Logius Standard - Draft

NLGov Profile for OpenID AuthZEN Authorization API

Logius Standard
Draft October 06, 2025

This version:
https://vng-realisatie.github.io/authzen-nlgov/

Latest published version:
https://gitdocumentatie.logius.nl/publicatie/dk/authzen-nlgov/

Latest editor's draft:
https://vng-realisatie.github.io/authzen-nlgov/

Previous version:
https://gitdocumentatie.logius.nl/publicatie/dk/authzen-nlgov/null/

Editor:
Project Federatieve Toegangsverlening (MinBZK)

Author:
Project Federatieve Toegangsverlening (MinBZK)

Participate:
GitHub vng-realisatie/authzen-nlgov
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Abstract

The Authorization API enables Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) to communicate authorization requests and decisions to each other without requiring knowledge of each other's inner workings. The Authorization API is served by the PDP and is called by the PEP. The Authorization API includes an Evaluation endpoint, which provides specific access decisions. Other endpoints may be added in the future for other scenarios, including searching for subjects, resources or actions.

Status of This Document

This is a draft that could be altered, removed or replaced by other documents. It is not a recommendation approved by TO.

Table of Contents

Abstract

Status of This Document

1. Introduction

2. Model

3. Features

4. API Version

5. Information Model

5.1 Subject

5.1.1 Subject Properties

5.1.1.1 IP Address

5.1.1.2 Device ID

5.2 Resource

5.2.1 Examples (non-normative)

5.3 Action

5.3.1 Action Properties

5.3.1.1 Processing Activity and Algorithm identifiers

5.4 Context

5.4.1 Context Properties

5.4.1.1 Time

5.4.1.2 W3C Trace Context

6. Access Evaluation API

6.1 The Access Evaluation API Request

6.1.1 Example (non-normative)

6.2 The Access Evaluation API Response

6.2.1 Access Evaluation Decision

6.2.2 Additional Context in a Response

6.2.3 Example Context

6.2.3.1 Reasons
6.2.3.1.1 Reason Field
6.2.3.1.2 Reason Object
6.2.4 Sample Response with additional context (non-normative)

7. Access Evaluations API
7.1 The Access Evaluations API Request
7.1.1 Default values
7.1.2 Evaluations options
7.1.2.1 Evaluations semantics
7.1.2.1.1 Example: Evaluate `read` action for three documents using all three semantics
7.2 Access Evaluations API Response
7.2.1 Errors

8. Subject Search API
8.1 Subject Search Semantics
8.2 The Subject Search API Request
8.2.1 Example (non-normative)
8.3 The Subject Search API Response
8.3.1 Paged requests
8.3.1.1 Example

9. Resource Search API
9.1 Resource Search Semantics
9.2 The Resource Search API Request
9.2.1 Example (non-normative)
9.3 The Resource Search API Response
9.3.1 Paged requests
9.3.1.1 Example

10. Action Search API
10.1 Action Search Semantics
10.2 The Action Search API Request
10.2.1 Example (non-normative)
10.3 The Action Search API Response
10.3.1 Paged requests
10.3.1.1 Example

11. Policy Decision Point Metadata
11.1 Data structure
11.1.1 Endpoint Parameters
11.1.2 Signature Parameter

11.2 Obtaining Policy Decision Point Metadata
11.2.1 Policy Decision Point Metadata Request
11.2.2 Policy Decision Point Metadata Response
11.2.3 Policy Decision Point Metadata Validation

12. Transport
12.1 HTTPS Binding
12.1.1 HTTPS Access Evaluation Request
12.1.2 HTTPS Access Evaluation Response
12.1.3 HTTPS Access Evaluations Request
12.1.4 HTTPS Access Evaluations Response
12.1.5 HTTPS Subject Search Request
12.1.6 HTTPS Subject Search Response
12.1.7 HTTPS Resource Search Request
12.1.8 HTTPS Resource Search Response
12.1.9 HTTPS Action Search Request
12.1.10 HTTPS Action Search Response
12.1.11 Error Responses
12.1.12 Request Identification
12.1.13 Request Identification in a Response

13. Security Considerations
13.1 Communication Integrity and Confidentiality
13.2 Policy Confidentiality and Sender Authentication
13.3 Trust
13.4 Availability & Denial of Service
13.5 Differences between Unsigned and Signed Metadata
13.6 Metadata Caching

14. IANA Considerations
14.1 AuthZEN Policy Decision Point Metadata Registry
14.1.1 Registration Template
14.1.2 Initial Registry Contents
14.2 Well-Known URI Registry
14.2.1 Registry Contents

15. Conformance

A. Index
A.1 Terms defined by this specification
A.2 Terms defined by reference

B. **References**
B.1 Normative references

1. Introduction

Computational services often implement access control within their components by separating Policy Decision Points (PDPs) from Policy Enforcement Points (PEPs). PDPs and PEPs are defined in [*XACML*] and NIST's ABAC SP 800-162. Communication between PDPs and PEPs follows similar patterns across different software and services that require or provide authorization information. The Authorization API described in this document enables different providers to offer PDP and PEP capabilities without having to bind themselves to one particular implementation of a PDP or PEP.

2. Model

The Authorization API is a transport-agnostic API published by the PDP, to which the PEP acts as a client. Possible bindings of this specification, such as HTTPS or gRPC, are described in 12. Transport.

Authorization for the Authorization API itself is out of scope for this document, since authorization for APIs is well-documented elsewhere. For example, the Authorization API's HTTPS binding *MAY* support authorization using an `Authorization` header, using a `basic` or `bearer` token. Support for OAuth 2.0 ([*RFC6749*]) is *RECOMMENDED*.

3. Features

The core feature of the Authorization API is the Access Evaluation API, which enables a PEP to find out if a specific request can be permitted to access a specific resource. The following are non-normative examples:

- Can Alice view document #123?
- Can Alice view document #123 at 16:30 on Tuesday, June 11, 2024?
- Can a manager print?

4. API Version

This document describes the API version 1.0. Any updates to this API through subsequent revisions of this document or other documents *MAY* augment this API, but *MUST NOT* modify the API described here. Augmentation *MAY* include additional API methods or additional parameters to existing API methods, additional authorization mechanisms, or additional optional headers in API requests. All API methods for version 1.0 *MUST* be immediately preceded by the relative URL path `/v1/`.

5. Information Model

The information model for requests and responses include the following entities: Subject, Action, Resource, Context, and Decision. These are all defined below.

