Stream: Internet Engineering Task Force (IETF)
RFC: 9901
Category: Standards Track
Published: November 2025
ISSN: 2070-1721
Authors: D. Fett, *Authlete*; K. Yasuda, *Keio University*; B. Campbell, *Ping Identity*

RFC 9901
Selective Disclosure for JSON Web Tokens

Abstract

This specification defines a mechanism for the selective disclosure of individual elements of a JSON data structure used as the payload of a JSON Web Signature (JWS). The primary use case is the selective disclosure of JSON Web Token (JWT) claims.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9901.

Copyright Notice

Table of Contents

1. Introduction

The exchange of JSON data between systems is often secured against modification using JSON Web Signatures (JWSs) [RFC7515]. A popular application of JWS is the JSON Web Token (JWT) [RFC7519], a format that is often used to represent a user's identity. An ID Token as defined in OpenID Connect [OpenID.Core], for example, is a JWT containing the user's claims created by the server for consumption by a relying party. In cases where the JWT is sent immediately from the server to the relying party, as in OpenID Connect, the server can select at the time of issuance which user claims to include in the JWT, minimizing the information shared with the relying party who validates the JWT.

Another model is emerging that fully decouples the issuance of a JWT from its presentation. In this model, a JWT containing many claims is issued to an intermediate party, who holds the JWT (the Holder). The Holder can then present the JWT to different verifying parties (Verifiers) that

each may only require a subset of the claims in the JWT. For example, the JWT may contain claims representing both an address and a birthdate. The Holder may elect to disclose only the address to one Verifier, and only the birthdate to a different Verifier.

Privacy principles of minimal disclosure in conjunction with this model demand a mechanism enabling selective disclosure of data elements while ensuring that Verifiers can still check the authenticity of the data provided. This specification defines such a mechanism for JSON payloads of JWSs, with JWTs as the primary use case.

Selectively Disclosable JWT (SD-JWT) is based on an approach called "salted hashes": For any data element that should be selectively disclosable, the Issuer of the SD-JWT does not include the cleartext of the data in the JSON payload of the JWS structure; instead, a digest of the data takes its place. For presentation to a Verifier, the Holder sends the signed payload along with the cleartext of those claims it wants to disclose. The Verifier can then compute the digest of the cleartext data and confirm it is included in the signed payload. To ensure that Verifiers cannot guess cleartext values of non-disclosed data elements, an additional salt value is used when creating the digest and sent along with the cleartext when disclosing it.

To prevent attacks in which an SD-JWT is presented to a Verifier without the Holder's consent, this specification additionally defines a mechanism for binding the SD-JWT to a key under the control of the Holder (Key Binding). When Key Binding is enforced, a Holder has to prove possession of a private key belonging to a public key contained in the SD-JWT itself. It usually does so by signing over a data structure containing transaction-specific data, herein defined as the Key Binding JWT. An SD-JWT with a Key Binding JWT is called "SD-JWT+KB" in this specification.

1.1. Feature Summary

This specification defines two primary data formats:

1. SD-JWT is a composite structure, consisting of a JWS plus optional Disclosures, enabling selective disclosure of portions of the JWS payload. It comprises the following:
   - A format for enabling selective disclosure in nested JSON data structures, supporting selectively disclosable object properties (name/value pairs) and array elements.
   - A format for encoding the selectively disclosable data items.
   - A format extending the JWS Compact Serialization, allowing for the combined transport of the Issuer-signed JSON data structure and the disclosable data items.
   - An alternate format extending the JWS JSON Serialization, also allowing for transport of the Issuer-signed JSON data structure and Disclosure data.
2. SD-JWT+KB is a composite structure of an SD-JWT and a cryptographic Key Binding that can be presented to and verified by the Verifier. It comprises the following:
   - A mechanism for associating an SD-JWT with a key pair.
   - A format for a Key Binding JWT (KB-JWT) that allows proof of possession of the private key of the associated key pair.

- A format extending the SD-JWT format for the combined transport of the SD-JWT and the KB-JWT.

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].

Claim:
: In this document, refers generally to object properties (name/value pairs) as well as array elements.

Selective Disclosure:
: Process of a Holder disclosing to a Verifier a subset of claims contained in a JWT issued by an Issuer.

Selectively Disclosable JWT (SD-JWT):
: A composite structure, consisting of an Issuer-signed JWT (JWS; see [RFC7515]) and zero or more Disclosures, which supports selective disclosure as defined in this document. It can contain both regular claims and digests of selectively disclosable claims.

Disclosure:
: A base64url-encoded string of a JSON array that contains a salt, a claim name (present when the claim is a name/value pair and absent when the claim is an array element), and a claim value. The Disclosure is used to calculate a digest for the respective claim. The term Disclosure refers to the whole base64url-encoded string.

Key Binding:
: Ability of the Holder to prove possession of an SD-JWT by proving control over a private key during the presentation. When utilizing Key Binding, an SD-JWT contains the public key corresponding to the private key controlled by the Holder (or a reference to this public key).

Key Binding JWT (KB-JWT):
: A Key Binding JWT is said to "be tied to" a particular SD-JWT when its payload is signed using the key included in the SD-JWT payload, and the KB-JWT contains a hash of the SD-JWT in its `sd_hash` claim. Its format is defined in Section 4.3.

Selectively Disclosable JWT with Key Binding (SD-JWT+KB):
: A composite structure, comprising an SD-JWT and a Key Binding JWT tied to that SD-JWT.

Processed SD-JWT Payload:
: The JSON object resulting from verification and processing of the Issuer-signed SD-JWT, with digest placeholders replaced by the corresponding values from the Disclosures.

Issuer:
: An entity that creates SD-JWTs.

Holder:
: An entity that received SD-JWTs from the Issuer and has control over them. In the context of this document, the term may refer to the actual user, the supporting hardware and software in their possession, or both.

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

2. Flow Diagram

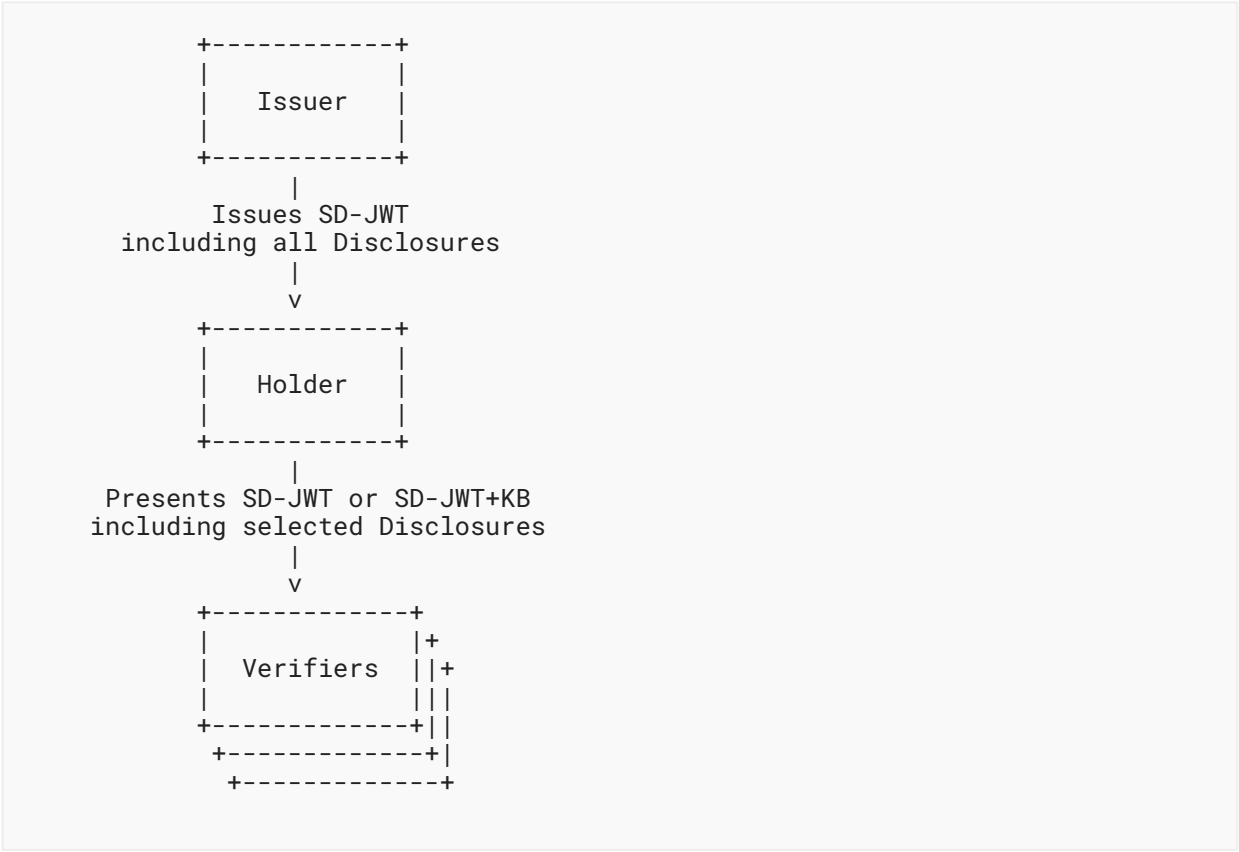

Figure 1: SD-JWT Issuance and Presentation Flow

3. 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 4.

3.1. 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 modifications to them can be detected. Selectively disclosable claims can be individual object properties (name/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 cryptographically secure random salt, the claim name (only when the claim is an object property), and the claim value. The Disclosures are sent to the Holder with the SD-JWT in the format defined in Section 4. When presenting an SD-JWT to a Verifier, the Holder only includes the Disclosures for the claims that it wants to reveal to that Verifier.

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

3.2. 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.

3.3. 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 described in Section 4.1.2.

When a Verifier requires Key Binding, the Holder presents an SD-JWT+KB, consisting of an SD-JWT as well as a Key Binding JWT tied to that SD-JWT. The Key Binding JWT encodes a signature by the Holder's private key over

- a hash of the SD-JWT,
- a nonce to ensure the freshness of the signature, and
- an audience value to indicate the intended Verifier for the document.

Details of the format of Key Binding JWTs are described in Section 4.3.

3.4. Verification

At a high level, the Verifier

- receives either an SD-JWT or an SD-JWT+KB from the Holder,
- verifies the signature on the SD-JWT (or the SD-JWT inside the SD-JWT+KB) using the Issuer's public key,

- verifies the signature on the KB-JWT using the public key included (or referenced) in the SD-JWT, if the Verifier's policy requires Key Binding, and
- 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 7.3.

4. SD-JWT and SD-JWT+KB Data Formats

An SD-JWT is composed of

- an Issuer-signed JWT, and
- zero or more Disclosures.

An SD-JWT+KB is composed of

- an SD-JWT (i.e., an Issuer-signed JWT and zero or more Disclosures), and
- a Key Binding JWT.

The Issuer-signed JWT, Disclosures, and Key Binding JWT are explained in Sections 4.1, 4.2, and 4.3, respectively.

The compact serialized format for the SD-JWT is the concatenation of each part delineated with a single tilde ('~') character as follows, where "D.1" to "D.N" represent the respective Disclosures:

```
<Issuer-signed JWT>~<D.1>~<D.2>~...~<D.N>~
```

The order of the concatenated parts **MUST** be the Issuer-signed JWT, a tilde character, zero or more Disclosures each followed by a tilde character, 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 serialized format for an SD-JWT+KB extends the SD-JWT format by concatenating a Key Binding JWT.

```
<Issuer-signed JWT>~<D.1>~<D.2>~...~<D.N>~<KB-JWT>
```

The two formats can be distinguished by the final `~` character that is present on an SD-JWT. A Verifier that expects an SD-JWT **MUST** verify that the final tilde-separated component is empty. A Verifier that expects an SD-JWT+KB **MUST** verify that its final tilde-separated component is a valid KB-JWT.

The Disclosures are linked to the Issuer-signed JWT through the digest values included therein.

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

When presenting to a Verifier, the Holder sends only the selected set of the Disclosures in the SD-JWT.

The Holder **MAY** send any subset of the Disclosures to the Verifier, i.e., none, some, 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 issued SD-JWT or send a Disclosure more than once.

To further illustrate the SD-JWT format, the following examples show a few different SD-JWT permutations, both with and without various constituent parts.

An SD-JWT without Disclosures:

```
<Issuer-signed JWT>~
```

An SD-JWT with Disclosures:

```
<Issuer-signed JWT>~<Disclosure 1>~<Disclosure N>~
```

An SD-JWT+KB without Disclosures:

```
<Issuer-signed JWT>~<KB-JWT>
```

An SD-JWT+KB with Disclosures:

```
<Issuer-signed JWT>~<Disclosure 1>~<Disclosure N>~<KB-JWT>
```

As an alternative illustration of the SD-JWT format, ABNF [RFC5234] for the SD-JWT, SD-JWT+KB, and various constituent parts is provided here (for those who celebrate):

```
ALPHA = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT = %x30-39 ; 0-9
BASE64URL = 1*(ALPHA / DIGIT / "-" / "_")
JWT = BASE64URL "." BASE64URL "." BASE64URL
DISCLOSURE = BASE64URL
SD-JWT = JWT "~" *(DISCLOSURE "~")
KB-JWT = JWT
SD-JWT-KB = SD-JWT KB-JWT
```

4.1. Issuer-Signed JWT

An SD-JWT has a JWT component that **MUST** be signed using the Issuer's private key. It **MUST NOT** use the none algorithm.

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 4.1.1.
2. The payload **MAY** contain one or more digests of Disclosures to enable selective disclosure of the respective claims, created and formatted as described in Section 4.2.
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 4.2.5.
4. The payload **MAY** contain one or more permanently disclosed claims.
5. The payload **MAY** contain the Holder's public key(s) or reference(s) thereto, as explained in Section 4.1.2.
6. The payload **MAY** contain further claims such as `iss`, `iat`, etc. as defined or required by the application using SD-JWTs.
7. The payload **MUST NOT** contain the claims `_sd` or `...` except for the purpose of conveying digests as described in Sections 4.2.4.1 and 4.2.4.2, respectively.

The same digest value **MUST NOT** appear more than once in the SD-JWT.

Application and profiles of SD-JWT **SHOULD** be explicitly typed. See Section 9.11 for more details.

It is the Issuer who decides which claims are selectively disclosable by the Holder and which are not. 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. See Section 9.7 for considerations on making validity-controlling claims such as `exp` selectively disclosable.

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

4.1.1. Hash Function Claim

The claim `_sd_alg` indicates the hash algorithm used by the Issuer to generate the digests as described in Section 4.2. 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.

This claim value is a case-sensitive string with the hash algorithm identifier. The hash algorithm identifier **MUST** be a hash algorithm value from the "Hash Name String" column in the "Named Information Hash Algorithm Registry" [Hash.Algs] 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.

4.1.2. Key Binding

If the Issuer wants to enable Key Binding, it includes a public key associated with the Holder, or a reference thereto, using the `cnf` claim as defined in [RFC7800]. The `jwk` confirmation method, as defined in Section 3.2 of [RFC7800], is suggested for doing so, however, other confirmation methods can be used.

> Note that, as was stated in [RFC7800], if an application needs to represent multiple proof-of-possession keys in the same SD-JWT, one way to achieve this is to use other claim names, in addition to `cnf`, to hold the additional proof-of-possession key information.

It is outside the scope of this document to describe how the Holder key pair is established. For example, the Holder **MAY** create a key pair and provide a public key 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 throughout this document use the `cnf` claim with the `jwk` member to include the raw public key by value in SD-JWT.

4.2. Disclosures

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

- For a claim that is an object property, the Issuer creates a Disclosure as described in Section 4.2.1.
- For a claim that is an array element, the Issuer creates a Disclosure as described in Section 4.2.2.

4.2.1. 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 a JSON array of three elements in the following order:
  1. A salt value. **MUST** be a string. See Section 9.3 for security considerations. To achieve the recommended entropy of the salt, the Issuer can base64url-encode 128 bits of cryptographically secure random data, producing a string. The salt value **MUST** be unique for each claim that is to be selectively disclosed. The Issuer **MUST NOT** reveal 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 payload. It **MUST** be a string and **MUST NOT** be `_sd`, `...`, or a claim name existing in the object as a permanently disclosed claim.

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

- base64url-encode the UTF-8 byte sequence of the JSON array. This string is the Disclosure.

> Note: The order was decided based on readability considerations: Salts 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 resultant Disclosure is:

```
WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsICJmYW1pbHlfbmFtZSIsICJNw7ZiaXVzIl0
```

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

- A different way to encode the unicode umlaut:

  ```
  WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsICJmYW1pbHlfbmFtZSIsICJNXHUwMGY2Yml1cyJd
  ```

- No white space:

  ```
  WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsImZhbWlseV9uYW1lIiwiTcO2Yml1cyJd
  ```

- Newline characters between elements:

  ```
  WwoiXzI2YmM0TFQtYWM2cTJLSTZjQlc1ZXMiLAoiZmFtaWx5X25hbWUiLAoiTcO2Yml1cyIKXQ
  ```

However, the digest is calculated over the respective base64url-encoded value itself, which effectively signs the variation chosen by the Issuer and makes it immutable in the context of the particular SD-JWT.

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

4.2.2. 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 4.2.1.
  2. The array element that is to be hidden. This value can be of any type that is allowed in JSON, including numbers, strings, booleans, arrays, and objects.

The Disclosure string is created by base64url-encoding the UTF-8 byte sequence of the resultant JSON array as described in Section 4.2.1. 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 JWT Claims Set:

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

could be created by first creating the following array:

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

The resultant Disclosure would be:

`WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIkZSIl0`

> Note that the size of an array alone can potentially reveal unintended information. The use of decoys, as described in Section 4.2.5, to consistently pad the size of an array can help obscure the actual number of elements present in any particular instance.

4.2.3. Hashing Disclosures

For embedding references to the Disclosures in the SD-JWT, each Disclosure is hashed using the hash algorithm specified in the `_sd_alg` claim described in Section 4.1.1, or SHA-256 if no algorithm is specified. The resultant digest is then included in the SD-JWT payload instead of the original claim value, as described next.

