Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 26 February 2023

D. Fett
yes.com
K. Yasuda
Microsoft

25 August 2022

Selective Disclosure for JWTs (SD-JWT)

draft-ietf-oauth-selective-disclosure-jwt-00

Abstract

This document specifies conventions for creating JSON Web Token (JWT) documents that support selective disclosure of JWT claim values.

Status of This Memo

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

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

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

This Internet-Draft will expire on 26 February 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

The JSON-based representation of claims in a signed JSON Web Token (JWT) [RFC7519] is secured against modification using JSON Web Signature (JWS) [RFC7515] digital signatures. A consumer of a signed JWT that has checked the signature can safely assume that the contents of the token have not been modified. However, anyone receiving an unencrypted JWT can read all of the claims and likewise, anyone with the decryption key receiving an encrypted JWT can also read all of the claims.

This document describes a format for signed JWTs that supports selective disclosure (SD-JWT), enabling sharing only a subset of the claims included in the original signed JWT instead of releasing all the claims to every verifier. During issuance, an SD-JWT is sent from the issuer to the holder alongside an SD-JWT Salt/Value Container (SVC), a JSON object that contains the mapping between raw claim values contained in the SD-JWT and the salts for each claim value.

This document also defines a format for SD-JWT Releases (SD-JWT-R), which convey a subset of the claim values of an SD-JWT to the verifier. For presentation, the holder creates an SD-JWT-R and sends it together with the SD-JWT to the verifier. To verify claim values received in SD-JWT-R, the verifier uses the salts values in the SD-JWT-R to compute the hash digests of the claim values and compare them to the ones in the SD-JWT.

One of the common use cases of a signed JWT is representing a user's identity created by an issuer. As long as the signed JWT is one-time use, it typically only contains those claims the user has consented to release to a specific verifier. However, when a signed JWT is intended to be multi-use, it needs to contain the superset of all claims the user might want to release to verifiers at some point. The ability to selectively disclose a subset of these claims depending on the verifier becomes crucial to ensure minimum disclosure and prevent verifiers from obtaining claims irrelevant for the transaction at hand.

One example of such a multi-use JWT is a verifiable credential, a tamper-evident credential with a cryptographically verifiable authorship that contains claims about a subject. SD-JWTs defined in this document enable such selective disclosure of claims.

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

This document also describes holder binding, or the concept of binding SD-JWT to key material controlled by the subject of SD-JWT. Holder binding is optional to implement.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

2. Terms and Definitions

Selectively Disclosable JWT (SD-JWT) A JWT [RFC7515] created by the issuer, which is signed as a JWS [RFC7515], that supports selective disclosure as defined in this document.

SD-JWT Salt/Value Container (SVC) A JSON object created by the issuer that contains mapping between raw claim values contained in the SD-JWT and the salts for each claim value.

SD-JWT Release (SD-JWT-R) A JWT created by the holder that contains a subset of the claim values of an SD-JWT in a verifiable way.

Holder binding Ability of the holder to prove legitimate possession of SD-JWT by proving control over the same private key during the issuance and presentation. SD-JWT signed by the issuer contains a public key or a reference to a public key that matches to the private key controlled by the holder.

Issuer An entity that creates SD-JWTs (2.1).

Holder An entity that received SD-JWTs (2.1) from the issuer and has control over them.

Verifier An entity that requests, checks and extracts the claims from SD-JWT-R (2.2)

Note: discuss if we want to include Client, Authorization Server for the purpose of ensuring continuity and separating the entity from the actor.

3. Flow Diagram

```
              +------------+
              |            |
              |   Issuer   |
              |            |
              +------------+
                    |
                  Issues
              SD-JWT and SVC
                    |
                    v
              +------------+
              |            |
              |   Holder   |
              |            |
              +------------+
                    |
```

```
                 Presents
            SD-JWT-R and SD-JWT
                    |
                    v
              +-------------+
              |             |+
              |  Verifiers  ||+
              |             |||
              +-------------+||
               +-------------+|
                +-------------+
```

Figure 1: SD-JWT Issuance and Presentation Flow

4. Concepts

In the following, the contents of SD-JWTs and SD-JWT Releases are described at a conceptual level, abstracting from the data formats described afterwards.

4.1. Creating an SD-JWT

An SD-JWT, at its core, is a digitally signed document containing hash digests over the claim values with unique random salts and other metadata. It MUST be digitally signed using the issuer's private key.

```
SD-JWT-DOC = (METADATA, SD-CLAIMS)
SD-JWT = SD-JWT-DOC | SIG(SD-JWT-DOC, ISSUER-PRIV-KEY)
```

SD-CLAIMS can be a simple object with claim names mapped to hash digests over the claim values with unique random salts:

```
SD-CLAIMS = (
    CLAIM-NAME: HASH(SALT | CLAIM-VALUE)
)*
```

SD-CLAIMS can also be nested deeper to capture more complex objects, as will be shown later.

SD-JWT is sent from the issuer to the holder, together with the mapping of the plain-text claim values, the salt values, and potentially some other information.

4.2. Creating an SD-JWT Release

To disclose to a verifier a subset of the SD-JWT claim values, a holder creates a JWT such as the following:

```
SD-JWT-RELEASE-DOC = (METADATA, SD-RELEASES)
SD-JWT-RELEASE = SD-JWT-RELEASE-DOC
```

SD-RELEASES follows the structure of SD-CLAIMS and can be a simple object with claim names mapped to values and salts:

```
SD-RELEASES = (
    CLAIM-NAME: (DISCLOSED-SALT, DISCLOSED-VALUE)
)
```

Just as SD-CLAIMS, SD-RELEASES can be more complex as well.

SD-JWT-RELEASE is sent together with SD-JWT from the holder to the verifier.

4.3. Optional Holder Binding

Some use-cases may require holder binding.

If holder binding is desired, SD-JWT must contain information about key material controlled by the holder:

```
SD-JWT-DOC = (METADATA, HOLDER-PUBLIC-KEY, SD-CLAIMS)
```

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

With holder binding, the SD-JWT-RELEASE is signed by the holder using its private key. It therefore looks as follows:

```
SD-JWT-RELEASE = SD-JWT-RELEASE-DOC | SIG(SD-JWT-RELEASE-DOC, HOLDER-PRIV-KEY)
```

4.4. Verifying an SD-JWT Release

A verifier checks that

- for each claim in SD-JWT-RELEASE, the hash digest HASH(DISCLOSED-SALT | DISCLOSED-VALUE) matches the one under the given claim name in SD-JWT.

- if holder binding is used, the SD-JWT-RELEASE was signed by the private key belonging to HOLDER-PUBLIC-KEY.

The detailed algorithm is described below.

5. Data Formats

This section defines data formats for SD-JWTs (containing hash digests of the salted claim values), SD-JWT Salt/Value Containers (containing the mapping of the plain-text claim values and the salt values), and SD-JWT Releases (containing a subset of the same mapping).

5.1. Format of an SD-JWT

An SD-JWT is a JWT that MUST be signed using the issuer's private key. The payload of an SD-JWT MUST contain the sd_digests and sd_hash_alg claims described in the following, and MAY contain a holder's public key or a reference thereto, as well as further claims such as iss, iat, etc. as defined or required by the application using SD-JWTs.