The information model *SHOULD* be incorporated into the meta-information model of the organization according to [*MIM*]. It is *RECOMMENDED* to use [*JSON-LD11*] to enable automatic integration into existing semantic models

5.1 Subject

A Subject is the user or machine principal about whom the Authorization API is being invoked. The Subject may be requesting access at the time the Authorization API is invoked.

A Subject is a JSON ([*RFC8259*]) object that contains two *REQUIRED* keys, `type` and `id`, which have a value typed `string`, and an *OPTIONAL* key, `properties`, with a value of a JSON object. The Subject *MAY* contain [*JSON-LD11*] keys starting with the `@`-symbol.

`type`: : *REQUIRED*. A `string` value that specifies the type of the Subject.

`id`: : *REQUIRED*. A `string` value containing the unique identifier of the Subject, scoped to the `type`.

`properties`: : *OPTIONAL*. A JSON object containing any number of key-value pairs, which can be used to express additional properties of a Subject.

The following is a non-normative example of a Subject:

EXAMPLE 1: Example Subject

```
{
  "type": "user",
  "id": "alice@acmecorp.com"
}
```

§ 5.1.1 Subject Properties

Many authorization systems are stateless, and expect the client (PEP) to pass in any properties or attributes that are expected to be used in the evaluation of the authorization policy. To satisfy this requirement, Subjects *MAY* include zero or more additional attributes as key-value pairs, under the `properties` object.

An attribute can be single-valued or multi-valued. It can be a primitive type (string, boolean, number) or a complex type such as a JSON object or JSON array.

The following is a non-normative example of a Subject which adds a string-valued `department` property:

EXAMPLE 2: Example Subject with Additional Property

```
{
  "type": "user",
  "id": "alice@acmecorp.com",
  "properties": {
    "department": "Sales"
  }
}
```

To increase interoperability, a few common properties are specified below:

§ *5.1.1.1 IP Address*

The IP Address of the Subject, identified by an `ip_address` field, whose value is a textual representation of an IP Address, as defined in `Textual Conventions for Internet`

`Network Addresses` [*RFC4001*].

The following is a non-normative example of a subject which adds the `ip_address` property:

EXAMPLE 3: Example Subject with IP Address

```
{
  "type": "user",
  "id": "alice@acmecorp.com",
  "properties": {
    "department": "Sales",
    "ip_address": "172.217.22.14"
  }
}
```

§ *5.1.1.2 Device ID*

The Device Identifier of the Subject, identified by a `device_id` field, whose value is a string representation of the device identifier.

The following is a non-normative example of a subject which adds the `device_id` property:

EXAMPLE 4: Example Subject with Device ID

```
{
  "type": "user",
  "id": "alice@acmecorp.com",
  "properties": {
    "department": "Sales",
    "ip_address": "172.217.22.14",
    "device_id": "8:65:ee:17:7e:0b"
  }
}
```

§ 5.2 Resource

A Resource is the target of an access request. It is a JSON ([*RFC8259*]) object that is constructed similar to a Subject entity. It has the follow keys:

`type`: : *REQUIRED*. A `string` value that specifies the type of the Resource.

`id`: : *REQUIRED*. A `string` value containing the unique identifier of the Resource, scoped to the `type`.

`properties`: : *OPTIONAL*. A JSON object containing any number of key-value pairs, which can be used to express additional properties of a Resource.

The Resource *MAY* contain [*JSON-LD11*] keys starting with the `@`-symbol.

§ 5.2.1 Examples (non-normative)

The following is a non-normative example of a Resource with a `type` and a simple `id`:

EXAMPLE 5: Example Resource

```
{
  "type": "book",
  "id": "123"
}
```

The following is a non-normative example of a Resource containing a `library_record` property, that is itself a JSON object:

EXAMPLE 6: Example Resource with Additional Property

```
{
  "type": "book",
  "id": "123",
  "properties": {
    "library_record":{
      "title": "AuthZEN in Action",
      "isbn": "978-0593383322"
    }
  }
}
```

§ 5.3 Action

An Action is the type of access that the requester intends to perform.

Action is a JSON ([*RFC8259*]) object that contains a *REQUIRED* `name` key with a `string` value, and an *OPTIONAL* `properties` key with a JSON object value. The Action *MAY* contain [*JSON-LD11*] keys starting with the `@`-symbol.

`name`: : *REQUIRED*. The name of the Action.

`properties`: : *OPTIONAL*. A JSON object containing any number of key-value pairs, which can be used to express additional properties of an Action.

The following is a non-normative example of an action:

EXAMPLE 7: Example Action

```
{
  "name": "can_read"
}
```

§ 5.3.1 Action Properties

Actions *MAY* include zero or more additional attributes as key-value pairs, under the properties object.

To increase interoperability, a few common properties are specified below:

§ *5.3.1.1 Processing Activity and Algorithm identifiers*

Under Dutch and EU legislation every action *SHOULD* include a relation to the processing activity - or algorithm under which it is performed. These identifiers *SHOULD* be included in the action properties and *MUST* use the property names as defined in [*Logboek dataverwerkingen*] standard and its extensions.

5.4 Context

The Context object is a set of attributes that represent environmental or contextual data about the request such as time of day. It is a JSON ([*RFC8259*]) object. The Context *MAY* contain [*JSON-LD11*] keys starting with the @-symbol.

As described in 13.3 Trust it is recommended to consider values in the information model as trusted and valid. For purposes of defense-in-depth and traceability proof of values *MAY* be provided. If provided, these *SHOULD* be included in the Context object.

The following is a non-normative example of a Context:

EXAMPLE 8: Example Context

```
{
  "time": "1985-10-26T01:22-07:00"
}
```

5.4.1 Context Properties

Context *MAY* include zero or more additional attributes as key-value pairs.

To increase interoperability, a few common properties are specified below:

5.4.1.1 Time

The logical time at which the action was considered to be initiated, identified by the `time` field, whose value is a textual representation of the time as defined in [*RFC3339*].

This timestamp *SHOULD* be used when a PDP evaluates the access request uses information from data sources that support temporal queries. See for example the [*REST API Design Rules*] and its temporal extension.

5.4.1.2 W3C Trace Context

To enable tracing of requests request identifiers *MUST* be included in the evaluation request. Request identifiers *SHOULD* be included in the Context object. They *SHOULD* be in the form of `tracestate`, `traceparent` as defined by [*Trace-Context*].