The digest **MUST** be computed over the US-ASCII bytes of the base64url-encoded value that is the 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 (sometimes used) hex representation of the bytes of the digest.

For example, the base64url-encoded SHA-256 digest of the Disclosure `WyJfMjZiYzRMVC1hYzZxMktJNmNCVzVlcyIsICJmYW1pbHlfbmFtZSIsICJNw7ZiaXVzIl0` for the `family_name` claim from Section 4.2.1 above is `X9yH0Ajrdm1Oij4tWso9UzzKJvPoDxwmuEcO3XAdRC0`.

4.2.4. Embedding Disclosure Digests in SD-JWTs

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

- For a claim that is an object property, the Issuer embeds a Disclosure digest as described in Section 4.2.4.1.
- For a claim that is an array element, the Issuer creates a Disclosure digest as described in Section 4.2.4.2.

4.2.4.1. 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 4.2.5. An `_sd` key can be present at any level of the JSON object hierarchy, including at the top-level, nested deeper as described in Section 6, or in recursive Disclosures as described in Section 4.2.6.

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 4.2.5. 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 Disclosure from Section 4.2.3, the Issuer could create the following SD-JWT payload to make `family_name` selectively disclosable:

```
{
  "given_name": "Alice",
  "_sd": ["X9yH0Ajrdm1Oij4tWso9UzzKJvPoDxwmuEcO3XAdRC0"]
}
```

4.2.4.2. 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 4.2.3. There **MUST NOT** be any other keys in the object. Note that the string `...` was chosen because the ellipsis character, typically entered as three period characters, is commonly used in places where content is omitted from the present context.

For example, using the digest of the array element Disclosure created in Section 4.2.2, the Issuer could create the following SD-JWT payload to make the second element of the `nationalities` array selectively disclosable:

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

As described in Section 7.3, 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 two elements, `["DE", "US"]`, unless the matching Disclosure for the second element is received, in which case the output would be a three-element array, `["DE", "FR", "US"]`.

4.2.5. 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 adversarial Verifier to see the original number of claims or array elements 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 10.4 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.

4.2.6. Recursive Disclosures

The algorithms above are compatible with "recursive Disclosures", in which one selectively disclosed field reveals the existence of more selectively disclosable fields. For example, consider the following JSON structure:

```
{
    "family_name": "Möbius",
    "nationalities": ["DE", "FR", "UK"]
}
```

When the Holder has multiple nationalities, the Issuer may wish to conceal the presence of any statement regarding nationalities while also allowing the Holder to reveal each of those nationalities individually. This can be accomplished by first making the entries within the "nationalities" array selectively disclosable, and then making the whole "nationalities" field selectively disclosable.

The following shows each of the entries within the "nationalities" array being made selectively disclosable:

```
{
    "family_name": "Möbius",
    "nationalities": [
        { "...": "PmnlrRjhLcwf8zTDdK15HVGwHtPYjddvD362WjBLwro" }
        { "...": "r823HFN6Ba_lpSANYtXqqCBAH-TsQlIzfOK0lRAFLCM" },
        { "...": "nP5GYjwhFm6ESlAeC4NCaIliW4tz0hTrUeoJB3lb5TA" }
    ]
}
```

Content of Disclosures:

```
PmnlrRj... = ["16_mAd0GiwaZokU26_0i0h","DE"]
r823HFN... = ["fn9fN0rD-fFs2n303ZI-0c","FR"]
nP5GYjw... = ["YIKesqOkXXNzMQtsX_-_lw","UK"]
```

Followed by making the whole "nationalities" array selectively disclosable:

```
{
    "family_name": "Möbius",
    "_sd": [ "5G1srw3RG5W4pVTwSsYxeOWosRBbzd18ZoWKkC-hBL4" ]
}
```

Content of Disclosures:

```
PmnlrRj... = ["16_mAd0GiwaZokU26_0i0h","DE"]
r823HFN... = ["fn9fN0rD-fFs2n303ZI-0c","FR"]
nP5GYjw... = ["YIKesqOkXXNzMQtsX_-_lw","UK"]
5G1srw3... = ["4drfeTtSUK3aY_-PF12gcX","nationalities",
    [
        { "...": "PmnlrRjhLcwf8zTDdK15HVGwHtPYjddvD362WjBLwro" },
        { "...": "r823HFN6Ba_lpSANYtXqqCBAH-TsQlIzfOK0lRAFLCM" },
        { "...": "nP5GYjwhFm6ESlAeC4NCaIliW4tz0hTrUeoJB3lb5TA" }
    ]
]
```

With this set of Disclosures, the Holder could include the Disclosure with hash `PmnlrRj...` to disclose only the "DE" nationality, or include both `PmnlrRj...` and `r823HFN...` to disclose both the "DE" and "FR" nationalities, but hide the "UK" nationality. In either case, the Holder would also need to include the Disclosure with hash `5G1srw3...` to disclose the `nationalities` field that contains the respective elements.

Note that making recursive redactions introduces dependencies between the Disclosure objects in an SD-JWT. The `r823HFN...` Disclosure cannot be used without the `5G1srw3...` Disclosure; since a Verifier would not have a matching hash that would tell it where the content of the `r823HFN...` Disclosure should be inserted. If a Disclosure object is included in an SD-JWT, then the SD-JWT **MUST** include any other Disclosure objects necessary to process the first Disclosure object. In other words, any Disclosure object in an SD-JWT must "connect" to the claims in the issuer-signed JWT, possibly via an intermediate Disclosure object. In the above example, it would be illegal to include any one of the `PmnlrRj...`, `r823HFN...`, `nP5GYjw...` Disclosure objects without also including the `5G1srw3...` Disclosure object.

4.3. Key Binding JWT

This section defines the Key Binding JWT, which encodes a signature over an SD-JWT by the Holder's private key.

The Key Binding JWT **MUST** be a JWT according to [RFC7519], and it **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 the IANA "JSON Web Signature and Encryption Algorithms" registry. It **MUST NOT** be "none".
- in the JWT payload,
  - `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 value **MUST** be a single string that identifies the intended receiver of the Key Binding JWT. How the value is represented is up to the protocol used and is out of scope for this specification.
- "nonce": **REQUIRED**. Ensures the freshness of the signature or its binding to the given transaction. The value type of this claim **MUST** be a string. How this value is obtained is up to the protocol used and is out of scope for this specification.
- `sd_hash`: **REQUIRED**. The base64url-encoded hash value over the Issuer-signed JWT and the selected Disclosures as defined below.

The general extensibility model of JWT means that additional claims and header parameters can be added to the Key Binding JWT. However, unless there is a compelling reason, this **SHOULD** be avoided, as it may harm interoperability and burden conceptual integrity.

4.3.1. Binding to an SD-JWT

The hash value in the `sd_hash` claim binds the KB-JWT to the specific SD-JWT. The `sd_hash` value **MUST** be computed over the US-ASCII bytes of the encoded SD-JWT, i.e., the Issuer-signed JWT, a tilde character, and zero or more Disclosures selected for presentation to the Verifier, each followed by a tilde character:

```
<Issuer-signed JWT>~<Disclosure 1>~<Disclosure 2>~...~<Disclosure N>~
```

The bytes of the digest **MUST** then be base64url encoded.

The same hash algorithm as for the Disclosures **MUST** be used (defined by the `_sd_alg` element in the Issuer-signed JWT or the default value, as defined in Section 4.1.1).

4.3.2. Validating the Key Binding JWT

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. See Section 9.5 for security considerations.

If the Verifier requires Key Binding, the Verifier **MUST** ensure that the key with which it validates the signature on the Key Binding JWT is the key specified in the SD-JWT as the Holder's public key. For example, if the SD-JWT contains a `cnf` value with a `jwk` member, the Verifier would parse the provided JWK and use it to verify the Key Binding JWT.

Details of the validation process are defined in Section 7.3.

5. Example SD-JWT

In this example, a simple SD-JWT is demonstrated. This example is split into issuance and presentation.

> Note: Throughout the examples in this document, line breaks were added to JSON strings and base64-encoded strings to adhere to the line-length limit in RFCs and for readability. JSON does not allow line breaks within strings.

5.1. Issuance

The following data about the user comprises the input JWT Claims Set used by the Issuer:

```
{
  "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"
  ]
}
```

In this example, the following decisions were made by the Issuer in constructing the SD-JWT:

- The `nationalities` array is always visible, but its contents are selectively disclosable.
- The `sub` element as well as essential verification data (`iss`, `exp`, `cnf`, etc.) are always visible.
- All other claims are selectively disclosable.
- For `address`, the Issuer is using a flat structure, i.e., all the claims in the `address` claim can only be disclosed in full. Other options are discussed in Section 6.

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://issuer.example.com",
  "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 respective Disclosures, created by the Issuer, are listed below. In the text below and in other locations in this specification, the label "SHA-256 Hash:" is used as a shorthand for the label "Base64url-Encoded SHA-256 Hash:".

- Claim `given_name`:
  - SHA-256 Hash:

    `jsu9yVulwQQlhFlM_3JlzMaSFzglhQG0DpfayQwLUK4`
  - Disclosure:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9obiJd`
  - Contents:

    `["2GLC42sKQveCfGfryNRN9w", "given_name", "John"]`
- Claim `family_name`:
  - SHA-256 Hash:

`TGf4oLbgwd5JQaHyKVQZU9UdGE0w5rtDsrZzfUaomLo`

- Disclosure:

`WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRvZSJd`

- Contents:

`["eluV5Og3gSNII8EYnsxA_A", "family_name", "Doe"]`

- Claim `email`:
  - SHA-256 Hash:

`JzYjH4svliH0R3PyEMfeZu6Jt69u5qehZo7F7EPYlSE`

  - Disclosure:

`WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImVtYWlsIiwgImpvaG5kb2VAZXhhbXBsZS5jb20iXQ`

  - Contents:

`["6Ij7tM-a5iVPGboS5tmvVA", "email", "johndoe@example.com"]`

- Claim `phone_number`:
  - SHA-256 Hash:

`PorFbpKuVu6xymJagvkFsFXAbRoc2JGlAUA2BA4o7cI`

  - Disclosure:

`WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgInBob25lX251bWJlciIsICIrMS0yMDItNTU1LTAxMDEiXQ`

  - Contents:

`["eI8ZWm9QnKPpNPeNenHdhQ", "phone_number", "+1-202-555-0101"]`

- Claim `phone_number_verified`:
  - SHA-256 Hash:

`XQ_3kPKt1XyX7KANkqVR6yZ2Va5NrPIvPYbyMvRKBMM`

  - Disclosure:

`WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bWJlcl92ZXJpZmllZCIsIHRydWVd`

  - Contents:

`["Qg_O64zqAxe412a108iroA", "phone_number_verified", true]`

- Claim `address`:
  - SHA-256 Hash:

`XzFrzwscM6Gn6CJDc6vVK8BkMnfG8vOSKfpPIZdAfdE`

  - Disclosure:

WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjog
IjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iLCAicmVnaW9uIjogIkFueXN0YXRl
IiwgImNvdW50cnkiOiAiVVMifV0

- Contents:

["AJx-095VPrpTtN4QMOqROA", "address", {"street_address": "123 Main St",
"locality": "Anytown", "region": "Anystate", "country": "US"}]

- Claim `birthdate`:
  - SHA-256 Hash:

gbOsI4Edq2x2Kw-w5wPEzakob9hV1cRD0ATN3oQL9JM

  - Disclosure:

WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImJpcnRoZGF0ZSIsICIxOTQwLTAxLTAxIl0

  - Contents:

["Pc33JM2LchcU_lHggv_ufQ", "birthdate", "1940-01-01"]

- Claim `updated_at`:
  - SHA-256 Hash:

CrQe7S5kqBAHt-nMYXgc6bdt2SH5aTY1sU_M-PgkjPI

  - Disclosure:

WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInVwZGF0ZWRfYXQiLCAxNTcwMDAwMDAwXQ

  - 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 the following Issuer-signed JWT:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJfc2QiOiBb
IkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkdDJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZ
akg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5dTVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBL
dVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHbEFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1
SlFhSHlLVlFaVTlVZEdFMHc1cnREc3JaemZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tB
TmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS0JNTSIsICJYekZyendzY002R242Q0pEYzZ2
Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFIiwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFr
b2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAianN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpn
bGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzIjogImh0dHBzOi8vaXNzdWVyLmV4YW1wbGUu
Y29tIiwgImlhdCI6IDE2ODMwMDAwMDAsICJleHAiOiAxODgzMDAwMDAwLCAic3ViIjog
InVzZXJfNDIiLCAibmF0aW9uYWxpdGllcyI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15
VGE2VWpsWm8zZGgta284YUlLUWM5RGxHemhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1
ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlclAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjog
InNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0y
NTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VH
ZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlG
MkhaUSJ9fX0.MczwjBFGtzf-6WMT-hIvYbkb11NrV1WMO-jTijpMPNbswNzZ87wY2uHz
-CXo6R04b7jYrpj9mNRAvVssXou1iw
```

Adding the Disclosures produces the SD-JWT:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJfc2QiOiBb
IkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkdDJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZ
akg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5dTVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBL
dVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHbEFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1
SlFhSHlLVlFaVTlVZEdFMHc1cnREc3JaemZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tB
TmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS0JNTSIsICJYekZyendzY002R242Q0pEYzZ2
Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFIiwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFr
b2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAianN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpn
bGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzIjogImh0dHBzOi8vaXNzdWVyLmV4YW1wbGUu
Y29tIiwgImlhdCI6IDE2ODMwMDAwMDAsICJleHAiOiAxODgzMDAwMDAwLCAic3ViIjog
InVzZXJfNDIiLCAibmF0aW9uYWxpdGllcyI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15
VGE2VWpsWm8zZGgta284YUlLUWM5RGxHemhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1
ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlclAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjog
InNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0y
NTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VH
ZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlG
MkhaUSJ9fX0.MczwjBFGtzf-6WMT-hIvYbkb11NrV1WMO-jTijpMPNbswNzZ87wY2uHz
-CXo6R04b7jYrpj9mNRAvVssXou1iw~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgI
mdpdmVuX25hbWUiLCAiSm9obiJd~WyJlbHVWNU9nM2dTTklOEVZbnN4QV9BIiwgImZh
bWlseV9uYW1lIiwgIkRvZSJd~WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImVtYWl
sIiwgImpvaG5kb2VAZXhhbXBsZS5jb20iXQ~WyJlSThaV205UW5LUHBOUGVOZW5IZGhR
IiwgInBob25lX251bWJlciIsICIrMS0yMDItNTU1LTAxMDEiXQ~WyJRZ19PNjR6cUF4Z
TQxMmExMDhpcm9BIiwgInBob25lX251bWJlcl92ZXJpZmllZCIsIHRydWVd~WyJBSngt
MDk1VlBycFR0TjRRTU9xUk9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjog
IjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iLCAicmVnaW9uIjogIkFu
eXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0~WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZR
IiwgImJpcnRoZGF0ZSIsICIxOTQwLTAxLTAxIl0~WyJHMDJOU3JRZmpGWFE3SW8wOXN5
YWpBIiwgInVwZGF0ZWRfYXQiLCAxNTcwMDAwMDAwXQ~WyJsa2x4RjVqTVlsR1RQVW92T
U5JdkNBIiwgIlVTIl0~WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgIkRFIl0~
```

5.2. Presentation

The following non-normative example shows an SD-JWT+KB as it would be sent from the Holder to the Verifier. Note that it consists of six tilde-separated parts, with the Issuer-signed JWT as shown above in the beginning, four Disclosures (for the claims `given_name`, `family_name`, `address`, and one of the `nationalities`) in the middle, and the Key Binding JWT as the last element.

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJfc2QiOiBb
IkNyUWU3UzVrcUJBSHQtbk1ZWGdjNmJkdDJTSDVhVFkxc1VfTS1QZ2tqUEkiLCAiSnpZ
akg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5dTVxZWhabzdGN0VQWWxTRSIsICJQb3JGYnBL
dVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHbEFVQTJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1
SlFhSHlLVlFaVTlVZEdFMHc1cnREc3JaemZVYW9tTG8iLCAiWFFfM2tQS3QxWHlYN0tB
TmtxVlI2eVoyVmE1TnJQSXZQWWJ5TXZSS0JNTSIsICJYekZyendzY002R242Q0pEYzZ2
Vks4QmtNbmZHOHZPU0tmcFBJWmRBZmRFIiwgImdiT3NJNEVkcTJ4Mkt3LXc1d1BFemFr
b2I5aFYxY1JEMEFUTjNvUUw5Sk0iLCAianN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpn
bGhRRzBEcGZheVF3TFVLNCJdLCAiaXNzIjogImh0dHBzOi8vaXNzdWVyLmV4YW1wbGUu
Y29tIiwgImlhdCI6IDE2ODMwMDAwMDAsICJleHAiOiAxODgzMDAwMDAwLCAic3ViIjog
InVzZXJfNDIiLCAibmF0aW9uYWxpdGllcyI6IFt7Ii4uLiI6ICJwRm5kamtaX1ZDem15
VGE2VWpsWm8zZGgta284YUlLUWM5RGxHemhhVllvIn0sIHsiLi4uIjogIjdDZjZKa1B1
ZHJ5M2xjYndIZ2VaOGtoQXYxVTFPU2xlclAwVmtCSnJXWjAifV0sICJfc2RfYWxnIjog
InNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0y
NTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VH
ZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlG
MkhaUSJ9fX0.MczwjBFGtzf-6WMT-hIvYbkb11NrV1WMO-jTijpMPNbswNzZ87wY2uHz
-CXo6R04b7jYrpj9mNRAvVssXou1iw~WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgI
mZhbWlseV9uYW1lIiwgIkRvZSJd~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImFk
ZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5
IjogIkFueXRvd24iLCAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMi
fV0~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9obiJd
~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIlVTIl0~eyJhbGciOiAiRVMyNTYiLCA
idHlwIjogImtiK2p3dCJ9.eyJub25jZSI6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodH
RwczovL3ZlcmlmaWVyLmV4YW1wbGUub3JnIiwgImlhdCI6IDE3NDg1MzcyNDQsICJzZF
9oYXNoIjogIjBfQWYtMkItRWhMV1g1eWRoX3cyeHp3bU82aU02NkJfMlFDRWFuSTRmVV
kifQ.T3SIus2OidNl41nmVkTZVCKKhOAX97aOldMyHFiYjHm261eLiJ1YiuONFiMN8Ql
CmYzDlBLAdPvrXh52KaLgUQ
```

The following Key Binding JWT payload was created and signed for this presentation by the Holder:

```
{
  "nonce": "1234567890",
  "aud": "https://verifier.example.org",
  "iat": 1748537244,
  "sd_hash": "0_Af-2B-EhLWX5ydh_w2xzwmO6iM66B_2QCEanI4fUY"
}
```

If the Verifier did not require Key Binding, then the Holder could have presented the SD-JWT with selected Disclosures directly, instead of encapsulating it in an SD-JWT+KB.

After validation, the Verifier will have the following Processed SD-JWT Payload available for further handling:

```
{
  "iss": "https://issuer.example.com",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "user_42",
  "nationalities": [
    "US"
  ],
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  },
  "family_name": "Doe",
  "address": {
    "street_address": "123 Main St",
    "locality": "Anytown",
    "region": "Anystate",
    "country": "US"
  },
  "given_name": "John"
}
```

6. Considerations on Nested Data in SD-JWTs

Being JSON, an object in an SD-JWT payload **MAY** contain name/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 or not the claim should be selectively disclosable. This choice can be made on each level independent of 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 structure to use, depending on, for example, the expected use cases for the SD-JWT, requirements for privacy, size considerations, or operating environment requirements. For more examples with nested structures, see Appendices A.1 and A.2.

The following input JWT Claims 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"
  }
}
```

> Note: The following examples of the structures are non-normative and are not intended to represent all possible options. They are also not meant to define or restrict how `address` claim can be represented in an SD-JWT.