5.1.1. sd_digests Claim (Digests of Selectively Disclosable Claims)

An SD-JWT MUST include hash digests of the salted claim values that are included by the issuer under the property sd_digests.

The issuer MUST choose a unique and cryptographically random salt value for each claim value. Each salt value SHOULD contain at least 128 bits of pseudorandom data, making it hard for an attacker to guess. The salt value MUST then be encoded as a string. It is RECOMMENDED to base64url-encode the salt value.

The issuer MUST build the digests by hashing over a string that is formed by JSON-encoding an ordered array containing the salt and the claim value, e.g.: ["6qMQvRL5haj","Peter"]. The digest value is then base64url-encoded. Note that the precise JSON encoding can vary, and therefore, the JSON encodings MUST be sent to the holder along with the SD-JWT, as described below.

5.1.1.1. Flat and Structured sd_digests objects

The sd_digests object can be a 'flat' object, directly containing all claim names and hashed claim values without any deeper structure. The sd_digests object can also be a 'structured' object, where some claims and their respective hash digests are contained in places deeper in the structure. It is at the issuer's discretion whether to use a 'flat' or 'structured' sd_digests SD-JWT object, and how to structure it such that it is suitable for the use case.

Example 1 below is a non-normative example of an SD-JWT using a 'flat' sd_digests object and Example 2 in the appendix shows a non-normative example of an SD-JWT using a 'structured' sd_digests object. The difference between the examples is how the address claim is disclosed.

Appendix 2 shows a more complex example using claims from eKYC (todo: reference).

5.1.2. Hash Function Claim

The claim sd_hash_alg indicates the hash algorithm used by the Issuer to generate the hashes of the salted claim values. The hash algorithm identifier MUST be a value from the "Hash Name String" column in the IANA "Named Information Hash Algorithm" registry [IANA.Hash.Algorithms]. SD-JWTs with hash algorithm identifiers not found in this registry are not considered valid and MUST NOT be accepted by verifiers.

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

5.1.3. Holder Public Key Claim

If the issuer wants to enable holder binding, it MAY include a public key associated with the holder, or a reference thereto.

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

Note: Examples in this document use cnf Claim defined in [RFC7800] to include raw public key by value in SD-JWT.

5.2. Example 1: SD-JWT

This example and Example 2 in the appendix use the following object as the set of claims that the Issuer is issuing:

```
{
  "sub": "6c5c0a49-b589-431d-bae7-219122a9ec2c",
  "given_name": "John",
  "family_name": "Doe",
  "email": "johndoe@example.com",
  "phone_number": "+1-202-555-0101",
  "address": {
    "street_address": "123 Main St",
    "locality": "Anytown",
    "region": "Anystate",
    "country": "US"
  },
  "birthdate": "1940-01-01"
}
```

The following non-normative example shows the payload of an SD-JWT. The issuer is using a flat structure, i.e., all of the claims the address claim can only be disclosed in full.

```
{
  "iss": "https://example.com/issuer",
  "cnf": {
    "jwk" : {
        "kty": "RSA",
        "n": "pm4bOHBg-oYhAyPWzR56AWX3rUIXp11_ICDkGgS6W3ZWLts-hzwI3x65659kg4
        "e": "AQAB"
    }
  },
  "iat": 1516239022,
  "exp": 1516247022,
  "sd_hash_alg": "sha-256",
  "sd_digests": {
    "sub": "z4xgEco94diTaSruISPiE7o_wtmcOfnH_8R7X9Pa578",
    "given_name": "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0",
    "family_name": "H-Relr4cEBMlenyK1gvyx16QVpnt4MEclT5tP0aTLFU",
    "email": "ET2A1JQLF85ZpBulh6UFstGrSfR4B3KM-bjQVllhxqY",
    "phone_number": "SJnciB2DIRVA5cXBrdKoH6n45788mZyUn2rnv74uMVU",
    "address": "0FldqLfGnERPPVDC17od9xb4w3iRJTEQbW_Yk9AmnDw",
    "birthdate": "-L0kMgIbLXe3OEkKTUGwz_QKhjehDeofKGwoPrxLuo4"
  }
}
```

The SD-JWT is then signed by the issuer to create a document like the following:

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZUk
F6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAic3ViX2p3ayI6IHsia3R5IjogIlJTQSIsICJuIjogInBtNGJPSEJnLW9ZaEF5UFd
6UjU2QVdYM3JVSVhwMTFfSUNEa0dnUzZXM1pXTHRzLWh6d0kzeDY1NjU5a2c0aFZvOWRiR
29DSkUzWkdGX2VhZXRFMzBVaEJVRWdwR3dyRHJRaUo5enFwcm1jRmZyM3F2dmtHanR0aDh
aZ2wxZU0yYkpjT3dFN1BDQkhXVEtXWXMxNTJSN2c2SmcyT1ZwaC1hOHJxLXE3OU1oS0c1U
W9XX21UejEwUVRfNkg0YzdQaldHMWZqaDhocFdObmJQX3B2NmQxelN3WmZjNWZsNnlWUkw
wRFYwVjNsR0hLZTJXcWZfZU5HakJyQkxWa2xEVGs4LXN0WF9NV0xjUi1FR21YQU92MFVCV
2l0U19kWEpLSnUtdlhKeXcxNG5IU0d1eFRJSzJoeDFwdHRNZnQ5Q3N2cWltWEtlRFRVMTR
xUUwxZUU3aWhjdyIsICJlIjogIkFRQUIifSwgImlhdCI6IDE1MTYyMzkwMjIsICJleHAiO
iAxNTE2MjQ3MDIyLCAiaGFzaF9hbGciOiAic2hhLTI1NiIsICJzZF9kaWdlc3RzIjogeyJ
zdWIiOiAiejR4Z0Vjbzk0ZGlUYVNydUlTUGlFN29fd3RtY09mbkhfOFI3WDlQYTU3OCIsI
CJnaXZlbl9uYW1lIjogIlB2VTdjV2p1SFVxNnctaTlYRnBRWmhqVC11cHJRTDNHSDNtS3N
BSmwwZTAiLCAiZmFtaWx5X25hbWUiOiAiSC1SZWxyNGNFQk1sZW55SzFndnl4MTZRVnBud
DRNRWNsVDV0UDBhVExGVSIsICJlbWFpbCI6ICJFVDJBMUpRTEY4NVpwQnVsaDZVRnN0R3J
TZlI0QjNLTS1ialFWbGxoeHFZIiwgInBob25lX251bWJlciI6ICJTSm5jaUIyRElSVkE1Y
1hCcmRLb0g2bjQ1Nzg4bVp5VW4ycm52NzR1TVZVIiwgImFkZHJlc3MiOiAiMEZsZHFMZkd
uRVJQUFZEQzE3b2Q5eGI0dzNpUkpURVFiV19ZazlBbW5EdyIsICJiaXJ0aGRhdGUiOiAiL
Uwwa01nSWJMWGUzT0VrS1RVR3d6X1FLaGplaERlb2ZLR3dvUHJ4THVvNCJ9fQ.TSxRFVMl
Nlt_drWLqQuzz9Sc19Bo5r-cgXZ0to8PfadImEJuPbhfkxCSt1xCT5IWQJrD6p_47h8Ac-
5yOtuu8s-wxrQzCUAU7OS2ThQGKiDKsGv0oiH3KQX4QyCAOIIaWOLxO6VQRxI7Kn6RFT0M
5jQxBNFmL582n_uJR31ANBO45Jc_pgp4iSTn2vf1lxxwU7B3bgrad0vHmvwmv1lu17Ov79
SGmT_fTjoybIJfHavTq61xCJ3UrMAGCEqm-_laOTxB1HRbPF38va0iuhqSSk2QaUpfDsVb
9VsPBVIx_MsGlK3kYUWcHzkJ0BBPIVu0xnVcLkg1_SO_5gPSxFWZ4g
```

(Line breaks for presentation only.)

5.3. Format of a SD-JWT Salt/Value Container (SVC)

Besides the SD-JWT itself, the holder needs to learn the raw claim values that are contained in the SD-JWT, along with the precise input to the hash calculation, and the salts. There MAY be other information the issuer needs to communicate to the holder, such as a private key if the issuer selected the holder key pair.

A SD-JWT Salt/Value Container (SVC) is a JSON object containing at least the top-level property sd_release. Its structure mirrors the one of sd_digests in the SD-JWT, but the values are the inputs to the hash calculations the issuer used, as strings.

The SVC MAY contain further properties, for example, to transport the holder private key.

5.4. Example: SVC for the Flat SD-JWT in Example 1

The SVC for Example 1 is as follows:

```
{
  "sd_release": {
    "sub": "[\"2GLC42sKQveCfGfryNRN9w\", \"6c5c0a49-b589-431d-bae7-219122a9e
    "given_name": "[\"eluV5Og3gSNII8EYnsxA_A\", \"John\"]",
    "family_name": "[\"6Ij7tM-a5iVPGboS5tmvVA\", \"Doe\"]",
    "email": "[\"eI8ZWm9QnKPpNPeNenHdhQ\", \"johndoe@example.com\"]",
    "phone_number": "[\"Qg_O64zqAxe412a108iroA\", \"+1-202-555-0101\"]",
    "address": "[\"AJx-095VPrpTtN4QMOqROA\", {\"street_address\": \"123 Main
    "birthdate": "[\"Pc33JM2LchcU_lHggv_ufQ\", \"1940-01-01\"]"
  }
}
```

Important: As described above, hash digests are calculated over the string formed by serializing a JSON array containing the salt and the claim value. This ensures that issuer and verifier use the same input to their hash functions and avoids issues with canonicalization of JSON values that would lead to different hash digests. The SVC therefore maps claim names to JSON-encoded arrays.

5.5. Sending SD-JWT and SVC during Issuance

For transporting the SVC together with the SD-JWT from the issuer to the holder, the SVC is base64url-encoded and appended to the SD-JWT using a period character . as the separator.

The SVC and SD-JWT are implicitly linked through the hash values of the claims in the SVC that is included in the SD-JWT. To ensure that the correct SVC and SD-JWT pairings are being used, the holder SHOULD verify the binding between SVC and SD-JWT as defined in the Verification Section of this document.

For Example 1, the combined format looks as follows:

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZUk
F6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAic3ViX2p3ayI6IHsia3R5IjogIlJTQSIsICJuIjogInBtNGJPSEJnLW9ZaEF5UFd
6UjU2QVdYM3JVSVhwMTFfSUNEa0dnUzZXM1pXTHRzLWh6d0kzeDY1NjU5a2c0aFZvOWRiR
29DSkUzWkdGX2VhZXRFMzBVaEJVRWdwR3dyRHJRaUo5enFwcm1jRmZyM3F2dmtHanR0aDh
aZ2wxZU0yYkpjT3dFN1BDQkhXVEtXWXMxNTJSN2c2SmcyT1ZwaC1hOHJxLXE3OU1oS0c1U
W9XX21UejEwUVRfNkg0YzdQaldHMWZqaDhocFdObmJQX3B2NmQxelN3WmZjNWZsNnlWUkw
wRFYwVjNsR0hLZTJXcWZfZU5HakJyQkxWa2xEVGs4LXN0WF9NV0xjUi1FR21YQU92MFVCV
2l0U19kWEpLSnUtdlhKeXcxNG5IU0d1eFRJSzJoeDFwdHRNZnQ5Q3N2cWltWEtlRFRVMTR
xUUwxZUU3aWhjdyIsICJlIjogIkFRQUIifSwgImlhdCI6IDE1MTYyMzkwMjIsICJleHAiO
iAxNTE2MjQ3MDIyLCAiaGFzaF9hbGciOiAic2hhLTI1NiIsICJzZF9kaWdlc3RzIjogeyJ
zdWIiOiAiejR4Z0Vjbzk0ZGlUYVNydUlTUGlFN29fd3RtY09mbkhfOFI3WDlQYTU3OCIsI
CJnaXZlbl9uYW1lIjogIlB2VTdjV2p1SFVxNnctaTlYRnBRWmhqVC11cHJRTDNHSDNtS3N
BSmwwZTAiLCAiZmFtaWx5X25hbWUiOiAiSC1SZWxyNGNFQk1sZW55SzFndnl4MTZRVnBud
DRNRWNsVDV0UDBhVExGVSIsICJlbWFpbCI6ICJFVDJBMUpRTEY4NVpwQnVsaDZVRnN0R3J
TZlI0QjNLTS1ialFWbGxoeHFZIiwgInBob25lX251bWJlciI6ICJTSm5jaUIyRElSVkE1Y
1hCcmRLb0g2bjQ1Nzg4bVp5VW4ycm52NzR1TVZVIiwgImFkZHJlc3MiOiAiMEZsZHFMZkd
uRVJQUFZEQzE3b2Q5eGI0dzNpUkpURVFiV19ZazlBbW5EdyIsICJiaXJ0aGRhdGUiOiAiL
```

```
Uwwa01nSWJMWGUzT0VrS1RVR3d6X1FLaGplaERlb2ZLR3dvUHJ4THVvNCJ9fQ.TSxRFVMl
Nlt_drWLqQuzz9Sc19Bo5r-cgXZ0to8PfadImEJuPbhfkxCSt1xCT5IWQJrD6p_47h8Ac-
5yOtuu8s-wxrQzCUAU7OS2ThQGKiDKsGv0oiH3KQX4QyCAOIIaWOLxO6VQRxI7Kn6RFT0M
5jQxBNFmL582n_uJR31ANBO45Jc_pgp4iSTn2vf1lxxwU7B3bgrad0vHmvwmv1lu17Ov79
SGmT_fTjoybIJfHavTq61xCJ3UrMAGCEqm-_laOTxB1HRbPF38va0iuhqSSk2QaUpfDsVb
9VsPBVIx_MsGlK3kYUWcHzkJ0BBPIVu0xnVcLkg1_SO_5gPSxFWZ4g.eyJzZF9yZWxlYXN
lIjogeyJzdWIiOiAiW1wiMkdMQzQyc0tRdmVDZkdmcnlOUk45d1wiLCBcIjZjNWMwYTQ5L
WI1ODktNDMxZC1iYWU3LTIxOTEyMmE5ZWMyY1wiXSIsICJnaXZlbl9uYW1lIjogIltcImV
sdVY1T2czZ1NOSUk4RVluc3hBX0FcIiwgXCJKb2huXCJdIiwgImZhbWlseV9uYW1lIjogI
ltcIjZJajd0TS1hNWlWUEdib1M1dG12VkFcIiwgXCJEb2VcIl0iLCAiZW1haWwiOiAiW1w
iZUk4WldtOVFuS1BwTlBlTmVuSGRoUVwiLCBcImpvaG5kb2VAZXhhbXBsZS5jb21cIl0iL
CAicGhvbmVfbnVtYmVyIjogIltcIlFnX082NHpxQXhlNDEyYTEwOGlyb0FcIiwgXCIrMS0
yMDItNTU1LTAxMDFcIl0iLCAiYWRkcmVzcyI6ICJbXCJBSngtMDk1VlBycFR0TjRRRTU9xU
k9BXCIsIHtcInN0cmVldF9hZGRyZXNzXCI6IFwiMTIzIE1haW4gU3RcIiwgXCJsb2NhbGl
0eVwiOiBcIkFueXRvd25cIiwgXCJyZWdpb25cIjogXCJBbnlzdGF0ZVwiLCBcImNvdW50c
nlcIjogXCJVU1wifV0iLCAiYmlydGhkYXRlIjogIltcIlBjMzNKTTJMY2hjVV9sSGdndl9
1ZlFcIiwgXCIxOTQwLTAxLTAxXCJdIn19
```

