this specification]], [RFC7519], and [RFC8725].

*Contact: Daniel Fett, mail@danielfett.de

*Author/Change controller: IESG

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

14. Informative References

[EUDIW.ARF] Commission, E., "The European Digital Identity Wallet Architecture and Reference Framework", <https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework>.

[I-D.terbu-sd-jwt-vc]
Terbu, O. and D. Fett, "SD-JWT-based Verifiable Credentials with JSON payloads (SD-JWT VC)", Work in Progress, Internet-Draft, draft-terbu-sd-jwt-vc-02, 26 May 2023, <https://datatracker.ietf.org/doc/html/draft-terbu-sd-jwt-vc-02>.

[IANA.Hash.Algorithms] IANA, "Named Information Hash Algorithm", <https://www.iana.org/assignments/named-information/named-information.xhtml>.

[IANA.JWS.Algorithms] IANA, "JSON Web Signature and Encryption Algorithms", <https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms>.

[IANA.MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types/media-types.xhtml>.

[IANA.StructuredSuffix] IANA, "Structured Syntax Suffixs", <https://www.iana.org/assignments/media-type-structured-suffix/media-type-structured-suffix.xhtml>.

[ISO.29100] ISO, "ISO/IEC 29100:2011 Information technology — Security techniques — Privacy framework", <https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>.

[OIDC.IDA] Lodderstedt, T., Fett, D., Haine, M., Pulido, A., Lehmann, K., and K. Koiwai, "OpenID Connect for Identity Assurance 1.0", <https://openid.net/specs/openid-connect-4-identity-assurance-1_0-13.html>.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/info/rfc8785>.

[VC_DATA_v2.0] Sporny, M., Steele, O., Jones, M. B., Cohen, G., and O. Terbu, "Verifiable Credentials Data Model 1.0", 7 March 2023, <https://www.w3.org/TR/vc-data-model-2.0/>.

Appendix A. Additional Examples

All of the following examples are non-normative.

A.1. Example 2: Handling Structured Claims

In this example, in contrast to Example 1, the Issuer decided to create a structured object for the address claim, allowing to separately disclose individual members of the claim.

The Issuer is using the following input claim set:

```
{
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "given_name": "  ",
  "family_name": "  ",
  "email": "\"unusual email address\"@example.jp",
  "phone_number": "+81-80-1234-5678",
  "address": {
    "street_address": "         – ",
    "locality": "   ",
    "region": "  ",
    "country": "JP"
  },
  "birthdate": "1940-01-01"
}
```

The Issuer also decided to add decoy digests to prevent the Verifier from deducing the true number of claims.

The following payload is used for the SD-JWT:

```
{
  "_sd": [
    "C9inp6YoRaEXR427zYJP7Qrk1WH_8bdwOA_YUrUnGQU",
    "Kuet1yAa0HIQvYnOVd59hcViO9Ug6J2kSfqYRBeowvE",
    "MMldOFFzB2d0umlmpTIaGerhWdU_PpYfLvKhh_f_9aY",
    "X6ZAYOII2vPN40V7xExZwVwz7yRmLNcVwt5DL8RLv4g",
    "Y34zmIo0QLLOtdMpXGwjBgLvr17yEhhYT0FGofR-aIE",
    "fyGp0WTwwPv2JDQln1lSiaeobZsMWA10bQ5989-9DTs",
    "ommFAicVT8LGHCB0uywx7fYuo3MHYKO15cz-RZEYM5Q",
    "s0BKYsLWxQQeU8tVlltM7MKsIRTrEIa1PkJmqxBBf5U"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "address": {
    "_sd": [
      "6aUhzYhZ7SJ1kVmagQAO3u2ETN2CC1aHheZpKnaF0_E",
      "AzLlFobkJ2xiaupREPyoJz-9-NSldB6Cgjr7fUyoHzg",
      "PzzcVu0qbMuBGSjulfewzkesD9zutOExn5EWNwkrQ-k",
      "b2Dkw0jcIF9rGg8_PF8ZcvncW7zwZj5ryBWvXfrpzek",
      "cPYJHIZ8Vu-f9CCyVub2UfgEk8jvvXezwK1p_JneeXQ",
      "glT3hrSU7fSWgwF5UDZmWwBTw32gnUldIhi8hGVCaV4",
      "rvJd6iq6T5ejmsBMoGwuNXh9qAAFATAci40oidEeVsA",
      "uNHoWYhXsZhVJCNE2Dqy-zqt7t69gJKy5QaFv7GrMX4"
    ]
  },
  "_sd_alg": "sha-256"
}
```

The following Disclosures are created:

Claim sub:

*SHA-256 Hash: X6ZAYOII2vPN40V7xExZwVwz7yRmLNcVwt5DL8RLv4g

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN1YiIsICI2YzVjMGE0OS1i
NTg5LTQzMWQtYmFlNy0yMTkxMjJhOWVjMmMiXQ

*Contents: ["2GLC42sKQveCfGfryNRN9w", "sub",
"6c5c0a49-b589-431d-bae7-219122a9ec2c"]

Claim given_name:

*SHA-256 Hash: ommFAicVT8LGHCB0uywx7fYuo3MHYKO15cz-RZEYM5Q

  *Disclosure:
   WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiXHU1
   OTJhXHU5MGNlIl0

  *Contents: ["eluV5Og3gSNII8EYnsxA_A", "given_name",
   "\u592a\u90ce"]

Claim family_name:

  *SHA-256 Hash: C9inp6YoRaEXR427zYJP7Qrk1WH_8bdwOA_YUrUnGQU

  *Disclosure:
   WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIlx1
   NWM3MVx1NzUzMCJd

  *Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "family_name",
   "\u5c71\u7530"]

Claim email:

  *SHA-256 Hash: Kuet1yAa0HIQvYnOVd59hcViO9Ug6J2kSfqYRBeowvE

  *Disclosure:
   WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImVtYWlsIiwgIlwidW51c3Vh
   bCBlbWFpbCBhZGRyZXNzXCJAZXhhbXBsZS5qcCJd

  *Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "email", "\"unusual email
   address\"@example.jp"]

Claim phone_number:

  *SHA-256 Hash: s0BKYsLWxQQeU8tVlltM7MKsIRTrEIa1PkJmqxBBf5U

  *Disclosure:
   WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bWJlciIsICIr
   ODEtODAtMTIzNC01Njc4Il0

  *Contents: ["Qg_O64zqAxe412a108iroA", "phone_number",
   "+81-80-1234-5678"]

Claim street_address:

  *SHA-256 Hash: 6aUhzYhZ7SJ1kVmagQAO3u2ETN2CC1aHheZpKnaF0_E

  *Disclosure:
   WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgInN0cmVldF9hZGRyZXNzIiwg
   Ilx1Njc3MVx1NGVhY1x1OTBmZFx1NmUyZlx1NTMzYVx1ODI5ZFx1NTE2Y1x1
   NTcxMlx1ZmYxNFx1NGUwMVx1NzZlZVx1ZmYxMlx1MjIxMlx1ZmYxOCJd

*Contents: ["AJx-095VPrpTtN4QMOqROA", "street_address",
"\u6771\u4eac\u
90fd\u6e2f\u533a\u829d\u516c\u5712\uff14\u4e01\u76ee\uff12\u
2212\uff18"]

Claim locality:

*SHA-256 Hash: rvJd6iq6T5ejmsBMoGwuNXh9qAAFATAci40oidEeVsA

*Disclosure:
WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImxvY2FsaXR5IiwgIlx1Njc3
MVx1NGVhY1x1OTBmZCJd

*Contents: ["Pc33JM2LchcU_lHggv_ufQ", "locality",
"\u6771\u4eac\u90fd"]

Claim region:

*SHA-256 Hash: PzzcVu0qbMuBGSjulfewzkesD9zutOExn5EWNwkrQ-k

*Disclosure:
WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInJlZ2lvbiIsICJcdTZlMmZc
dTUzM2EiXQ

*Contents: ["G02NSrQfjFXQ7Io09syajA", "region", "\u6e2f\u533a"]

Claim country:

*SHA-256 Hash: uNHoWYhXsZhVJCNE2Dqy-zqt7t69gJKy5QaFv7GrMX4

*Disclosure:
WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImNvdW50cnkiLCAiSlAiXQ

*Contents: ["lklxF5jMYlGTPUovMNIvCA", "country", "JP"]

Claim birthdate:

*SHA-256 Hash: MMldOFFzB2d0umlmpTIaGerhWdU_PpYfLvKhh_f_9aY

*Disclosure:
WyJ5eXRWYmRBUEdjZ2wyckk0QzlHU29nIiwgImJpcnRoZGF0ZSIsICIxOTQw
LTAxLTAxIl0

*Contents: ["yytVbdAPGcgl2rI4C9GSog", "birthdate", "1940-01-01"]

The following decoy digests are added:

*AzLlFobkJ2xiaupREPyoJz-9-NSldB6Cgjr7fUyoHzg

*cPYJHIZ8Vu-f9CCyVub2UfgEk8jvvXezwK1p_JneeXQ

*glT3hrSU7fSWgwF5UDZmWwBTw32gnUldIhi8hGVCaV4

*b2Dkw0jcIF9rGg8_PF8ZcvncW7zwZj5ryBWvXfrpzek

*fyGp0WTwwPv2JDQln1lSiaeobZsMWA10bQ5989-9DTs

*Y34zmIo0QLLOtdMpXGwjBgLvr17yEhhYT0FGofR-aIE

The following is how a presentation of the SD-JWT that discloses only region and country of the address property and without a Key Binding JWT could look like:


```
eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIkM5aW5wNllvUmFFWFI0Mjd6WUpQN1Fya
zFXSF84YmR3T0FfWVVyVW5HUVUiLCAiS3VldDF5QWEwSElRdlluT1ZkNTloY1ZpTzlVZ
zZKMmtTZnFZUkJlb3d2RSIsICJNTWxkT0ZGekIyZDB1bWxtcFRJYUdlcmhXZFVfUHBZZ
kx2S2hoX2ZfOWFZIiwgIlg2WkFZT0lJMnZQTjQwVjd4RXhad1Z3ejd5Um1MTmNWd3Q1R
Ew4Ukx2NGciLCAiWTM0em1JbzBRTExPdGRNcFhHd2pCZ0x2cjE3eUVoaFlUMEZHb2ZSL
WFJRSIsICJmeUdwMFdUd3dQdjJKRFFsbjFsU2lhZW9iWnNNV0ExMGJRNTk4OS05RFRzI
iwgIm9tbUZBaWNWVDhMR0hDQjB1eXd4N2ZZdW8zTUhZS08xNWN6LVJaRVlNNVEiLCAic
zBCS1lzTFd4UVFlVTh0VmxsdE03TUtzSVJUckVJYTFQa0ptcXhCQmY1VSJdLCAiaXNzI
jogImh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlhdCI6IDE2ODMwMDAwMDAsI
CJleHAiOiAxODgzMDAwMDAwLCAiYWRkcmVzcyI6IHsiX3NkIjogWyI2YVVoelloWjdTS
jFrVm1hZ1FBTzN1MkVUTjJDQzFhSGhlWnBLbmFGMF9FIiwgIkF6TGxGb2JrSjJ4aWF1c
FJFUHlvSnotOS1OU2xkQjZDZ2pyN2ZVeW9IemciLCAiUHp6Y1Z1MHFiTXVCR1NqdWxmZ
Xd6a2VzRDl6dXRPRXhuNUVXTndrclEtayIsICJiMkRrdzBqY0lGOXJHZzhfUEY4WmN2b
mNXN3p3Wmo1cnlCV3ZYZnJwemVrIiwgImNQWUpISVo4VnUtZjlDQ3lWdWIyVWZnRWs4a
nZ2WGV6d0sxcF9KbmVlWFEiLCAiZ2xUM2hyU1U3ZlNXZ3dGNVVEWm1Xd0JUdzMyZ25Vb
GRJaGk4aEdWQ2FWNCIsICJydkpkNmlxNlQ1ZWptc0JNb0d3dU5YaDlxQUFGQVRBY2k0M
G9pZEVlVnNBIiwgInVOSG9XWWhYc1poVkpDTkUyRHF5LXpxdDd0NjlnSkt5NVFhRnY3R
3JNWDQiXX0sICJfc2RfYWxnIjogInNoYS0yNTYifQ.rFsowW-KSZe7EITlWsGajR9nnG
BLlQ78qgtdGIZg3FZuZnxtapP0H8CUMnffJAwPQJmGnpFpulTkLWHiI1kMmw~WyJHMDJ
OU3JRZmpGWFE3SW8wOXN5YWpBIiwgInJlZ2lvbiIsICJcdTZlMmZcdTUzM2EiXQ~WyJs
a2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImNvdW50cnkiLCAiSlAiXQ~
```


A.2. Example 3 - Complex Structured SD-JWT

In this example, an SD-JWT with a complex object is represented. The data structures defined in OIDC4IDA [OIDC.IDA] are used.

The Issuer is using the following input claim set:

```
{
  "verified_claims": {
    "verification": {
      "trust_framework": "de_aml",
      "time": "2012-04-23T18:25Z",
      "verification_process": "f24c6f-6d3f-4ec5-973e-b0d8506f3bc7",
      "evidence": [
        {
          "type": "document",
          "method": "pipp",
          "time": "2012-04-22T11:30Z",
          "document": {
            "type": "idcard",
            "issuer": {
              "name": "Stadt Augsburg",
              "country": "DE"
            },
            "number": "53554554",
            "date_of_issuance": "2010-03-23",
            "date_of_expiry": "2020-03-22"
          }
        }
      ]
    },
    "claims": {
      "given_name": "Max",
      "family_name": "Müller",
      "nationalities": [
        "DE"
      ],
      "birthdate": "1956-01-28",
      "place_of_birth": {
        "country": "IS",
        "locality": "Þykkvabæjarklaustur"
      },
      "address": {
        "locality": "Maxstadt",
        "postal_code": "12344",
        "country": "DE",
        "street_address": "Weidenstraße 22"
      }
    }
  },
  "birth_middle_name": "Timotheus",
  "salutation": "Dr.",
  "msisdn": "49123456789"
}
```


The following payload is used for the SD-JWT:

```
{
  "_sd": [
    "-aSznId9mWM8ocuQolCllsxVggq1-vHW4OtnhUtVmWw",
    "IKbrYNn3vA7WEFrysvbdBJjDDU_EvQIr0W18vTRpUSg",
    "otkxuT14nBiwzNJ3MPaOitOl9pVnXOaEHal_xkyNfKI"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "verified_claims": {
    "verification": {
      "_sd": [
        "7h4UE9qScvDKodXVCuoKfKBJpVBfXMF_TmAGVaZe3Sc",
        "vTwe3raHIFYgFA3xaUD2aMxFz5oDo8iBu05qKlOg9Lw"
      ],
      "trust_framework": "de_aml",
      "evidence": [
        {
          "...": "tYJ0TDucyZZCRMbROG4qRO5vkPSFRxFhUELc18CSl3k"
        }
      ]
    },
    "claims": {
      "_sd": [
        "RiOiCn6_w5ZHaadkQMrcQJf0Jte5RwurRs54231DTlo",
        "S_498bbpKzB6Eanftss0xc7cOaoneRr3pKr7NdRmsMo",
        "WNA-UNK7F_zhsAb9syWO6IIQ1uHlTmOU8r8CvJ0cIMk",
        "Wxh_sV3iRH9bgrTBJi-aYHNCLt-vjhX1sd-igOf_9lk",
        "_O-wJiH3enSB4ROHntToQT8JmLtz-mhO2f1c89XoerQ",
        "hvDXhwmGcJQsBCA2OtjuLAcwAMpDsaU0nkovcKOqWNE"
      ]
    }
  },
  "_sd_alg": "sha-256"
}
```


The following Disclosures are created by the Issuer:

Claim time:

*SHA-256 Hash: vTwe3raHIFYgFA3xaUD2aMxFz5oDo8iBu05qKlOg9Lw

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInRpbWUiLCAiMjAxMi0wNC0y
M1QxODoyNVoiXQ

*Contents: ["2GLC42sKQveCfGfryNRN9w", "time", "2012-04-23T18:25Z"]

Claim verification_process:

*SHA-256 Hash: 7h4UE9qScvDKodXVCuoKfKBJpVBfXMF_TmAGVaZe3Sc

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgInZlcmlmaWNhdGlvbl9wcm9j
ZXNzIiwgImYyNGM2Zi02ZDNmLTRlYzUtOTczZS1iMGQ4NTA2ZjNiYzciXQ

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "verification_process",
"f24c6f-6d3f-4ec5-973e-b0d8506f3bc7"]

Claim type:

*SHA-256 Hash: G5EnhOAOoU9X_6QMNvzFXjpEA_Rc-AEtm1bG_wcaKIk

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInR5cGUiLCAiZG9jdW1lbnQi
XQ

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "type", "document"]

Claim method:

*SHA-256 Hash: WpxQ4HSoEtcTmCCKOeDslB_emucYLz2oO8oHNr1bEVQ

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgIm1ldGhvZCIsICJwaXBwIl0

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "method", "pipp"]

Claim time:

*SHA-256 Hash: 9wpjVPWuD7PK0nsQDL8B06lmdgV3LVybhHydQpTNyLI

*Disclosure:
WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInRpbWUiLCAiMjAxMi0wNC0y
MlQxMTozMFoiXQ

*Contents: ["Qg_O64zqAxe412a108iroA", "time", "2012-04-22T11:30Z"]

Claim document:

*SHA-256 Hash: IhwFrWUB63RcZq9yvgZ0XPc7Gowh3O2kqXeBIswg1B4

*Disclosure:
WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImRvY3VtZW50IiwgeyJ0eXBl
IjogImlkY2FyZCIsICJpc3N1ZXIiOiB7Im5hbWUiOiAiU3RhZHQgQXVnc2J1
cmciLCAiY291bnRyeSI6ICJERSJ9LCAibnVtYmVyIjogIjUzNTU0NTU0Iiwg
ImRhdGVfb2ZfaXNzdWFuY2UiOiAiMjAxMC0wMy0yMyIsICJkYXRlX29mX2V4
cGlyeSI6ICIyMDIwLTAzLTIyIn1d

*Contents: ["AJx-095VPrpTtN4QMOqROA", "document", {"type":
"idcard",
"issuer": {"name": "Stadt Augsburg", "country": "DE"},
"number": "53554554", "date_of_issuance": "2010-03-23",
"date_of_expiry": "2020-03-22"}]

Array Entry:

*SHA-256 Hash: tYJ0TDucyZZCRMbROG4qRO5vkPSFRxFhUELc18CSl3k

*Disclosure:
WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgeyJfc2QiOiBbIjl3cGpWUFd1
RDdQSzBuc1FETDhCMDZsbWRnVjNMVnliaEh5ZFFwVE55TEkiLCAiRzVFbmhP
QU9vVTlYXzZRTU52ekZYanBFQV9SYy1BRXRtMWJHX3djYUtJayIsICJJaHdG
cldVQjYzUmNacTl5dmdaMFhQYzdHb3doM08ya3FYZUJJc3dnMUI0IiwgIldw
eFE0SFNvRXRjVG1DQ0tPZURzbEJfZW11Y1lMejJvTzhvSE5yMWJFVlEiXX1d

