GA4GH Passport Visa Conditions

Authors: craigv@google.com
Last updated: Sept 18, 2019

Background

Given the scope of changes to the rest of the passport spec, the "condition" field was left pretty much untouched. Now that the Passport Visa pieces are in place (perhaps with some edits to come as we find gaps), this is one of the last opportunities to revisit the format and figure out what we want to do about it for v1.0.

It is difficult to introduce new condition syntax and semantics later without making it a breaking change. New condition types cannot simply be ignored as they must be enforced. Therefore, we should assume that we do not have an opportunity to materially change conditions until v2.0 at a minimum.

Current Syntax

```
"condition": {
  "<PassportVisaType1>" : {
    "<FieldName1>": [
      "<Value1a>",
      "<Value1b>"
    ],
    "<FieldName2>": ["<Value2>"],
    ...
  },
```

```
  "<PassportVisaType2>" : { ... }
}
```

Current Semantics

- The Passport Clearinghouse MUST verify that for each condition claim and each condition field present, a single corresponding Passport Visa Object and its corresponding fields match as per the matching algorithms described elsewhere in this specification, along with the following requirements:
- Checking the correctness of the condition MUST be performed first. For example, the field name but be a valid choice.
- A condition field matches when any one string within the specified list matches a corresponding claim's field in the Passport.
- All condition fields that are specified MUST match the same Passport Visa Object in the Passport.

Current Example

```
"condition": {
  "AffiliationAndRole": {
    "value": [
      "faculty@uni-heidelberg.de",
      "student@uni-heidelberg.de"
    ],
    "by": [
      "so",
      "system"
    ]
  }
}
```

Feedback

1. The format was chosen to match the format of the former "ga4gh" JWT claim. Should it look more like a Passport Visa now?
2. Wording about "condition fields" doesn't seem to align as well to new Passport Visa format.
3. No boolean logic is available. Martin K suggested this would be desirable.

Proposals

Proposal 1: Refactor object layout

```
"conditions": [
  {
    "type": "AffiliationAndRole",
    "value": [
      "faculty@uni-heidelberg.de",
      "student@uni-heidelberg.de"
    ],
    "by": [
      "so",
      "system"
    ]
  },
  {
    ...
  }
]
```

Description:

- Move conditions to be a list of conditions, ALL of which must be met (implicit AND).
- Perhaps "type" is single valued while other "fields" are all list of ORed options.
- Leave most of the semantics the same for how evaluation works.
- Additional **Option 1a**: If value fields like "faculty@example.org" start with "^" AND end with "$", then the string it to be interpreted as a regex. Example: "^faculty@.*$".

Pros	**Cons**
Looks more aligned with Passport Visas	Non-intuitive
Allows the AND of multiple Passport Visas with the same Passport Visa Type	More difficult to extend in a fluid, backwards compatible way.
With option 1a: allows matching of substrings of structured values like AffiliationAndRole, LinkedIdentities, etc.	Still a fairly limited syntax

Proposal 2: Boolean Language

```
"conditions": [
  `AffiliationAndRole.value="faculty@cam.ac.uk"`,
  `AffiliationAndRole.value in [
     "faculty@uni-heidelberg.de", "student@uni-heidelberg.de"
   ]
   and
   AffiliationAndRole.by in [ "so", "system" ]`,
]
```

The above example introduces the following "conditions" to a given Passport Visa:

- The Passport must also have a Passport Visa that asserts an AffiliationAndRole of faculty@cam.ac.uk; and
- The Passport must also have a Passport Visa that asserts a faculty or student AffilationAndRole at Heidelberg by either an "so" or "system".
- Both of the above conditions must be met in the same Passport.

Description:

- Within each expression string (there are two in the above example), the expression must match a single Passport Visa. Use multiple entries within "conditions" to AND across Passport Visas similar to Proposal 1 and existing semantics.
- Language could be a simple grammar of:
    - {expression} one of:
        - X.Y = "string"
        - X.Y in ["string", "string", ...]
    - {condition} one of:
        - {expression}
        - {expression} AND {expression}
    - Allow "string" to also be regex via one of the following:
        - Same as Option 1a: "^matcher$"
        - Modify {expression} to include /matcher/ or regexp("matcher")
        - Introduce regex operator ~=
- If the language does not parse, then permission denied (i.e. backwards compliant by not being overly permissive).
- Easy to make fully backwards compatible as the language gets extended.
- **Option 2a**: alternative is to allow a full language like CEL off the start. Finding an OSS library that works in many common languages could be a challenge, and difficult to specify in a spec.

Pros	Cons
More expressive	Complexity of finding a reliable library available in many languages
More extensible with backwards compatibility	Possible security issues due to complexity of implementation and vulnerabilities
Restrictive enough to avoid open language problems / attacks (can address differently in future spec versions)	

Proposal 3: Boolean Flexible Struct

```
"condition": {
  "and": [
    { "check": "=",
      "type": "AffiliationAndRole"
    },
    {
      "or": [
        {
          "check": "=",
          "value": "faculty@cam.ac.uk"
        },
        {
          "check": "regex",
          "value": "^faculty@"
        }
      ]
    },
    { "check": "=",
      "by": "so"
    }
  ]
}
```

Description:

- Parsed boolean logic into a structure
    - {expression} = {check} | {and} | {or}
    - {and} = [{expression}, ...]
    - {or} = [{expression}, ...]
    - {check} = JSON { "check": "=" | "!=" | "regex", {field}: {value}, … }

    - {field} = "type" | "value" | "source" | "by"
    - {value} = {{string}}
- If the language does not parse, then permission denied (i.e. backwards compliant by not being overly permissive).

Pros	**Cons**
Expressive	Hard for humans to encode
	Need tools to help non-programmers

Proposal 4: Boolean Fixed Struct

```
"conditions": [
  [
    {
      "type": "AffiliationAndRole",
      "value": "const:faculty@uni-heidelberg.de",
      "by": "const:so"
    }, // AND
    {
      ...
    }
  ], // OR
  [
    {
      "type": "AffiliationAndRole",
      "value": "pattern:faculty@*",
      "source": "const:https://visas.elixir.org"
      "by": "const:system"
    }
  ]
]
```

Description:

- Conditions is a two-nested-lists structure (Disjunctive Normal Form):
    - Outer level list is a set of OR clauses
    - Inner level list is a set of AND clauses that contain Condition Objects
    - A Condition Object MUST specify a "type" to match the Passport Visa Type along with at least one other field with a name that matches a Passport Visa Object field name.

- The values of Condition Object fields MUST have a string prefix followed by a colon and then string suffix except for "type" where it MUST be assumed to be "const" and is not specified.
  - If prefix is "const", then suffix MUST match using a case sensitive value by comparing full strings.
  - If prefix is "pattern", then suffix MUST use full string Glob pattern matching.
  - If prefix is "split_pattern", then split the input string on ";" and then use "pattern" to match any one substring. (this supports cases when value fields encode more than one value like LinkedIdentities)
  - If prefix is unknown or unsupported, then condition must fail to match.

Pros	**Cons**
Expressive enough to handle foreseeable use cases	Hard for humans to remember the definitions of the different levels
Reasonable in terms of implementation complexity	Need tools to help non-programmers
Easier to ensure security than some alternatives (avoids regex, deep substructures causing JSON parser issues, etc)	