(Line breaks for presentation only.)

5.6. Format of an SD-JWT Release

SD-JWT-R contains claim values and the salts of the claims that the holder has consented to release to the Verifier. This enables the Verifier to verify the claims received from the holder by computing the hash digests of the claim values and the salts revealed in the SD-JWT-R using the hashing algorithm specified in SD-JWT and comparing them to the hash digests included in SD-JWT.

For each claim, an array of the salt and the claim value is contained in the sd_release object. The structure of sd_release object in the SD-JWT-R is the same as in SD-JWT.

The SD-JWT-R MAY contain further claims, for example, to ensure a binding to a concrete transaction (in the example the nonce and aud claims).

When the holder sends the SD-JWT-R to the Verifier, the SD-JWT-R MUST be a JWS represented as the JWS Compact Serialization as described in Section 7.1 of [RFC7515].

If holder binding is desired, the SD-JWT-R is signed by the holder. If no holder binding is to be used, the none algorithm is used, i.e., the document is not signed. TODO: Change to plain base64 to avoid alg=none issues

5.7. Example: SD-JWT Release for Example 1

The following is a non-normative example of the contents of an SD-JWT-R for Example 1:

```
{
  "nonce": "XZOUco1u_gEPknxS78sWWg",
  "aud": "https://example.com/verifier",
  "sd_release": {
    "given_name": "[\"eluV5Og3gSNII8EYnsxA_A\", \"John\"]",
    "family_name": "[\"6Ij7tM-a5iVPGboS5tmvVA\", \"Doe\"]",
    "address": "[\"AJx-095VPrpTtN4QMOqROA\", {\"street_address\": \"123 Main
  }
}
```

For each claim, an array of the salt and the claim value is contained in the sd_release object.

Again, the SD-JWT-R follows the same structure as the sd_digests in the SD-JWT.

Below is a non-normative example of a representation of the SD-JWT-R JWS Compact Serialization:

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIkxkeVRYd0F5ZnJpcjRfVjZORzFSYzEwVThKZE
xZVHJFQktKaF9oNWlfclUifQ.eyJub25jZSI6ICJYWk9VY28xdV9nRVBrbnhTNzhzV1dnI
iwgImF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29tL3ZlcmlmaWVyIiwgInNkX3JlbGVhc2U
iOiB7ImdpdmVuX25hbWUiOiAiW1wiZWx1VjVPZzNnU05JSThFWW5zeEFfQVwiLCBcIkpva
```