*Contents: ["Pc33JM2LchcU_lHggv_ufQ", {"_sd":
["9wpjVPWuD7PK0nsQDL8B06lmdgV3LVybhHydQpTNyLI",
"G5EnhOAOoU9X_6QMNvzFXjpEA_Rc-AEtm1bG_wcaKIk",
"IhwFrWUB63RcZq9yvgZ0XPc7Gowh3O2kqXeBIswg1B4",
"WpxQ4HSoEtcTmCCKOeDslB_emucYLz2oO8oHNr1bEVQ"]}]

Claim given_name:

*SHA-256 Hash: S_498bbpKzB6Eanftss0xc7cOaoneRr3pKr7NdRmsMo

*Disclosure:
WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImdpdmVuX25hbWUiLCAiTWF4
Il0

*Contents: ["G02NSrQfjFXQ7Io09syajA", "given_name", "Max"]

Claim family_name:

*SHA-256 Hash: Wxh_sV3iRH9bgrTBJi-aYHNCLt-vjhX1sd-igOf_9lk

*Disclosure:
WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImZhbWlseV9uYW1lIiwgIk1c
dTAwZmNsbGVyIl0

*Contents: ["lklxF5jMYlGTPUovMNIvCA", "family_name",
"M\u00fcller"]

Claim nationalities:

*SHA-256 Hash: hvDXhwmGcJQsBCA2OtjuLAcwAMpDsaU0nkovcKOqWNE

*Disclosure:
WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgIm5hdGlvbmFsaXRpZXMiLCBb
IkRFIl1d

*Contents: ["nPuoQnkRFq3BIeAm7AnXFA", "nationalities", ["DE"]]

Claim birthdate:

*SHA-256 Hash: WNA-UNK7F_zhsAb9syWO6IIQ1uHlTmOU8r8CvJ0cIMk

*Disclosure:
WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImJpcnRoZGF0ZSIsICIxOTU2
LTAxLTI4Il0

*Contents: ["5bPs1IquZNa0hkaFzzzZNw", "birthdate", "1956-01-28"]

Claim place_of_birth:

*SHA-256 Hash: RiOiCn6_w5ZHaadkQMrcQJf0Jte5RwurRs54231DTlo

*Disclosure:
WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13IiwgInBsYWNlX29mX2JpcnRoIiwg
eyJjb3VudHJ5IjogIklTIiwgImxvY2FsaXR5IjogIlx1MDBkZXlra3ZhYlx1
MDBlNmphcmtsYXVzdHVyIn1d

*Contents: ["5a2W0_NrlEZzfqmk_7Pq-w", "place_of_birth",
{"country":
"IS", "locality": "\u00deykkvab\u00e6jarklaustur"}]

Claim address:

*SHA-256 Hash: _O-wJiH3enSB4ROHntToQT8JmLtz-mhO2f1c89XoerQ

*Disclosure:
WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImFkZHJlc3MiLCB7ImxvY2Fs
aXR5IjogIk1heHN0YWR0IiwgInBvc3RhbF9jb2RlIjogIjEyMzQ0IiwgImNv
dW50cnkiOiAiREUiLCAic3RyZWV0X2FkZHJlc3MiOiAiV2VpZGVuc3RyYVx1
MDBkZmUgMjIifV0

*Contents: ["y1sVU5wdfJahVdgwPgS7RQ", "address", {"locality":
"Maxstadt", "postal_code": "12344", "country": "DE",
"street_address": "Weidenstra\u00dfe 22"}]

Claim birth_middle_name:

*SHA-256 Hash: otkxuT14nBiwzNJ3MPaOitOl9pVnXOaEHal_xkyNfKI

*Disclosure:
WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImJpcnRoX21pZGRsZV9uYW1l
IiwgIlRpbW90aGV1cyJd

  *Contents: ["HbQ4X8srVW3QDxnIJdqyOA", "birth_middle_name",
   "Timotheus"]

Claim salutation:

  *SHA-256 Hash: -aSznId9mWM8ocuQolCllsxVggq1-vHW4OtnhUtVmWw

  *Disclosure:
   WyJDOUdTb3VqdmlKcXVFZ1lmb2pDYjFBIiwgInNhbHV0YXRpb24iLCAiRHIu
   Il0

  *Contents: ["C9GSoujviJquEgYfojCb1A", "salutation", "Dr."]

Claim msisdn:

  *SHA-256 Hash: IKbrYNn3vA7WEFrysvbdBJjDDU_EvQIr0W18vTRpUSg

  *Disclosure:
   WyJreDVrRjE3Vi14MEptd1V4OXZndnR3IiwgIm1zaXNkbiIsICI0OTEyMzQ1
   Njc4OSJd

  *Contents: ["kx5kF17V-x0JmwUx9vgvtw", "msisdn", "49123456789"]

The following is how a presentation of the SD-JWT without a Key Binding JWT could look like:

eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIi1hU3puSWQ5bVdNOG9jdVFvbENsbHN4V
mdncTEtdkhXNE90bmhVdFZtV3ciLCAiSUticllObjN2QTdXRUZyeXN2YmRCSmpERFVfR
XZRSXIwVzE4dlRScFVTZyIsICJvdGt4dVQxNG5CaXd6TkozTVBhT2l0T2w5cFZuWE9hR
UhhbF94a3lOZktJIl0sICJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1ZXIiL
CAiaWF0IjogMTY4MzAwMDAwMCwgImV4cCI6IDE4ODMwMDAwMDAsICJ2ZXJpZmllZF9jb
GFpbXMiOiB7InZlcmlmaWNhdGlvbiI6IHsiX3NkIjogWyI3aDRVRTlxU2N2REtvZFhWQ
3VvS2ZLQkpwVkJmWE1GX1RtQUdWYVplM1NjIiwgInZUd2UzcmFISUZZZ0ZBM3hhVUQyY
U14Rno1b0RvOGlCdTA1cUtsT2c5THciXSwgInRydXN0X2ZyYW1ld29yayI6ICJkZV9hb
WwiLCAiZXZpZGVuY2UiOiBbeyIuLi4iOiAidFlKMFREdWN5WlpDUk1iUk9HNHFSTzV2a
1BTRlJ4RmhVRUxjMThDU2wzayJ9XX0sICJjbGFpbXMiOiB7Il9zZCI6IFsiUmlPaUNuN
l93NVpIYWFka1FNcmNRSmYwSnRlNVJ3dXJSczU0MjMxRFRsbyIsICJTXzQ5OGJicEt6Q
jZFYW5mdHNzMHhjN2NPYW9uZVJyM3BLcjdOZFJtc01vIiwgIldOQS1VTks3Rl96aHNBY
jlzeVdPNklJUTF1SGxUbU9VOHI4Q3ZKMGNJTWsiLCAiV3hoX3NWM2lSSDliZ3JUQkppL
WFZSE5DTHQtdmpoWDFzZC1pZ09mXzlsayIsICJfTy13SmlIM2VuU0I0Uk9IbnRUb1FUO
EptTHR6LW1oTzJmMWM4OVhvZXJRIiwgImh2RFhod21HY0pRc0JDQTJPdGp1TEFjd0FNc
ERzYVUwbmtvdmNLT3FXTkUiXX19LCAiX3NkX2FsZyI6ICJzaGEtMjU2In0.Xtpp8nvAq
22k6wNRiYHGRoRnkn3EBaHdjcaa0sf0sYjCiyZnmSRlxv_C72gRwfVQkSA36ID_I46QS
TZvBrgm3g~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInRpbWUiLCAiMjAxMi0wNC
0yM1QxODoyNVoiXQ~WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgeyJfc2QiOiBbIjl
3cGpWUFd1RDdQSzBuc1FETDhCMDZsbWRnVjNMVnliaEh5ZFFwVE55TEkiLCAiRzVFbmh
PQU9vVTlYXzZRTU52ekZYanBFQV9SYy1BRXRtMWJHX3djYUtJayIsICJJaHdGcldVQjY
zUmNacTl5dmdaMFhQYzdHb3doM08ya3FYZUJJc3dnMUI0IiwgIldweFE0SFNvRXRjVG1
DQ0tPZURzbEJfZW11Y1lMejJvTzhvSE5yMWJFVlEiXX1d~WyJlSThaV205UW5LUHBOUG
VOZW5IZGhRIiwgIm1ldGhvZCIsICJwaXBwIl0~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YW
pBIiwgImdpdmVuX25hbWUiLCAiTWF4Il0~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIi
wgImZhbWlseV9uYW1lIiwgIk1cdTAwZmNsbGVyIl0~WyJ5MXNWVTV3ZGZKYWhWZGd3UG
dTN1JRIiwgImFkZHJlc3MiLCB7ImxvY2FsaXR5IjogIk1heHN0YWR0IiwgInBvc3RhbF
9jb2RlIjogIjEyMzQ0IiwgImNvdW50cnkiOiAiREUiLCAic3RyZWV0X2FkZHJlc3MiOi
AiV2VpZGVuc3RyYVx1MDBkZmUgMjIifV0~

After the validation, the Verifier will have the following data for further processing:

```
{
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "verified_claims": {
    "verification": {
      "trust_framework": "de_aml",
      "evidence": [
        {
          "method": "pipp"
        }
      ],
      "time": "2012-04-23T18:25Z"
    },
    "claims": {
      "given_name": "Max",
      "family_name": "Müller",
      "address": {
        "locality": "Maxstadt",
        "postal_code": "12344",
        "country": "DE",
        "street_address": "Weidenstraße 22"
      }
    }
  },
  "_sd_alg": "sha-256"
}
```

A.3. Example 4a - SD-JWT-based Verifiable Credentials (SD-JWT VC)

In this example, the artifacts defined in this specification are used to represent SD-JWT-based Verifiable Credentials (SD-JWT VC) as defined in [I-D.terbu-sd-jwt-vc]. Person Identification Data (PID) defined in [EUDIW.ARF] is used.