6.1. Example: 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://issuer.example.com",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "_sd_alg": "sha-256"
}
```

The Issuer would create the following Disclosure referenced by the one hash in the SD-JWT:

- Claim `address`:
  - SHA-256 Hash:

    `fOBUSQvo46yQO-wRwXBcGqvnbKIueISEL961_Sjd4do`

  - Disclosure:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImFkZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjogIlNjaHVsc3RyLiAxMiIsICJsb2NhbGl0eSI6ICJTY2h1bHBmb3J0YSIsICJyZWdpb24iOiAiU2FjaHNlbi1BbmhhbHQiLCAiY291bnRyeSI6ICJERSJ9XQ`

  - Contents:

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

6.2. Example: Structured SD-JWT

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

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

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN0cmVldF9hZGRyZXNzIiwgIlNjaHVsc3RyLiAxMiJd`
  - Contents:

    `["2GLC42sKQveCfGfryNRN9w", "street_address", "Schulstr. 12"]`
- Claim `locality`:
  - SHA-256 Hash:

    `6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0`
  - Disclosure:

    `WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImxvY2FsaXR5IiwgIlNjaHVscGZvcnRhIl0`
  - Contents:

    `["eluV5Og3gSNII8EYnsxA_A", "locality", "Schulpforta"]`

- Claim `region`:
  - SHA-256 Hash:

    `KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88`
  - Disclosure:

    `WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInJlZ2lvbiIsICJTYWNoc2VuLUFuaGFsdCJd`
  - 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` permanently disclosed and hide only the other sub-claims:

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

In this case, there would be no Disclosure for `country`, since it is provided in the clear.

6.3. Example: 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://issuer.example.com",
  "iat": 1683000000,
  "exp": 1883000000,
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "_sd_alg": "sha-256"
}
```

The Issuer first creates Disclosures 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:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN0cmVldF9hZGRyZXNzIiwgIlNjaHVsc3RyLiAxMiJd`
  - Contents:

    `["2GLC42sKQveCfGfryNRN9w", "street_address", "Schulstr. 12"]`
- Claim `locality`:
  - SHA-256 Hash:

    `6vh9bq-zS4GKM_7GpggVbYzzu6oOGXrmNVGPHP75Ud0`
  - Disclosure:

    `WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImxvY2FsaXR5IiwgIlNjaHVscGZvcnRhIl0`
  - Contents:

    `["eluV5Og3gSNII8EYnsxA_A", "locality", "Schulpforta"]`
- Claim `region`:
  - SHA-256 Hash:

    `KURDPh4ZC19-3tiz-Df39V8eidy1oV3a3H1Da2N0g88`
  - Disclosure:

    `WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInJlZ2lvbiIsICJTYWNoc2VuLUFuaGFsdCJd`
  - 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:

    `WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImFkZHJlc3MiLCB7Il9zZCI6IFsiNnZoOWJxLXpTNEdLTV83R3BnZ1ZiWXp6dTZvT0dYcm1OVkdQSFA3NVVkMCIsICI5Z2pWdVh0ZEZST0NnUnJ0TmNHVVhtRjY1cmRlemlfNkVyX2o3NmttWXlNIiwgIktVURQaDRaQzE5LTN0aXotRGYzOVY4ZWlkeTFvVjNhM0gxRGEyTjBnODgiLCAiV045cjlkQ0JKOEhUQ3NTMmpLQVN4VGpFeVc1bTV4NjVfWl8ycm8yamZYTSJdfV0`
  - Contents:

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

7. Verification and Processing

7.1. Verification of the SD-JWT

Upon receiving an SD-JWT, either directly or as a component of an SD-JWT+KB, a Holder or Verifier needs to ensure that:

- the Issuer-signed JWT is valid, and
- all Disclosures are valid and correspond to a respective digest value in the Issuer-signed JWT (directly in the payload or recursively included in the contents of other Disclosures).

The Holder or the Verifier **MUST** perform the following checks when receiving an SD-JWT to validate the SD-JWT and extract the payload:

1. Separate the SD-JWT into the Issuer-signed JWT and the Disclosures (if any).

2. Validate the Issuer-signed JWT:
   a. Ensure that the used signing algorithm was deemed secure for the application. Refer to [RFC8725], Sections 3.1 and 3.2 for details. The "none" algorithm **MUST NOT** be accepted.
   b. Validate the signature over the Issuer-signed JWT per Section 5.2 of [RFC7515].
   c. Validate the Issuer and that the signing key belongs to this Issuer.
   d. Check that the `_sd_alg` claim value is understood and the hash algorithm is deemed secure according to the Holder or Verifier's policy (see Section 4.1.1).
3. Process the Disclosures and embedded digests in the Issuer-signed JWT as follows:
   a. For each Disclosure provided:
      i. Calculate the digest over the base64url-encoded string as described in Section 4.2.3.
   b. (*) Identify all embedded digests in the Issuer-signed JWT as follows:
      i. Find all objects having an `_sd` key that refers to an array of strings.
      ii. Find all array elements that are objects with one key, that key being `...` and referring to a string.
   c. (**) For each embedded digest found in the previous step:
      i. 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.
      ii. If the digest was found in an object's `_sd` key:
         1. If the contents of the respective Disclosure is not a JSON array of three elements (salt, claim name, claim value), the SD-JWT **MUST** be rejected.
         2. If the claim name is `_sd` or `...`, the SD-JWT **MUST** be rejected.
         3. If the claim name already exists at the level of the `_sd` key, the SD-JWT **MUST** be rejected.
         4. Insert, at the level of the `_sd` key, a new claim using the claim name and claim value from the Disclosure.
         5. Recursively process the value using the steps described in (*) and (**).
      iii. If the digest was found in an array element:
         1. If the contents of the respective Disclosure is not a JSON array of two elements (salt, value), the SD-JWT **MUST** be rejected.
         2. Replace the array element with the value from the Disclosure.
         3. Recursively process the value using the steps described in (*) and (**).
   d. Remove all array elements for which the digest was not found in the previous step.
   e. Remove all `_sd` keys and their contents from the Issuer-signed JWT payload. If this results in an object with no properties, it should be represented as an empty object `{}`.
   f. Remove the claim `_sd_alg` from the SD-JWT payload.

4. If any digest value is encountered more than once in the Issuer-signed JWT payload (directly or recursively via other Disclosures), the SD-JWT **MUST** be rejected.
5. If any Disclosure was not referenced by digest value in the Issuer-signed JWT (directly or recursively via other Disclosures), the SD-JWT **MUST** be rejected.
6. Check that the SD-JWT is valid using claims such as `nbf`, `exp`, and `aud` in the processed payload, if present. If a required validity-controlling claim is missing (see Section 9.7), the SD-JWT **MUST** be rejected.

If any step fails, the SD-JWT is not valid, and processing **MUST** be aborted. Otherwise, the JSON document resulting from the preceding processing and verification steps, herein referred to as the "Processed SD-JWT Payload", can be made available to the application to be used for its intended purpose.

> Note that these processing steps do not yield any guarantees to the Holder about having received a complete set of Disclosures. That is, for some digest values in the Issuer-signed JWT (which are not decoy digests), there may be no corresponding Disclosures, for example, if the message from the Issuer was truncated. 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 user when needed.

7.2. Processing by the Holder

The Issuer provides the Holder with an SD-JWT, not an SD-JWT+KB. If the Holder receives an SD-JWT+KB, it **MUST** be rejected.

When receiving an SD-JWT, the Holder **MUST** do the following:

1. Process the SD-JWT as defined in Section 7.1 to validate it and extract the payload.
2. Ensure that the contents of claims in the payload are acceptable (depending on the application; for example, check that any values the Holder can check are correct).

For presentation to a Verifier, the Holder **MUST** perform the following (or equivalent) steps (in addition to the checks described in Section 7.1 performed after receiving the SD-JWT):

1. Decide which Disclosures to release to the Verifier, obtaining consent if necessary (note that if and how consent is attained is out of scope for this document).
2. Verify that each selected Disclosure satisfies one of the two following conditions:
   a. The hash of the Disclosure is contained in the Issuer-signed JWT claims.
   b. The hash of the Disclosure is contained in the claim value of another selected Disclosure.
3. Assemble the SD-JWT, including the Issuer-signed JWT and the selected Disclosures (see Section 4 for the format).
4. If Key Binding is not required:
   a. Send the SD-JWT to the Verifier.

5. If Key Binding is required:
   a. Create a Key Binding JWT tied to the SD-JWT.
   b. Assemble the SD-JWT+KB by concatenating the SD-JWT and the Key Binding JWT.
   c. Send the SD-JWT+KB to the Verifier.

7.3. Verification by the Verifier

Upon receiving a presentation from a Holder, in the form of either an SD-JWT or an SD-JWT+KB, in addition to the checks described in Section 7.1, Verifiers need to ensure that

- if Key Binding is required, then the Holder has provided an SD-JWT+KB, and
- 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 or not a Key Binding JWT is provided by the Holder. Refer to Section 9.5 for details.
2. If Key Binding is required and the Holder has provided an SD-JWT (without Key Binding), the Verifier **MUST** reject the presentation.
3. If the Holder has provided an SD-JWT+KB, parse it into an SD-JWT and a Key Binding JWT.
4. Process the SD-JWT as defined in Section 7.1 to validate the presentation and extract the payload.
5. If Key Binding is required:
   a. Determine the public key for the Holder from the SD-JWT (see Section 4.1.2).
   b. 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.
   c. Validate the signature over the Key Binding JWT per Section 5.2 of [RFC7515].
   d. Check that the `typ` of the Key Binding JWT is `kb+jwt` (see Section 4.3).
   e. Check that the creation time of the Key Binding JWT, as determined by the `iat` claim, is within an acceptable window.
   f. Determine that the Key Binding JWT is bound to the current transaction and was created for this Verifier (replay detection) by validating `nonce` and `aud` claims.
   g. Calculate the digest over the Issuer-signed JWT and Disclosures as defined in Section 4.3.1 and verify that it matches the value of the `sd_hash` claim in the Key Binding JWT.
   h. Check that the Key Binding JWT is a valid JWT 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.

8. JWS JSON Serialization

This section describes an alternative format for SD-JWTs and SD-JWT+KBs using the JWS JSON Serialization from [RFC7515]. Supporting this format is **OPTIONAL**.

8.1. New Unprotected Header Parameters

For both the General and Flattened JSON Serialization, the SD-JWT or SD-JWT+KB is represented as a JSON object according to Section 7.2 of [RFC7515]. The following new unprotected header parameters are defined:

`disclosures`: An array of strings where each element is an individual Disclosure as described in Section 4.2.

`kb_jwt`: Present only in an SD-JWT+KB, the Key Binding JWT as described in Section 4.3.

In an SD-JWT+KB, `kb_jwt` **MUST** be present when using the JWS JSON Serialization, and the digest in the `sd_hash` claim **MUST** be computed over the SD-JWT as described in Section 4.3.1. This means that even when using the JWS JSON Serialization, the representation as a regular SD-JWT Compact Serialization **MUST** be created temporarily to calculate the digest. In detail, the SD-JWT Compact Serialization part is built by concatenating the protected header, the payload, and the signature of the JWS JSON serialized SD-JWT using a `.` character as a separator, and using the Disclosures from the `disclosures` member of the unprotected header.

Unprotected headers other than `disclosures` are not covered by the digest, and therefore, as usual, are not protected against tampering.

8.2. Flattened JSON Serialization

In the case of Flattened JSON Serialization, there is only one unprotected header.

The following is a non-normative example of a JWS JSON serialized SD-JWT as issued using the Flattened JSON Serialization:

```
{
  "header": {
    "disclosures": [
      "WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN1YiIsICJqb2huX2RvZV80M
        iJd",
      "WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiSm9ob
        iJd",
      "WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIkRvZ
        SJd",
      "WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImJpcnRoZGF0ZSIsICIxOTQwL
        TAxLTAxIl0"
    ]
  },
  "payload": "eyJfc2QiOiBbIjRIQm42YUlZM1d0dUdHV1R4LXFVajZjZGs2V0JwWn
    lnbHRkRmF2UGE3TFkiLCAiOHNtMVFDZjAyMXBObkhBQ0k1c1A0bTRLWmd5Tk9PQV
    ljVGo5SE5hQzF3WSIsICJjZ0ZkaHFQbzgzeFlObEpmYWNhQ2FhN3VQOVJDUjUwVk
    U1UjRMQVE5aXFVIiwgImpNQ1hWei0tOWI4eDM3WWNvRGZYUWluencxd1pjY2NmRl
    JCQ0ZHcWRHMm8iXSwgImlzcyI6ICJodHRwczovL2lzc3Vlci5leGFtcGxlLmNvbS
    IsICJpYXQiOiAxNjgzMDAwMDAwLCAiZXhwIjogMTg4MzAwMDAwMCwgIl9zZF9hbG
    ciOiAic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydi
    I6ICJQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTG
    lsRGxzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVm
    Z1ZWNDRTZ0NGpUOUYySFpRIn19fQ",
  "protected":
    "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0",
  "signature": "3oOtvPxU3QdDWUmfGexVB5rWyON2f1atg5rL825bvvD1g7ywjKDK
    y2UHqHoH2QS4FA99JbG5qnlqFaGXFChfjQ"
}
```

The following is an SD-JWT+KB with two Disclosures:

```
{
  "header": {
    "disclosures": [
      "WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIkRvZ
        SJd",
      "WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiSm9ob
        iJd"
    ],
    "kb_jwt": "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25j
      ZSI6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL3ZlcmlmaWVyLmV4YW
      1wbGUub3JnIiwgImlhdCI6IDE3NDg1MzcyNDQsICJzZF9oYXNoIjogIlZqdFBz
      Z1pwUVRSeEtKdkRwU0otblhsWktFOVo5TGdENEZ5Q3d3b05NUncifQ.GrDvJ2j
      hYNmUvqdwVEIrxeTFEuI5qKSM7I6P95JmA6Wko-FBB5vPGQn0wvmdgjLCE2iDR
      h1r82zchjmABQ3V8w"
  },
  "payload": "eyJfc2QiOiBbIjRIQm42YUlZM1d0dUdHV1R4LXFVajZjZGs2V0JwWn
    lnbHRkRmF2UGE3TFkiLCAiOHNtMVFDZjAyMXBObkhBQ0k1c1A0bTRLWmd5Tk9PQV
    ljVGo5SE5hQzF3WSIsICJjZ0ZkaHFQbzgzeFlObEpmYWNhQ2FhN3VQOVJDUjUwVk
    U1UjRMQVE5aXFVIiwgImpNQ1hWei0tOWI4eDM3WWNvRGZYUWluencxd1pjY2NmRl
    JCQ0ZHcWRHMm8iXSwgImlzcyI6ICJodHRwczovL2lzc3Vlci5leGFtcGxlLmNvbS
    IsICJpYXQiOiAxNjgzMDAwMDAwLCAiZXhwIjogMTg4MzAwMDAwMCwgIl9zZF9hbG
    ciOiAic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydi
    I6ICJQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTG
    lsRGxzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVm
    Z1ZWNDRTZ0NGpUOUYySFpRIn19fQ",
  "protected":
    "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0",
  "signature": "3oOtvPxU3QdDWUmfGexVB5rWyON2f1atg5rL825bvvD1g7ywjKDK
    y2UHqHoH2QS4FA99JbG5qnlqFaGXFChfjQ"
}
```


8.3. General JSON Serialization

In the case of General JSON Serialization, there are multiple unprotected headers (one per signature). If present, `disclosures` and `kb_jwt` **MUST** be included in the first unprotected header and **MUST NOT** be present in any following unprotected headers.

