OIDC
Advanced Syntax for Claims
(ASC)

Transformed Claims & Selective Abort/Omit

Mark Haine (with thanks to Daniel Fett)

OIDC Advanced Syntax for Claims (ASC)

Defines extensions for OIDC around **requesting and receiving Claims**

But… why?

- Fine-grained control over data delivery
  - … for privacy
  - … for billing
    - Clients need to be able to define **what they want,**
    - and **don’t pay** for data useless to them.
- Handling of complex Claims
  - OpenID Connect Core: **2** levels (root+1, e.g., in ‘address’)
  - OpenID for Identity Assurance: **5+** levels

OIDC Advanced Syntax for Claims (ASC)

- No dependency on OpenID Connect for Identity Assurance (OIDC4IDA), but:
  - Requirements inspired from discussions in eKYC & IDA working group
  - Special provisions for combination cases with OIDC4IDA
- Two independent extensions:
  - ASC/TC: Transformed Claims (among others, for age verification)
  - ASC/SAO: Selective Abort/Omit

ASC/TC

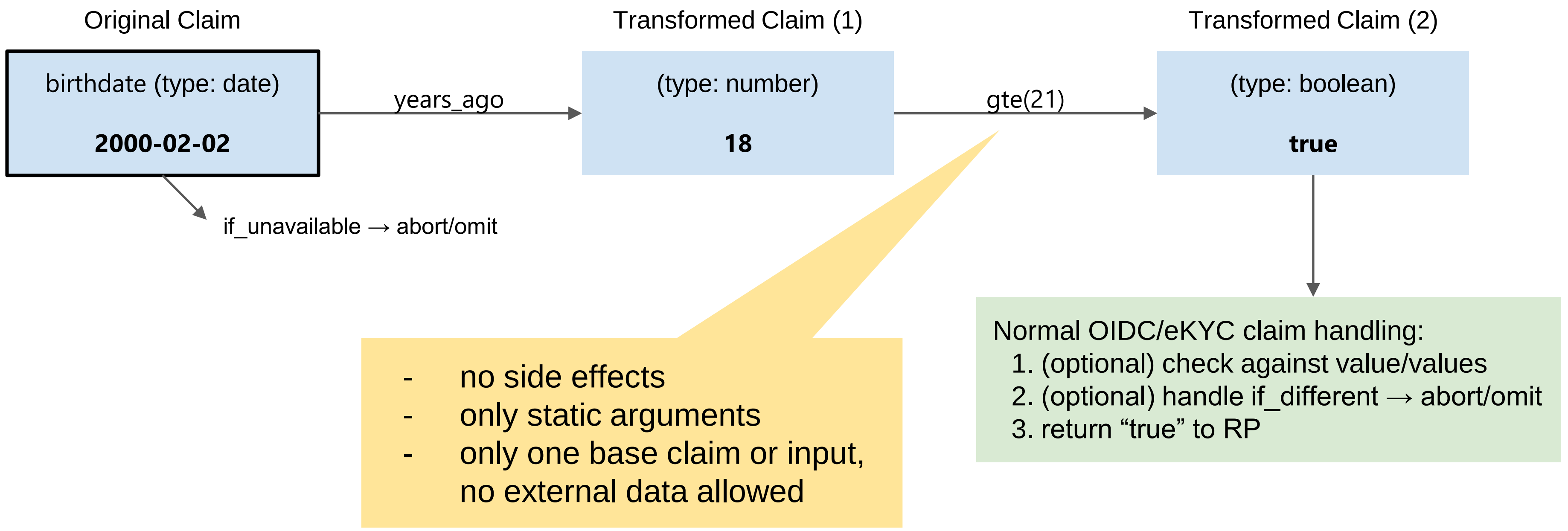
Transformed Claims

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 },
     "email_verified": { "value": true },
     "verified_claims": {
        "claims": {
           ":nationality_usa": { "value": true }
        },
        "verification": { "trust_framework": null }
     }
  }
}
```

1st function

1st function

2nd function

Simplifying Implementations

- 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 over-approximation because:
    - no side effects,
    - no expressions over multiple Claims,
    - no dynamic arguments

ASC/SAO

Selective Abort/Omit

“the feature you expected essential:true to be”

What SAO provides

Enables further data minimisation

Allows for elements of the response to be conditionally left out

- conditions are expressed
- for a given action

Trouble with SAO

- Our first couple of iterations of this were flawed
- Major issue was a race condition where order of processing really affected the outcome

Outcome

A more flexible way of expressing the SAO feature (without race condition)

Uses:

- JSON schema based definition to identify the element the condition needs to use as input
- The action that should be done if the condition matches
- The element the action should apply to

Example Combine JSONSchema Partial Application and “filter”

Allow both options to provide a less verbose option.

“filter” is essentially a shorthand version for a simple schema.

```
{
    "id_token": {
        "verified_claims": {
            "verification": {
                "trust_framework": null,
                "assurance_level": {
                    "value": "example_assurance_level"
                }
            },
            "claims": {
                "family_name": {
                    "value": "nonexistent_family_name"
                },
                "given_name": null,
                "birthdate": null
            }
        },
        "asc/sao-schemas": [
            {
                "location": "/verified_claims/claims",
                "schema": {
                    "$schema": "http://json-schema.org/draft-07/schema#",
                    "type": "object",
                    "properties": {
                        "birthdate": {
                            "type": "string",
                            "const": "1900-01-01"
                        }
                    }
                },
                "otherwise": "omit",
                "what": [
                    "/verified_claims/claims"
                ]
            },
            {
                "pointer": "/verified_claims/claims/birthdate",
                "filter": {
                    "type": "string",
                    "const": "1990-01-01"
                },
                "otherwise": "omit",
                "what": [
                    "/verified_claims/claims"
                ]
            }
        ]
    }
}
```

Thank You