Key Binding is applied using the Holder's public key passed in a cnf claim in the SD-JWT.

The Issuer is using the following input claim set:

```
{
  "iss": "https://pid-provider.memberstate.example.eu",
  "iat": 1541493724,
  "type": "PersonIdentificationData",
  "first_name": "Erika",
  "family_name": "Mustermann",
  "nationalities": [
    "DE"
  ],
  "birth_family_name": "Schmidt",
  "birthdate": "1973-01-01",
  "address": {
    "postal_code": "12345",
    "locality": "Irgendwo",
    "street_address": "Sonnenstrasse 23",
    "country_code": "DE"
  },
  "is_over_18": true,
  "is_over_21": true,
  "is_over_65": false
}
```


The following is the issued SD-JWT (with line breaks for formatting only):

eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIjA5VGJTdW8xMmkyQ3FaYmczMUFGZ2JHe
V9Vbk1JWElIb01qc0VMcHVrcWciLCAiMG45eXpGU1d2S19CVUhpYU1obTEyZ2hyQ3RWY
WhyR0o2Xy1rWlAteVNxNCIsICI0Vm9BM2ExVm1QeG1kQzhXSW4zcE9xUWYzZ2ZPVk92R
FlzTjVFNVI1S2QwIiwgIjVBODhBbWF1QWFvLVFBTmFvOTVDWVVrVVBOVGlkX2dBSzhhW
XRaOVJad2MiLCAiOTEwYnlyM1VWUnFSelFvUHpCc2MyMG0tZU1ncFpBaExONno4Tm9HR
jVtYyIsICJDaC1EQmNMM2tiNFZiSEl3dGtublpkTlVIdGhFcTlNWmpvRmRnNmlkaWhvI
iwgIkkwMGZjRlVvRFhDdWNwNXl5MnVqcVBzc0RWR2FXTmlVbGlOel9hd0QwZ2MiLCAiW
DlNYVBhRldtUVlwZkhFZHl0UmRhY2xuWW9FcnU4RXp0QkVVUXVXT2U0NCIsICJZMXVyV
0pWXy1IQkduU2Y5dEZPd3ZINGNJQ1JCQ2lLd0VIZmtYRlNmanBvIiwgInJOaEtvcmFhc
S0teDdCV1dJVmhiR1h1MVhYWExNOGl2WlhEM20yRlpNZ3MiLCAieHBzcTZjeFFIRHNPb
lpXaHJxQmNrVGtPTV9lZkVsVW5ERlhPRm1vd0xTRSIsICJ6VTQ1MmxrR2JFS2g4WnVIX
zhLeDNDVXZuMUY0eTFnWkxxbERUZ1hfOFBrIl0sICJpc3MiOiAiaHR0cHM6Ly9waWQtc
HJvdmlkZXIubWVtYmVyc3RhdGUuZXhhbXBsZS5ldSIsICJpYXQiOiAxNTQxNDkzNzI0L
CAiZXhwIjogMTg4MzAwMDAwMCwgInR5cGUiOiAiUGVyc29uSWRlbnRpZmljYXRpb25EY
XRhIiwgIl9zZF9hbGciOiAic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogI
kVDIiwgImNydiI6ICJQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb
0hJUDFJTGlsRGxzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJa
XJzVmZ1ZWNDRTZ0NGpUOUYySFpRIn19fQ.K5Ol1OgKQtP3FGBoJ8pmPdraIsSeOHxOAE
-64L3Bc3q_aq2ANQSRNh4hqPYjuK4CnqyCK1reyHLO2iiMDwleOw~WyIyR0xDNDJzS1F
2ZUNmR2ZyeU5STjl3IiwgImZpcnN0X25hbWUiLCAiRXJpa2EiXQ~WyJlbHVWNU9nM2dT
TklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIk11c3Rlcm1hbm4iXQ~WyI2SWo3d
E0tYTVpVlBHYm9TNXRtdlZBIiwgIkRFIl0~WyJlSThaV205UW5LUHBOUGVOZW5IZGhRI
iwgIm5hdGlvbmFsaXRpZXMiLCBbeyIuLi4iOiAiSnVMMzJRWER6aXpsLUw2Q0xyZnhma
nBac1gzTzZ2c2ZwQ1ZkMWprd0pZZyJ9XV0~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BI
iwgImJpcnRoX2ZhbWlseV9uYW1lIiwgIlNjaG1pZHQiXQ~WyJBSngtMDk1VlBycFR0Tj
RRTU9xUk9BIiwgImJpcnRoZGF0ZSIsICIxOTczLTAxLTAxIl0~WyItNmhDZVJFV3JjRW
g0Q2JzQ2RjVTVRIiwgImFkZHJlc3MiLCB7Il9zZCI6IFsiOHo4ejlYOWpVdGI5OWdqZW
pDd0ZBR3o0YXFsSGYtc0NxUTZlTV9xbXBVUSIsICJDeHE0ODcyVVhYbmdHVUxUX2tsOG
Zkd1ZGa3lLNkFKZlBaTHk3TDVfMGtJIiwgImxjMk8weER4WXdiZ2M4cjJERXc3eWhfSU
RXZXhXOENiZ3R6WVBRRlJpNGMiXSwgInBvc3RhbF9jb2RlIjogIjEyMzQ1IiwgImxvY2
FsaXR5IjogIklyZ2VuZHdvIiwgInN0cmVldF9hZGRyZXNzIjogIlNvbm5lbnN0cmFzc2
UgMjMiLCAiY291bnRyeV9jb2RlIjogIkRFIn1d~WyJzLXpVaXE1azFyU0dSb1hQUE5rM
zVRIiwgImlzX292ZXJfMTgiLCB0cnVlXQ~WyJyWmJUODJ1VVJuTi1xYVRfcy1yN3l3Ii
wgImlzX292ZXJfMjEiLCB0cnVlXQ~WyJXeFZUbkIxOGxxRlYyREEtQkx0RkhRIiwgIml
zX292ZXJfNjUiLCBmYWxzZV0~