The following is a non-normative example of a presentation of a JWS JSON serialized SD-JWT, including a Key Binding JWT using the General JSON Serialization:

```
{
  "payload": "eyJfc2QiOiBbIjRIQm42YUlZM1d0dUdHV1R4LXFVajZjZGs2V0JwWn
    lnbHRkRmF2UGE3TFkiLCAiOHNtMVFDZjAyMXBObkhBQ0k1c1A0bTRLWmd5Tk9PQV
    ljVGo5SE5hQzF3WSIsICJjZ0ZkaHFQbzgzeFlObEpmYWNhQ2FhN3VQOVJDUjUwVk
    U1UjRMQVE5aXFVIiwgImpNQ1hWei0tOWI4eDM3WWNvRGZYUWluencxd1pjY2NmRl
    JCQ0ZHcWRHMm8iXSwgImlzcyI6ICJodHRwczovL2lzc3Vlci5leGFtcGxlLmNvbS
    IsICJpYXQiOiAxNjgzMDAwMDAwLCAiZXhwIjogMTg4MzAwMDAwMCwgIl9zZF9hbG
    ciOiAic2hhLTI1NiIsICJjbmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydi
    I6ICJQLTI1NiIsICJ4IjogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTG
    lsRGxzN3ZDZUdlbWMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVm
    Z1ZWNDRTZ0NGpUOUYySFpRIn19fQ",
  "signatures": [
    {
      "header": {
        "disclosures": [
          "WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgI
            kRvZSJd",
          "WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiS
            m9obiJd"
        ],
        "kid": "issuer-key-1",
        "kb_jwt": "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJu
          b25jZSI6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL3ZlcmlmaW
          VyLmV4YW1wbGUub3JnIiwgImlhdCI6IDE3NDg1MzcyNDQsICJzZF9oYXNo
          IjogInFieUlXUDNwaFZneEVzRFJpd2R3OVc2QkozZHhpUEx1bWNZcFBidT
          RFYjgifQ.VyZqxaVHh1XE6M-kuax_7Laq42uFDrx17lLG2jluyKgy_PqC8
          5z4DVpISdMZDdSANGs-0zN2N7xnM-E1Pg0sOw"
      },
      "protected":
        "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0",
      "signature": "dz1N3uvhVHJjldyXwppmBLieTj0vuBMbzL06rnrLIuxEQb9B
        HoIOwGrWh-UadW4orRpEiEtjf7xyHDONMJ6tBw"
    },
    {
      "header": {
        "kid": "issuer-key-2"
      },
      "protected":
        "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0",
      "signature": "kuXio_U88RH_-fihAPET4AFUjj0BpxsT6yddMFIr6pfHKtAe
        0FOJNWQxU42rfnORuNQNTgGsf2A8LjEba5inNg"
    }
  ]
}
```


8.4. Verification of the JWS JSON Serialized SD-JWT

Verification of the JWS JSON serialized SD-JWT follows the rules defined in Section 3.4, except for the following aspects:

- The SD-JWT or SD-JWT+KB does not need to be split into component parts and the Disclosures can be found in the `disclosures` member of the unprotected header.

- To verify the digest in `sd_hash` in the Key Binding JWT of an SD-JWT+KB, the Verifier **MUST** assemble the string to be hashed as described in Section 8.1.

9. Security Considerations

The security considerations 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 9.5, the application of Key Binding can ensure that the presenter of an SD-JWT credential is the Holder of the credential.

9.1. Mandatory Signing of the Issuer-Signed JWT

The JWT **MUST** be signed by the Issuer to protect the 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 its 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. Per the last paragraph of Section 5.2 of [RFC7515], it is an application-specific decision to choose the appropriate JWS algorithm from [JWS.Algs], 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 7.3 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 high entropy random data of the salt as additional input to the hash function. The randomness ensures that the same plaintext claim value does not produce the same digest value. It also makes it infeasible to guess the preimage of the digest (thereby learning the plaintext claim value) by enumerating the potential value space for a claim into the hash function to search for a matching digest value. It is therefore vitally important that unrevealed salts cannot be learned or guessed, even if other salts have been revealed. As such, each salt **MUST** be created in such a manner that it is cryptographically random, sufficiently long, and has high enough entropy that it is infeasible to guess. A new salt **MUST** be chosen for each claim independently of other salts. See "Randomness Requirements for Security" [RFC4086] for considerations on generating random values.

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 the example in Appendix A.2, where multiple claims with the name `type` appear, but each of them has a different salt.

9.4. Choice of a Hash Algorithm

To ensure privacy of claims that are selectively disclosable but are not being disclosed in a given presentation, the hash function **MUST** ensure that it is infeasible to calculate any portion of the three elements (salt, claim name, claim value) from a particular digest. This implies the hash function **MUST** be preimage resistant and should also not allow an observer to infer any partial information about the undisclosed content. In the terminology of cryptographic commitment schemes, the hash function needs to be computationally hiding.

To ensure the integrity of selectively disclosable claims, the hash function **MUST** be second-preimage resistant. That is, for any combination of salt, claim name, and claim value, it is infeasible to find a different combination of salt, claim name, and claim value that results in the same digest.

The hash function **SHOULD** also be collision resistant. Although not essential to the anticipated uses of SD-JWT, without collision resistance an Issuer may be able to find multiple Disclosures that have the same hash value. In which case, the signature over the SD-JWT would not then commit the Issuer to the contents of the JWT. The collision resistance of the hash function used to generate digests **SHOULD** match the collision resistance of the hash function used by the signature scheme. For example, use of the ES512 signature algorithm would require a Disclosure hash function with at least 256-bit collision resistance, such as SHA-512.

Inclusion in the "Named Information Hash Algorithm Registry" [Hash.Algs] 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).

9.5. Key Binding

Key Binding aims to ensure that the presenter of an SD-JWT credential is actually the Holder of the credential. An SD-JWT compatible 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 the credential was leaked to an attacker, the attacker can present the credential to any Verifier that does not require a binding. 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 not make its security policy decisions based on data that can be influenced by an attacker. For this reason, when deciding whether or not Key Binding is required, Verifiers **MUST NOT** take into account whether the Holder has provided an SD-JWT+KB or a bare SD-JWT; otherwise, an attacker could strip the KB-JWT from an SD-JWT+KB and present the resultant SD-JWT.

Furthermore, Verifiers should be aware that Key Binding information may have been added to an SD-JWT in a format that they do not recognize and therefore may not be able to tell whether or not the SD-JWT supports Key Binding.

If a Verifier determines that Key Binding is required for a particular use case and the Holder presents either a bare SD-JWT or an SD-JWT+KB with an invalid Key Binding JWT, then the Verifier will reject the presentation when following the verification steps described in Section 7.3.

9.6. Concealing Claim Names

SD-JWT ensures that names of claims that are selectively disclosable are always concealed unless the claim's value is disclosed. This prevents an attacker from learning the names of such claims. However, the names of the claims that are permanently disclosed are not hidden. This includes the keys of objects that themselves are not concealed, but contain concealed claims. This limitation needs to be taken into account by Issuers when creating the structure of the SD-JWT.

9.7. Selectively Disclosable Validity Claims

An Issuer **MUST NOT** allow any content to be selectively disclosable that is critical for evaluating the SD-JWT's authenticity or validity. The exact list of such content will depend on the application and **SHOULD** be listed by any application-specific profiles of SD-JWT. The following is a list of registered JWT claim names that **SHOULD** be considered as security critical:

- `iss` (Issuer)
- `aud` (Audience), although issuers **MAY** allow individual entries in the array to be selectively disclosable
- `exp` (Expiration Time)
- `nbf` (Not Before)
- `cnf` (Confirmation Key)

Issuers will typically include claims controlling the validity of the SD-JWT in plaintext in the SD-JWT payload, but there is no guarantee they will do so. Therefore, Verifiers cannot reliably depend on that and need to operate as though security-critical claims might be selectively disclosable.

Verifiers therefore **MUST** ensure that all claims they deem necessary for checking the validity of an SD-JWT in the given context are present (or disclosed, respectively) during validation of the SD-JWT. This is implemented in the last step of the verification defined in Section 7.1.

The precise set of required validity claims will typically be defined by operating environment rules, an application-specific profile, or the credential format, and **MAY** include claims other than those listed herein.

9.8. Distribution and Rotation of Issuer Signature Verification Key

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

Any entity in possession of an SD-JWT (including an SD-JWT extracted from an SD-JWT+KB) can forward it to any third party that does not enforce Key Binding. 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; if it is not, Issuers need to support and Verifiers need to enforce Key Binding.

9.10. Integrity of SD-JWTs and SD-JWT+KBs

With an SD-JWT, the Issuer-signed JWT is integrity protected by the Issuer's signature, and the values of the Disclosures are integrity protected by the digests included therein. The specific set of Disclosures, however, is not integrity protected; the SD-JWT can be modified by adding or removing Disclosures and still be valid.

With an SD-JWT+KB, the set of selected Disclosures is integrity protected. The signature in the Key Binding JWT covers a specific SD-JWT, with a specific Issuer-signed JWT and a specific set of Disclosures. Thus, the signature on the Key Binding JWT, in addition to proving Key Binding, also assures the authenticity and integrity of the set of Disclosures the Holder disclosed. The set of Disclosures in an SD-JWT+KB is the set that the Holder intended to send; no intermediate party has added, removed, or modified the list of Disclosures.

9.11. Explicit Typing

Section 3.11 of [RFC8725] describes the use of explicit typing as one mechanism to prevent confusion attacks (described in Section 2.8 of [RFC8725]) in which one kind of JWT is mistaken for another. SD-JWTs are also potentially subject to such confusion attacks, so in the absence of other techniques, it is **RECOMMENDED** that application profiles of SD-JWT specify an explicit type by including the `typ` header parameter when the SD-JWT is issued, and that Verifiers check this value.

When explicit typing using the `typ` header 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.

Use of the `cty` content type header parameter to indicate the content type of the SD-JWT payload can also be used to distinguish different types of JSON objects or different kinds of JWT Claim Sets.

9.12. Key Pair Generation and Lifecycle Management

Implementations of SD-JWT rely on asymmetric cryptographic keys and must therefore ensure that key pair generation, handling, storage, and lifecycle management are performed securely.

While the specific mechanisms for secure key management are out of scope for this document, implementers should follow established best practices, such as those outlined in NIST SP 800-57 Part 1 [NIST.SP.800-57pt1r5]. This includes:

- Secure Generation: Using cryptographically secure methods and random number generators.
- Secure Storage: Protecting private keys from unauthorized access.
- Lifecycle Management: Ensuring secure key rotation, revocation, and disposal as needed.

Appropriate key management is essential, as any compromise can lead to unauthorized disclosure or forgery of SD-JWTs.

10. Privacy Considerations

10.1. Unlinkability

Unlinkability is a property whereby adversaries are prevented from correlating credential presentations of the same user beyond the user's consent. Without unlinkability, an adversary might be able to learn more about the user than the user intended to disclose, for example:

- Cooperating Verifiers might want to track users across services to build advertising profiles.
- Issuers might want to track where users present their credentials to enable surveillance.
- After a data breach at multiple Verifiers, publicly available information might allow linking identifiable information presented to Verifier A with originally anonymous information presented to Verifier B, therefore revealing the identities of users of Verifier B.

The following types of unlinkability are discussed below:

- Presentation Unlinkability: A Verifier should not be able to link two presentations of the same credential.
- Verifier/Verifier Unlinkability: The presentations made to two different Verifiers should not reveal that the same credential was presented (e.g., if the two Verifiers collude, or if they are forced by a third party to reveal the presentations made to them, or data leaks from one Verifier to the other).
- Issuer/Verifier Unlinkability (Honest Verifier): An Issuer of a credential should not be able to know that a user presented this credential unless the Verifier is sharing presentation data with the Issuer accidentally, deliberately, or because it is forced to do so.
- Issuer/Verifier Unlinkability (Careless/Colluding/Compromised/Coerced Verifier): >An Issuer of a credential should under no circumstances be able to tell that a user presented this credential to a certain Verifier. In particular, this includes cases when the Verifier accidentally or deliberately shares presentation data with the Issuer or is forced to do so.

In all cases, unlinkability is limited to cases where the disclosed claims do not contain information that directly or indirectly identifies the user. For example, when a taxpayer identification number is contained in the disclosed claims, the Issuer and Verifier can easily link

the user's transactions. However, when the user only discloses a birthdate to one Verifier and a postal code to another Verifier, the two Verifiers should not be able to determine that they were interacting with the same user.

Issuer/Verifier unlinkability with a careless, colluding, compromised, or coerced Verifier cannot be achieved in salted hash-based selective disclosure approaches, such as SD-JWT, as the issued credential with the Issuer's signature is directly presented to the Verifier, who can forward it to the Issuer. To reduce the risk of revealing the data later on, Section 10.2 defines requirements to reduce the amount of data stored.

In considering Issuer/Verifier unlinkability, it is important to note the potential for an asymmetric power dynamic between Issuers and Verifiers. This dynamic can compel an otherwise Honest Verifier into collusion. For example, a governmental Issuer might have the authority to mandate that a Verifier report back information about the credentials presented to it. Legal requirements could further enforce this, explicitly undermining Issuer/Verifier unlinkability. Similarly, a large service provider issuing credentials might implicitly pressure Verifiers into collusion by incentivizing participation in their larger operating environment. Deployers of SD-JWT must be aware of these potential power dynamics, mitigate them as much as possible, and/or make the risks transparent to the user.

Contrary to that, Issuer/Verifier unlinkability with an Honest Verifier can generally be achieved. However, a callback from the Verifier to the Issuer, such as a revocation check, could potentially disclose information about the credential's usage to the Issuer. Where such callbacks are necessary, they need to be executed in a manner that preserves privacy and does not disclose details about the credential to the Issuer (the mechanism described in [TSL] is an example of an approach that discloses minimal information towards the Issuer). It is important to note that the timing of such requests could potentially serve as a side channel.

Verifier/Verifier unlinkability and presentation unlinkability can be achieved using batch issuance: A batch of credentials based on the same claims is issued to the Holder instead of just a single credential. The Holder can then use a different credential for each Verifier or even for each session with a Verifier. New Key Binding keys and salts **MUST** be used for each credential in the batch to ensure that the Verifiers cannot link the credentials using these values. Likewise, claims carrying time information, like `iat`, `exp`, and `nbf`, **MUST** either be randomized within a time period considered appropriate (e.g., randomize `iat` within the last 24 hours and calculate `exp` accordingly) or rounded (e.g., rounded down to the beginning of the day).

SD-JWT only conceals the value of claims that are not revealed. It does not meet the security properties for anonymous credentials [CL01]. In particular, colluding Verifiers and Issuers can know when they have seen the same credential no matter what fields have been disclosed, even when none have been disclosed. This behavior may not align with what users naturally anticipate or are guided to expect from user-interface interactions, potentially causing them to make decisions they might not otherwise make. Workarounds such as batch issuance, as described above, help with keeping Verifiers from linking different presentations, but cannot work for Issuer/Verifier unlinkability. This issue applies to all salted hash-based approaches, including mDL/mDoc [ISO.18013-5] and SD-CWT [SD-CWT].

10.2. Storage of User Data

Wherever 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 [OpenID.Core], 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 users to third parties. Signed user data, the authenticity of which can be easily verified by third parties, further exacerbates the risk. As discussed in Section 9.5, 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.

Due to these risks, and the risks described in Section 10.1, systems implementing SD-JWT **SHOULD** be designed to minimize the amount of data that is stored. All involved parties **SHOULD NOT** store SD-JWTs longer than strictly necessary, including in log files.

After Issuance, Issuers **SHOULD NOT** store the Issuer-signed JWT or the respective Disclosures.

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 user devices, **SHOULD** be preferred for user credentials over centralized storage. Expired SD-JWTs **SHOULD** be deleted as soon as possible.

After Verification, Verifiers **SHOULD NOT** store the Issuer-signed JWT or the respective Disclosures. It may be sufficient to store the result of the verification and any user data that is needed for the application.

Exceptions from the rules above can be made if there are strong requirements to do so (e.g., functional requirements or legal audit requirements), secure storage can be ensured, and the privacy impact has been assessed.

10.3. Confidentiality During Transport

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

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 user data or correlation attacks by passive observers are a concern.

To encrypt an SD-JWT or SD-JWT+KB during transit over potentially insecure or leakage-prone channels, implementers **MAY** use JSON Web Encryption (JWE) [RFC7516], encapsulating the SD-JWT or SD-JWT+KB as the plaintext payload of the 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.

10.4. 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 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 user's data.

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 a cancer research 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. IANA Considerations

11.1. JSON Web Token Claims Registration

IANA has registered the following Claims in the "JSON Web Token Claims" registry [JWT.Claims] established by [RFC7519].

Claim Name: `_sd`
Claim Description: Digests of Disclosures for object properties
Change Controller: IETF
Specification Document(s): Section 4.2.4.1 of RFC 9901

Claim Name: `...`
Claim Description: Digest of the Disclosure for an array element
Change Controller: IETF
Specification Document(s): Section 4.2.4.2 of RFC 9901

Claim Name: `_sd_alg`
Claim Description: Hash algorithm used to generate Disclosure digests and digest over presentation
Change Controller: IETF
Specification Document(s): Section 4.1.1 of RFC 9901

Claim Name: `sd_hash`
Claim Description: Digest of the SD-JWT to which the KB-JWT is tied
Change Controller: IETF
Specification Document(s): Section 4.3 of RFC 9901

11.2. Media Type Registrations

IANA has registered the following media types [RFC2046] in the "Media Types" registry [MediaTypes] in the manner described in [RFC6838].

> Note: For the media type value used in the `typ` header in the Issuer-signed JWT itself, see Section 9.11.

11.2.1. SD-JWT Content

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 ('.') and tilde ('~') characters.

Security considerations: See the Security Considerations sections of RFC 9901, [RFC7519], and [RFC8725].

Interoperability considerations: n/a

Published specification: RFC 9901

Applications that use this media type: Applications requiring selective disclosure of integrity-protected content.

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: IETF

11.2.2. JWS JSON Serialized SD-JWT Content

To indicate that the content is a JWS JSON serialized SD-JWT:

Type name: application

Subtype name: sd-jwt+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: binary; application/sd-jwt+json values are represented as a JSON Object.

Security considerations: See the Security Considerations sections of RFC 9901 and [RFC7515].

Interoperability considerations: n/a

Published specification: RFC 9901

Applications that use this media type: Applications requiring selective disclosure of content protected by ETSI JAdES compliant signatures.

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: IETF

11.2.3. Key Binding JWT Content

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 separated by period ('.') characters.