6. Access Evaluation API

The Access Evaluation API defines the message exchange pattern between a client (PEP) and an authorization service (PDP) for executing a single access evaluation.

6.1 The Access Evaluation API Request

The Access Evaluation request is a 4-tuple constructed of the four previously defined entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. The context (or environment) of type Context.

The Access Evaluation Request *MAY* contain [*JSON-LD11*] keys starting with the `@`-symbol.

6.1.1 Example (non-normative)

EXAMPLE 9: Example Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "action": {
    "name": "can_read",
    "properties": {
      "method": "GET"
    }
  },
  "context": {
    "time": "1985-10-26T01:22-07:00"
  }
}
```

6.2 The Access Evaluation API Response

The simplest form of a response is simply a boolean representing a Decision, indicated by a `"decision"` field.

`decision`: : *REQUIRED*. A boolean value that specifies whether the Decision is to allow or deny the operation.

In this specification, assuming the evaluation was successful, there are only 2 possible responses:

- `true`: The access request is permitted to go forward.
- `false`: The access request is denied and *MUST NOT* be permitted to go forward.

The response object *MUST* contain this boolean-valued Decision key.

§ 6.2.1 Access Evaluation Decision

The following is a non-normative example of a simple Decision:

> EXAMPLE 10: Example Decision
>
> ```
> {
>   "decision": true
> }
> ```

§ 6.2.2 Additional Context in a Response

In addition to a `"decision"`, a response may contain a `"context"` field which can be any JSON object. This context can convey additional information that can be used by the PEP as part of the decision evaluation process. Examples include:

- XACML's notion of "advice" and "obligations"
- Hints for rendering UI state
- Instructions for step-up authentication

§ 6.2.3 Example Context

An implementation *MAY* follow a structured approach to `"context"`, in which it presents the reasons that an authorization request failed.

- A list of identifiers representing the items (policies, graph nodes, tuples) that were used in the decision-making process.
- A list of reasons as to why access is permitted or denied.

§ *6.2.3.1 Reasons*

Reasons *MAY* be provided by the PDP.

§ 6.2.3.1.1 Reason Field

A Reason Field is a JSON object that has keys and values of type `string`. The following are non-normative examples of Reason Field objects:

EXAMPLE 11: Example Reason

```
{
  "en": "location restriction violation"
}
```

§ 6.2.3.1.2 Reason Object

A Reason Object specifies a particular reason. It is a JSON object that has the following fields:

`id`: : *REQUIRED*. A string value that specifies the reason within the scope of a particular response.

`reason_admin`: : *OPTIONAL*. The reason, which *MUST NOT* be shared with the user, but useful for administrative purposes that indicates why the access was denied. The value of this field is a Reason Field object.

`reason_user`: : *OPTIONAL*. The reason, which *MAY* be shared with the user that indicates why the access was denied. The value of this field is a Reason Field object.

The following is a non-normative example of a Reason Object:

EXAMPLE 12: Example of a Reason Object

```
{
  "id": "0",
  "reason_admin": {
    "en": "Request failed policy C076E82F"
  },
  "reason_user": {
    "en-403": "Insufficient privileges. Contact your administrator",
    "es-403": "Privilegios insuficientes. Póngase en contacto con su a
  }
}
```

§ 6.2.4 Sample Response with additional context (non-normative)

EXAMPLE 13: Example Response with Context

```
{
  "decision": false,
  "context": {
    "id": "0",
    "reason_admin": {
      "en": "Request failed policy C076E82F"
    },
    "reason_user": {
      "en-403": "Insufficient privileges. Contact your administrator",
      "es-403": "Privilegios insuficientes. Póngase en contacto con su
    }
  }
}
```

§ 7. Access Evaluations API

The Access Evaluations API defines the message exchange pattern between a client (PEP) and an authorization service (PDP) for evaluating multiple access evaluations within the scope of a single message exchange (also known as "boxcarring" requests).

§ 7.1 The Access Evaluations API Request

The Access Evaluation API Request builds on the information model presented in 5. Information Model and the 4-tuple defined in 6.1 The Access Evaluation API Request.

To send multiple access evaluation requests in a single message, the caller *MAY* add an `evaluations` key to the request. The `evaluations` key is an array which contains a list of JSON objects, each typed as the 4-tuple as defined defined in 6.1 The Access Evaluation API Request, and specifying a discrete request.

If an `evaluations` array is NOT present, the Access Evaluations Request behaves in a backwards-compatible manner with the (single) Access Evaluation API Request.

If an `evaluations` array IS present and contains one or more objects, these form distinct requests that the PDP will evaluate. These requests are independent from each other, and may be executed sequentially or in parallel, left to the discretion of each implementation.

If the `evaluations` array IS present and contains one or more objects, the top-level `subject`, `action`, `resource`, and `context` keys (4-tuple) in the request object *MAY* be omitted. However, if one or more of these values is present, they provide default values for their respective fields in the evaluation requests. This behavior is described in 7.1.1 Default values.

The following is a non-normative example for specifying three requests, with no default values:

```
{
  "evaluations": [
    {
      "subject": {
        "type": "user",
        "id": "alice@acmecorp.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    },
    {
      "subject": {
        "type": "user",
        "id": "alice@acmecorp.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    },
    {
```

```
      "subject": {
        "type": "user",
        "id": "alice@acmecorp.com"
      },
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      },
      "context": {
        "time": "2024-05-31T15:22-07:00"
      }
    }
  ]
}
```

§ 7.1.1 Default values

While the example above provides the most flexibility in specifying distinct values in each request for every evaluation, it is common for boxcarred requests to share one or more values of the 4-tuple. For example, evaluations *MAY* all refer to a single subject, and/or have the same contextual (environmental) attributes.

Default values offer a more compact syntax that avoids over-duplication of request data.

If any of the top-level `subject`, `action`, `resource`, and `context` keys are provided, the value of the top-level key is treated as the default value for the 4-tuples specified in the `evaluations` array. If a top-level key is specified in the 4-tuples present in the `evaluations` array then the value of that will take precedence over these default values.

The following is a non-normative example for specifying three requests that refer to a single subject and context:

```
{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
```

```
  "evaluations": [
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_read"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

