OIDC Advanced Syntax for Claims (ASC)

Transformed Claims & Selective Abort/Omit

Daniel Fett, 2021-05-12

OIDC Advanced Syntax for Claims (ASC)

- Defines extensions for OIDC around **requesting and receiving Claims**
- No dependency on OIDC4IA, but:
    - Requirements derived from eKYC work
    - Special provisions for combination cases with OIDC4IA
- Two independent extensions:
    - ASC/SAO: Selective Abort/Omit
    - ASC/TC: Transformed Claims (among others, for age verification)

ASC/SAO:
Selective Abort/Omit

Selective Abort/Omit

Formerly known as PR #52.

```
{
   "id_token": {
       "phone_number": {
           "if_unavailable": "abort"
       },
       "custom_paid_claim": {
           "if_unavailable": "omit_set"
       },
       "verified_claims": {
           "verification": {
               "trust_framework": {
                   "value": "de_aml",
                   "if_different": "abort"
               },
               "verification_process": {
                   "if_unavailable": "omit_verified_claims"
               }
           },
           "claims": {
               "given_name": null,
               "family_name": null,
               "place_of_birth": {
                   "if_unavailable": "omit_set"
               }
           }
       }
   }
}
```

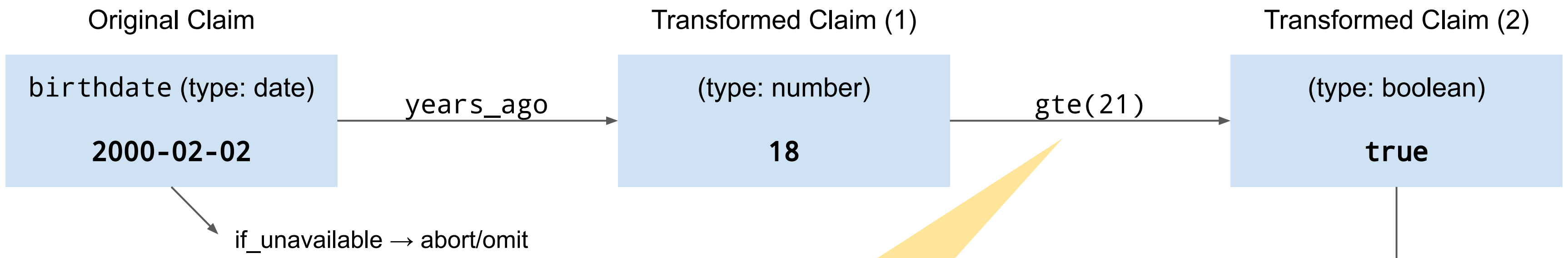
ASC/TC:
Transformed Claims

Use Cases

- Age Verification:
    - Above 16? Above 18? Above 21? Under 99?
- Partial matching:
    - E-Mail ends with ‘@company.com’
    - ZIP code is ‘90210’
    - address/country is not empty
    - Nationalities contains ‘JPN’
- Data minimization:
    - Return only address/country instead of address

Idea

Claims values can be transformed using a small set of functions before any further evaluation is performed:



- no side effects
- only static arguments
- only one base claim or input, no external data allowed

Normal OIDC/eKYC claim handling:

1. (optional) check against value/values
2. (optional) handle if_different → abort/omit
3. return “true” to RP

Example: Age Verification

Definition

Use - Prefix ‘:’

```
claims=
{
 "transformed_claims": {
   "above_18": {
     "claim": "birthdate",
     "fn": [
       "years_ago",
       ["gte", 18]
     ]
   }
 },
 "id_token": {
   "given_name": null,
   "family_name": null,
   ":above_18": null
 }
}
```

base claim

1st function

2nd function

```
Response:
{
 ...
 "given_name": "Max",
 "family_name": "Mustermann",
 ":above_18": true,
 ...
}
```

Simple, Self-Contained Functions

- `years_ago(optional date ReferenceDate): date → number`
  Takes a date (or datetime), calculates the number of years since the date. Optionally, a reference date is given.

- `gt(number Threshold): number → boolean`
  `lt(number Threshold): number → boolean`
  Evaluate whether a number is above/below a certain threshold.

- `any(): array of booleans → boolean`
  `all(): array of booleans → boolean`
  `none(): array of booleans → boolean`
  Evaluate whether, in an array of booleans, any, all, or none of the values are “true”.

- `eq(any Compare): any → boolean`
  Evaluates equality - useful in combination with any/all/none for arrays.

- `get(string Key): JSON object → any`
  Access the key of a JSON object; returns the value.

- `match(string Regex): string → bool`
  Match a string against a regular expression. (Todo: Define a regex dialect and/or subset to support.)

Example: Partial Matching

```
claims=
{
   "transformed_claims": {
       "company_email": {
           "claim": "email",
           "fn": [["match", "@company\\.com$"]]
       },
       "nationality_usa": {
           "claim": "nationalities",
           "fn": [["eq", "USA"], "any"]
       }
   },
   "id_token": {
       ...
       ":company_email": { "value": true, "if_different": "abort" },
       "email_verified": { "value": true, "if_different": "abort" },
       "verified_claims": {
           "claims": {
               ":nationality_usa": { "value": true, "if_different": "abort" }
           },
           "verification": { "trust_framework": null }
       }
   }
}
```

Simplifying Common Use Cases

- OPs can opt to support only a limited subset of functions:

OP Metadata:
```
"transformed_claims_functions_supported": ["years_ago", "gte"]
```

- OPs can provide Predefined Transformed Claims (PTC):

OP Metadata:
```
"transformed_claims_predefined": {
   "above_18": {
     "claim": "birthdate",
     "fn": [
       "years_ago",
       ["gte", 18]
     ]
   }
 }
```

- OPs can limit support to PTCs only:

OP Metadata:
```
"transformed_claims_restricted": true,
```

Example: Age Verification with PTC

:: indicates PTC

```
claims=
{
 "id_token": {
   "given_name": null,
   "family_name": null,
   "::above_18": null
 }
}
```

```
Response:
{
 ...
 "given_name": "Max",
 "family_name": "Mustermann",
 "::above_18": true,
 ...
}
```

With PTCs, simple use cases can be handled with **minimal implementation overhead,** both for OP and RP.

The PTC is handled just like any other custom Claim, but has a precisely-defined meaning.

UX Considerations

- For PTCs, OPs can trivially show a meaningful consent prompt
- For Custom TCs, OPs can try to match patterns:
  - e.g. birthdate / years_ago / gte(x) → Consent: “RP wants to know whether you are x years old or above”.
- Safe fallback:
  - Show consent to release of full Claim (“wants to know your birth date”)
  - → safe overapproximation because:
    - no side effects,
    - no expressions over multiple Claims,
    - no dynamic arguments

Compatibility Considerations

- New element “transformed_claims” will be ignored by non-supporting OPs
- Transformed Claims will be ignored by non-supporting OPs
- RPs can check OP support in metadata
- Ecosystems can define custom functions
- Can be used with and without ASC/SAO.