Security considerations: See the Security Considerations sections of RFC 9901, [RFC7519], and [RFC8725].

Interoperability considerations: n/a

Published specification: RFC 9901

Applications that use this media type: Applications utilizing a JWT-based proof-of-possession mechanism.

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: IETF

11.3. Structured Syntax Suffixes Registration

IANA has registered "+sd-jwt" in the "Structured Syntax Suffixes" registry [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: RFC 9901

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 sections of RFC 9901, [RFC7519], and [RFC8725].

Contact: Daniel Fett, mail@danielfett.de

Author/Change controller: IETF

12. References

12.1. 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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[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>.

[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>.

[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>.

[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>.

12.2. Informative References

[CL01] Camenisch, J. and A. Lysyanskaya, "An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation", Cryptology ePrint Archive, Paper 2001/019, 2001, <https://eprint.iacr.org/2001/019.pdf>.

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

[ISO.18013-5] ISO/IEC, "Personal identification - ISO-compliant driving license — Part 5: Mobile driving license (mDL) application", ISO/IEC 18013-5:2021, September 2021, <https://www.iso.org/standard/69084.html>.

[JWS.Algs] IANA, "JSON Web Signature and Encryption Algorithms", <https://www.iana.org/assignments/jose>.

[JWT.Claims] IANA, "JSON Web Token Claims", <https://www.iana.org/assignments/jwt>.

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

[NIST.SP.800-57pt1r5] Barker, E., "Recommendation for Key Management: Part 1 - General", NIST SP 800-57pt1r5, DOI 10.6028/NIST.SP.800-57pt1r5, May 2020, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>.

[OIDC.IDA] Lodderstedt, T., Fett, D., Haine, M., Pulido, A., Lehmann, K., and K. Koiwai, "OpenID Connect for Identity Assurance 1.0", 1 October 2024, <https://openid.net/specs/openid-connect-4-identity-assurance-1_0.html>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

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

[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>.

[SD-CWT] Prorock, M., Steele, O., Birkholz, H., and R. Mahy, "Selective Disclosure CBOR Web Tokens (SD-CWT)", Work in Progress, Internet-Draft, draft-ietf-spice-sd-cwt-05, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-spice-sd-cwt-05>.

[SD-JWT-VC] Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", Work in Progress, Internet-Draft, draft-ietf-oauth-sd-jwt-vc-13, 6 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-13>.

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

[TSL] Looker, T., Bastian, P., and C. Bormann, "Token Status List (TSL)", Work in Progress, Internet-Draft, draft-ietf-oauth-status-list-13, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-status-list-13>.

[VC_DATA_v2.0] Sporny, M., Ed., Thiboeau, T., Ed., Jones, M. B., Ed., Cohen, G., Ed., and I. Herman, Ed., "Verifiable Credentials Data Model 2.0", W3C Recommendation, May 2025, <https://www.w3.org/TR/vc-data-model-2.0/>.

Appendix A. Additional Examples

The following examples are not normative and are provided for illustrative purposes only. In particular, neither the structure of the claims nor the selection of selectively disclosable claims is normative.

Line breaks have been added for readability.

A.1. Simple Structured SD-JWT

In this example, in contrast to Section 5, the Issuer decided to create a structured object for the `address` claim, allowing individual members of the claim to be disclosed separately.

The following data about the user comprises the input JWT Claims Set used by the Issuer:

```
{
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "given_name": "太郎",
  "family_name": "山田",
  "email": "\"unusual email address\"@example.jp",
  "phone_number": "+81-80-1234-5678",
  "address": {
    "street_address": "東京都港区芝公園４丁目２-８",
    "locality": "東京都",
    "region": "港区",
    "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://issuer.example.com",
  "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 digests in the SD-JWT payload reference the following Disclosures:

- Claim `sub`:
  - SHA-256 Hash:

    `X6ZAYOII2vPN40V7xExZwVwz7yRmLNcVwt5DL8RLv4g`

  - Disclosure:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInN1YiIsICI2YzVjMGE0OS1iNTg5LTQzMWQtYmFlNy0yMTkxMjJhOWVjMmMiXQ`

  - Contents:

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

- Claim `given_name`:
  - SHA-256 Hash:

    `ommFAicVT8LGHCB0uywx7fYuo3MHYKO15cz-RZEYM5Q`

  - Disclosure:

    `WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImdpdmVuX25hbWUiLCAiXHU1OTJhXHU5MGNlIl0`

  - Contents:

    `["eluV5Og3gSNII8EYnsxA_A", "given_name", "\u592a\u90ce"]`

- Claim `family_name`:
  - SHA-256 Hash:

    `C9inp6YoRaEXR427zYJP7Qrk1WH_8bdwOA_YUrUnGQU`

  - Disclosure:

    `WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImZhbWlseV9uYW1lIiwgIlx1NWM3MVx1NzUzMCJd`

  - Contents:

    `["6Ij7tM-a5iVPGboS5tmvVA", "family_name", "\u5c71\u7530"]`

- Claim `email`:
  - SHA-256 Hash:

    `Kuet1yAa0HIQvYnOVd59hcViO9Ug6J2kSfqYRBeowvE`

  - Disclosure:

    `WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgImVtYWlsIiwgIlwidW51c3VhbCBlbWFpbCBhZGRyZXNzXCJAZXhhbXBsZS5qcCJd`

  - Contents:

    `["eI8ZWm9QnKPpNPeNenHdhQ", "email", "\"unusual email address\"@example.jp"]`

- Claim `phone_number`:
  - SHA-256 Hash:

`s0BKYsLWxQQeU8tVlltM7MKsIRTrEIa1PkJmqxBBf5U`

- Disclosure:

`WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInBob25lX251bWJlciIsICIrODEtODAtMTIzNC01Njc4Il0`

- Contents:

```
["Qg_O64zqAxe412a108iroA", "phone_number", "+81-80-1234-5678"]
```

- Claim `street_address`:
  - SHA-256 Hash:

`6aUhzYhZ7SJ1kVmagQAO3u2ETN2CC1aHheZpKnaF0_E`

  - Disclosure:

`WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgInN0cmVldF9hZGRyZXNzIiwgIlx1Njc3MVx1NGVhY1x1OTBmZFx1NmUyZlx1NTMzYVx1ODI5ZFx1NTE2Y1x1NTcxMlx1ZmYxNFx1NGUwMVx1NzZlZVx1ZmYxMlx1MjIxMlx1ZmYxOCJd`

  - Contents:

```
["AJx-095VPrpTtN4QMOqROA", "street_address",
"\u6771\u4eac\u90fd\u6e2f\u533a\u829d\u516c\u5712\uff14\u4e01\u76ee\uff12\u2212\uff18"]
```

- Claim `locality`:
  - SHA-256 Hash:

`rvJd6iq6T5ejmsBMoGwuNXh9qAAFATAci40oidEeVsA`

  - Disclosure:

`WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImxvY2FsaXR5IiwgIlx1Njc3MVx1NGVhY1x1OTBmZCJd`

  - Contents:

```
["Pc33JM2LchcU_lHggv_ufQ", "locality", "\u6771\u4eac\u90fd"]
```

- Claim `region`:
  - SHA-256 Hash:

`PzzcVu0qbMuBGSjulfewzkesD9zutOExn5EWNwkrQ-k`

  - Disclosure:

`WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInJlZ2lvbiIsICJcdTZlMmZcdTUzM2EiXQ`

  - 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:

`WyJ5eXRWYmRBUEdjZ2wyckk0QzlHU29nIiwgImJpcnRoZGF0ZSIsICIxOTQwLTAxLTAxIl0`

  - 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 a presentation of the SD-JWT that discloses only `region` and `country` of the `address` property:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJfc2QiOiBb
IkM5aW5wNllvUmFFWFI0Mjd6WUpQN1FyazFXSF84YmR3T0FfWVVyVW5HUVUiLCAiS3Vl
dDF5QWEwSElRdlluT1ZkNTloY1ZpTzlVZzZKMmtTZnFZUkJlb3d2RSIsICJNTWxkT0ZG
ekIyZDB1bWxtcFRJYUdlcmhXZFVfUHBZZkx2S2hoX2ZfOWFZIiwgIlg2WkFZT0lJMnZQ
TjQwVjd4RXhad1Z3ejd5Um1MTmNWd3Q1REw4Ukx2NGciLCAiWTM0em1JbzBRTExPdGRN
cFhHd2pCZ0x2cjE3eUVoaFlUMEZHb2ZSLWFJRSIsICJmeUdwMFdUd3dQdjJKRFFsbjFs
U2lhZW9iWnNNV0ExMGJRNTk4OS05RFRzIiwgIm9tbUZBaWNWVDhMR0hDQjB1eXd4N2ZZ
dW8zTUhZS08xNWN6LVJaRVlNNVEiLCAiczBCS1lzTFd4UVFlVTh0VmxsdE03TUtzSVJU
ckVJYTFQa0ptcXhCQmY1VSJdLCAiaXNzIjogImh0dHBzOi8vaXNzdWVyLmV4YW1wbGUu
Y29tIiwgImlhdCI6IDE2ODMwMDAwMDAsICJleHAiOiAxODgzMDAwMDAwLCAiYWRkcmVz
cyI6IHsiX3NkIjogWyI2YVVoelloWjdTSjFrVm1hZ1FBTzN1MkVUTjJDQzFhSGhlWnBL
bmFGMF9FIiwgIkF6TGxGb2JrSjJ4aWF1cFJFUHlvSnotOS1OU2xkQjZDZ2pyN2ZVeW9I
emciLCAiUHp6Y1Z1MHFiTXVCR1NqdWxmZXd6a2VzRDl6dXRPRXhuNUVXTndrclEtayIs
ICJiMkRrdzBqY0lGOXJHZzhfUEY4WmN2bmNXN3p3Wmo1cnlCV3ZYZnJwemVrIiwgImNQ
WUpISVo4VnUtZjlDQ3lWdWIyVWZnRWs4anZ2WGV6d0sxcF9KbmVlWFEiLCAiZ2xUM2hy
U1U3ZlNXZ3dGNVVEWm1Xd0JUdzMyZ25VbGRJaGk4aEdWQ2FWNCIsICJydkpkNmlxNlQ1
ZWptc0JNb0d3dU5YaDlxQUFGQVRBY2k0MG9pZEVlVnNBIiwgInVOSG9XWWhYc1poVkpD
TkUyRHF5LXpxdDd0NjlnSkt5NVFhRnY3R3JNWDQiXX0sICJfc2RfYWxnIjogInNoYS0y
NTYifQ.EOZa2YqK8j4i7cqBDkfPcTMaFsgPwcx3aYJkFoMfvV46LxL-PPqrWsIyNukB4
x8Y2LT31eIHDc4Wg4XNzaqu4w~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgInJlZ2
lvbiIsICJcdTZlMmZcdTUzM2EiXQ~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImN
vdW50cnkiLCAiSlAiXQ~
```

After validation, the Verifier will have the following Processed SD-JWT Payload available for further handling:

```
{
  "iss": "https://issuer.example.com",
  "iat": 1683000000,
  "exp": 1883000000,
  "address": {
    "region": "港区",
    "country": "JP"
  }
}
```

A.2. Complex Structured SD-JWT

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

The Issuer is using the following user data as the input JWT Claims 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://issuer.example.com",
  "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 digests in the SD-JWT payload reference the following Disclosures:

- Claim `time`:
  - SHA-256 Hash:

    `vTwe3raHIFYgFA3xaUD2aMxFz5oDo8iBu05qKlOg9Lw`

  - Disclosure:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgInRpbWUiLCAiMjAxMi0wNC0yM1QxODoyNVoiXQ`

  - Contents:

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

- Claim `verification_process`:
  - SHA-256 Hash:

    `7h4UE9qScvDKodXVCuoKfKBJpVBfXMF_TmAGVaZe3Sc`

- Disclosure:

  WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgInZlcmlmaWNhdGlvbl9wcm9jZXNzIiwgImYyNGM2Zi02ZDNmLTRlYzUtOTczZS1iMGQ4NTA2ZjNiYzciXQ

- Contents:

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

- Claim `type`:
  - SHA-256 Hash:

    G5EnhOAOoU9X_6QMNvzFXjpEA_Rc-AEtm1bG_wcaKIk

  - Disclosure:

    WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgInR5cGUiLCAiZG9jdW1lbnQiXQ

  - 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:

    WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgInRpbWUiLCAiMjAxMi0wNC0yMlQxMTozMFoiXQ

  - Contents:

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

- Claim `document`:
  - SHA-256 Hash:

    IhwFrWUB63RcZq9yvgZ0XPc7Gowh3O2kqXeBIswg1B4

  - Disclosure:

WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImRvY3VtZW50IiwgeyJ0eXBlIjogImlkY2FyZCIs
ICJpc3N1ZXIiOiB7Im5hbWUiOiAiU3RhZHQgQXVnc2J1cmciLCAiY291bnRyeSI6ICJERSJ9LCAi
bnVtYmVyIjogIjUzNTU0NTU0IiwgImRhdGVfb2ZfaXNzdWFuY2UiOiAiMjAxMC0wMy0yMyIsICJk
YXRlX29mX2V4cGlyeSI6ICIyMDIwLTAzLTIyIn1d

◦ 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:

WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgeyJfc2QiOiBbIjl3cGpWUFd1RDdQSzBuc1FETDhC
MDZsbWRnVjNMVnliaEh5ZFFwVE55TEkiLCAiRzVFbmhPQU9vVTlYXzZRTU52ekZYanBFQV9SYy1B
RXRtMWJHX3djYUtJayIsICJJaHdGcldVQjYzUmNacTl5dmdaMFhQYzdHb3doM08ya3FYZUJJc3dn
MUI0IiwgIldweFE0SFNvRXRjVG1DQ0tPZURzbEJfZW11Y1lMejJvTzhvSE5yMWJFVlEiXX1d

◦ Contents:

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

• Claim `given_name`:

◦ SHA-256 Hash:

S_498bbpKzB6Eanftss0xc7cOaoneRr3pKr7NdRmsMo

◦ Disclosure:

WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImdpdmVuX25hbWUiLCAiTWF4Il0

◦ Contents:

```
["G02NSrQfjFXQ7Io09syajA", "given_name", "Max"]
```

• Claim `family_name`:

◦ SHA-256 Hash:

Wxh_sV3iRH9bgrTBJi-aYHNCLt-vjhX1sd-igOf_9lk

◦ Disclosure:

WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImZhbWlseV9uYW1lIiwgIk1cdTAwZmNsbGVyIl0

◦ Contents:

```
["lklxF5jMYlGTPUovMNIvCA", "family_name", "M\u00fcller"]
```

- Claim `nationalities`:
  - SHA-256 Hash:

    `hvDXhwmGcJQsBCA2OtjuLAcwAMpDsaU0nkovcKOqWNE`
  - Disclosure:

    `WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgIm5hdGlvbmFsaXRpZXMiLCBbIkRFIl1d`
  - Contents:

    `["nPuoQnkRFq3BIeAm7AnXFA", "nationalities", ["DE"]]`
- Claim `birthdate`:
  - SHA-256 Hash:

    `WNA-UNK7F_zhsAb9syWO6IIQ1uHlTmOU8r8CvJ0cIMk`
  - Disclosure:

    `WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImJpcnRoZGF0ZSIsICIxOTU2LTAxLTI4Il0`
  - Contents:

    `["5bPs1IquZNa0hkaFzzzZNw", "birthdate", "1956-01-28"]`
- Claim `place_of_birth`:
  - SHA-256 Hash:

    `RiOiCn6_w5ZHaadkQMrcQJf0Jte5RwurRs54231DTlo`
  - Disclosure:

    `WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13IiwgInBsYWNlX29mX2JpcnRoIiwgeyJjb3VudHJ5Ijog IklTIiwgImxvY2FsaXR5IjogIlx1MDBkZXlra3ZhYlx1MDBlNmphcmtsYXVzdHVyIn1d`
  - Contents:

    `["5a2W0_NrlEZzfqmk_7Pq-w", "place_of_birth", {"country": "IS", "locality": "\u00deykkvab\u00e6jarklaustur"}]`
- Claim `address`:
  - SHA-256 Hash:

    `_O-wJiH3enSB4ROHntToQT8JmLtz-mhO2f1c89XoerQ`
  - Disclosure:

    `WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImFkZHJlc3MiLCB7ImxvY2FsaXR5IjogIk1heHN0 YWR0IiwgInBvc3RhbF9jb2RlIjogIjEyMzQ0IiwgImNvdW50cnkiOiAiREUiLCAic3RyZWV0X2Fk ZHJlc3MiOiAiV2VpZGVuc3RyYVx1MDBkZmUgMjIifV0`
  - 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:

    `WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImJpcnRoX21pZGRsZV9uYW1lIiwgIlRpbW90aGV1cyJd`
  - Contents:

    `["HbQ4X8srVW3QDxnIJdqyOA", "birth_middle_name", "Timotheus"]`
- Claim `salutation`:
  - SHA-256 Hash:

    `-aSznId9mWM8ocuQolCllsxVggq1-vHW4OtnhUtVmWw`
  - Disclosure:

    `WyJDOUdTb3VqdmlKcXVFZ1lmb2pDYjFBIiwgInNhbHV0YXRpb24iLCAiRHIuIl0`
  - Contents:

    `["C9GSoujviJquEgYfojCb1A", "salutation", "Dr."]`
- Claim `msisdn`:
  - SHA-256 Hash:

    `IKbrYNn3vA7WEFrysvbdBJjDDU_EvQIr0W18vTRpUSg`
  - Disclosure:

    `WyJreDVrRjE3Vi14MEptd1V4OXZndnR3IiwgIm1zaXNkbiIsICI0OTEyMzQ1Njc4OSJd`
  - Contents:

    `["kx5kF17V-x0JmwUx9vgvtw", "msisdn", "49123456789"]`

The following is a presentation of the SD-JWT:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJfc2QiOiBb
Ii1hU3puSWQ5bVdNOG9jdVFvbENsbHN4VmdncTEtdkhXNE90bmhVdFZtV3ciLCAiSUti
cllObjN2QTdXRUZyeXN2YmRCSmpERFVfRXZRSXIwVzE4dlRScFVTZyIsICJvdGt4dVQx
NG5CaXd6TkozTVBhT2l0T2w5cFZuWE9hRUhhbF94a3lOZktJIl0sICJpc3MiOiAiaHR0
cHM6Ly9pc3N1ZXIuZXhhbXBsZS5jb20iLCAiaWF0IjogMTY4MzAwMDAwMCwgImV4cCI6
IDE4ODMwMDAwMDAsICJ2ZXJpZmllZF9jbGFpbXMiOiB7InZlcmlmaWNhdGlvbiI6IHsi
X3NkIjogWyI3aDRVRTlxU2N2REtvZFhWQ3VvS2ZLQkpwVkJmWE1GX1RtQUdWYVplM1Nj
IiwgInZUd2UzcmFISUZZZ0ZBM3hhVUQyYU14Rno1b0RvOGlCdTA1cUtsT2c5THciXSwg
InRydXN0X2ZyYW1ld29yayI6ICJkZV9hbWwiLCAiZXZpZGVuY2UiOiBbeyIuLi4iOiAi
dFlKMFREdWN5WlpDUk1iUk9HNHFSTzV2a1BTRlJ4RmhVRUxjMThDU2wzayJ9XX0sICJj
bGFpbXMiOiB7Il9zZCI6IFsiUmlPaUNuNl93NVpIYWFka1FNcmNRSmYwSnRlNVJ3dXJS
czU0MjMxRFRsbyIsICJTXzQ5OGJicEt6QjZFYW5mdHNzMHhjN2NPYW9uZVJyM3BLcjdO
ZFJtc01vIiwgIldOQS1VTks3Rl96aHNBYjlzeVdPNklJUTF1SGxUbU9VOHI4Q3ZKMGNJ
TWsiLCAiV3hoX3NWM2lSSDliZ3JUQkppLWFZSE5DTHQtdmpoWDFzZC1pZ09mXzlsayIs
ICJfTy13SmlIM2VuU0I0Uk9IbnRUb1FUOEptTHR6LW1oTzJmMWM4OVhvZXJRIiwgImh2
RFhod21HY0pRc0JDQTJPdGp1TEFjd0FNcERzYVUwbmtvdmNLT3FXTkUiXX19LCAiX3Nk
X2FsZyI6ICJzaGEtMjU2In0.QoWYWtikm-AtjmPnNVshbGXQl5raEz15PByTmZwfTQg9
W2O3oR6j2tMmysTZZawdo6mNLR_PsZSI25qrUpiNTg~WyIyR0xDNDJzS1F2ZUNmR2Zye
U5STjl3IiwgInRpbWUiLCAiMjAxMi0wNC0yM1QxODoyNVoiXQ~WyJQYzMzSk0yTGNoY1
VfbEhnZ3ZfdWZRIiwgeyJfc2QiOiBbIjl3cGpWUFd1RDdQSzBuc1FETDhCMDZsbWRnVj
NMVnliaEh5ZFFwVE55TEkiLCAiRzVFbmhPQU9vVTlYXzZRTU52ekZYanBFQV9SYy1BRX
RtMWJHX3djYUtJayIsICJJaHdGcldVQjYzUmNacTl5dmdaMFhQYzdHb3doM08ya3FYZU
JJc3dnMUI0IiwgIldweFE0SFNvRXRjVG1DQ0tPZURzbEJfZW11Y1lMejJvTzhvSE5yMW
JFVlEiXX1d~WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgIm1ldGhvZCIsICJwaXBwI
l0~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImdpdmVuX25hbWUiLCAiTWF4Il0~W
yJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImZhbWlseV9uYW1lIiwgIk1cdTAwZmNsb
GVyIl0~WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImFkZHJlc3MiLCB7ImxvY2Fsa
XR5IjogIk1heHN0YWR0IiwgInBvc3RhbF9jb2RlIjogIjEyMzQ0IiwgImNvdW50cnki0
iAiREUiLCAic3RyZWV0X2FkZHJlc3MiOiAiV2VpZGVuc3RyYVx1MDBkZmUgMjIifV0~
```

The Verifier will have this Processed SD-JWT Payload available after validation:

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

A.3. SD-JWT-Based Verifiable Credentials (SD-JWT VC)

This example shows how the artifacts defined in this specification could be used in the context of SD-JWT-based Verifiable Credentials (SD-JWT VC) [SD-JWT-VC] to represent a hypothetical identity credential with the data of a fictional German citizen.

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

The following citizen data is the input JWT Claims Set:

```
{
  "vct": "urn:eudi:pid:de:1",
  "iss": "https://pid-issuer.bund.de.example",
  "given_name": "Erika",
  "family_name": "Mustermann",
  "birthdate": "1963-08-12",
  "address": {
    "street_address": "Heidestraße 17",
    "locality": "Köln",
    "postal_code": "51147",
    "country": "DE"
  },
  "nationalities": [
    "DE"
  ],
  "sex": 2,
  "birth_family_name": "Gabler",
  "place_of_birth": {
    "locality": "Berlin",
    "country": "DE"
  },
  "age_equal_or_over": {
    "12": true,
    "14": true,
    "16": true,
    "18": true,
    "21": true,
    "65": false
  },
  "age_in_years": 62,
  "age_birth_year": 1963,
  "issuance_date": "2020-03-11",
  "expiry_date": "2030-03-12",
  "issuing_authority": "DE",
  "issuing_country": "DE"
}
```

The following is the issued SD-JWT:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImRjK3NkLWp3dCJ9.eyJfc2QiOiBbIjBIWm1
uU0lQejMzN2tTV2U3QzM0bC0tODhnekppLWVCSjJWel9ISndBVGciLCAiMUNybjAzV21
VZVJXcDR6d1B2dkNLWGw5WmFRcC1jZFFWX2dIZGFHU1dvdyIsICIycjAwOWR6dkh1VnJ
XclJYVDVrSk1tSG5xRUhIbldlME1MVlp3OFBBVEI4IiwgIjZaTklTRHN0NjJ5bWxyT0F
rYWRqZEQ1WnVsVDVBMjk5Sjc4U0xoTV9fT3MiLCAiNzhqZzc3LUdZQmVYOElRZm9FTFB
5TDBEWVBkbWZabzBKZ1ZpVjBfbEtDTSIsICI5MENUOEFhQlBibjVYOG5SWGtlc2p1MWk
wQnFoV3FaM3dxRDRqRi1xREdrIiwgIkkwMGZjRlVvRFhDdWNwNXl5MnVqcVBzc0RWR2F
XTmlVbGlOel9hd0QwZ2MiLCAiS2pBWGdBQTlONVdIRUR0UkloNHU1TW4xWnNXaXhoaFd
BaVgtQTRRaXdnQSIsICJMYWk2SVU2ZDdHUWFnWFI3QXZHVHJuWGdTbGQzejhFSWdfZnY
zZk9aMVdnIiwgIkxlemphYlJxaVpPWHpFWW1WWmY4Uk1pOXhBa2QzX00xTFo4VTdFNHM
zdTQiLCAiUlR6M3FUbUZOSGJwV3JyT01aUzQxRjQ3NGtGcVJ2M3ZJUHF0aDZQVWhsTSI
sICJXMTRYSGJVZmZ6dVc0SUZNanBTVGIxbWVsV3hVV2Y0Tl9vMmxka2tJcWM4IiwgIld
UcEk3UmNNM2d4WnJ1UnBYemV6U2JrYk9yOTNQVkZ2V3g4d29KM2oxY0UiLCAiX29oSlZ
JUUlCc1U0dXBkTlM0X3c0S2IxTUhxSjBMOXFMR3NoV3E2SlhRcyIsICJ5NTBjemMwSVN
DaHlfYnNiYTFkTW9VdUFPUTVBTW1PU2ZHb0VlODF2MUZVIl0sICJpc3MiOiAiaHR0cHM
```

```
6Ly9waWQtaXNzdWVyLmJ1bmQuZGUuZXhhbXBsZSIsICJpYXQiOiAxNjgzMDAwMDAwLCA
iZXhwIjogMTg4MzAwMDAwMCwgInZjdCI6ICJ1cm46ZXVkaTpwaWQ6ZGU6MSIsICJfc2R
fYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnY
iOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbER
sczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U
2dDRqVDlGMkhaUSJ9fX0.ZOZQTqmq8X1mCyFXi0wbV8xjctX1AlEa5TkdnkKOyWvLfW4
0XDb5oj9tzkgwff5s44IDnrfAdgLtmTcojs97_Q~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5S
Tjl3IiwgImdpdmVuX25hbWUiLCAiRXJpa2EiXQ~WyJlbHVWNU9nM2dTTklOEVZbnN4Q
V9BIiwgImZhbWlseV9uYW1lIiwgIk11c3Rlcm1hbm4iXQ~WyI2SWo3dE0tYTVpVlBHYm
9TNXRtdlZBIiwgImJpcnRoZGF0ZSIsICIxOTYzLTA4LTEyIl0~WyJlSThaV205UW5LUH
BOUGVOZW5IZGhRIiwgInN0cmVldF9hZGRyZXNzIiwgIkhlaWRlc3RyYVx1MDBkZmUgMT
ciXQ~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImxvY2FsaXR5IiwgIktcdTAwZjZ
sbiJd~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgInBvc3RhbF9jb2RlIiwgIjUxMT
Q3Il0~WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImNvdW50cnkiLCAiREUiXQ~WyJ
HMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImFkZHJlc3MiLCB7Il9zZCI6IFsiQUxaRVJ
zU241V05pRVhkQ2tzVzhJNXFRdzNfTnBBblJxcFNBWkR1ZGd3OCIsICJEX19XX3VZY3Z
SejN0dlVuSUp2QkRIaVRjN0NfX3FIZDB4Tkt3SXNfdzlrIiwgImVCcENYVTFKNWRoSDJ
nNHQ4UVlOVzVFeFM5QXhVVmJsVW9kb0xZb1BobzAiLCAieE9QeTktZ0pBTEs2VWJXS0Z
MUjg1Y09CeVVEM0FiTndGZzNJM1lmUUVfSSJdfV0~WyJsa2x4RjVqTVlsR1RQVW92TU5
JdkNBIiwgIm5hdGlvbmFsaXRpZXMiLCBbIkRFIl1d~WyJuUHVvUW5rUkZxM0JJZUFtN0
FuWEZBIiwgInNleCIsIDJd~WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImJpcnRoX
2ZhbWlseV9uYW1lIiwgIkdhYmxlciJd~WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13Iiwg
ImxvY2FsaXR5IiwgIkJlcmxpbiJd~WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImN
vdW50cnkiLCAiREUiXQ~WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgInBsYWNlX29m
X2JpcnRoIiwgeyJfc2QiOiBbIktVVmlhYUxuWTVqU01MOTBHMjlPT0xFTlBiYlhmaFNq
U1BNalphR2t4QUUiLCAiWWJzVDBTNzZWcVhDVnNkMWpVU2x3S1BEZ21BTGVCMXVaY2xG
SFhmLVVTUSJdfV0~WyJDOUdTb3VqdmlKcXVFZ1lmb2pDYjFBIiwgIjEyIiwgdHJ1ZV0~
WyJreDVrRjE3Vi14MEptd1V4OXZndnR3IiwgIjE0IiwgdHJ1ZV0~WyJIM28xdXN3UDc2
MEZpMnllR2RWQ0VRIiwgIjE2IiwgdHJ1ZV0~WyJPQktsVFZsdkxnLUFkd3FZR2JQOFpB
IiwgIjE4IiwgdHJ1ZV0~WyJNMEpiNTd0NDF1YnJrU3V5ckRUM3hBIiwgIjIxIiwgdHJ1
ZV0~WyJEc210S05ncFY0ZEFIcGpyY2Fvc0F3IiwgIjY1IiwgZmFsc2Vd~WyJlSzVvNXB
IZmd1cFBwbHRqMXFoQUp3IiwgImFnZV9lcXVhbF9vcl9vdmVyIiwgeyJfc2QiOiBbIjF
0RWl5elBSWU9Lc2Y3U3NZR01nUFpLc09UMWxRWlJ4SFhBMHI1X0J3a2siLCAiQ1ZLbmx
5NVA5MHlKczNFd3R4UWlPdFVjemFYQ1lOQTRJY3pSYW9ock1EZyIsICJhNDQtZzJHcjh
fM0FtSncyWFo4a0kxeTBRel96ZTlpT2NXMlczUkxwWEdnIiwgImdrdnkwRnV2QkJ2ajB
oczJaTnd4Y3FPbGY4bXUyLWtDRTctTmIyUXh1QlUiLCAiaHJZNEhubUY1YjVKd0M5ZVR
6YUZDVWNlSVFBYUlkaHJxVVhRTkNXYmZaSSIsICJ5NlNGclZGUnlxNTBJYlJKdmlUWnF
xalFXejB0TGl1Q21NZU8wS3FhekdJIl19XQ~WyJqN0FEZGIwVVZiMExpMGNpUGNQMGV3
IiwgImFnZV9pbl95ZWFycyIsIDYyXQ~WyJXcHhKckZ1WDh1U2kycDRodDA5anZ3IiwgI
mFnZV9iaXJ0aF95ZWFyIiwgMTk2M10~WyJhdFNtRkFDWU1iSlZLRDA1bzNKZ3RRIiwgI
mlzc3VhbmNlX2RhdGUiLCAiMjAyMC0wMy0xMSJd~WyI0S3lSMzJvSVp0LXprV3ZGcWJV
TEtnIiwgImV4cGlyeV9kYXRlIiwgIjIwMzAtMDMtMTIiXQ~WyJjaEJDc3loeWgtSjg2S
S1hd1FEaUNRIiwgImlzc3VpbmdfYXV0aG9yaXR5IiwgIkRFIl0~WyJmbE5QMW5jTXo5T
GctYzlxTUl6XzlnIiwgImlzc3VpbmdfY291bnRyeSIsICJERSJd~
```

The following payload is used for the SD-JWT:

```
{
  "_sd": [
    "0HZmnSIPz337kSWe7C34l--88gzJi-eBJ2Vz_HJwATg",
    "1Crn03WmUeRWp4zwPvvCKXl9ZaQp-cdQV_gHdaGSWow",
    "2r009dzvHuVrWrRXT5kJMmHnqEHHnWe0MLVZw8PATB8",
    "6ZNISDst62ymlrOAkadjdD5ZulT5A299J78SLhM__Os",
    "78jg77-GYBeX8IQfoELPyL0DYPdmfZo0JgViV0_lKCM",
    "90CT8AaBPbn5X8nRXkesju1i0BqhWqZ3wqD4jF-qDGk",
    "I00fcFUoDXCucp5yy2ujqPssDVGaWNiUliNz_awD0gc",
    "KjAXgAA9N5WHEDtRIh4u5Mn1ZsWixhhWAiX-A4QiwgA",
    "Lai6IU6d7GQagXR7AvGTrnXgSld3z8EIg_fv3fOZ1Wg",
    "LezjabRqiZOXzEYmVZf8RMi9xAkd3_M1LZ8U7E4s3u4",
    "RTz3qTmFNHbpWrrOMZS41F474kFqRv3vIPqth6PUhlM",
    "W14XHbUffzuW4IFMjpSTb1melWxUWf4N_o2ldkkIqc8",
    "WTpI7RcM3gxZruRpXzezSbkbOr93PVFvWx8woJ3j1cE",
    "_ohJVIQIBsU4updNS4_w4Kb1MHqJ0L9qLGshWq6JXQs",
    "y50czc0ISChy_bsba1dMoUuAOQ5AMmOSfGoEe81v1FU"
  ],
  "iss": "https://pid-issuer.bund.de.example",
  "iat": 1683000000,
  "exp": 1883000000,
  "vct": "urn:eudi:pid:de:1",
  "_sd_alg": "sha-256",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```

The digests in the SD-JWT payload reference the following Disclosures:

- Claim `given_name`:
  - SHA-256 Hash:

    `0HZmnSIPz337kSWe7C34l--88gzJi-eBJ2Vz_HJwATg`
  - Disclosure:

    `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiRXJpa2EiXQ`
  - Contents:

    `["2GLC42sKQveCfGfryNRN9w", "given_name", "Erika"]`
- Claim `family_name`:
  - SHA-256 Hash:

    `I00fcFUoDXCucp5yy2ujqPssDVGaWNiUliNz_awD0gc`
  - Disclosure:

    `WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIk11c3Rlcm1hbm4iXQ`

- Contents:

  ```
  ["eluV5Og3gSNII8EYnsxA_A", "family_name", "Mustermann"]
  ```

- Claim `birthdate`:
  - SHA-256 Hash:

    ```
    Lai6IU6d7GQagXR7AvGTrnXgSld3z8EIg_fv3fOZ1Wg
    ```

  - Disclosure:

    ```
    WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImJpcnRoZGF0ZSIsICIxOTYzLTA4LTEyIl0
    ```

  - Contents:

    ```
    ["6Ij7tM-a5iVPGboS5tmvVA", "birthdate", "1963-08-12"]
    ```

- Claim `street_address`:
  - SHA-256 Hash:

    ```
    ALZERsSn5WNiEXdCksW8I5qQw3_NpAnRqpSAZDudgw8
    ```

  - Disclosure:

    ```
    WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgInN0cmVldF9hZGRyZXNzIiwgIkhlaWRlc3RyYVx1MDBkZmUgMTciXQ
    ```

  - Contents:

    ```
    ["eI8ZWm9QnKPpNPeNenHdhQ", "street_address", "Heidestra\u00dfe 17"]
    ```

- Claim `locality`:
  - SHA-256 Hash:

    ```
    D__W_uYcvRz3tvUnIJvBDHiTc7C__qHd0xNKwIs_w9k
    ```

  - Disclosure:

    ```
    WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImxvY2FsaXR5IiwgIktcdTAwZjZsbiJd
    ```

  - Contents:

    ```
    ["Qg_O64zqAxe412a108iroA", "locality", "K\u00f6ln"]
    ```

- Claim `postal_code`:
  - SHA-256 Hash:

    ```
    xOPy9-gJALK6UbWKFLR85cOByUD3AbNwFg3I3YfQE_I
    ```

  - Disclosure:

    ```
    WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgInBvc3RhbF9jb2RlIiwgIjUxMTQ3Il0
    ```

  - Contents:

    ```
    ["AJx-095VPrpTtN4QMOqROA", "postal_code", "51147"]
    ```

- Claim `country`:
  - SHA-256 Hash:

    `eBpCXU1J5dhH2g4t8QYNW5ExS9AxUVblUodoLYoPho0`
  - Disclosure:

    `WyJQYzMzSk0yTGNoY1VfbEhnZ3ZfdWZRIiwgImNvdW50cnkiLCAiREUiXQ`
  - Contents:

    `["Pc33JM2LchcU_lHggv_ufQ", "country", "DE"]`
- Claim `address`:
  - SHA-256 Hash:

    `RTz3qTmFNHbpWrrOMZS41F474kFqRv3vIPqth6PUhlM`
  - Disclosure:

    `WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiwgImFkZHJlc3MiLCB7Il9zZCI6IFsiQUxaRVJzU241V05pRVhkQ2tzVzhJNXFRdzNfTnBBblJxcFNBWkR1ZGd3OCIsICJEX19XX3VZY3ZSejN0dlVuSUp2QkRIaVRjN0NfX3FIZDB4Tkt3SXNfdzlrIiwgImVCcENYVTFKNWRoSDJnNHQ4UVlOVzVFeFM5QXhVVmJsVW9kb0xZb1BobzAiLCAieE9QeTktZ0pBTEs2VWJXS0ZMUjg1Y09CeVVEM0FiTndGZzNJM1lmUUVfSSJdfV0`
  - Contents:

    `["G02NSrQfjFXQ7Io09syajA", "address", {"_sd": ["ALZERsSn5WNiEXdCksW8I5qQw3_NpAnRqpSAZDudgw8", "D__W_uYcvRz3tvUnIJvBDHiTc7C__qHd0xNKwIs_w9k", "eBpCXU1J5dhH2g4t8QYNW5ExS9AxUVblUodoLYoPho0", "xOPy9-gJALK6UbWKFLR85cOByUD3AbNwFg3I3YfQE_I"]}]`
- Claim `nationalities`:
  - SHA-256 Hash:

    `y50czc0ISChy_bsba1dMoUuAOQ5AMmOSfGoEe81v1FU`
  - Disclosure:

    `WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIm5hdGlvbmFsaXRpZXMiLCBbIkRFIl1d`
  - Contents:

    `["lklxF5jMYlGTPUovMNIvCA", "nationalities", ["DE"]]`
- Claim `sex`:
  - SHA-256 Hash:

    `90CT8AaBPbn5X8nRXkesju1i0BqhWqZ3wqD4jF-qDGk`
  - Disclosure:

    `WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgInNleCIsIDJd`
  - Contents:

`["nPuoQnkRFq3BIeAm7AnXFA", "sex", 2]`

- Claim `birth_family_name`:
  - SHA-256 Hash:

    `KjAXgAA9N5WHEDtRIh4u5Mn1ZsWixhhWAiX-A4QiwgA`

  - Disclosure:

    `WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImJpcnRoX2ZhbWlseV9uYW1lIiwgIkdhYmxlciJd`

  - Contents:

    `["5bPs1IquZNa0hkaFzzzZNw", "birth_family_name", "Gabler"]`

- Claim `locality`:
  - SHA-256 Hash:

    `KUViaaLnY5jSML90G29OOLENPbbXfhSjSPMjZaGkxAE`

  - Disclosure:

    `WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13IiwgImxvY2FsaXR5IiwgIkJlcmxpbiJd`

  - Contents:

    `["5a2W0_NrlEZzfqmk_7Pq-w", "locality", "Berlin"]`

- Claim `country`:
  - SHA-256 Hash:

    `YbsT0S76VqXCVsd1jUSlwKPDgmALeB1uZclFHXf-USQ`

  - Disclosure:

    `WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImNvdW50cnkiLCAiREUiXQ`

  - Contents:

    `["y1sVU5wdfJahVdgwPgS7RQ", "country", "DE"]`

- Claim `place_of_birth`:
  - SHA-256 Hash:

    `1Crn03WmUeRWp4zwPvvCKXl9ZaQp-cdQV_gHdaGSWow`

  - Disclosure:

    `WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgInBsYWNlX29mX2JpcnRoIiwgeyJfc2QiOiBbIktV VmlhYUxuWTVqU01MOTBHMjlPT0xFTlBiYlhmaFNqU1BNalphR2t4QUUiLCAiWWJzVDBTNzZWcVhD VnNkMWpVU2x3S1BEZ21BTGVCMXVaY2xGSFhmLVVTUSJdfV0`

  - Contents:

    `["HbQ4X8srVW3QDxnIJdqyOA", "place_of_birth", {"_sd": ["KUViaaLnY5jSML90G29OOLENPbbXfhSjSPMjZaGkxAE", "YbsT0S76VqXCVsd1jUSlwKPDgmALeB1uZclFHXf-USQ"]}]`

- Claim 12:
  - SHA-256 Hash:
    `gkvy0FuvBBvj0hs2ZNwxcqOlf8mu2-kCE7-Nb2QxuBU`
  - Disclosure:
    `WyJDOUdTb3VqdmlKcXVFZ1lmb2pDYjFBIiwgIjEyIiwgdHJ1ZV0`
  - Contents:
    `["C9GSoujviJquEgYfojCb1A", "12", true]`
- Claim 14:
  - SHA-256 Hash:
    `y6SFrVFRyq50IbRJviTZqqjQWz0tLiuCmMeO0KqazGI`
  - Disclosure:
    `WyJreDVrRjE3Vi14MEptd1V4OXZndnR3IiwgIjE0IiwgdHJ1ZV0`
  - Contents:
    `["kx5kF17V-x0JmwUx9vgvtw", "14", true]`
- Claim 16:
  - SHA-256 Hash:
    `hrY4HnmF5b5JwC9eTzaFCUceIQAaIdhrqUXQNCWbfZI`
  - Disclosure:
    `WyJIM28xdXN3UDc2MEZpMnllR2RWQ0VRIiwgIjE2IiwgdHJ1ZV0`
  - Contents:
    `["H3o1uswP760Fi2yeGdVCEQ", "16", true]`
- Claim 18:
  - SHA-256 Hash:
    `CVKnly5P90yJs3EwtxQiOtUczaXCYNA4IczRaohrMDg`
  - Disclosure:
    `WyJPQktsVFZsdkxnLUFkd3FZR2JQOFpBIiwgIjE4IiwgdHJ1ZV0`
  - Contents:
    `["OBKlTVlvLg-AdwqYGbP8ZA", "18", true]`
- Claim 21:
  - SHA-256 Hash:
    `1tEiyzPRYOKsf7SsYGMgPZKsOT1lQZRxHXA0r5_Bwkk`
  - Disclosure:

WyJNMEpiNTd0NDF1YnJrU3V5ckRUM3hBIiwgIjIxIiwgdHJ1ZV0

- Contents:

["M0Jb57t41ubrkSuyrDT3xA", "21", true]

- Claim 65:
  - SHA-256 Hash:

a44-g2Gr8_3AmJw2XZ8kI1y0Qz_ze9iOcW2W3RLpXGg

  - Disclosure:

WyJEc210S05ncFY0ZEFIcGpyY2Fvc0F3IiwgIjY1IiwgZmFsc2Vd

  - Contents:

["DsmtKNgpV4dAHpjrcaosAw", "65", false]

- Claim age_equal_or_over:
  - SHA-256 Hash:

2r009dzvHuVrWrRXT5kJMmHnqEHHnWe0MLVZw8PATB8

  - Disclosure:

WyJlSzVvNXBIZmd1cFBwbHRqMXFoQUp3IiwgImFnZV9lcXVhbF9vcl9vdmVyIiwgeyJfc2QiOiBb
IjF0RWl5elBSWU9Lc2Y3U3NZR01nUFpLc09UMWxRWlJ4SFhBMHI1X0J3a2siLCAiQ1ZLbmx5NVA5
MHlKczNFd3R4UWlPdFVjemFYQ1lOQTRJY3pSYW9ock1EZyIsICJhNDQtZzJHcjhfM0FtSncyWFo4
a0kxeTBRel96ZTlpT2NXMlczUkxwWEdnIiwgImdrdnkwRnV2QkJ2ajBoczJaTnd4Y3FPbGY4bXUy
LWtDRTctTmIyUXh1QlUiLCAiaHJZNEhubUY1YjVKd0M5ZVR6YUZDVWNlSVFBYUlkaHJxVVhRTkNX
YmZaSSIsICJ5NlNGclZGUnlxNTBJYlJKdmlUWnFxalFXejB0TGl1Q21NZU8wS3FhekdJIl19XQ

  - Contents:

["eK5o5pHfgupPpltj1qhAJw", "age_equal_or_over", {"_sd":
["1tEiyzPRYOKsf7SsYGMgPZKsOT1lQZRxHXA0r5_Bwkk",
"CVKnly5P90yJs3EwtxQiOtUczaXCYNA4IczRaohrMDg", "a44-
g2Gr8_3AmJw2XZ8kI1y0Qz_ze9iOcW2W3RLpXGg", "gkvy0FuvBBvj0hs2ZNwxcqOlf8mu2-
kCE7-Nb2QxuBU", "hrY4HnmF5b5JwC9eTzaFCUceIQAaIdhrqUXQNCWbfZI",
"y6SFrVFRyq50IbRJviTZqqjQWz0tLiuCmMeO0KqazGI"]}]

- Claim age_in_years:
  - SHA-256 Hash:

WTpI7RcM3gxZruRpXzezSbkbOr93PVFvWx8woJ3j1cE

  - Disclosure:

WyJqN0FEZGIwVVZiMExpMGNpUGNQMGV3IiwgImFnZV9pbl95ZWFycyIsIDYyXQ

  - Contents:

["j7ADdb0UVb0Li0ciPcP0ew", "age_in_years", 62]

- Claim `age_birth_year`:
  - SHA-256 Hash:

    `LezjabRqiZOXzEYmVZf8RMi9xAkd3_M1LZ8U7E4s3u4`
  - Disclosure:

    `WyJXcHhKckZ1WDh1U2kycDRodDA5anZ3IiwgImFnZV9iaXJ0aF95ZWFyIiwgMTk2M10`
  - Contents:

    `["WpxJrFuX8uSi2p4ht09jvw", "age_birth_year", 1963]`
- Claim `issuance_date`:
  - SHA-256 Hash:

    `W14XHbUffzuW4IFMjpSTb1melWxUWf4N_o2ldkkIqc8`
  - Disclosure:

    `WyJhdFNtRkFDWU1iSlZLRDA1bzNKZ3RRIiwgImlzc3VhbmNlX2RhdGUiLCAiMjAyMC0wMy0xMSJd`
  - Contents:

    `["atSmFACYMbJVKD05o3JgtQ", "issuance_date", "2020-03-11"]`
- Claim `expiry_date`:
  - SHA-256 Hash:

    `78jg77-GYBeX8IQfoELPyL0DYPdmfZo0JgViV0_lKCM`
  - Disclosure:

    `WyI0S3lSMzJvSVp0LXprV3ZGcWJVTEtnIiwgImV4cGlyeV9kYXRlIiwgIjIwMzAtMDMtMTIiXQ`
  - Contents:

    `["4KyR32oIZt-zkWvFqbULKg", "expiry_date", "2030-03-12"]`
- Claim `issuing_authority`:
  - SHA-256 Hash:

    `6ZNISDst62ymlrOAkadjdD5ZulT5A299J78SLhM__Os`
  - Disclosure:

    `WyJjaEJDc3loeWgtSjg2SS1hd1FEaUNRIiwgImlzc3VpbmdfYXV0aG9yaXR5IiwgIkRFIl0`
  - Contents:

    `["chBCsyhyh-J86I-awQDiCQ", "issuing_authority", "DE"]`
- Claim `issuing_country`:
  - SHA-256 Hash:

    `_ohJVIQIBsU4updNS4_w4Kb1MHqJ0L9qLGshWq6JXQs`
  - Disclosure:

`WyJmbE5QMW5jTXo5TGctYzlxTUl6XzlnIiwgImlzc3VpbmdfY291bnRyeSIsICJERSJd`

- Contents:

`["flNP1ncMz9Lg-c9qMIz_9g", "issuing_country", "DE"]`

The following is an example of an SD-JWT+KB that discloses only nationality and the fact that the person is over 18 years old:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImRjK3NkLWp3dCJ9.eyJfc2QiOiBbIjBIWm1
uU0lQejMzN2tTV2U3QzM0bC0tODhnekppLWVCSjJWel9ISndBVGciLCAiMUNybjAzV21
VZVJXcDR6d1B2dkNLWGw5WmFRcC1jZFFWX2dIZGFHU1dvdyIsICIycjAwOWR6dkh1VnJ
XclJYVDVrSk1tSG5xRUhIbldlME1MVlp3OFBBVEI4IiwgIjZaTklTRHN0NjJ5bWxyT0F
rYWRqZEQ1WnVsVDVBMjk5Sjc4U0xoTV9fT3MiLCAiNzhqZzc3LUdZQmVYOElRZm9FTFB
5TDBEWVBkbWZabzBKZ1ZpVjBfbEtDTSIsICI5MENUOEFhQlBibjVYOG5SWGtlc2p1MWk
wQnFoV3FaM3dxRDRqRi1xREdrIiwgIkkwMGZjRlVvRFhDdWNwNXl5MnVqcVBzc0RWR2F
XTmlVbGlOel9hd0QwZ2MiLCAiS2pBWGdBQTlONVdIRUR0UkloNHU1TW4xWnNXaXhoaFd
BaVgtQTRRaXdnQSIsICJMYWk2SVU2ZDdHUWFnWFI3QXZHVHJuWGdTbGQzejhFSWdfZnY
zZk9aMVdnIiwgIkxlemphYlJxaVpPWHpFWW1WWmY4Uk1pOXhBa2QzX00xTFo4VTdFNHM
zdTQiLCAiUlR6M3FUbUZOSGJwV3JyT01aUzQxRjQ3NGtGcVJ2M3ZJUHF0aDZQVWhsTSI
sICJXMTRYSGJVZmZ6dVc0SUZNanBTVGIxbWVsV3hVV2Y0Tl9vMmxka2tJcWM4IiwgIld
UcEk3UmNNM2d4WnJ1UnBYemV6U2JrYk9yOTNQVkZ2V3g4d29KM2oxY0UiLCAiX29oSlZ
JUUlCc1U0dXBkTlM0X3c0S2IxTUhxSjBMOXFMR3NoV3E2SlhRcyIsICJ5NTBjemMwSVN
DaHlfYnNiYTFkTW9VdUFPUTVBTW1PU2ZHb0VlODF2MUZVIl0sICJpc3MiOiAiaHR0cHM
6Ly9waWQtaXNzdWVyLmJ1bmQuZGUuZXhhbXBsZSIsICJpYXQiOiAxNjgzMDAwMDAwLCA
iZXhwIjogMTg4MzAwMDAwMCwgInZjdCI6ICJ1cm46ZXVkaTpwaWQ6ZGU6MSIsICJfc2R
fYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnY
iOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbER
sczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U
2dDRqVDlGMkhaUSJ9fX0.ZOZQTqmq8X1mCyFXi0wbV8xjctX1AlEa5TkdnkKOyWvLfW4
0XDb5oj9tzkgwff5s44IDnrfAdgLtmTcojs97_Q~WyJlSzVvNXBIZmd1cFBwbHRqMXFo
QUp3IiwgImFnZV9lcXVhbF9vcl9vdmVyIiwgeyJfc2QiOiBbIjF0RWl5elBSWU9Lc2Y3
U3NZR01nUFpLc09UMWxRWlJ4SFhBMHI1X0J3a2siLCAiQ1ZLbmx5NVA5MHlKczNFd3R4
UWlPdFVjemFYQ1lOQTRJY3pSYW9ock1EZyIsICJhNDQtZzJHcjhfM0FtSncyWFo4a0kx
eTBRel96ZTlpT2NXMlczUkxxWEdnIiwgImdrdnkwRnV2QkJ2ajBoczJaTnd4Y3FPbGY4
bXUyLWtDRTctTmIyUXh1QlUiLCAiaHJZNEhubUY1YjVKd0M5ZVR6YUZDVWNlSVFBYUlk
aHJxVVhRTkNXYmZaSSIsICJ5NlNGclZGUnlxNTBJYlJKdmlUWnFxalFXejB0TGl1Q21N
ZU8wS3FhekdJIl19XQ~WyJPQktsVFZsdkxnLUFkd3FZR2JQOFpBIiwgIjE4IiwgdHJ1Z
V0~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgIm5hdGlvbmFsaXRpZXMiLCBbIkRFI
l1d~eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZSI6ICIxMjM
0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL3ZlcmlmaWVyLmV4YW1wbGUub3JnIiwgIml
hdCI6IDE3NDg1MzcyNDQsICJzZF9oYXNoIjogIlBqTVlmTTA3VmJKZE14TElsdXZSTmI
4OEpGbGpTWDRuLUc0M1VjX0JTUk0ifQ.f3TeS_1BWEG78EbIJRh5wgv8nYumk7euzu6x
gbgpNB4pbQQqgRPWK-vQjlhhgU1EFGZ9LFakFX_0mgul1G_3mw
```

This is the payload of the corresponding Key Binding JWT:

```
{
  "nonce": "1234567890",
  "aud": "https://verifier.example.org",
  "iat": 1748537244,
  "sd_hash": "PjMYfM07VbJdMxLIluvRNb88JFljSX4n-G43Uc_BSRM"
}
```

After validation, the Verifier will have the following Processed SD-JWT Payload available for further handling:

```
{
  "iss": "https://pid-issuer.bund.de.example",
  "iat": 1683000000,
  "exp": 1883000000,
  "vct": "urn:eudi:pid:de:1",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  },
  "age_equal_or_over": {
    "18": true
  },
  "nationalities": [
    "DE"
  ]
}
```

A.4. W3C Verifiable Credentials Data Model v2.0

This non-normative example illustrates how the artifacts defined in this specification could be used to express a W3C Verifiable Credentials Data Model v2.0 payload [VC_DATA_v2.0].