The following is a non-normative example for specifying three requests that refer to a single `subject` and `context`, with a default value for `action`, that is overridden by the third request:

```
{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
  "action": {
    "name": "can_read"
  },
  "evaluations": [
```

```
    {
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_edit"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

§ 7.1.2 Evaluations options

The `evaluations` request payload includes an *OPTIONAL* `options` key, with a JSON value containing a set of key-value pairs.

This provides a general-purpose mechanism for providing caller-supplied metadata on how the request is to be executed.

One such option controls *evaluation semantics*, and is described in 7.1.2.1 Evaluations semantics.

A non-normative example of the `options` field is shown below, following an `evaluations` array provided for the sake of completeness:

```
{
  "evaluations": [{
    "resource": {
      "type": "doc",
      "id": "1"
    },
    "subject": {
```

```
      "type": "doc",
      "id": "2"
    }
  }],
  "options": {
    "evaluation_semantics": "execute_all",
    "another_option": "value"
  }
}
```

§ *7.1.2.1 Evaluations semantics*

By default, every request in the `evaluations` array is executed and a response returned in the same array order. This is the most common use-case for boxcarring multiple evaluation requests in a single payload.

With that said, three evaluation semantics are supported:

1. *Execute all of the requests (potentially in parallel), return all of the results.* Any failure can be denoted by `decision: false` and *MAY* provide a reason code in the context.
2. *Deny on first denial (or failure).* This semantic could be desired if a PEP wants to issue a few requests in a particular order, with any denial (error, or `decision: false`) "short-circuiting" the evaluations call and returning on the first denial. This essentially works like the && operator in programming languages.
3. *Permit on first permit.* This is the converse "short-circuiting" semantic, working like the `||` operator in programming languages.

To select the desired evaluations semantic, a caller can pass in `options.evaluations_semantic` with exactly one of the following values:

- `execute_all`
- `deny_on_first_deny`
- `permit_on_first_permit`

`execute_all` is the default semantic, so an `evaluations` request without the `options.evaluations_semantic` flag will execute using this semantic.

§ 7.1.2.1.1 Example: Evaluate `read` action for three documents using all three semantics

Execute all requests:

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "execute_all"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "3"
      }
    }
  ]
}
```

Response:

```
{
  "evaluations": [
    {
```

```
      "decision": true
    },
    {
      "decision": false
    },
    {
      "decision": true
    }
  ]
}
```

Deny on first deny:

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "deny_on_first_deny"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "3"
      }
    }
  ]
}
```

Response:

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "id": "200",
        "reason": "deny_on_first_deny"
      }
    }
  ]
}
```

Permit on first permit:

```
{
  "subject": {
    "type": "user",
    "id": "alice@example.com"
  },
  "action": {
    "name": "read"
  },
  "options": {
    "evaluations_semantic": "permit_on_first_permit"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "1"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "2"
      }
    },
    {
      "resource": {
        "type": "document",
```

```
        "id": "3"
      }
    }
  ]
}
```

Response:

```
{
  "evaluations": [
    {
      "decision": true
    }
  ]
}
```

§ 7.2 Access Evaluations API Response

Like the request format, the Access Evaluations Response format for an Access Evaluations Request adds an `evaluations` array that lists the decisions in the same order they were provided in the `evaluations` array in the request. Each value of the evaluations array is typed as an Access Evaluation Response (6.2 The Access Evaluation API Response).

In case the `evaluations` array is present, it is *RECOMMENDED* that the `decision` key of the response will be omitted. If present, it can be ignored by the caller.

The following is a non-normative example of a Access Evaluations Response to an Access Evaluations Request containing three evaluation objects:

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "reason": "resource not found"
      }
    },
    {
      "decision": false,
      "context": {
```

```
        "reason": "Subject is a viewer of the resource"
      }
    }
  ]
}
```

§ 7.2.1 Errors

There are two types of errors, and they are handled differently:

1. Transport-level errors, or errors that pertain to the entire payload.
2. Errors in individual evaluations.

The first type of error is handled at the transport level. For example, for the HTTP binding, the 4XX and 5XX codes indicate a general error that pertains to the entire payload, as described in 12. Transport.

The second type of error is handled at the payload level. Decisions default to *closed* (i.e. `false`), but the `context` field can include errors that are specific to that request.

The following is a non-normative example of a response to an Access Evaluations Request containing three evaluation objects, two of them demonstrating how errors can be returned for two of the evaluation requests:

```
{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "error": {
          "status": 404,
          "message": "Resource not found"
        }
      }
    },
    {
      "decision": false,
      "context": {
        "reason": "Subject is a viewer of the resource"
      }
```

```
    }
  ]
}
```

8. Subject Search API

The Subject Search API defines the message exchange pattern between a client (PEP) and an authorization service (PDP) for returning all of the subjects that match the search criteria.

The Subject Search API is based on the Access Evaluation information model, but omits the Subject ID.

8.1 Subject Search Semantics

While the evaluation of a search is implementation-specific, it is expected that any returned results that are then fed into an `evaluation` call *MUST* result in a `decision: true` response.

In addition, it is *RECOMMENDED* that a subject search is performed transitively, traversing intermediate attributes and/or relationships. For example, if the members of group G are designated as viewers on a document D, then a search for all users that are viewers of document D will include all the members of group G.

8.2 The Subject Search API Request

The Subject Search request is a 4-tuple constructed of three previously defined entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject. NOTE that the Subject type is *REQUIRED* but the Subject ID can be omitted, and if present, is IGNORED.

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page token for paged requests.

8.2.1 Example (non-normative)

The following payload defines a request for the subjects of type `user` that can perform the `can_read` action on the resource of type `account` and ID `123`.

EXAMPLE 14: Example Request

```
{
  "subject": {
    "type": "user"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  }
}
```

8.3 The Subject Search API Response

The response is a paged array of Subjects.