The following payload is used for the SD-JWT:

```
{
  "_sd": [
    "09TbSuo12i2CqZbg31AFgbGy_UnMIXIHoMjsELpukqg",
    "0n9yzFSWvK_BUHiaMhm12ghrCtVahrGJ6_-kZP-ySq4",
    "4VoA3a1VmPxmdC8WIn3pOqQf3gfOVOvDYsN5E5R5Kd0",
    "5A88AmauAao-QANao95CYUkUPNTid_gAK8aYtZ9RZwc",
    "910byr3UVRqRzQoPzBsc20m-eMgpZAhLN6z8NoGF5mc",
    "Ch-DBcL3kb4VbHIwtknnZdNUHthEq9MZjoFdg6idiho",
    "I00fcFUoDXCucp5yy2ujqPssDVGaWNiUliNz_awD0gc",
    "X9MaPaFWmQYpfHEdytRdaclnYoEru8EztBEUQuWOe44",
    "Y1urWJV_-HBGnSf9tFOwvH4cICRBCiKwEHfkXFSfjpo",
    "rNhKoraaq--x7BWWIVhbGXu1XXXLM8ivZXD3m2FZMgs",
    "xpsq6cxQHDsOnZWhrqBckTkOM_efElUnDFXOFmowLSE",
    "zU452lkGbEKh8ZuH_8Kx3CUvn1F4y1gZLqlDTgX_8Pk"
  ],
  "iss": "https://pid-provider.memberstate.example.eu",
  "iat": 1541493724,
  "exp": 1883000000,
  "type": "PersonIdentificationData",
  "_sd_alg": "sha-256",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```


The following Disclosures are created by the Issuer:

Claim first_name:

*SHA-256 Hash: Ch-DBcL3kb4VbHIwtknnZdNUHthEq9MZjoFdg6idiho

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImZpcnN0X25hbWUiLCAiRXJp
a2EiXQ

*Contents: ["2GLC42sKQveCfGfryNRN9w", "first_name", "Erika"]

Claim family_name:

*SHA-256 Hash: I00fcFUoDXCucp5yy2ujqPssDVGaWNiUliNz_awD0gc

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIk11
c3Rlcm1hbm4iXQ

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "family_name", "Mustermann"]

Array Entry:

*SHA-256 Hash: JuL32QXDzizl-L6CLrfxfjpZsX3O6vsfpCVd1jkwJYg

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgIkRFIl0

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA", "DE"]

Claim nationalities:

*SHA-256 Hash: zU452lkGbEKh8ZuH_8Kx3CUvn1F4y1gZLqlDTgX_8Pk

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgIm5hdGlvbmFsaXRpZXMiLCBb
eyIuLi4iOiAiSnVMMzJRWER6aXpsLUw2Q0xyZnhmanBac1gzTzZ2c2ZwQ1Zk
MWprd0pZZyJ9XV0

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "nationalities", [{"...":
"JuL32QXDzizl-L6CLrfxfjpZsX3O6vsfpCVd1jkwJYg"}]]

Claim birth_family_name:

*SHA-256 Hash: X9MaPaFWmQYpfHEdytRdaclnYoEru8EztBEUQuWOe44

*Disclosure:
WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImJpcnRoX2ZhbWlseV9uYW1l
IiwgIlNjaG1pZHQiXQ

*Contents: ["Qg_O64zqAxe412a108iroA", "birth_family_name",
"Schmidt"]

Claim birthdate:

*SHA-256 Hash: 0n9yzFSWvK_BUHiaMhm12ghrCtVahrGJ6_-kZP-ySq4

*Disclosure:
WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImJpcnRoZGF0ZSIsICIxOTcz
LTAxLTAxIl0

*Contents: ["AJx-095VPrpTtN4QMOqROA", "birthdate", "1973-01-01"]

Claim address:

*SHA-256 Hash: Y1urWJV_-HBGnSf9tFOwvH4cICRBCiKwEHfkXFSfjpo

*Disclosure:
WyItNmhDZVJFV3JjRWg0Q2JzQ2RjVTVRIiwgImFkZHJlc3MiLCB7Il9zZCI6

IFsiOHo4ejlYOWpVdGI5OWdqZWpDd0ZBR3o0YXFsSGYtc0NxUTZlTV9xbXBV
USIsICJDeHE0ODcyVVhYbmdHVUxUX2tsOGZkd1ZGa3lLNkFKZlBaTHk3TDVf
MGtJIiwgImxjMk8weER4WXdiZ2M4cjJERXc3eWhfSURXZXhXOENiZ3R6WVBR
RlJpNGMiXSwgInBvc3RhbF9jb2RlIjogIjEyMzQ1IiwgImxvY2FsaXR5Ijog
IklyZ2VuZHdvIiwgInN0cmVldF9hZGRyZXNzIjogIlNvbm5lbnN0cmFzc2Ug
MjMiLCAiY291bnRyeV9jb2RlIjogIkRFIn1d

*Contents: ["-6hCeREWrcEh4CbsCdcU5Q", "address", {"_sd":
["8z8z9X9jUtb99gjejCwFAGz4aqlHf-sCqQ6eM_qmpUQ",
"Cxq4872UXXngGULT_kl8fdwVFkyK6AJfPZLy7L5_0kI",
"lc2O0xDxYwbgc8r2DEw7yh_IDWexW8CbgtzYPQFRi4c"],
"postal_code": "12345", "locality": "Irgendwo",
"street_address": "Sonnenstrasse 23", "country_code": "DE"}]

Claim is_over_18:

*SHA-256 Hash: rNhKoraaq--x7BWWIVhbGXu1XXXLM8ivZXD3m2FZMgs

*Disclosure:
WyJzLXpVaXE1azFyU0dSb1hQUE5rMzVRIiwgImlzX292ZXJfMTgiLCB0cnVl
XQ

*Contents: ["s-zUiq5k1rSGRoXPPNk35Q", "is_over_18", true]

Claim is_over_21:

*SHA-256 Hash: 09TbSuo12i2CqZbg31AFgbGy_UnMIXIHoMjsELpukqg

*Disclosure:
WyJyWmJUODJ1VVJuTi1xYVRfcy1yN3l3IiwgImlzX292ZXJfMjEiLCB0cnVl
XQ

*Contents: ["rZbT82uURnN-qaT_s-r7yw", "is_over_21", true]

Claim is_over_65:

*SHA-256 Hash: 910byr3UVRqRzQoPzBsc20m-eMgpZAhLN6z8NoGF5mc

*Disclosure:
WyJXeFZUbkIxOGxxRlYyREEtQkx0RkhRIiwgImlzX292ZXJfNjUiLCBmYWxz
ZV0

*Contents: ["WxVTnB18lqFV2DA-BLtFHQ", "is_over_65", false]

The following decoy digests are added:

*AzLlFobkJ2xiaupREPyoJz-9-NSldB6Cgjr7fUyoHzg

*cPYJHIZ8Vu-f9CCyVub2UfgEk8jvvXezwK1p_JneeXQ

*glT3hrSU7fSWgwF5UDZmWwBTw32gnUldIhi8hGVCaV4

*b2Dkw0jcIF9rGg8_PF8ZcvncW7zwZj5ryBWvXfrpzek

*fyGp0WTwwPv2JDQln1lSiaeobZsMWA10bQ5989-9DTs

*Y34zmIo0QLLOtdMpXGwjBgLvr17yEhhYT0FGofR-aIE

The following is how a presentation of the SD-JWT with a Key Binding JWT that discloses only nationality and the fact that the person is over 18 years old could look like:

```
eyJhbGciOiAiRVMyNTYifQ.eyJfc2QiOiBbIjA5VGJTdW8xMmkyQ3FaYmczMUFGZ2JHe
V9Vbk1JWElIb01qc0VMcHVrcWciLCAiMG45eXpGU1d2S19CVUhpYU1obTEyZ2hyQ3RWY
WhyR0o2Xy1rWlAteVNxNCIsICI0Vm9BM2ExVm1QeG1kQzhXSW4zcE9xUWYzZ2ZPVk92R
FlzTjVFNVI1S2QwIiwgIjVBODhBbWF1QWFvLVFBTmFvOTVDWVVrVVBOVGlkX2dBSzhhW
XRaOVJad2MiLCAiOTEwYnlyM1VWUnFSelFvUHpCc2MyMG0tZU1ncFpBaExONno4Tm9HR
jVtYyIsICJDaC1EQmNMM2tiNFZiSEl3dGtublpkTlVIdGhFcTlNWmpvRmRnNmlkaWhvI
iwgIkkwMGZjRlVvRFhDdWNwNXl5MnVqcVBzc0RWR2FXTmlVbGlOel9hd0QwZ2MiLCAiW
DlNYVBhRldtUVlwZkhFZHl0UmRhY2xuWW9FcnU4RXp0QkVVUXVXT2U0NCIsICJZMXVyV
0pWXy1IQkduU2Y5dEZPd3ZINGNJQ1JCQ2lLd0VIZmtYRlNmanBvIiwgInJOaEtvcmFhc
S0teDdCV1dJVmhiR1h1MVhYWExNOGl2WlhEM20yRlpNZ3MiLCAieHBzcTZjeFFIRHNPb
lpXaHJxQmNrVGtPTV9lZkVsVW5ERlhPRm1vd0xTRSIsICJ6VTQ1MmxrR2JFS2g4WnVIX
zhLeDNDVXZuMUY0eTFnWkxxbERUZ1hfOFBrIl0sICJpc3MiOiAiaHR0cHM6Ly9waWQtc
HJvdmlkZXIubWVtYmVyc3RhdGUuZXhhbXBsZS5ldSIsICJpYXQiOiAxNTQxNDkzNzI0L
CAiZXhwIjogMTg4MzAwMDAwMCwgInR5cGUiOiAiUGVyc29uSWRlbnRpZmljYXRpb25EY
XRhIiwgIl9zZF9hbGciOiAic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogI
kVDIiwgImNydiI6ICJQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb
0hJUDFJTGlsRGxzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJa
XJzVmZ1ZWNDRTZ0NGpUOUYySFpRIn19fQ.K5Ol1OgKQtP3FGBoJ8pmPdraIsSeOHxOAE
-64L3Bc3q_aq2ANQSRNh4hqPYjuK4CnqyCK1reyHLO2iiMDwleOw~WyJzLXpVaXE1azF
yU0dSb1hQUE5rMzVRIiwgImlzX292ZXJfMTgiLCB0cnVlXQ~WyJlSThaV205UW5LUHBO
UGVOZW5IZGhRIiwgIm5hdGlvbmFsaXRpZXMiLCBbeyIuLi4iOiAiSnVMMzJRWER6aXps
LUw2Q0xyZnhmanBac1gzTzZ2c2ZwQ1ZkMWprd0pZZyJ9XV0~WyI2SWo3dE0tYTVpVlBH
Ym9TNXRtdlZBIiwgIkRFIl0~eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9
.eyJub25jZSI6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29
tL3ZlcmlmaWVyIiwgImlhdCI6IDE2ODgxNjA0ODN9.duRIKesDpGY-5GkRcr98uhud64
PfmPhL0qMcXFeBL5x2IGbAc_buglOrpd0LZA_cgCGXDx4zQoMou2kKrl-WCA
```

The following is the payload of a corresponding Key Binding JWT:

```
{
  "nonce": "1234567890",
  "aud": "https://example.com/verifier",
  "iat": 1688160483
}
```

After the validation, the Verifier will have the following data for further processing:

```
{
  "iss": "https://pid-provider.memberstate.example.eu",
  "iat": 1541493724,
  "exp": 1883000000,
  "type": "PersonIdentificationData",
  "_sd_alg": "sha-256",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  },
  "is_over_18": true,
  "nationalities": [
    "DE"
  ]
}
```

A.4. Example 4b - W3C Verifiable Credentials Data Model v2.0

In this example, the artifacts defined in this specification are used to represent a payload that is represented as a W3C Verifiable Credentials Data Model v2.0 [VC_DATA_v2.0].

Key Binding is applied using the Holder's public key passed in a cnf claim in the SD-JWT.

The Issuer is using the following input claim set:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/vaccination/v1"
  ],
  "type": [
    "VerifiableCredential",
    "VaccinationCertificate"
  ],
  "issuer": "https://example.com/issuer",
  "issuanceDate": "2023-02-09T11:01:59Z",
  "expirationDate": "2028-02-08T11:01:59Z",
  "name": "COVID-19 Vaccination Certificate",
  "description": "COVID-19 Vaccination Certificate",
  "credentialSubject": {
    "vaccine": {
      "type": "Vaccine",
      "atcCode": "J07BX03",
      "medicinalProductName": "COVID-19 Vaccine Moderna",
      "marketingAuthorizationHolder": "Moderna Biotech"
    },
    "nextVaccinationDate": "2021-08-16T13:40:12Z",
    "countryOfVaccination": "GE",
    "dateOfVaccination": "2021-06-23T13:40:12Z",
    "order": "3/3",
    "recipient": {
      "type": "VaccineRecipient",
      "gender": "Female",
      "birthDate": "1961-08-17",
      "givenName": "Marion",
      "familyName": "Mustermann"
    },
    "type": "VaccinationEvent",
    "administeringCentre": "Praxis Sommergarten",
    "batchNumber": "1626382736",
    "healthProfessional": "883110000015376"
  }
}
```