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

The following is the input JWT Claims 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:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJAY29udGV4
dCI6IFsiaHR0cHM6Ly93d3cudzMub3JnLzIwMTgvY3JlZGVudGlhbHMvdjEiLCAiaHR0
cHM6Ly93M2lkLm9yZy92YWNjaW5hdGlvbi92MSJdLCAidHlwZSI6IFsiVmVyaWZpYWJs
ZUNyZWRlbnRpYWwiLCAiVmFjY2luYXRpb25DZXJ0aWZpY2F0ZSJdLCAiaXNzdWVyIjog
Imh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlzc3VhbmNlRGF0ZSI6ICIyMDIz
LTAyLTA5VDExOjAxOjU5WiIsICJleHBpcmF0aW9uRGF0ZSI6ICIyMDI4LTAyLTA4VDEx
OjAxOjU5WiIsICJuYW1lIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRl
IiwgImRlc2NyaXB0aW9uIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRl
IiwgImNyZWRlbnRpYWxTdWJqZWN0IjogeyJfc2QiOiBbIjFWX0stOGxEUThpRlhCRlhi
Wlk5ZWhxUjRIYWJXQ2k1VDB5Ykl6WlBld3ciLCAiSnpqTGd0UDI5ZFAtQjN0ZDEyUDY3
NGdGbUsyenk4MUhNdEJnZjZDSk5XZyIsICJSMmZHYmZBMDdaX1lsa3FtTlp5bWExeHl5
eDFYc3RJaVM2QjFZYmwySlo0IiwgIlRDbXpybDdLMmdldl9kdTdwY01JeXpSTEhwLVll
Zy1GbF9jeHRyVXZQeGciLCAiVjdrSkJMSzc4VG1WRE9tcmZKN1p1VVBIdUtfMmNjN3la
UmE0cVYxdHh3TSIsICJiMGVVc3ZHUC1PRERkRm9ZNE5semxYYzN0RHNsV0p0Q0pGNzVO
dzhPal9nIiwgInpKS19lU01YandNOGRYbU1aTG5JOEZHTTA4ekozX3ViR2VFTUotNVRC
eTAiXSwgInZhY2NpbmUiOiB7Il9zZCI6IFsiMWNGNWhMd2toTU5JYXFmV0pyWEk3Tk1X
ZWRMLTlmNlkyUEE1MnlQalNaSSIsICJIaXk2V1d1ZUxENWJuMTYyOTh0UHY3R1hobWxk
TURPVG5CaS1DWmJwaE5vIiwgIkxiMDI3cTY5MWpYWGwtakM3M3ZpOGViT2o5c214M0Mt
X29nN2dBNFRCUUUiXSwgInR5cGUiOiAiVmFjY2luZSJ9LCAicmVjaXBpZW50IjogeyJf
c2QiOiBbIjFsU1FCTlkyNHEwVGg2T0d6dGhxLTctNGw2Y0FheHJZWE9HWnBlV19sbkEi
LCAiM256THE4MU0yb04wNndkdjFzaEh2T0VKVnhaNUtMbWREa0hFREpBQldFSSIsICJQ
bjFzV2kwNkc0TEpybm4tX1JUMFJiTV9IVGR4blBKUXVYMmZ6V3ZfSk9VIiwgImxGOXV6
ZHN3N0hwbEdMYzcxNFRyNFdPN01HSnphN3R0N1FGbGVDWDRJdHciXSwgInR5cGUiOiAi
VmFjY2luZVJlY2lwaWVudCJ9LCAidHlwZSI6ICJWYWNjaW5hdGlvbkV2ZW50In0sICJf
c2RfYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJj
cnYiOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxp
bERsczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVj
Q0U2dDRqVDlGMkhaUSJ9fX0.OZomvwO8iw4db89MYCeeomBVStXkT6u7G7FkicPWZnd2
_hGgr0l_u1NHgPVocuOt-m32Uu6kwtPmYFxKk0AOeA~WyIyR0xDNDJzS1F2ZUNmR2Zye
U5STjl3IiwgImF0Y0NvZGUiLCAiSjA3QlgwMyJd~WyJlbHVWNU9nM2dTTklJOEVZbnN4
QV9BIiwgIm1lZGljaW5hbFByb2R1Y3ROYW1lIiwgIkNPVklELTE5IFZhY2NpbmUgTW9k
ZXJuYSJd~WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgIm1hcmtldGluZ0F1dGhvcml
6YXRpb25Ib2xkZXIiLCAiTW9kZXJuYSBCaW90ZWNoIl0~WyJlSThaV205UW5LUHBOUGV
OZW5IZGhRIiwgIm5leHRWYWNjaW5hdGlvbkRhdGUiLCAiMjAyMS0wOC0xNlQxMzo0MDo
xMloiXQ~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImNvdW50cnlPZlZhY2NpbmF0
aW9uIiwgIkdFIl0~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwgImRhdGVPZlZhY2Np
bmF0aW9uIiwgIjIwMjEtMDYtMjNUMTM6NDA6MTJaIl0~WyJQYzMzSk0yTGNoY1VfbEhn
Z3ZfdWZRIiwgIm9yZGVyIiwgIjMvMyJd~WyJHMDJOU3JRZmpGWFE3SW8wOXN5YWpBIiw
gImdlbmRlciIsICJGZW1hbGUiXQ~WyJsa2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImJp
cnRoRGF0ZSIsICIxOTYxLTA4LTE3Il0~WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwg
ImdpdmVuTmFtZSIsICJNYXJpb24iXQ~WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgI
mZhbWlseU5hbWUiLCAiTXVzdGVybWFubiJd~WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13
IiwgImFkbWluaXN0ZXJpbmdDZW50cmUiLCAiUHJheGlzIFNvbW1lcmdhcnRlbiJd~WyJ
5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImJhdGNoTnVtYmVyIiwgIjE2MjYzODI3MzY
iXQ~WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImhlYWx0aFByb2Zlc3Npb25hbCIs
ICI4ODMxMTAwMDAwMTUzNzYiXQ~
```

The following payload is used for the SD-JWT:

```
{
  "@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",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```

The digests in the SD-JWT payload reference the following Disclosures:

- Claim `atcCode`:
  - SHA-256 Hash:

    `1cF5hLwkhMNIaqfWJrXI7NMWedL-9f6Y2PA52yPjSZI`

  - Disclosure:

WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImF0Y0NvZGUiLCAiSjA3QlgwMyJd

- Contents:

["2GLC42sKQveCfGfryNRN9w", "atcCode", "J07BX03"]

- Claim `medicinalProductName`:
  - SHA-256 Hash:

Hiy6WWueLD5bn16298tPv7GXhmldMDOTnBi-CZbphNo

  - Disclosure:

WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgIm1lZGljaW5hbFByb2R1Y3ROYW1lIiwgIkNPVklELTE5IFZhY2NpbmUgTW9kZXJuYSJd

  - Contents:

["eluV5Og3gSNII8EYnsxA_A", "medicinalProductName", "COVID-19 Vaccine Moderna"]

- Claim `marketingAuthorizationHolder`:
  - SHA-256 Hash:

Lb027q691jXXl-jC73vi8ebOj9smx3C-_og7gA4TBQE

  - Disclosure:

WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgIm1hcmtldGluZ0F1dGhvcml6YXRpb25Ib2xkZXIiLCAiTW9kZXJuYSBCaW90ZWNoIl0

  - Contents:

["6Ij7tM-a5iVPGboS5tmvVA", "marketingAuthorizationHolder", "Moderna Biotech"]

- Claim `nextVaccinationDate`:
  - SHA-256 Hash:

R2fGbfA07Z_YlkqmNZyma1xyyx1XstIiS6B1Ybl2JZ4

  - Disclosure:

WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgIm5leHRWYWNjaW5hdGlvbkRhdGUiLCAiMjAyMS0wOC0xNlQxMzo0MDoxMloiXQ

  - 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:

    `WyJuUHVvUW5rUkZxM0JJZUFtN0FuWEZBIiwgImdpdmVuTmFtZSIsICJNYXJpb24iXQ`
  - Contents:

    `["nPuoQnkRFq3BIeAm7AnXFA", "givenName", "Marion"]`
- Claim `familyName`:
  - SHA-256 Hash:

    `1lSQBNY24q0Th6OGzthq-7-4l6cAaxrYXOGZpeW_lnA`
  - Disclosure:

    `WyI1YlBzMUlxdVpOYTBoa2FGenp6Wk53IiwgImZhbWlseU5hbWUiLCAiTXVzdGVybWFubiJd`
  - Contents:

    `["5bPs1IquZNa0hkaFzzzZNw", "familyName", "Mustermann"]`
- Claim `administeringCentre`:
  - SHA-256 Hash:

    `TCmzrl7K2gev_du7pcMIyzRLHp-Yeg-Fl_cxtrUvPxg`
  - Disclosure:

    `WyI1YTJXMF9OcmxFWnpmcW1rXzdQcS13IiwgImFkbWluaXN0ZXJpbmdDZW50cmUiLCAiUHJheGlz IFNvbW1lcmdhcnRlbiJd`
  - Contents:

    `["5a2W0_NrlEZzfqmk_7Pq-w", "administeringCentre", "Praxis Sommergarten"]`
- Claim `batchNumber`:
  - SHA-256 Hash:

    `V7kJBLK78TmVDOmrfJ7ZuUPHuK_2cc7yZRa4qV1txwM`
  - Disclosure:

    `WyJ5MXNWVTV3ZGZKYWhWZGd3UGdTN1JRIiwgImJhdGNoTnVtYmVyIiwgIjE2MjYzODI3MzYiXQ`
  - Contents:

    `["y1sVU5wdfJahVdgwPgS7RQ", "batchNumber", "1626382736"]`
- Claim `healthProfessional`:
  - SHA-256 Hash:

    `1V_K-8lDQ8iFXBFXbZY9ehqR4HabWCi5T0ybIzZPeww`
  - Disclosure:

`WyJIYlE0WDhzclZXM1FEeG5JSmRxeU9BIiwgImhlYWx0aFByb2Zlc3Npb25hbCIsICI4ODMxMTAwMDAwMTUzNzYiXQ`

- Contents:

```
["HbQ4X8srVW3QDxnIJdqyOA", "healthProfessional", "883110000015376"]
```

This is an example of an SD-JWT+KB that discloses only `type`, `medicinalProductName`, `atcCode` of the vaccine, `type` of the `recipient`, `type`, `order`, and `dateOfVaccination`:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImV4YW1wbGUrc2Qtand0In0.eyJAY29udGV4
dCI6IFsiaHR0cHM6Ly93d3cudzMub3JnLzIwMTgvY3JlZGVudGlhbHMvdjEiLCAiaHR0
cHM6Ly93M2lkLm9yZy92YWNjaW5hdGlvbi92MSJdLCAidHlwZSI6IFsiVmVyaWZpYWJs
ZUNyZWRlbnRpYWwiLCAiVmFjY2luYXRpb25DZXJ0aWZpY2F0ZSJdLCAiaXNzdWVyIjog
Imh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlzc3VhbmNlRGF0ZSI6ICIyMDIz
LTAyLTA5VDExOjAxOjU5WiIsICJleHBpcmF0aW9uRGF0ZSI6ICIyMDI4LTAyLTA4VDEx
OjAxOjU5WiIsICJuYW1lIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRl
IiwgImRlc2NyaXB0aW9uIjogIkNPVklELTE5IFZhY2NpbmF0aW9uIENlcnRpZmljYXRl
IiwgImNyZWRlbnRpYWxTdWJqZWN0IjogeyJfc2QiOiBbIjFWX0stOGxEUThpRlhCRlhi
Wlk5ZWhxUjRIYWJXQ2k1VDB5Ykl6WlBld3ciLCAiSnpqTGd0UDI5ZFAtQjN0ZDEyUDY3
NGdGbUsyenk4MUhNdEJnZjZDSk5XZyIsICJSMmZHYmZBMDdaX1lsa3FtTlp5bWExeHl5
eDFYc3RJaVM2QjFZYmwySlo0IiwgIlRDbXpybDdLMmdld19kdTdwY01JeXpSTEhwLVll
Zy1GbF9jeHRyVXZQeGciLCAiVjdrSkJMSzc4VG1WRE9tcmZKN1p1VVBIdUtfMmNjN3la
UmE0cVYxdHh3TSIsICJiMGVVc3ZHUC1PRERkRm9ZNE5semxYYzN0RHNsV0p0Q0pGNzVO
dzhPal9nIiwgInpKS19lU01YandNOGRYbU1aTG5JOEZHTTA4ekozX3ViR2VFTUotNVRC
eTAiXSwgInZhY2NpbmUiOiB7Il9zZCI6IFsiMWNGNWhMd2toTU5JYXFmV0pyWEk3Tk1X
ZWRMLTlmNlkyUEE1MnlQalNaSSIsICJIaXk2V1d1ZUxENWJuMTYyOTh0UHY3R1hobWxk
TURPVG5CaS1DWmJwaE5vIiwgIkxiMDI3cTY5MWpYWGwtakM3M3ZpOGViT2o5c214M0Mt
X29nN2dBNFRCUUUiXSwgInR5cGUiOiAiVmFjY2luZSJ9LCAicmVjaXBpZW50IjogeyJf
c2QiOiBbIjFsU1FCTlkyNHEwVGg2T0d6dGhxLTctNGw2Y0FheHJZWE9HWnBlV19sbkEi
LCAiM256THE4MU0yb04wNndkdjFzaEh2T0VKVnhaNUtMbWREa0hFREpBQldFSSIsICJQ
bjFzV2kwNkc0TEpybm4tX1JUMFJiTV9IVGR4blBKUXVYMmZ6V3ZfSk9VIiwgImxGOXV6
ZHN3N0hwbEdMYzcxNFRyNFdPN01HSnphN3R0N1FGbGVDWDRJdHciXSwgInR5cGUiOiAi
VmFjY2luZVJlY2lwaWVudCJ9LCAidHlwZSI6ICJWYWNjaW5hdGlvbkV2ZW50In0sICJf
c2RfYWxnIjogInNoYS0yNTYiLCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJj
cnYiOiAiUC0yNTYiLCAieCI6ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxp
bERsczd2Q2VHZW1jIiwgInkiOiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVj
Q0U2dDRqVDlGMkhaUSJ9fX0.OZomvwO8iw4db89MYCeeomBVStXkT6u7G7FkicPWZnd2
_hGgr0l_u1NHgPVocuOt-m32Uu6kwtPmYFxKk0AOeA~WyJQYzMzSk0yTGNoY1VfbEhnZ
3ZfdWZRIiwgIm9yZGVyIiwgIjMvMyJd~WyJBSngtMDk1VlBycFR0TjRRTU9xUk9BIiwg
ImRhdGVPZlZhY2NpbmF0aW9uIiwgIjIwMjEtMDYtMjNUMTM6NDA6MTJaIl0~WyIyR0xD
NDJzS1F2ZUNmR2ZyeU5STjl3IiwgImF0Y0NvZGUiLCAiSjA3QlgwMyJd~WyJlbHVWNU9
nM2dTTklOEVZbnN4QV9BIiwgIm1lZGljaW5hbFByb2R1Y3ROYW1lIiwgIkNPVklELTE
5IFZhY2NpbmUgTW9kZXJuYSJd~eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dC
J9.eyJub25jZSI6ICIxMjM0NTY3ODkwIiwgImF1ZCI6ICJodHRwczovL3Zlcmlma WVyL
mV4YW1wbGUub3JnIiwgImlhdCI6IDE3NDg1MzcyNDQsICJzZF9oYXNoIjogIklvV1VIO
TFsbGYzWEVybDQyYlEzc3hfNTNWMW8xdWpDejA4aERxSEs3RGsifQ.n0vzyIwCFMDVau
EaeJIWEKZZchxXMpXTQewHgAkARbOSZxB09IbXXtHfpoGqO_BtNFN2lShJEIQBGyc-Xp
HigA
```

After the validation, the Verifier will have the following Processed SD-JWT Payload available for further handling:

```
{
  "@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"
  },
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```

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 4.2, the Disclosure structure is JSON containing a 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 different representations but is semantically equivalent, thus avoiding canonicalization. Some further discussion of the considerations around this design decision follows.

When receiving an SD-JWT, a Verifier must be able to recompute 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 needs 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 they yield equivalent objects. The same is true for the representations of `address`, which vary 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 and 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 plaintext contents 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 7.3 for details.

Acknowledgements

We would like to thank Alen Horvat, Alex Hodder, Anders Rundgren, Arjan Geluk, Chad Parry, Christian Bormann, Christian Paquin, Dale Bowie, Dan Moore, David Bakker, David Waite, Deb Cooley, Fabian Hauck, Filip Skokan, Giuseppe De Marco, Jacob Ward, Jeffrey Yasskin, John Preuß Mattsson, Joseph Heenan, Justin Richer, Kushal Das, Martin Thomson, Matthew Miller, Michael Fraser, Michael B. Jones, Mike Prorock, Nat Sakimura, Neil Madden, Oliver Terbu, Orie Steele, Paul Bastian, Peter Altmann, Pieter Kasselman, Richard Barnes, Rohan Mahy, Roman Danyliw, Ryosuke Abe, Sami Rosendahl, Shawn Butterfield, Shawn Emery, Simon Schulz, Takahiko Kawasaki, Tobias Looker, Torsten Lodderstedt, Vittorio Bertocci, Watson Ladd, and Yaron Sheffer for their contributions (some of which were substantial) to this draft and to the initial set of implementations.

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

Authors' Addresses

Daniel Fett
Authlete
Email: mail@danielfett.de
URI: https://danielfett.de/

Kristina Yasuda
Keio University
Email: kristina@sfc.keio.ac.jp

Brian Campbell
Ping Identity
Email: bcampbell@pingidentity.com