```
{
  "results": [
    {
      "type": "user",
      "id": "alice@acmecorp.com"
    },
    {
      "type": "user",
      "id": "bob@acmecorp.com"
    }
  ],
  "page": {
```

```
    "next_token": ""
  }
}
```

§ 8.3.1 Paged requests

A response that needs to be split across page boundaries returns a non-empty `page.next_token`.

§ *8.3.1.1 Example*

```
{
  "results": [
    {
      "type": "user",
      "id": "alice@acmecorp.com"
    },
    {
      "type": "user",
      "id": "bob@acmecorp.com"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

To retrieve the next page, provide `page.next_token` in the next request:

```
{
  "subject": {
    "type": "user"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
```

```
    "time": "2024-10-26T01:22-07:00"
  },
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

Note: page size is implementation-dependent.

§ 9. Resource Search API

The Resource Search API defines the message exchange pattern between a client (PEP) and an authorization service (PDP) for returning all of the resources that match the search criteria.

The Resource Search API is based on the Access Evaluation information model, but omits the Resource ID.

§ 9.1 Resource Search Semantics

While the evaluation of a search is implementation-specific, it is expected that any returned results that are then fed into an `evaluation` call *MUST* result in a `decision: true` response.

In addition, it is *RECOMMENDED* that a resource search is performed transitively, traversing intermediate attributes and/or relationships. For example, if user U is a viewer of folder F that contains a set of documents, then a search for all documents that user U can view will include all of the documents in folder F.

§ 9.2 The Resource Search API Request

The Resource Search request is a 4-tuple constructed of three previously defined entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject.

`action`: : *REQUIRED*. The action (or verb) of type Action.

`resource`: : *REQUIRED*. The resource of type Resource. NOTE that the Resource type is *REQUIRED* but the Resource ID is omitted, and if present, is IGNORED.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page token for paged requests.

9.2.1 Example (non-normative)

The following payload defines a request for the resources of type `account` on which the subject of type `user` and ID `alice@acmecorp.com` can perform the `can_read` action.

EXAMPLE 15: Example Request

```
{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  }
}
```

9.3 The Resource Search API Response

The response is a paged array of Resources.

```
{
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ],
  "page": {
```

```
    "next_token": ""
  }
}
```

§ 9.3.1 Paged requests

A response that needs to be split across page boundaries returns a non-empty `page.next_token`.

§ *9.3.1.1 Example*

```
{
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

To retrieve the next page, provide `page.next_token` in the next request:

```
{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  },
  "page": {
```

```
    "next_token": "alsehrq3495u8"
  }
}
```

Note: page size is implementation-dependent.

10. Action Search API

The Action Search API defines the message exchange pattern between a client (PEP) and an authorization service (PDP) for returning all of the actions that match the search criteria.

The Action Search API is based on the Access Evaluation information model, but omits the Action Name.

10.1 Action Search Semantics

While the evaluation of a search is implementation-specific, it is expected that any returned results that are then fed into an `evaluation` call *MUST* result in a `decision: true` response.

10.2 The Action Search API Request

The Action Search request is a 3-tuple constructed of three previously defined entities:

`subject`: : *REQUIRED*. The subject (or principal) of type Subject.

`resource`: : *REQUIRED*. The resource of type Resource.

`context`: : *OPTIONAL*. Contextual data about the request.

`page`: : *OPTIONAL*. A page token for paged requests.

10.2.1 Example (non-normative)

The following payload defines a request for the actions that the subject of type user and ID may perform on the resource of type account and ID 123 at 01:22 AM.

EXAMPLE 16: Example Request

```
{
  "subject": {
    "type": "user",
    "id": "123"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  }
}
```

§ 10.3 The Action Search API Response

The response is a paged array of Actions.

EXAMPLE 17: Example Response

```
{
  "results": [
    {
      "name": "can_read"
    },
    {
      "name": "can_write"
    }
  ],
  "page": {
    "next_token": ""
  }
}
```

§ 10.3.1 Paged requests

A response that needs to be split across page boundaries returns a non-empty `page.next_token`.

§ *10.3.1.1 Example*

EXAMPLE 18: Example Paged Response

```
{
  "results": [
    {
      "name": "can_read"
    },
    {
      "name": "can_write"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

To retrieve the next page, provide `page.next_token` in the next request:

EXAMPLE 19: Example Paged Request

```
{
  "subject": {
    "type": "user",
    "id": "123"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  },
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

Note: page size is implementation-dependent.

11. Policy Decision Point Metadata

Policy Decision Points can have metadata describing their configuration.

11.1 Data structure

The following Policy Decision Point metadata parameters are used by this specification and are registered in the IANA "AuthZEN PDP Metadata" registry established in 14.1 AuthZEN Policy Decision Point Metadata Registry.

11.1.1 Endpoint Parameters

`policy_decision_point`: : *REQUIRED*. The policy decision point's policy decision point identifier, which is a URL that uses the "https" scheme and has no query or fragment components. Policy Decision Point metadata is published at a location that is ".well-known" according to

[*RFC5785*] derived from this policy decision point identifier, as described in . The policy decision point identifier is used to prevent policy decision point mix-up attacks.

`access_evaluation_endpoint`: : *REQUIRED*. URL of Policy Decision Point Access Evaluation API endpoint

`access_evaluations_endpoint`: : *OPTIONAL*. URL of Policy Decision Point Access Evaluations API endpoint

`search_subject_endpoint`: : *OPTIONAL*. URL of Policy Decision Point Search API endpoint for subject element

`search_action_endpoint`: : *OPTIONAL*. URL of Policy Decision Point Search API endpoint for action element

`search_resource_endpoint`: : *OPTIONAL*. URL of Policy Decision Point Search API endpoint for resource element

Note that the non presence of any of those parameter is sufficient for the policy enforcement point to determine that the policy decision point is not capable and therefore will not return a result for the associated API.

§ 11.1.2 Signature Parameter

In addition to JSON elements, metadata values *MAY* also be provided as a `signed_metadata` value, which is a JSON Web Token [*RFC7519*] that asserts metadata values about the policy decision point as a bundle. A set of metadata parameters that can be used in signed metadata as claims are defined in 11.1.1 Endpoint Parameters. The signed metadata *MUST* be digitally signed or MACed using JSON Web Signature [*RFC7515*] and *MUST* contain an `iss` (issuer) claim denoting the party attesting to the claims in the signed metadata.

Consumers of the metadata *MAY* ignore the signed metadata if they do not support this feature. If the consumer of the metadata supports signed metadata, metadata values conveyed in the signed metadata *MUST* take precedence over the corresponding values conveyed using plain JSON elements. Signed metadata is included in the policy decision point metadata JSON object using this *OPTIONAL* metadata parameter:

`signed_metadata`: : A JWT containing metadata parameters about the protected resource as claims. This is a string value consisting of the entire signed JWT. A `signed_metadata` parameter *SHOULD NOT* appear as a claim in the JWT; it is *RECOMMENDED* to reject any metadata in which this occurs.

§ 11.2 Obtaining Policy Decision Point Metadata

Policy Decision Point supporting metadata *MUST* make a JSON document containing metadata as specified in 11.1.1 Endpoint Parameters available at a URL formed by inserting a well-known URI string between the host component and the path and/or query components, if any. The well-known URI string used is `/.well-known/authzen-configuration`.