The following is the issued SD-JWT (with line breaks for formatting only):

```
eyJhbGciOiAiRVMyNTYifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAiaWF0IjogMTY4MzAwMDAwMCwgImV4cCI6IDE4ODMwMDAwMDAsICJAY29udGV4d
CI6IFsiaHR0cHM6Ly93d3cudzMub3JnLzIwMTgvY3JlZGVudGlhbHMvdjEiLCAiaHR0c
HM6Ly93M2lkLm9yZy92YWNjaW5hdGlvbi92MSJdLCAidHlwZSI6IFsiVmVyaWZpYWJsZ
UNyZWRlbnRpYWwiLCAiVmFjY2luYXRpb25DZXJ0aWZpY2F0ZSJdLCAiaXNzdWVyIjogI
mh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlzc3VhbmNlRGF0ZSI6ICIyMDIzL
TAyLTA5VDExOjAxOjU5WiIsICJleHBpcmF0aW9uRGF0ZSI6ICIyMDI4LTAyLTA4VDExO
jAxOjU5WiIsICJuYW1lIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRlI
iwgImRlc2NyaXB0aW9uIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRlI
iwgImNyZWRlbnRpYWxTdWJqZWN0IjogeyJfc2QiOiBbIjFWX0stOGxEUThpRlhCRlhiW
lk5ZWhxUjRIYWJXQ2k1VDB5Ykl6WlBld3ciLCAiSnpqTGd0UDI5ZFAtQjN0ZDEyUDY3N
GdGbUsyenk4MUhNdEJnZjZDSk5XZyIsICJSMmZHYmZBMDdaX1lsa3FtTlp5bWExeHl5e
DFYc3RJaVM2QjFZYmwySlo0IiwgIlRDbXpybDdLMmdldl9kdTdwY01JeXpSTEhwLVllZ
y1GbF9jeHRyVXZQeGciLCAiVjdrSkJMSzc4VG1WRE9tcmZKN1p1VVBIdUtfMmNjN3laU
mE0cVYxdHh3TSIsICJiMGVVc3ZHUC1PRERkRm9ZNE5semxYYzN0RHNsV0p0Q0pGNzVOd
zhPal9nIiwgInpKS19lU01YandNOGRYbU1aTG5JOEZHTTA4ekozX3ViR2VFTUotNVRCe
TAiXSwgInZhY2NpbmUiOiB7Il9zZCI6IFsiMWNGNWhMd2toTU5JYXFmV0pyWEk3Tk1XZ
WRMLTlmNlkyUEE1MnlQalNaSSIsICJIaXk2V1d1ZUxENWJuMTYyOTh0UHY3R1hobWxkT
URPVG5CaS1DWmJwaE5vIiwgIkxiMDI3cTY5MWpYWGwtakM3M3ZpOGViT2o5c214M0MtX
29nN2dBNFRCUUUiXSwgInR5cGUiOiAiVmFjY2luZSJ9LCAicmVjaXBpZW50IjogeyJfc
2QiOiBbIjFsU1FCTlkyNHEwVGg2T0d6dGhxLTctNGw2Y0FheHJZWE9HWnBlV19sbkEiL
CAiM256THE4MU0yb04wNndkdjFzaEh2T0VKVnhaNUtMbWREa0hFREpBQldFSSIsICJQb
jFzV2kwNkc0TEpybm4tX1JUMFJiTV9IVGR4blBKUXVYMmZ6V3ZfSk9VIiwgImxGOXV6Z
HN3N0hwbEdMYzcxNFRyNFdPN01HSnphN3R0N1FGbGVDWDRJdHciXSwgInR5cGUiOiAiV
mFjY2luZVJlY2lwaWVudCJ9LCAidHlwZSI6ICJWYWNjaW5hdGlvbkV2ZW50In0sICJfc
2RfYWxnIjogInNoYS0yNTYifQ.5jJEqiRViN_DJ4VMxHQIN4KK-Cdfn30nY3nRe5jGxS
5Pths5G7mF1GWy9TawYuyFwCze7qiCIx_MhJ68Uu5zeQ~WyIyR0xDNDJzS1F2ZUNmR2Z
yeU5STjl3IiwgImF0Y0NvZGUiLCAiSjA3QlgwMyJd~WyJlbHVWNU9nM2dTTklJOEVZbn
N4QV9BIiwgIm1lZGljaW5hbFByb2R1Y3ROYW1lIiwgIkNPVklELTE5IFZhY2NpbmUgTW
9kZXJuYSJd~WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgIm1hcmtldGluZ0F1dGhvc
ml6YXRpb25Ib2xkZXIiLCAiTW9kZXJuYSBCaW90ZWNoIl0~WyJlSThaV205UW5LUHBOU
GVOZW5IZGhRIiwgIm5leHRWYWNjaW5hdGlvbkRhdGUiLCAiMjAyMS0wOC0xNlQxMzo0M
DoxMloiXQ~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImNvdW50cnlPZlZhY2Npbm
F0aW9uIiwgIkdFIl0~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImRhdGVPZlZhY2
NpbmF0aW9uIiwgIjIwMjEtMDYtMjNUMTM6NDA6MTJaIl0~WyJQYzMzSk0yTGNoY1VfbE
hnZ3ZfdWZRIiwgIm9yZGVyIiwgIjMvMyJd~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBI
iwgImdlbmRlciIsICJGZW1hbGUiXQ~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIm
JpcnRoRGF0ZSIsICIxOTYxLTA4LTE3Il0~WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIi
wgImdpdmVuTmFtZSIsICJNYXJpb24iXQ~WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53Iiw
gImZhbWlseU5hbWUiLCAiTXVzdGVybWFubiJd~WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS
13IiwgImFkbWluaXN0ZXJpbmdDZW50cmUiLCAiUHJheGlzIFNvbW1lcmdhcnRlbiJd~W
yJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImJhdGNoTnVtYmVyIiwgIjE2MjYzODI3M
zYiXQ~WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImhlYWx0aFByb2Zlc3Npb25hbC
IsICI4ODMxMTAwMDAwMTUzNzYiXQ~
```


The following payload is used for the SD-JWT:

```
{
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/vaccination/v1"
  ],
  "type": [
    "VerifiableCredential",
    "VaccinationCertificate"
  ],
  "issuer": "https://example.com/issuer",
  "issuanceDate": "2023-02-09T11:01:59Z",
  "expirationDate": "2028-02-08T11:01:59Z",
  "name": "COVID-19 Vaccination Certificate",
  "description": "COVID-19 Vaccination Certificate",
  "credentialSubject": {
    "_sd": [
      "1V_K-8lDQ8iFXBFXbZY9ehqR4HabWCi5T0ybIzZPeww",
      "JzjLgtP29dP-B3td12P674gFmK2zy81HMtBgf6CJNWg",
      "R2fGbfA07Z_YlkqmNZyma1xyyx1XstIiS6B1Ybl2JZ4",
      "TCmzrl7K2gev_du7pcMIyzRLHp-Yeg-Fl_cxtrUvPxg",
      "V7kJBLK78TmVDOmrfJ7ZuUPHuK_2cc7yZRa4qV1txwM",
      "b0eUsvGP-ODDdFoY4NlzlXc3tDslWJtCJF75Nw8Oj_g",
      "zJK_eSMXjwM8dXmMZLnI8FGM08zJ3_ubGeEMJ-5TBy0"
    ],
    "vaccine": {
      "_sd": [
        "1cF5hLwkhMNIaqfWJrXI7NMWedL-9f6Y2PA52yPjSZI",
        "Hiy6WWueLD5bn16298tPv7GXhmldMDOTnBi-CZbphNo",
        "Lb027q691jXXl-jC73vi8ebOj9smx3C-_og7gA4TBQE"
      ],
      "type": "Vaccine"
    },
    "recipient": {
      "_sd": [
        "1lSQBNY24q0Th6OGzthq-7-4l6cAaxrYXOGZpeW_lnA",
        "3nzLq81M2oN06wdv1shHvOEJVxZ5KLmdDkHEDJABWEI",
        "Pn1sWi06G4LJrnn-_RT0RbM_HTdxnPJQuX2fzWv_JOU",
        "lF9uzdsw7HplGLc714Tr4WO7MGJza7tt7QFleCX4Itw"
      ],
      "type": "VaccineRecipient"
    },
    "type": "VaccinationEvent"
  },
  "_sd_alg": "sha-256"
}
```

The following Disclosures are created by the Issuer:

Claim atcCode:

*SHA-256 Hash: 1cF5hLwkhMNIaqfWJrXI7NMWedL-9f6Y2PA52yPjSZI

*Disclosure:
WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImF0Y0NvZGUiLCAiSjA3Qlgw
MyJd

*Contents: ["2GLC42sKQveCfGfryNRN9w", "atcCode", "J07BX03"]

Claim medicinalProductName:

*SHA-256 Hash: Hiy6WWueLD5bn16298tPv7GXhmldMDOTnBi-CZbphNo

*Disclosure:
WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgIm1lZGljaW5hbFByb2R1Y3RO
YW1lIiwgIkNPVklELTE5IFZhY2NpbmUgTW9kZXJuYSJd

*Contents: ["eluV5Og3gSNII8EYnsxA_A", "medicinalProductName",
"COVID-19
Vaccine Moderna"]

Claim marketingAuthorizationHolder:

*SHA-256 Hash: Lb027q691jXXl-jC73vi8ebOj9smx3C-_og7gA4TBQE

*Disclosure:
WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgIm1hcmtldGluZ0F1dGhvcml6
YXRpb25Ib2xkZXIiLCAiTW9kZXJuYSBCaW90ZWNoIl0

