Decoupling Issuance & Presentation

Identity Federation

Identity Provider

End-User

Set of Claims

Relying Party

Decoupled Model

Issuer

Reusable, contains **all claims** known to the Issuer

End-User

(Holder)

Subset from the Issuer-signed credential relevant to given request.

Verifier

RP chooses which claims are required to be released to receive service.

Only claims relevant to given request: Already selectively disclosed!
Selective Disclosure

Issuer issued a whole set of claims:

{
    "iss": "https://server.example.com",
    "sub": "some-user-identifier",
    "aud": "s6BhdRkqt3",
    "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"
}

But Verifier needs only a subset in a given request:

{
    "iss": "https://server.example.com",
    "sub": "some-user-identifier",
    "aud": "s6BhdRkqt3",
    "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"
}
SD-JWT

Selective Disclosure for JWTs
“Simple” is a feature.
## Why another Spec?

<table>
<thead>
<tr>
<th></th>
<th>Existing SD schemes</th>
<th>SD-JWT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complexity</strong></td>
<td>(Very) Complex</td>
<td>As simple as possible</td>
</tr>
<tr>
<td><strong>Algorithms</strong></td>
<td>Advanced cryptography</td>
<td>Standard cryptography with salted hashes</td>
</tr>
<tr>
<td><strong>Format</strong></td>
<td>Binary formats</td>
<td>JSON &amp; JWT</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Hard to audit</td>
<td>Easy to understand &amp; verify</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>Lack of implementations</td>
<td>Widely-available JWT libraries can be leveraged;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Already 4 independent implementations</td>
</tr>
<tr>
<td><strong>Use Cases</strong></td>
<td>often limited to the identity use cases</td>
<td>Universal (beyond identity use cases)</td>
</tr>
</tbody>
</table>
Salted Hash Approach

Idea:
- Issuer hashes each claim value together with a random salt
  - "John" → hash(salt, "John") → "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0"
- Issuer-signed credential only contains digests
  - "given_name": "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0"
- Holder selectively releases salt + plain-text value
  - "given_name" = hash("eluV50g3gSNII8EYnsxA_A", "John")
- Verifier checks by calculating hashes & issuer’s signature
Overview: Salted Hash Approach
Issuer hashes each claim value together with a random salt

- "John" → hash(salt, "John") → "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0"

Issuer-signed credential only contains digests

- "given_name": "PvU7cWjuHUq6w-i9XFpQZhjT-uprQL3GH3mKsAJl0e0"
Issuer sends a mapping of plain-text claim values and unique salts

-> given_name" : "["eluV50g3gSNII8EYnsx_A", "John"]
Holder selectively releases salt + plain-text value
-> "given_name" = hash("eluV5Og3gSNII8EYnsxA_A", "John")

Verifier checks by calculating hashes & issuer’s signature
Deep-dive
Deep-dive on SD-JWT

SD-JWT
hashed claim values

Issuer

Issuance

End-User
(Holder)

SD-JWT
hashed claim values

Veriﬁer

Presentation

Salt/Value Container (SVC)
mapping of plain-text claim values and unique salts used in hashing

SD-JWT-Release
plain-text claim values and unique salts of selectively disclosed claims

✓ signed by Issuer

 ✓ signed by Issuer

✓ optionally signed by holder

✓ signed by Issuer

sent as-is
Example: SD-JWT

Issuer creates & sends to holder:

```
{
    "iss": "https://example.com/issuer",
    "sub_jwk": {  # optional: public key of holder
        "kty": "RSA",
        "n": "pm4bOHBg-oYhAyP(...)7ihcw",
        "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-ReI4cEBMlenyK1gvxy16QVpnt4MEclT5tP0aTLFU",
        "email": "ET2A1JQLF85ZpBulh6UFstGrSFr4B3KM-bjQVllhxqY",
        "phone_number": "SJnciB2DIRVA5cXBrdKoH6n45788mZyUn2rnv74uMVU",
        "address": "0FldqLfGnERPPVDC17od9xb4w3iRJTEQbw_Yk9AmmDw",
        "birthdate": "-L0kMgIbLXe30EktUgewz_QKhjehDeofKGwoPrxLuo4"
    }
}
```
Deep-dive on SVC

**Issuance**

- **Issuer**
  - SD-JWT: hashed claim values
  - Salt/Value Container (SVC) mapping of plain-text claim values and unique salts used in hashing

**Presentation**

- **End-User** (Holder)
  - SD-JWT: hashed claim values
  - SD-JWT-Release: plain-text claim values and unique salts of selectively disclosed claims