The syntax and semantics of .well-known are defined in [*RFC8615*]. The well-known URI path suffix used is registered in the IANA "Well-Known URIs" registry [*IANA.well-known-uris*].

§ 11.2.1 Policy Decision Point Metadata Request

A policy decision point metadata document *MUST* be queried using an HTTP GET request at the previously specified URL. The consumer of the metadata would make the following request when the resource identifier is https://pdp.mycompany.com:

```
GET /.well-known/authzen-configuration HTTP/1.1
Host: pdp.mycompany.com
```

§ 11.2.2 Policy Decision Point Metadata Response

The response is a set of metadata parameters about the protected resource's configuration. A successful response *MUST* use the `200 OK HTTP` status code and return a JSON object using the `application/json` content type that contains a set of metadata parameters as its members that are a subset of the metadata parameters defined in 11.1.1 Endpoint Parameters. Additional metadata parameters *MAY* be defined and used; any metadata parameters that are not understood *MUST* be ignored.

Parameters with multiple values are represented as JSON arrays. Parameters with zero values *MUST* be omitted from the response.

An error response uses the applicable HTTP status code value.

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
  "policy_decision_point": "https://pdp.mycompany.com",
  "access_evaluation_endpoint": "https://pdp.mycompany.com/access/v1/eval
  "search_subject_endpoint": "https://pdp.mycompany.com/access/v1/search/
  "search_resource_endpoint": "https://pdp.mycompany.com/access/v1/search
}
```

11.2.3 Policy Decision Point Metadata Validation

The "`policy_decision_point`" value returned *MUST* be identical to the policy decision point identifier value into which the well-known URI string was inserted to create the URL used to retrieve the metadata. If these values are not identical, the data contained in the response *MUST NOT* be used.

The recipient *MUST* validate that any signed metadata was signed by a key belonging to the issuer and that the signature is valid. If the signature does not validate or the issuer is not trusted, the recipient *SHOULD* treat this as an error condition.

12. Transport

This specification defines an HTTPS binding which *MUST* be implemented by a compliant PDP.

Additional transport bindings (e.g. gRPC) *MAY* be defined in the future in the form of profiles, and *MAY* be implemented by a PDP.

12.1 HTTPS Binding

12.1.1 HTTPS Access Evaluation Request

The Access Evaluation Request is an HTTPS request with `content-type` of `application/json`. Its body is a JSON object that contains the Access Evaluation Request, as defined in 6.1 The Access Evaluation API Request.

The following is a non-normative example of the HTTPS binding of the Access Evaluation Request:

EXAMPLE 20: Example of an HTTPS Access Evaluation Request

```
POST /access/v1/evaluation HTTP/1.1
Host: pdp.mycompany.com
Authorization: Bearer <myoauthtoken>
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "resource": {
    "type": "todo",
    "id": "1"
  },
  "action": {
    "name": "can_read"
  },
  "context": {
    "time": "1985-10-26T01:22-07:00"
  }
}
```

§ 12.1.2 HTTPS Access Evaluation Response

The success response to an Access Evaluation Request is an Access Evaluation Response. It is an HTTPS response with a `status` code of `200`, and `content-type` of `application/json`. Its body is a JSON object that contains the Access Evaluation Response, as defined in 6.2 The Access Evaluation API Response.

Following is a non-normative example of an HTTPS Access Evaluation Response:

EXAMPLE 21: Example of an HTTP Access Evaluation Response

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "decision": true
}
```

12.1.3 HTTPS Access Evaluations Request

The Access Evaluations Request is an HTTPS request with `content-type` of `application/json`. Its body is a JSON object that contains the Access Evaluations Request, as defined in 7.1 The Access Evaluations API Request.

The following is a non-normative example of a the HTTPS binding of the Access Evaluations Request:

EXAMPLE 22: Example of an HTTPS Access Evaluations Request

```
POST /access/v1/evaluations HTTP/1.1
Host: pdp.mycompany.com
Authorization: Bearer <myoauthtoken>
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "context": {
    "time": "2024-05-31T15:22-07:00"
  },
  "action": {
    "name": "can_read"
  },
  "evaluations": [
    {
      "resource": {
        "type": "document",
        "id": "boxcarring.md"
      }
    },
    {
      "resource": {
        "type": "document",
        "id": "subject-search.md"
      }
    },
    {
      "action": {
        "name": "can_edit"
      },
      "resource": {
        "type": "document",
        "id": "resource-search.md"
      }
    }
  ]
}
```

§ 12.1.4 HTTPS Access Evaluations Response

The success response to an Access Evaluations Request is an Access Evaluations Response. It is a HTTPS response with a `status` code of `200`, and `content-type` of `application/json`. Its body is a JSON object that contains the Access Evaluations Response, as defined in 7.2 Access Evaluations API Response.

The following is a non-normative example of an HTTPS Access Evaluations Response:

EXAMPLE 23: Example of an HTTPS Access Evaluations Response

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "evaluations": [
    {
      "decision": true
    },
    {
      "decision": false,
      "context": {
        "error": {
          "status": 404,
          "message": "Resource not found"
        }
      }
    },
    {
      "decision": false,
      "context": {
        "reason": "Subject is a viewer of the resource"
      }
    }
  ]
}
```

§ 12.1.5 HTTPS Subject Search Request

The Subject Search Request is an HTTPS request with `content-type` of `application/json`. Its body is a JSON object that contains the Subject Search Request, as defined in 8.2 The Subject

Search API Request.

The following is a non-normative example of the HTTPS binding of the Subject Search Request:

EXAMPLE 24: Example of an HTTPS Subject Search Request

```
POST /access/v1/search/subject HTTP/1.1
Host: pdp.mycompany.com
Authorization: Bearer <myoauthtoken>
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account",
    "id": "123"
  }
}
```

12.1.6 HTTPS Subject Search Response

The success response to a Subject Search Request is a Subject Search Response. It is an HTTPS response with a `status` code of `200`, and `content-type` of `application/json`. Its body is a JSON object that contains the Subject Search Response, as defined in 8.3 The Subject Search API Response.

The following is a non-normative example of an HTTPS Subject Search Response:

EXAMPLE 25: Example of an HTTPS Subject Search Response

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "results": [
    {
      "type": "user",
      "id": "alice@acmecorp.com"
    },
    {
      "type": "user",
      "id": "bob@acmecorp.com"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

§ 12.1.7 HTTPS Resource Search Request

The Resource Search Request is an HTTPS request with `content-type` of `application/json`. Its body is a JSON object that contains the Resource Search Request, as defined in 9.2 The Resource Search API Request.

The following is a non-normative example of the HTTPS binding of the Resource Search Request:

EXAMPLE 26: Example of an HTTPS Resource Search Request