```
G5cIl0iLCAiZmFtaWx5X25hbWUiOiAiW1wiNklqN3RNLWE1aVZQR2JvUzV0bXZWQVwiLCB
cIkRvZVwiXSIsICJhZGRyZXNzIjogIltcIkFKeC0wOTVWUHJwVHRONFFNT3FST0FcIiwge
1wic3RyZWV0X2FkZHJlc3NcIjogXCIxMjMgTWFpbiBTdFwiLCBcImxvY2FsaXR5XCI6IFw
iQW55dG93blwiLCBcInJlZ2lvblwiOiBcIkFueXN0YXRlXCIsIFwiY291bnRyeVwiOiBcI
lVTXCJ9XSJ9fQ.j5jtArW1QDK1BNM13sJbQaE00GsAhhiPRYi6oK-iJRtLWE6DAgcWxDir
TvxTnuo7Mbb6gSqGTmdEEtmscWxweFfGQoddObPTDiapjWiR1bUMMqPDKNNkRe0CBkU-pW
ieYWN-fQxlRa4BKUqs18jcvGtTGA8Ye-i6t2xLROeXf2U_Seko8b7MQWIFHdbc0LvEFt_-
JAcqeshH5wjVkwhHofVuZq1vGLlINKBveKA2dmn6wuEzi6XRceTwFrG_hTECagfobdO-bY
MF3FSiCQM2KxC_6_aLApYo0aH3zjBv9rm0qNmnL_JGN5FIu6YqwhvPzfdsfkjMd68o8LTW
d7F6kQ
```

(Line breaks for presentation only.)

5.8. Sending SD-JWT and SD-JWT-R during Presentation

The SD-JWT and the SD-JWT-R can be combined into one document using period character . as a separator (here for Example 1):

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZUk
F6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAic3ViX2p3ayI6IHsia3R5IjogIlJTQSIsICJuIjogInBtNGJPSEJnLW9ZaEF5UFd
6UjU2QVdYM3JVSVhwMTFfSUNEa0dnUzZXM1pXTHRzLWh6d0kzeDY1NjU5a2c0aFZvOWRiR
29DSkUzWkdGX2VhZXRFMzBVaEJVRWdwR3dyRHJRaUo5enFwcm1jRmZyM3F2dmtHanR0aDh
aZ2wxZU0yYkpjT3dFN1BDQkhXVEtXWXMxNTJSN2c2SmcyT1ZwaC1hOHJxLXE3OU1oS0c1U
W9XX21UejEwUVRfNkg0YzdQaldHMWZqaDhocFdObmJQX3B2NmQxelN3WmZjNWZsNnlWUkw
wRFYwVjNsR0hLZTJXcWZfZU5HakJyQkxWa2xEVGs4LXN0WF9NV0xjUi1FR21YQU92MFVCV
2l0U19kWEpLSnUtdlhKeXcxNG5IU0d1eFRJSzJoeDFwdHRNZnQ5Q3N2cWltWEtlRFRVMTR
xUUwxZUU3aWhjdyIsICJlIjogIkFRQUIifSwgImlhdCI6IDE1MTYyMzkwMjIsICJleHAiO
iAxNTE2MjQ3MDIyLCAiaGFzaF9hbGciOiAic2hhLTI1NiIsICJzZF9kaWdlc3RzIjogeyJ
zdWIiOiAiejR4Z0Vjbzk0ZGlUYVNydUlTUGlFN29fd3RtY09mbkhfOFI3WDlQYTU3OCIsI
CJnaXZlbl9uYW1lIjogIlB2VTdjV2p1SFVxNnctaTlYRnBRWmhqVC11cHJRTDNHSDNtS3N
BSmwwZTAiLCAiZmFtaWx5X25hbWUiOiAiSC1SZWxyNGNFQk1sZW55SzFndnl4MTZRVnBud
DRNRWNsVDV0UDBhVExGVSIsICJlbWFpbCI6ICJFVDJBMUpRTEY4NVpwQnVsaDZVRnN0R3J
TZlI0QjNLTS1ialFWbGxoeHFZIiwgInBob25lX251bWJlciI6ICJTSm5jaUIyRElSVkE1Y
1hCcmRLb0g2bjQ1Nzg4bVp5VW4ycm52NzR1TVZVIiwgImFkZHJlc3MiOiAiMEZsZHFMZkd
uRVJQUFZEQzE3b2Q5eGI0dzNpUkpURVFiV19ZazlBbW5EdyIsICJiaXJ0aGRhdGUiOiAiL
Uwwa01nSWJMWGUzT0VrS1RVR3d6X1FLaGplaERlb2ZLR3dvUHJ4THVvNCJ9fQ.TSxRFVMl
Nlt_drWLqQuzz9Sc19Bo5r-cgXZ0to8PfadImEJuPbhfkxCSt1xCT5IWQJrD6p_47h8Ac-
5yOtuu8s-wxrQzCUAU7OS2ThQGKiDKsGv0oiH3KQX4QyCAOIIaWOLxO6VQRxI7Kn6RFT0M
5jQxBNFmL582n_uJR31ANBO45Jc_pgp4iSTn2vf1lxxwU7B3bgrad0vHmvwmv1lu17Ov79
SGmT_fTjoybIJfHavTq61xCJ3UrMAGCEqm-_laOTxB1HRbPF38va0iuhqSSk2QaUpfDsVb
9VsPBVIx_MsGlK3kYUWcHzkJ0BBPIVu0xnVcLkg1_SO_5gPSxFWZ4g.eyJhbGciOiAiUlM
yNTYiLCAia2lkIjogIkxkeVRYd0F5ZnJpcjRfVjZORzFSYzEwVThKZExxZVHJFQktKaF9oN
WlfclUifQ.eyJub25jZSI6ICJYWk9VY28xdV9nRVBrbnhTNzhzV1dnIiwgImF1ZCI6ICJo
dHRwczovL2V4YW1wbGUuY29tL3ZlcmlmaWVyIiwgInNkX3JlbGVhc2UiOiB7ImdpdmVuX2
5hbWUiOiAiW1wiZWx1VjVPZzNnU05JSThFWW5zeEFfQVwiLCBcIkpvaG5cIl0iLCAiZmFt
aWx5X25hbWUiOiAiW1wiNklqN3RNLWE1aVZQR2JvUzV0bXZWQVwiLCBcIkRvZVwiXSIsIC
JhZGRyZXNzIjogIltcIkFKeC0wOTVWUHJwVHRONFFNT3FST0FcIiwge1wic3RyZWV0X2Fk
ZHJlc3NcIjogXCIxMjMgTWFpbiBTdFwiLCBcImxvY2FsaXR5XCI6IFwiQW55dG93blwiLC
BcInJlZ2lvblwiOiBcIkFueXN0YXRlXCIsIFwiY291bnRyeVwiOiBcIlVTXCJ9XSJ9fQ.j
5jtArW1QDK1BNM13sJbQaE00GsAhhiPRYi6oK-iJRtLWE6DAgcWxDirTvxTnuo7Mbb6gSq
GTmdEEtmscWxweFfGQoddObPTDiapjWiR1bUMMqPDKNNkRe0CBkU-pWieYWN-fQxlRa4BK
Uqs18jcvGtTGA8Ye-i6t2xLROeXf2U_Seko8b7MQWIFHdbc0LvEFt_-JAcqeshH5wjVkwh
HofVuZq1vGLlINKBveKA2dmn6wuEzi6XRceTwFrG_hTECagfobdO-bYMF3FSiCQM2KxC_6
_aLApYo0aH3zjBv9rm0qNmnL_JGN5FIu6YqwhvPzfdsfkjMd68o8LTWd7F6kQ
```

(Line breaks for presentation only.)

6. Verification

6.1. Verification by the Holder when Receiving SD-JWT and SVC

The holder SHOULD verify the binding between SD-JWT and SVC by performing the following steps: 1. Check that all the claims in the SVC are present in the SD-JWT and that there are no claims in the SD-JWT that are not in the SVC 2. Check that the hashes of the claims in the SVC match those in the SD-JWT

6.2. Verification by the Verifier when Receiving SD-JWT and SD-JWT-R

Verifiers MUST follow [RFC8725] for checking the SD-JWT and, if signed, the SD-JWT Release.

Verifiers MUST go through (at least) the following steps before trusting/using any of the contents of an SD-JWT:

1. Determine if holder binding is to be checked for the SD-JWT. Refer to Section 7.6 for details.

2. Check that the presentation consists of six period-separated (.) elements; if holder binding is not required, the last element can be empty.

3. Separate the SD-JWT from the SD-JWT Release.

4. Validate the SD-JWT:

    1. Ensure that a signing algorithm was used that was deemed secure for the application. Refer to [RFC8725], Sections 3.1 and 3.2 for details.

    2. Validate the signature over the SD-JWT.

    3. Validate the issuer of the SD-JWT and that the signing key belongs to this issuer.

    4. Check that the SD-JWT is valid using nbf, iat, and exp claims, if provided in the SD-JWT.

    5. Check that the claim sd_digests is present in the SD-JWT.

    6. Check that the sd_hash_alg claim is present and its value is understand and the hash algorithm is deemed secure.

5. Validate the SD-JWT Release:

    1. If holder binding is required, validate the signature over the SD-JWT using the same steps as for the SD-JWT plus the following steps:

        1. Determine that the public key for the private key that used to sign the SD-JWT-R is bound to the SD-JWT, i.e., the SD-JWT either contains a reference to the public key or contains the public key itself.

        2. Determine that the SD-JWT-R is bound to the current transaction and was created for this verifier (replay protection). This is usually achieved by a nonce and aud field within the SD-JWT Release.

    2. For each claim in the SD-JWT Release:

        1. Ensure that the claim is present as well in sd_release in the SD-JWT. If sd_release is structured, the claim MUST be present at the same place within the structure.

        2. Compute the base64url-encoded hash of a claim revealed from the Holder using the claim value and the salt included in the SD-JWT-R and the sd_hash_alg in SD-JWT.

        3. Compare the hash digests computed in the previous step with the one of the same claim in the SD-JWT. Accept the claim only when the two hash digests match.

        4. Ensure that the claim value in the SD-JWT-R is a JSON-encoded array of exactly two values.

        5. Store the second of the two values.

3. Once all necessary claims have been verified, their values can be validated and used according to the requirements of the application. It MUST be ensured that all claims required for the application have been released.

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

7. Security Considerations

7.1. Mandatory hash computation of the revealed claim values by the Verifier

ToDo: add text explaining mechanisms that should be adopted to ensure that verifiers validate the claim values received in SD-JWT-R by calculating the hashes of those values and comparing them with the hashes in the SD-JWT: - create a test suite that forces hash computation by the Verifiers, and includes negative test cases in test vectors - use only implementations/libraries that are compliant to the test suite - etc.