**Verifier**

- Signed by Issuer
- Optionally signed by holder

- Issuance: SD-JWT signed by Issuer
- Presentation: SD-JWT signed by Issuer, optionally signed by holder
- SVC: selectively disclosed

- "signed" and "sent as-is" symbols indicate action status.
Issuer creates & sends to holder together with SD-JWT:

```
{
    "sd_release": {
        "sub": "["2GLC42sKQveCfGfryNRN9w", "6c5c0a49-b589-431d-bae7-219122a9ec2c"]",
        "given_name": "["eluV5Og3gSNII8EYnsxA_A", "John"]",
        "family_name": "["6Ij7tM-a5iVPgboS5tmvVA", "Doe"]",
        "email": "["eI8ZWM9QbKppNPeNnHDhQ", "johndoe@example.com"]",
        "phone_number": "["Qg_064zAxe412a108iroA", "+1-202-555-0101"]",
        "address": "["AJx-095VPrpN4QMoqROA", {"street_address": "123 Main St", "locality": "Anytown", "region": "Anystate", "country": "US"}]",
        "birthdate": "["Pc33JM2LchCULHggv_ufQ", "1940-01-01"]"
    }
}
```

JSON string literals...
- ...ensure all parties hash the same values
- ...obsolete canonicalization even for complex claim values
- ...with extremely simple implementation

**Example: Salt/Value Container**

JSON strings?!
Deep-dive on SD-JWT-Release

Issuer

SD-JWT
hashed claim values

End-User
(Holder)

SD-JWT
hashed claim values

Presentation

Salt/Value Container (SVC)
mapping of plain-text claim values and
unique salts used in hashing

Verifier

SD-JWT-Release
plain-text claim values and unique salts of
selectively disclosed claims

✓ signed by Issuer

✓ signed by Issuer

✓ optionally signed by holder

Issuance

Sent as-is

selectively disclosed
Example: SD-JWT-Release

Holder creates from SVC & sends to verifier together with SD-JWT:

```
{
  "nonce": "XZ0Ucolu_gEPknxS78sWWg",
  "aud": "https://example.com/verifier",
  "sd_release": {
    "given_name": ["eluV50g3gSNI18EyNsxA_A", "John"],
    "family_name": ["6Ij7tM-a5iVGboS5mvVA", "Doe"]
  }
}
```

Verifier checks:
- SD-JWT is valid (signature, etc.)
- SD-JWT-Release is valid (if signed: signature, nonce, etc.)
- digests in SD-JWT match hashes of released claims

Then: Extract plain-text values from released claims

Foot-gun: Verifier MUST calculate and check digests before using claims!

Salts and claim values of the claims that the Holder is releasing to the Verifier
Granularity of hashing in SD-JWT

{  
  "address": {  
    "street_address": "123 Main St",  
    "locality": "Anytown",  
    "region": "Anystate",  
    "country": "US"  
  }  
}

or

one digest per claim

{  
  "sd_digests": {  
    "address": "0FldqLfGnERPPVDC17od9xb4w...",  
  }  
}

digest per component of the claim

{  
  "sd_digests": {  
    "address": {  
      "street_address": "07_1sdGcmZqcSobPVPmMgmJwB41hPUHHG8jg5L5f8YzY",  
      "locality": "w-Ztf6ljkQLTvVyp_3Nyd3t5Waj-B2vb0AXH1q80Sy",  
      "region": "nTvQpG6YQwEZipVBK9WKh0wriqsRjEhrxhQ24",  
      "country": "u-01yDqDTq0gUBSjWlqkMLzq_QOTELMF2rRT5e6k"  
    }  
  }  
}
Works with Complex Objects!