```
POST /access/v1/search/resource HTTP/1.1
Host: pdp.mycompany.com
Authorization: Bearer <myoauthtoken>
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "action": {
    "name": "can_read"
  },
  "resource": {
    "type": "account"
  }
}
```

12.1.8 HTTPS Resource Search Response

The success response to a Resource Search Request is a Resource Search Response. It is an HTTPS response with a `status` code of `200`, and `content-type` of `application/json`. Its body is a JSON object that contains the Resource Search Response, as defined in 9.3 The Resource Search API Response.

The following is a non-normative example of an HTTPS Resource Search Response:

EXAMPLE 27: Example of an HTTPS Resource Search Response

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "results": [
    {
      "type": "account",
      "id": "123"
    },
    {
      "type": "account",
      "id": "456"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

12.1.9 HTTPS Action Search Request

The Action Search Request is an HTTPS request with `content-type` of `application/json`. Its body is a JSON object that contains the Action Search Request, as defined in 10.2 The Action Search API Request.

The following is a non-normative example of the HTTPS binding of the Action Search Request:

EXAMPLE 28: Example of an HTTPS Action Search Request

```
POST /access/v1/search/action HTTP/1.1
Host: pdp.mycompany.com
Authorization: Bearer <myoauthtoken>
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "subject": {
    "type": "user",
    "id": "alice@acmecorp.com"
  },
  "resource": {
    "type": "account",
    "id": "123"
  },
  "context": {
    "time": "2024-10-26T01:22-07:00"
  },
}
```

12.1.10 HTTPS Action Search Response

The success response to an Action Search Request is an Action Search Response. It is an HTTPS response with a `status` code of `200`, and `content-type` of `application/json`. Its body is a JSON object that contains the Action Search Response, as defined in 10.3 The Action Search API Response.

The following is a non-normative example of an HTTPS Action Search Response:

EXAMPLE 29: Example of an HTTPS Action Search Response

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716

{
  "results": [
    {
      "name": "can_read"
    },
    {
      "name": "can_write"
    }
  ],
  "page": {
    "next_token": "alsehrq3495u8"
  }
}
```

§ 12.1.11 Error Responses

The following error responses are common to all methods of the Authorization API. The error response is indicated by an HTTPS status code (Section 15 of [*RFC9110*]) that indicates error.

The following errors are indicated by the status codes defined below:

Code	Description	HTTPS Body Content
400	Bad Request	An error message string
401	Unauthorized	An error message string
403	Forbidden	An error message string
500	Internal error	An error message string

Note: HTTPS errors are returned by the PDP to indicate an error condition relating to the request or its processing, and are unrelated to the outcome of an authorization decision, which is always returned with a `200` status code and a response payload.

To make this concrete:

- a `401` HTTPS status code indicates that the caller (policy enforcement point) did not properly authenticate to the PDP - for example, by omitting a required `Authorization` header, or using an invalid access token.
- the PDP indicates to the caller that the authorization request is denied by sending a response with a `200` HTTPS status code, along with a payload of `{ "decision": false }`.

§ 12.1.12 Request Identification

All requests to the API *MAY* have request identifiers to uniquely identify them. The API client (PEP) is responsible for generating the request identifier. If present, the request identifier *SHALL* be provided using the HTTPS Header `X-Request-ID`. The value of this header is an arbitrary string. The following non-normative example describes this header:

EXAMPLE 30: Example HTTPS request with a Request Id Header

```
POST /access/v1/evaluation HTTP/1.1
Authorization: Bearer mF_9.B5f-4.1JqM
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716
```

§ 12.1.13 Request Identification in a Response

A PDP responding to an Authorization API request that contains an `X-Request-ID` header *MUST* include a request identifier in the response. The request identifier is specified in the HTTPS Response header: `X-Request-ID`. If the PEP specified a request identifier in the request, the PDP *MUST* include the same identifier in the response to that request.

The following is a non-normative example of an HTTPS Response with this header:

EXAMPLE 31: Example HTTPS response with a Request Id Header

```
HTTP/1.1 OK
Content-type: application/json
X-Request-ID: bfe9eb29-ab87-4ca3-be83-a1d5d8305716
```

§ 13. Security Considerations

§ 13.1 Communication Integrity and Confidentiality

In the ABAC architecture, the PEP-PDP connection is the most sensitive one and needs to be secured to guarantee:

- Integrity
- Confidentiality

As a result, the connection between the PEP and the PDP *MUST* be secured using the most adequate means given the choice of transport (e.g. TLS for HTTP REST).

§ 13.2 Policy Confidentiality and Sender Authentication

Additionally, the PDP *SHOULD* authenticate requests fromthe calling PEP. There are several ways authentication can be established. These ways are out of scope of this specification. They *MAY* include:

- Mutual TLS
- OAuth-based authentication
- API key

The choice and strength of either mechanism is not in scope.

Authenticating requests from the PEP allows the PDP to avoid common attacks (such as DoS - see below) and/or reveal its internal policies. A malicious actor could craft a large number of requests to try and understand what policies the PDP is configured with. Requesting a client (PEP) be authenticated mitigates that risk.

§ 13.3 Trust

In ABAC, there is occasionally conversations around the trust between PEP and PDP: how can the PDP trust the PEP to send the right values in? This is a misplaced concern. The PDP must trust the

PEP as ultimately, the PEP is the one responsible for enforcing the decision the PDP produces.

§ 13.4 Availability & Denial of Service

The PDP *SHOULD* apply reasonable protections to avoid common attacks tied to request payload size, the number of requests, invalid JSON, nested JSON attacks, or memory consumption. Rate limiting is one such way to address such issues.

§ 13.5 Differences between Unsigned and Signed Metadata

Unsigned metadata is integrity protected by use of TLS at the site where it is hosted. This means that its security is dependent upon the Internet Public Key Infrastructure (PKI) [*RFC9525*]. Signed metadata is additionally integrity protected by the JWS signature applied by the issuer, which is not dependent upon the Internet PKI. When using unsigned metadata, the party issuing the metadata is the policy decision point itself. Whereas, when using signed metadata, the party issuing the metadata is represented by the `iss` (issuer) claim in the signed metadata. When using signed metadata, applications can make trust decisions based on the issuer that performed the signing -- information that is not available when using unsigned metadata. How these trust decisions are made is out of scope for this specification.