7.2. Mandatory signing of the SD-JWT

The SD-JWT MUST be signed by the issuer to protect integrity of the issued claims. An attacker can modify or add claims if an SD-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 SD-JWT signature to ensure that the SD-JWT has not been tampered with since its issuance. If the signature on the SD-JWT cannot be verified, the SD-JWT MUST be rejected.

7.3. Entropy and Uniqueness of the salt

The security model relies on the fact that the salt is not learned or guessed by the attacker. It is vitally important to adhere to this principle. As such, the salt MUST be created in such a manner that it is cryptographically random, long enough and has high entropy that it is not practical for the attacker to guess. Each salt value MUST be unique.

7.4. Minimum length of the salt

The length of the randomly-generated portion of the salt MUST be at least 128 bits.

7.5. Choice of a hash function

For the security of this scheme, the hash function is required to be preimage and collision resistant, i.e., it is infeasible to calculate the salt and claim value that result in a particular digest, and it is infeasible to find a different salt and claim value pair that result in a matching digest, respectively.

Furthermore the hash algorithms MD2, MD4, MD5, RIPEMD-160, and SHA-1 revealed fundamental weaknesses and they MUST NOT be used.

7.6. Holder Binding

TBD

8. Privacy Considerations

8.1. Claim Names

Claim names are not hashed in the SD-JWT and are used as keys in a key-value pair, where the value is the hash. This is because SD-JWT already reveals information about the issuer and the schema, and revealing the claim names does not provide any additional information.

8.2. Unlinkability

Colluding issuer/verifier or verifier/verifier pairs could link issuance/presentation or two presentation sessions to the same user on the basis of unique values encoded in the SD-JWT (issuer signature, salts, digests, etc.). More advanced cryptographic schemes, outside the scope of this specification, can be used to prevent this type of linkability.

9. Acknowledgements

We would like to thank Alen Horvat, Brian Campbell, Christian Paquin, Fabian Hauck, Giuseppe De Marco, Kushal Das, Mike Jones, Nat Sakimura, Pieter Kasselman, and Torsten Lodderstedt for their contributions (some of which substantial) to this draft and to the initial set of implementations.

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

10. IANA Considerations

TBD

11. Normative References

[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>.

[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>.

[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>.

[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>.

12. Informative References

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-

Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

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

[VC_DATA] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and D. Chadwick, "Verifiable Credentials Data Model 1.0", 19 November 2019, <https://www.w3.org/TR/vc_data>.

Appendix A. Additional Examples

A.1. Example 2 - Structured SD-JWT

This non-normative example is based on the same claim values as Example 1, but this time the issuer decided to create a structured object for the hashes. This allows for the release of individual members of the address claim separately.

```
{
  "iss": "https://example.com/issuer",
  "cnf": {
    "jwk" : {
        "kty": "RSA",
```

```
        "n": "pm4bOHBg-oYhAyPWzR56AWX3rUIXp11_ICDkGgS6W3ZWLts-hzwI3x65659kg4
        "e": "AQAB"
    }
  },
  "iat": 1516239022,
  "exp": 1516247022,
  "sd_hash_alg": "sha-256",
  "sd_digests": {
    "sub": "z4xgEco94diTaSruISPiE7o_wtmcOfnH_8R7X9Pa578",
    "given_name": "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0",
    "family_name": "H-Relr4cEBMlenyK1gvyx16QVpnt4MEclT5tP0aTLFU",
    "email": "ET2A1JQLF85ZpBulh6UFstGrSfR4B3KM-bjQVllhxqY",
    "phone_number": "SJnciB2DIRVA5cXBrdKoH6n45788mZyUn2rnv74uMVU",
    "address": {
      "street_address": "O7_Isd6CmZqcSobPVpMgmJwB41hPUHHG8jg5LJ8YzfY",
      "locality": "w-zTF6ljkQLTvVyp_JNyD3t5Waj-B2vb0AXH1q8OsjI",
      "region": "nTvoKpGA6YQwEZipVBIM4WVH9KWEnwiqsRjEhrxhQz4",
      "country": "u-O1yDQqDTTqOgUBSjWilgkMLzg_QOTELMfZrRT5e6k"
    },
    "birthdate": "TipyoxD43PZJF8ZEmKPrbxMElpFX_M7aBLkUpC-W53o"
  }
}
```

The SVC for this SD-JWT is as follows:

```
{
  "sd_release": {
    "sub": "[\"2GLC42sKQveCfGfryNRN9w\", \"6c5c0a49-b589-431d-bae7-219122a9e
    "given_name": "[\"eluV5Og3gSNII8EYnsxA_A\", \"John\"]",
    "family_name": "[\"6Ij7tM-a5iVPGboS5tmvVA\", \"Doe\"]",
    "email": "[\"eI8ZWm9QnKPpNPeNenHdhQ\", \"johndoe@example.com\"]",
    "phone_number": "[\"Qg_O64zqAxe412a108iroA\", \"+1-202-555-0101\"]",
    "address": {
      "street_address": "[\"AJx-095VPrpTtN4QMOqROA\", \"123 Main St\"]",
      "locality": "[\"Pc33JM2LchcU_lHggv_ufQ\", \"Anytown\"]",
      "region": "[\"G02NSrQfjFXQ7Io09syajA\", \"Anystate\"]",
      "country": "[\"lklxF5jMYlGTPUovMNIvCA\", \"US\"]"
    },
    "birthdate": "[\"nPuoQnkRFq3BIeAm7AnXFA\", \"1940-01-01\"]"
  }
}
```

An SD-JWT-R for the SD-JWT above that discloses only region and country of the address property:

```
{
  "nonce": "XZOUco1u_gEPknxS78sWWg",
  "aud": "https://example.com/verifier",
  "sd_release": {
    "given_name": "[\"eluV5Og3gSNII8EYnsxA_A\", \"John\"]",
    "family_name": "[\"6Ij7tM-a5iVPGboS5tmvVA\", \"Doe\"]",
    "birthdate": "[\"nPuoQnkRFq3BIeAm7AnXFA\", \"1940-01-01\"]",
    "address": {
      "region": "[\"G02NSrQfjFXQ7Io09syajA\", \"Anystate\"]",
      "country": "[\"lklxF5jMYlGTPUovMNIvCA\", \"US\"]"
    }
  }
}
```

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

In this example, a complex object such as those used for OIDC4IDA (todo reference) is used.