Example of OpenID for Identity Assurance

For full example, see the spec

```
{
  "sd_digests": {
    "verified_claims": {
      "trust_framework": "w1mP4oPc_J9thBex0TaQilvgxFmrqJxZYLFnkNFMaI",
      "time": "Pu3i0CWrPVLJW-LT30yF1bFBPP15B6-uKK3PnGdflv8",
      "verification_process": "8HqIXRmczs3dQZcGfLqI5-L9xN5Q5bK2XDtxmdfH7z-4",
      "evidence": [{
        "type": "TnLuqGQm6jeOoa5uXldiKANUPuh-zHrpBFdX9MR-g",
        "method": "Sagmakosu-X-XUPIC3EdgrEEwIWXwKXVx-w468X9TyEo",
        "time": "ld2c5oYDR7QcU6pogPkv_95WYqhqIJNVRMnfsacy",
        "document": {
          "type": "ufWjDaAa54MnHeji2ZUUhDdpnZ9zx6CU6uR28VMtsQ",
          "issuer": {
            "name": "a4GMucU7Zb060r0Svd7huY6Qho1bFv3l15Bv4PR8q6Y",
            "country": "135k9M0m2SCnYRvu0HFuYScyYSv2q3eeY1ltyRsaBT8"
          },
          "number": "cUVoXlU8pRV7TVliEiu-TQIel-LsEBE-xfUgL5gk",
          "date_of_issuance": "NIs8tlJn3ov4j1qIEBKvIt2sEs4fG3JNhM6XdxQt7E",
          "date_of_expiry": "HTR3vLtANT6MwK-9d8qekFpCvaTG7zNflze56rnV64"
        }
      },
      "birth_middle_name": "FeFSwed9dreyPteWvG1z42N9j_yostt1Ds5PBpxT3Rng",
      "salutation": "57CMhvASQMNuzuQ0a_B1_VX5XH73TcuPyWioj5g",
      "msisdn": "leKb80ro6Q3jrVraCq443uaGZViszD3iGrKuKE2mQm"
    }
  }
}
```
Running Code:
4 independent implementations!

- **Python**: https://github.com/oauthstuff/draft-selective-disclosure-jwt
- Kotlin: https://github.com/IDunion/SD-JWT-Kotlin
- Rust: https://github.com/kushaldas/sd_jwt
- TypeScript: https://github.com/christianpaquin/sd-jwt

---

### Produce SD-JWT and SVC

```python
sdjwt = SDJWT(
    user_claims,
    issuer,
    ISSUER_KEY,
    HOLDER_KEY,
    claims_structure,
    blinded_claim_names,
    iat,
    exp,
)
```

- Reference implementation in python: ~500 LoC.
- Evolves with the spec.
- Examples in the spec generated from the code.
### Use-Case: W3C VC-Data-Model

#### JWT-VC (= SD-JWT)

```json
{"sub": "did:example:ebfeb1f712ebc6f1c276e1e2ec21",
 "jti": "http://example.edu/credentials/3732",
 "iss": "https://example.com/keys/foo.jwk",
 "nbf": 1541493724,
 "exp": 1573029723,
 "vc": {
   "@context": [
     "https://www.w3.org/2018/credentials/v1",
     "https://www.w3.org/2018/credentials/examples/v1"
   ],
   "type": ["VerifiableCredential", "UniversityDegreeCredential"],
   "credentialSubject": {
     "sd_digests": {
       "given_name": "fUMdn8aoyKTHrvZd6AuLmPraGhPJ0xF5r_JhxCVZs",
       "family_name": "9h5vgv6TpFV6GmPtugJML5tHeeb5X_2cKHjN7cw",
       "birthdate": "fvLCnDm3r4V5YcBF3pILXP4ulEoHuH0f6_YmFZEuxPQ"
     }
   }
  }
}
```

#### JWT-VP (= SD-JWT-Release)

```json
{"iss": "did:example:ebfeb1f712ebc6f1c276e1e2ec21",
 "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": ["eI8Zwm9QnPpNPeNenHdHQ", "Doe"]
   }
  }
}
```

Note: specific syntax is under discussion.
Current Status &
Next Steps
Draft Status

- Functional specification complete and somewhat stable
- New: Approach to blinding claim names
- Some missing parts in security and privacy considerations
- Already considered a serious alternative to many existing formats
- Adopted in EU-projects & by Microsoft
OAuth WG?

- JWT was developed in OAuth WG
- Expected usage in OAuth and OAuth-based technologies
- SD-JWT is universal: Selectively disclosable access tokens?
Call for adoption?
Additional Slides
Blinding Claim Names

Hashing not only claim values but also claim names.

Credential

```json
{
    "iss": "https://server.example.com",
    "sub": "some-user-identifier",
    "aud": "s6BhdRkqt3",
    "given_name": "John",
    "family_name": "Doe",
    "email": "johndoe@example.com",
    "phone_number": "+1-202-555-0101",
    "secret_club_membership_no": "42-23-23"
}
```

✓ signed by Issuer

```json
{
    "iss": "https://server.example.com",
    "sub": "some-user-identifier",
    "aud": "s6BhdRkqt3",
    "given_name": "John",
    "family_name": "Doe",
    "email": "johndoe@example.com",
    "phone_number": "+1-202-555-0101",
    "secret_club_membership_no": "42-23-23"
}
```

✓ signed by Issuer