§ 13.6 Metadata Caching

Policy decision point metadata is retrieved using an HTTP GET request, as specified in 11.2.1 Policy Decision Point Metadata Request. Normal HTTP caching behaviors apply, meaning that the GET may retrieve a cached copy of the content, rather than the latest copy. Implementations should utilize HTTP caching directives such as Cache-Control with max-age, as defined in [*RFC7234*], to enable caching of retrieved metadata for appropriate time periods.

§ 14. IANA Considerations

The following registration procedure is used for the registry established by this specification.

Values are registered on a Specification Required [*RFC8126*] basis after a two-week review period on the openid-specs-authzen@lists.openid.net mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication of the final

version of a specification, the Designated Experts may approve registration once they are satisfied that the specification will be completed and published. However, if the specification is not completed and published in a timely manner, as determined by the Designated Experts, the Designated Experts may request that IANA withdraw the registration.

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register AuthZEN Policy Decision Point Metadata: example").

Within the review period, the Designated Experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. The IANA escalation process is followed when the Designated Experts are not responsive within 14 days.

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration makes sense.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly-informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

The reason for the use of the mailing list is to enable public review of registration requests, enabling both Designated Experts and other interested parties to provide feedback on proposed registrations. The reason to allow the Designated Experts to allocate values prior to publication as a final specification is to enable giving authors of specifications proposing registrations the benefit of review by the Designated Experts before the specification is completely done, so that if problems are identified, the authors can iterate and fix them before publication of the final specification.

§ 14.1 AuthZEN Policy Decision Point Metadata Registry

This specification establishes the IANA "AuthZEN Policy Decision Point Metadata" registry for AuthZEN policy decision point metadata names. The registry records the policy decision point metadata parameter and a reference to the specification that defines it.

§ 14.1.1 Registration Template

Metadata Name: : The name requested (e.g., "resource"). This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Metadata Description: : Brief description of the metadata (e.g., "Resource identifier URL").

Change Controller: : For IETF stream RFCs, list the "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s): : Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

§ 14.1.2 Initial Registry Contents

Metadata name: : `policy_decision_point`

Metadata description: : Base URL of the Policy Decision Point

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

Metadata name: : `access_evaluation_endpoint`

Metadata description: : URL of Policy Decision Point Access Evaluation API endpoint

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

Metadata name: : `access_evaluations_endpoint`

Metadata description: : URL of Policy Decision Point Access Evaluations API endpoint

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

Metadata name: : `search_subject_endpoint`

Metadata description: : URL of the Search Endpooint based on Subject element

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

Metadata name: : `search_resource_endpoint`

Metadata description: : URL of the Search Endpooint based on Resource element

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

Metadata name: : `signed_metadata`

Metadata description: : JWT containing metadata parameters about the protected resource as claims.

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Specification Document(s): : 11.1.1 Endpoint Parameters

§ 14.2 Well-Known URI Registry

This specification registers the well-known URI defined in Section 3 in the IANA "Well-Known URIs" registry [*IANA.well-known-uris*].

§ 14.2.1 Registry Contents

URI Suffix: : authzen-configuration

Reference: : 11.1.1 Endpoint Parameters

Status: : permanent

Change Controller: : OpenID_Foundation_AuthZEN_Working_Group : mailto:openid-specs-authzen@lists.openid.net

Related Information: : (none)

§ 15. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *OPTIONAL*, *RECOMMENDED*, *REQUIRED*, *SHALL*, *SHOULD*, and *SHOULD NOT* 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.

§ A. Index

§ A.1 Terms defined by this specification

§ A.2 Terms defined by reference

§ B. References

§ B.1 Normative references

[IANA.well-known-uris]
Well-Known URIs. 2010-01-20. URL: https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml

[JSON-LD11]
JSON-LD 1.1. Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. W3C. 16 July 2020. W3C Recommendation. URL: https://www.w3.org/TR/json-ld11/

[Logboek dataverwerkingen]

Logboek dataverwerkingen. Eelco Hotting; Vedran Bilanovic. n.t.b.. URL: https://logius-standaarden.github.io/logboek-dataverwerkingen/

[MIM]

Metamodel Informatie Modellering. 13 juni 2024. URL: https://docs.geostandaarden.nl/mim/mim/

[REST API Design Rules]

NLGov REST API Design Rules 2.0.0. Jasper Roes; Joost Farla. Maart 2024. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC3339]

Date and Time on the Internet: Timestamps. G. Klyne; C. Newman. IETF. July 2002. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc3339

[RFC4001]

Textual Conventions for Internet Network Addresses. M. Daniele; B. Haberman; S. Routhier; J. Schoenwaelder. IETF. February 2005. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc4001

[RFC5785]

Defining Well-Known Uniform Resource Identifiers (URIs). M. Nottingham; E. Hammer-Lahav. IETF. April 2010. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc5785

[RFC6749]

The OAuth 2.0 Authorization Framework. D. Hardt, Ed. IETF. October 2012. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6749

[RFC7234]

Hypertext Transfer Protocol (HTTP/1.1): Caching. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7234.html

[RFC7515]

JSON Web Signature (JWS). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7515

[RFC7519]

JSON Web Token (JWT). M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7519

[RFC8126]

Guidelines for Writing an IANA Considerations Section in RFCs. M. Cotton; B. Leiba; T. Narten. IETF. June 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8126

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[RFC8259]

The JavaScript Object Notation (JSON) Data Interchange Format. T. Bray, Ed. IETF. December 2017. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc8259

[RFC8615]

Well-Known Uniform Resource Identifiers (URIs). M. Nottingham. IETF. May 2019. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8615

[RFC9110]

HTTP Semantics. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2022. Internet Standard. URL: https://httpwg.org/specs/rfc9110.html

[RFC9525]

Service Identity in TLS. P. Saint-Andre; R. Salz. IETF. November 2023. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc9525

[Trace-Context]

Trace Context. Sergey Kanzhelev; Morgan McLean; Alois Reitbauer; Bogdan Drutu; Nik Molnar; Yuri Shkuro. W3C. 23 November 2021. W3C Recommendation. URL: https://www.w3.org/TR/trace-context-1/

[XACML]

eXtensible Access Control Markup Language (XACML) Version 1.1. Simon Godik; Tim Moses (Ed.). 2006. URL: https://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf

↑