In this example, the Issuer is using a following object as a set of claims to issue to the Holder:

```
   {
     "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": "Meier",
         "birthdate": "1956-01-28",
         "place_of_birth": {
           "country": "DE",
           "locality": "Musterstadt"
         },
         "nationalities": [
           "DE"
         ],
         "address": {
           "locality": "Maxstadt",
           "postal_code": "12344",
           "country": "DE",
           "street_address": "An der Weide 22"
         }
       }
```

```
  },
  "birth_middle_name": "Timotheus",
  "salutation": "Dr.",
  "msisdn": "49123456789"
}
```

The following shows the resulting SD-JWT payload:

```
{
  "iss": "https://example.com/issuer",
  "cnf": {
    "jwk" : {
        "kty": "RSA",
        "n": "pm4bOHBg-oYhAyPWzR56AWX3rUIXp11_ICDkGgS6W3ZWLts-hzwI3x65659kg4
        "e": "AQAB"
    }
  },
  "iat": 1516239022,
  "exp": 1516247022,
  "sd_hash_alg": "sha-256",
  "sd_digests": {
    "verified_claims": {
      "verification": {
        "trust_framework": "w1mP4oPc_J9thBex0TaQi1vgxFmruQJxZYLFnkNFMaI",
        "time": "Pu3i0CWrPVLJW-LT30yF1bFBPP15B6-uKk3PnGDflv8",
        "verification_process": "8HqIXRmczsdYOZzGcLqI5-l9xN5QbK2XDtXmdfH7z-4
        "evidence": [
          {
            "type": "TnLuqGGQm6jfeOoa5uX1diKANUPuh-zHrpBFdX9MR-g",
            "method": "SagmakoSu-X-XUPIC3EgdrEEwIWxRWXX4-i68X9TyEo",
            "time": "ld2c5oYDRtQcfU6PzogPkx_95WYqhqIJNVRMnfcsicY",
            "document": {
              "type": "ufWjDaAa54MnHeji2ZUUHDdnpZ9zx6CUG6uR28VMtsQ",
              "issuer": {
                "name": "a4GMucU7Zb060r0Svd7huY6Qho1bIf3v1U5BvPR8q6Y",
                "country": "135k9M0m2SCnYRuOfHuYScYVS2q3eeY7IItgyRsaBT8"
              },
              "number": "cUvOxLUp8RV7TTVliEiu-TQIel-LsE8E-XfUgfqk5gk",
              "date_of_issuance": "NIs8olJnJOv4J1qIEBKuTs2sEFs4fgGJhNqM6xdQt
              "date_of_expiry": "HTR37vLtANT6MWk-9dBqekFpCvaTG7zNf1ze56rnV64
            }
```

```
             }
           ]
         },
         "claims": {
           "given_name": "NB9XH_yJKqKOhXDmXkZKpMCkRbOmOTd8bqJFYDJYQnQ",
           "family_name": "hAUbJ66ZYL9VJLbjsDpmSs2e9Ff_Ohim_WR4bwZyvoQ",
           "birthdate": "6XOR4k56BgWk5tnNismbmEHvoGX7RRfy6Z8HENl96cU",
           "place_of_birth": {
             "country": "CLTlhuy13WWc3_ISon1kEypFwvCmfhLSpGUMCyAUg68",
             "locality": "AQoX8ixGpz-ipweEGlC-2umqwyQdhjIeiUB_TKWcE2E"
           },
           "nationalities": "nfoc__QKlMUHodmxwlY-Kp-6ewgX3CdK7Ia0RJHIXVo",
           "address": "ngnO4uQeOktM7YdFD8x82doS7WJnlZnq-rQE_RfuBSI"
```

```
         }
       },
       "birth_middle_name": "FeFSwd9drypEPtWVgIZ42N9j_yostt1Ds5PBpxT3Rng",
       "salutation": "57CMhvASQMNuzuQ0a_B1_VX5XdH73TcuPxyWGiorj5g",
       "msisdn": "leKbB0ro6q3jrVraCqt443uaGZVZisD3iGrKuKE2mqM"
     }
   }
```

The SD-JWT is then signed by the issuer to create a document like the following:

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogImNBRUlVcUowY21MekQxa3pHemhlaUJhZzBZUk
F6VmRsZnhOMjgwTmdIYUEifQ.eyJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1Z
XIiLCAic3ViX2p3ayI6IHsia3R5IjogIlJTQSIsICJuIjogInBtNGJPSEJnLW9ZaEF5UFd
6UjU2QVdYM3JVSVhwMTFfSUNEa0dnUzZXM1pXTHRzLWh6d0kzeDY1NjU5a2c0aFZvOWRiR
29DSkUzWkdGX2VhZXRFMzBVaEJVRWdwR3dyRHJRaUo5enFwcm1jRmZyM3F2dmtHanR0aDh
aZ2wxZU0yYkpjT3dFN1BDQkhXVEtXWXMxNTJSN2c2SmcyT1ZwaC1hOHJxLXE3OU1oS0c1U
W9XX21UejEwUVRfNkg0YzdQaldHMWZqaDhocFdObmJQX3B2NmQxelN3WmZjNWZsNnlWUkw
wRFYwVjNsR0hLZTJXcWZfZU5HakJyQkxWa2xEVGs4LXN0WF9NV0xjUi1FR21YQU92MFVCV
2l0U19kWEpLSnUtdlhKeXcxNG5IU0d1eFRJSzJoeDFwdHRNZnQ5Q3N2cWltWEtlRFRVMTR
xUUwxZUU3aWhjdyIsICJlIjogIkFRQUIifSwgImlhdCI6IDE1MTYyMzkwMjIsICJleHAiO
iAxNTE2MjQ3MDIyLCAiaGFzaF9hbGciOiAic2hhLTI1NiIsICJzZF9kaWdlc3RzIjogeyJ
2ZXJpZmllZF9jbGFpbXMiOiB7InZlcmlmaWNhdGlvbiI6IHsidHJ1c3RfZnJhbWV3b3JrI
jogIncxbVA0b1BjX0o5dGhCZXgwVGFRaTF2Z3hGbXJ1UUp4WllMRm5rTkZNYUkiLCAidGl
tZSI6ICJQdTNpMENXclBWTEpXLUxUMzB5RjFiRkJQUDE1QjYtdUtrM1BuR0RmbHY4IiwgI
nZlcmlmaWNhdGlvbl9wcm9jZXNzIjogIjhIcUlYUm1jenNkWU9aekdjTHFJNS1sOXhONVF
iSzJYRHRYbWRmSDd6LTQiLCAiZXZpZGVuY2UiOiBbeyJ0eXBlIjogIlRuTHVxR0dRbTZqZ
mVPb2E1dVgxZGlLQU5VUHVoLXpIcnBCRmRYOU1SLWciLCAibWV0aG9kIjogIlNhZ21ha29
TdS1YLVhVUElDM0VnZHJFRXdJV3hSV1hYNC1pNjhYOVR5RW8iLCAidGltZSI6ICJsZDJjN
W9ZRFJ0UWNmVTZQem9nUGt4Xzk1V1lxaHFJSk5WUk1uZmNzaWNZIiwgImRvY3VtZW50Ijo
geyJ0eXBlIjogInVmV2pEYUFhNTRNbkhlamkyWlVVSERkbnBaOXp4NkNVRzZ1UjI4Vk10c
1EiLCAiaXNzdWVyIjogeyJuYW1lIjogImE0R011Y1U3WmIwNjByMFN2ZDdodVk2UWhvMWJ
```