*Contents: ["6Ij7tM-a5iVPGboS5tmvVA",
"marketingAuthorizationHolder",
"Moderna Biotech"]

Claim nextVaccinationDate:

*SHA-256 Hash: R2fGbfA07Z_YlkqmNZyma1xyyx1XstIiS6B1Ybl2JZ4

*Disclosure:
WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgIm5leHRWYWNjaW5hdGlvbkRh
dGUiLCAiMjAyMS0wOC0xNlQxMzo0MDoxMloiXQ

*Contents: ["eI8ZWm9QnKPpNPeNenHdhQ", "nextVaccinationDate",
"2021-08-16T13:40:12Z"]

Claim countryOfVaccination:

*SHA-256 Hash: JzjLgtP29dP-B3td12P674gFmK2zy81HMtBgf6CJNWg

*Disclosure:
WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImNvdW50cnlPZlZhY2NpbmF0aW9uIiwgIkdFIl0

*Contents: ["Qg_O64zqAxe412a108iroA", "countryOfVaccination", "GE"]

Claim dateOfVaccination:

*SHA-256 Hash: zJK_eSMXjwM8dXmMZLnI8FGM08zJ3_ubGeEMJ-5TBy0

*Disclosure:
WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImRhdGVPZlZhY2NpbmF0aW9uIiwgIjIwMjEtMDYtMjNUMTM6NDA6MTJaIl0

*Contents: ["AJx-095VPrpTtN4QMOqROA", "dateOfVaccination", "2021-06-23T13:40:12Z"]

Claim order:

*SHA-256 Hash: b0eUsvGP-ODDdFoY4NlzlXc3tDslWJtCJF75Nw8Oj_g

*Disclosure:
WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgIm9yZGVyIiwgIjMvMyJd

*Contents: ["Pc33JM2LchcU_lHggv_ufQ", "order", "3/3"]

Claim gender:

*SHA-256 Hash: 3nzLq81M2oN06wdv1shHvOEJVxZ5KLmdDkHEDJABWEI

*Disclosure:
WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImdlbmRlciIsICJGZW1hbGUiXQ

*Contents: ["G02NSrQfjFXQ7Io09syajA", "gender", "Female"]

Claim birthDate:

*SHA-256 Hash: Pn1sWi06G4LJrnn-_RT0RbM_HTdxnPJQuX2fzWv_JOU

*Disclosure:
WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImJpcnRoRGF0ZSIsICIxOTYxLTA4LTE3Il0

*Contents: ["lklxF5jMYlGTPUovMNIvCA", "birthDate", "1961-08-17"]

Claim givenName:

*SHA-256 Hash: lF9uzdsw7HplGLc714Tr4WO7MGJza7tt7QFleCX4Itw

*Disclosure:
WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgImdpdmVuTmFtZSIsICJNYXJp
b24iXQ

*Contents: ["nPuoQnkRFq3BIeAm7AnXFA", "givenName", "Marion"]

Claim familyName:

*SHA-256 Hash: 1lSQBNY24q0Th6OGzthq-7-4l6cAaxrYXOGZpeW_lnA

*Disclosure:
WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImZhbWlseU5hbWUiLCAiTXVz
dGVybWFubiJd

*Contents: ["5bPs1IquZNa0hkaFzzzZNw", "familyName", "Mustermann"]

Claim administeringCentre:

*SHA-256 Hash: TCmzrl7K2gev_du7pcMIyzRLHp-Yeg-Fl_cxtrUvPxg

*Disclosure:
WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13IiwgImFkbWluaXN0ZXJpbmdDZW50
cmUiLCAiUHJheGlzIFNvbW1lcmdhcnRlbiJd

*Contents: ["5a2W0_NrlEZzfqmk_7Pq-w", "administeringCentre",
"Praxis
Sommergarten"]

Claim batchNumber:

*SHA-256 Hash: V7kJBLK78TmVDOmrfJ7ZuUPHuK_2cc7yZRa4qV1txwM

*Disclosure:
WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImJhdGNoTnVtYmVyIiwgIjE2
MjYzODI3MzYiXQ

*Contents: ["y1sVU5wdfJahVdgwPgS7RQ", "batchNumber", "1626382736"]

Claim healthProfessional:

*SHA-256 Hash: 1V_K-8lDQ8iFXBFXbZY9ehqR4HabWCi5T0ybIzZPeww

*Disclosure:
WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImhlYWx0aFByb2Zlc3Npb25h
bCIsICI4ODMxMTAwMDAwMTUzNzYiXQ

*Contents: ["HbQ4X8srVW3QDxnIJdqyOA", "healthProfessional",
"883110000015376"]

The following is how a presentation of the SD-JWT with Key Binding JWT that discloses only type, medicinalProductName, atcCode of the vaccine, type of the recipient, type, order and dateOfVaccination could look like:


```
eyJhbGciOiAiRVMyNTYifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAiaWF0IjogMTY4MzAwMDAwMCwgImV4cCI6IDE4ODMwMDAwMDAsICJAY29udGV4d
CI6IFsiaHR0cHM6Ly93d3cudzMub3JnLzIwMTgvY3JlZGVudGlhbHMvdjEiLCAiaHR0c
HM6Ly93M2lkLm9yZy92YWNjaW5hdGlvbi92MSJdLCAidHlwZSI6IFsiVmVyaWZpYWJsZ
UNyZWRlbnRpYWwiLCAiVmFjY2luYXRpb25DZXJ0aWZpY2F0ZSJdLCAiaXNzdWVyIjogI
mh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlzc3VhbmNlRGF0ZSI6ICIyMDIzL
TAyLTA5VDExOjAxOjU5WiIsICJleHBpcmF0aW9uRGF0ZSI6ICIyMDI4LTAyLTA4VDExO
jAxOjU5WiIsICJuYW1lIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRlI
iwgImRlc2NyaXB0aW9uIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRlI
iwgImNyZWRlbnRpYWxTdWJqZWN0IjogeyJfc2QiOiBbIjFWX0stOGxEUThpRlhCRlhiW
lk5ZWhxUjRIYWJXQ2k1VDB5Ykl6WlBld3ciLCAiSnpqTGd0UDI5ZFAtQjN0ZDEyUDY3N
GdGbUsyenk4MUhNdEJnZjZDSk5XZyIsICJSMmZHYmZBMDdaX1lsa3FtTlp5bWExeHl5e
DFYc3RJaVM2QjFZYmwySlo0IiwgIlRDbXpybDdLMmdldl9kdTdwY01JeXpSTEhwLVllZ
y1GbF9jeHRyVXZQeGciLCAiVjdrSkJMSzc4VG1WRE9tcmZKN1p1VVBIdUtfMmNjN3laU
mE0cVYxdHh3TSIsICJiMGVVc3ZHUC1PRERkRm9ZNE5semxYYzN0RHNsV0p0Q0pGNzVOd
zhPal9nIiwgInpKS19lU01YandNOGRYbU1aTG5JOEZHTTA4ekozX3ViR2VFTUotNVRCe
TAiXSwgInZhY2NpbmUiOiB7Il9zZCI6IFsiMWNGNWhMd2toTU5JYXFmV0pyWEk3Tk1XZ
WRMLTlmNlkyUEE1MnlQalNaSSIsICJIaXk2V1d1ZUxENWJuMTYyOTh0UHY3R1hobWxkT
URPVG5CaS1DWmJwaE5vIiwgIkxiMDI3cTY5MWpYWGwtakM3M3ZpOGViT2o5c214M0MtX
29nN2dBNFRCUUUiXSwgInR5cGUiOiAiVmFjY2luZSJ9LCAicmVjaXBpZW50IjogeyJfc
2QiOiBbIjFsU1FCTlkyNHEwVGg2T0d6dGhxLTctNGw2Y0FheHJZWE9HWnBlV19sbkEiL
CAiM256THE4MU0yb04wNndkdjFzaEh2T0VKVnhaNUtMbWREa0hFREpBQldFSSIsICJQb
jFzV2kwNkc0TEpybm4tX1JUMFJiTV9IVGR4blBKUXVYMmZ6V3ZfSk9VIiwgImxGOXV6Z
HN3N0hwbEdMYzcxNFRyNFdPN01HSnphN3R0N1FGbGVDWDRJdHciXSwgInR5cGUiOiAiV
mFjY2luZVJlY2lwaWVudCJ9LCAidHlwZSI6ICJWYWNjaW5hdGlvbkV2ZW50In0sICJfc
2RfYWxnIjogInNoYS0yNTYifQ.5jJEqiRViN_DJ4VMxHQIN4KK-Cdfn30nY3nRe5jGxS
5Pths5G7mF1GWy9TawYuyFwCze7qiCIx_MhJ68Uu5zeQ~WyJQYzMzSk0yTGNoY1VfbEh
nZ3ZfdWZRIiwgIm9yZGVyIiwgIjMvMyJd~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIi
wgImRhdGVPZlZhY2NpbmF0aW9uIiwgIjIwMjEtMDYtMjNUMTM6NDA6MTJaIl0~WyIyR0
xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImF0Y0NvZGUiLCAiSjA3QlgwMyJd~WyJlbHVWN
U9nM2dTTklOEVZbnN4QV9BIiwgIm1lZGljaW5hbFByb2R1Y3ROYW1lIiwgIkNPVklEL
TE5IFZhY2NpbmUgTW9kZXJuYSJd~
```