```
JZjN2MVU1QnZQUjhxNlkiLCAiY291bnRyeSI6ICIxMzVrOU0wbTJTQ25ZUnVPZkh1WVNjW
VZTMnEzZWVZN0lJdGd5UnNhQlQ4In0sICJudW1iZXIiOiAiY1V2T3hMVXA4UlY3VFRWbGl
FaXUtVFFJZWwtTHNFOEUtWGZVZ2ZxazVnayIsICJkYXRlX29mX2lzc3VhbmNlIjogIk5Jc
zhvbEpuSk92NEoxcUlFQkt1VHMyc0VGczRmZ0dKaE5xTTZ4ZFF0N0UiLCAiZGF0ZV9vZl9
leHBpcnkiOiAiSFRSMzd2THRBTlQ2TVdrLTlkQnFla0ZwQ3ZhVEc3ek5mMXplNTZyblY2N
CJ9fV19LCAiY2xhaW1zIjogeyJnaXZlbl9uYW1lIjogIk5COVhIX3lKS3FLT2hYRG1Ya1p
LcE1Da1JiT21PVGQ4YnFKRllEUllRblEiLCAiZmFtaWx5X25hbWUiOiAiaEFVYko2NlpZT
DlWSkxianNEcG1TczJlOUZmX09oaW1fV1I0YndaeXZvUSIsICJiaXJ0aGRhdGUiOiAiNlh
PUjRrNTZCZ1drNXRuTmlzbWJtRUh2b0dYN1JSZnk2WjhIRU5sOTZjVSIsICJwbGFjZV9vZ
l9iaXJ0aCI6IHsiY291bnRyeSI6ICJDTFRsaHV5MTNXV2MzX0lTb24xa0V5cEZ3dkNtZmh
MU3BHVU1DeUFVZzY4IiwgImxvY2FsaXR5IjogIkFRb1g4aXhHcHotaXB3ZUVHbEMtMnVtc
Xd5UWRoakllaVVCX1RLV2NFMkUifSwgIm5hdGlvbmFsaXRpZXMiOiAibmZvY19fUUtsTVV
Ib2RteHdsWS1LcC02ZXdnWDNDZEs3SWEwUkpISVhWbyIsICJhZGRyZXNzIjogIm5nbk80d
VFlT2t0TTdZZEZEOHg4MmRvUzdXSm5sWm5xLXJRRV9SZnVCU0kifX0sICJiaXJ0aF9taWR
kbGVfbmFtZSI6ICJGZUZTd2Q5ZHJ5cEVQdFdWZ0laNDJOOWpfeW9zdHQxRHM1UEJweFQzU
m5nIiwgInNhbHV0YXRpb24iOiAiNTdDTWh2QVNRTU51enVRMGFfQjFfVlg1WGRINzNUY3V
QeHlXR2lvcmo1ZyIsICJtc2lzZG4iOiAibGVLYkIwcm82cTNqclZyYUNxdDQ0M3VhR1pWW
mlzRDNpR3JLdUtFMm1xTSJ9fQ.i6xjiYhdM0pnDamaDfvsLWrjD2JOQBYbkWpLAykGFenK
B4nBN1E2gfbexMvlyRJc1AcU2wKdGQnRZsaNR-UYZ7yQPXrlBQH2RMkbRuQcI0K4n9KP8c
I8JnUVgERRBQSbtrInYrgRDvQ9K7nlyj4y6ULvVybcZpbw2sUv5YJwdBsdB8kAvGWKCEkW
NMvk-UGKZAKO78RPemWOTezLZq3Ubfj5GinITb6SN_Gu1rdLgA3ebKNkVqDBjI9q483d-9
g_MP8owwGzE_tCd8oj2bjRlRE1YgwN1lxAAGVpDrIJwWbuivh7BqUArvblEHe4L0ZITMhz
F5KKsEkYhkrr8q6qFg
```

(Line breaks for presentation only.)

A SD-JWT-R for some of the claims:

```
{
  "nonce": "XZOUco1u_gEPknxS78sWWg",
  "aud": "https://example.com/verifier",
  "sd_release": {
    "verified_claims": {
      "verification": {
        "trust_framework": "[\"2GLC42sKQveCfGfryNRN9w\", \"de_aml\"]",
        "time": "[\"eluV5Og3gSNII8EYnsxA_A\", \"2012-04-23T18:25Z\"]",
        "evidence": [
          {
            "type": "[\"eI8ZWm9QnKPpNPeNenHdhQ\", \"document\"]"
          }
        ]
      },
```

```
        "claims": {
          "given_name": "[\"y1sVU5wdfJahVdgwPgS7RQ\", \"Max\"]",
          "family_name": "[\"HbQ4X8srVW3QDxnIJdqyOA\", \"Meier\"]",
          "birthdate": "[\"C9GSoujviJquEgYfojCb1A\", \"1956-01-28\"]",
          "place_of_birth": {
            "country": "[\"kx5kF17V-x0JmwUx9vgvtw\", \"DE\"]"
          }
        }
      }
    }
  }
```

A.3. Example 4 - W3C Verifiable Credentials Data Model

This example illustrates how the artifacts defined in this specification can be represented using W3C Verifiable Credentials Data Model as defined in [VC_DATA].

Below is a non-normative example of an SD-JWT represented as a verifiable credential encoded as JSON and signed as JWS compliant to [VC_DATA].

SVC sent alongside this SD-JWT as a JWT-VC is same as in Example 1.

```
  {
    "sub": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "jti": "http://example.edu/credentials/3732",
    "iss": "https://example.com/keys/foo.jwk",
    "nbf": 1541493724,
    "iat": 1541493724,
    "exp": 1573029723,
```

```
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": [
      "VerifiableCredential",
      "UniversityDegreeCredential"
    ]
  },
  "sd_digests": {
    "given_name": "fUMdn88aaoyKTHrvZd6AuLmPraGhPJ0zF5r_JhxCVZs",
    "family_name": "9h5vgv6TpFV6GmnPtugiMLl5tHetHeb5X_2cKHjN7cw",
    "birthdate": "fvLCnDm3r4VSYcBF3pIlXP4ulEoHuHOfG_YmFZEuxpQ"
  }
}
```

Below is a non-normative example of an SD-JWT-R represented as a verifiable presentation encoded as JSON and signed as a JWS compliant to [VC_DATA].

```
{
  "iss": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "aud": "s6BhdRkqt3",
  "nbf": 1560415047,
  "iat": 1560415047,
  "exp": 1573029723,
  "nonce": "660!6345FSer",
  "vp": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1"
    ],
    "type": [
      "VerifiablePresentation"
    ],
    "verifiableCredential": ["eyJhb...npyXw"]
  },
  "sd_release": {
    "given_name": "[\"6Ij7tM-a5iVPGboS5tmvVA\", \"John\"]",
    "family_name": "[\"eI8ZWm9QnKPpNPeNenHdhQ\", \"Doe\"]",
    "birthdate": "[\"5bPs1IquZNa0hkaFzzzZNw\", \"1940-01-01\"]"
  }
}
```

Appendix B. Document History

[[To be removed from the final specification]]

-02

- Added acknowledgements
- Improved Security Considerations
- Stressed uniqueness requirements for salts
- Python reference implementation clean-up and refactoring
- hash_alg renamed to sd_hash_alg

-01

- Editorial fixes
- Added hash_alg claim
- Renamed _sd to sd_digests and sd_release
- Added descriptions on holder binding - more work to do

- Clarify that signing the SD-JWT is mandatory

-00

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

- Make holder binding optional

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

- Improved the structure of the description

- Described verification steps

- All examples generated from python demo implementation

- Examples for structured objects

Authors' Addresses

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

Kristina Yasuda
Microsoft
Email: Kristina.Yasuda@microsoft.com