After the validation, the Verifier will have the following data for further processing:

```
{
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/vaccination/v1"
  ],
  "type": [
    "VerifiableCredential",
    "VaccinationCertificate"
  ],
  "issuer": "https://example.com/issuer",
  "issuanceDate": "2023-02-09T11:01:59Z",
  "expirationDate": "2028-02-08T11:01:59Z",
  "name": "COVID-19 Vaccination Certificate",
  "description": "COVID-19 Vaccination Certificate",
  "credentialSubject": {
    "vaccine": {
      "type": "Vaccine",
      "atcCode": "J07BX03",
      "medicinalProductName": "COVID-19 Vaccine Moderna"
    },
    "recipient": {
      "type": "VaccineRecipient"
    },
    "type": "VaccinationEvent",
    "order": "3/3",
    "dateOfVaccination": "2021-06-23T13:40:12Z"
  },
  "_sd_alg": "sha-256"
}
```

A.5. Elliptic Curve Key Used in the Examples

The following Elliptic Curve public key, represented in JWK format, can be used to validate the Issuer signatures in the above examples:

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "b28d4MwZMjw8-00CG4xfnn9SLMVMM19SlqZpVb_uNtQ",
  "y": "Xv5zWwuoaTgdS6hV43yI6gBwTnjukmFQQnJ_kCxzqk8"
}
```

The public key used to validate a Key Binding JWT can be found in the examples as the content of the cnf claim.

Appendix B. Disclosure Format Considerations

As described in Section 5.2, the Disclosure structure is JSON containing salt and the cleartext content of a claim, which is base64url encoded. The encoded value is the input used to calculate a digest for the respective claim. The inclusion of digest value in the signed JWT ensures the integrity of the claim value. Using encoded content as the input to the integrity mechanism is conceptually similar to the approach in JWS and particularly useful when the content, like JSON, can have differences but be semantically equivalent. Some further discussion of the considerations around this design decision follows.

When receiving an SD-JWT, a Verifier must be able to re-compute digests of the disclosed claim values and, given the same input values, obtain the same digest values as signed by the Issuer.

Usually, JSON-based formats transport claim values as simple properties of a JSON object such as this:

```
...
  "family_name": "Möbius",
  "address": {
    "street_address": "Schulstr. 12",
    "locality": "Schulpforta"
  }
...
```

However, a problem arises when computation over the data need to be performed and verified, like signing or computing digests. Common signature schemes require the same byte string as input to the signature verification as was used for creating the signature. In the digest approach outlined above, the same problem exists: for the Issuer and the Verifier to arrive at the same digest, the same byte string must be hashed.

JSON, however, does not prescribe a unique encoding for data, but allows for variations in the encoded string. The data above, for example, can be encoded as

```
...
"family_name": "M\u00f6bius",
"address": {
  "street_address": "Schulstr. 12",
  "locality": "Schulpforta"
}
...
```

or as

```
...
"family_name": "Möbius",
"address": {"locality":"Schulpforta", "street_address":"Schulstr. 12"}
...
```

The two representations of the value in family_name are very different on the byte-level, but yield equivalent objects. Same for the representations of address, varying in white space and order of elements in the object.

The variations in white space, ordering of object properties, and encoding of Unicode characters are all allowed by the JSON specification, including further variations, e.g., concerning floating-point numbers, as described in [RFC8785]. Variations can be introduced whenever JSON data is serialized or deserialized and unless dealt with, will lead to different digests and the inability to verify signatures.

There are generally two approaches to deal with this problem:

1. Canonicalization: The data is transferred in JSON format, potentially introducing variations in its representation, but is transformed into a canonical form before computing a digest. Both the Issuer and the Verifier must use the same canonicalization algorithm to arrive at the same byte string for computing a digest.

2. Source string hardening: Instead of transferring data in a format that may introduce variations, a representation of the data is serialized. This representation is then used as the hashing input at the Verifier, but also transferred to the Verifier and used for the same digest calculation there. This means that the Verifier can easily compute and check the digest of the byte string before finally deserializing and accessing the data.

Mixed approaches are conceivable, i.e., transferring both the original JSON data plus a string suitable for computing a digest, but such approaches can easily lead to undetected inconsistencies resulting in time-of-check-time-of-use type security vulnerabilities.

In this specification, the source string hardening approach is used, as it allows for simple and reliable interoperability without the requirement for a canonicalization library. To harden the source string, any serialization format that supports the necessary data types could be used in theory, like protobuf, msgpack, or pickle. In this specification, JSON is used and plain text values of each Disclosure are encoded using base64url-encoding for transport. This

approach means that SD-JWTs can be implemented purely based on widely available JWT, JSON, and Base64 encoding and decoding libraries.

A Verifier can then easily check the digest over the source string before extracting the original JSON data. Variations in the encoding of the source string are implicitly tolerated by the Verifier, as the digest is computed over a predefined byte string and not over a JSON object.

It is important to note that the Disclosures are neither intended nor suitable for direct consumption by an application that needs to access the disclosed claim values after the verification by the Verifier. The Disclosures are only intended to be used by a Verifier to check the digests over the source strings and to extract the original JSON data. The original JSON data is then used by the application. See Section 6.3 for details.

Appendix C. Document History

[[To be removed from the final specification]]

-05

* Consolidate processing rules for Holder and Verifier
* Add support for selective disclosure of array elements.
* Consolidate SD-JWT terminology and format
* Use the term Key Binding rather than Holder Binding
* Defined the structure of the Key Binding JWT
* Added a JWS JSON Serialization
* Added initial IANA media type and structured suffix registration requests
* Added recommendation for explicit typing of SD-JWTs
* Added considerations around forwarding credentials
* Removed Example 2b and merged the demo of decoy digests into Example 2a
* Improved example for allowed variations in Disclosures
* Added some text to the Abstract and Introduction to be more inclusive of JWS with JSON

* Added some security considerations text about the scope of the Key Binding JWT

* Aligned examples structure and used the term input claim set

* Replaced the general SD-JWT VC example with one based on Person Identification Data (PID) from the European Digital Identity Wallet Architecture and Reference Framework

* Added/clarified some privacy considerations in Confidentiality during Transport

* No longer recommending a claim name for enveloped SD-JWTs

* Mention prospective future PQ algs for JWS

* Include the public key in the draft, which can be used to verify the issuer signature examples

* Clarify that _sd_alg can only be at the top level of the SD-JWT payload

* Externalized the SD-JWT library that generates examples

* Attempt to improve description of security properties

-04

* Improve description of processing of disclosures

-03

* Clarify that other specifications may define enveloping multiple Combined Formats for Presentation

* Add an example of W3C vc-data-model that uses a JSON-LD object as the claims set

* Clarify requirements for the combined formats for issuance and presentation

* Added overview of the Security Considerations section

* Enhanced examples in the Privacy Considerations section

* Allow for recursive disclosures

* Discussion on holder binding and privacy of stored credentials

* Add some context about SD-JWT being general-purpose despite being a product of the OAuth WG

* More explicitly say that SD-JWTs have to be signed asymmetrically (no MAC and no none)

* Make sha-256 the default hash algorithm, if the hash alg claim is omitted

* Use ES256 instead of RS256 in examples

* Rename and move the c14n challenges section to an appendix

* A bit more in security considerations for Choice of a Hash Algorithm (1st & 2nd preimage resistant and not majorly truncated)

* Remove the notational figures from the Concepts section

* Change salt to always be a string (rather than any JSON type)

* Fix the Document History (which had a premature list for -03)

-02

* Disclosures are now delivered not as a JWT but as separate base64url-encoded JSON objects.

* In the SD-JWT, digests are collected under a _sd claim per level.

* Terms "II-Disclosures" and "HS-Disclosures" are replaced with "Disclosures".

* Holder Binding is now separate from delivering the Disclosures and implemented, if required, with a separate JWT.

* Examples updated and modified to properly explain the specifics of the new SD-JWT format.

* Examples are now pulled in from the examples directory, not inlined.

* Updated and automated the W3C VC example.

* Added examples with multibyte characters to show that the specification and demo code work well with UTF-8.

* reverted back to hash alg from digest derivation alg (renamed to _sd_alg)

* reformatted

-01

* introduced blinded claim names

* explained why JSON-encoding of values is needed

* explained merging algorithm ("processing model")

* generalized hash alg to digest derivation alg which also enables HMAC to calculate digests

* _sd_hash_alg renamed to sd_digest_derivation_alg

* Salt/Value Container (SVC) renamed to Issuer-Issued Disclosures (II-Disclosures)

* SD-JWT-Release (SD-JWT-R) renamed to Holder-Selected Disclosures (HS-Disclosures)

* sd_disclosure in II-Disclosures renamed to sd_ii_disclosures

* sd_disclosure in HS-Disclosures renamed to sd_hs_disclosures

* clarified relationship between sd_hs_disclosure and SD-JWT

* clarified combined formats for issuance and presentation

* clarified security requirements for blinded claim names

* improved description of Holder Binding security considerations - especially around the usage of "alg=none".

* updated examples

* text clarifications

* fixed cnf structure in examples

* added feature summary

-00

* Upload as draft-ietf-oauth-selective-disclosure-jwt-00

[[pre Working Group Adoption:]]

-02

* Added acknowledgements

* Improved Security Considerations

*Stressed entropy requirements for salts

*Python reference implementation clean-up and refactoring

*hash_alg renamed to _sd_hash_alg

-01

*Editorial fixes

*Added hash_alg claim

*Renamed _sd to sd_digests and sd_release

*Added descriptions on Holder Binding - more work to do

*Clarify that signing the SD-JWT is mandatory

-00

*Renamed to SD-JWT (focus on JWT instead of JWS since signature is optional)

*Make Holder Binding optional

*Rename proof to release, since when there is no signature, the term "proof" can be misleading

*Improved the structure of the description

*Described verification steps

*All examples generated from python demo implementation

*Examples for structured objects

Authors' Addresses

Daniel Fett
yes.com

Email: mail@danielfett.de
URI: https://danielfett.de/

Kristina Yasuda
Microsoft

Email: Kristina.Yasuda@microsoft.com

Brian Campbell
Ping Identity

Email: bcampbell@pingidentity.com