Logius Standard - Consultation version

Authorization Decision Log 1.0.0

Logius Standard

Consultation version April 29, 2026

This version:

https://gitdocumentatie.logius.nl/publicatie/ftv/adl/1.0.0/

Latest published version:

https://logius-standaarden.github.io/authorization-decision-log/

Latest editor's draft:

https://logius-standaarden.github.io/authorization-decision-log/

Editors:

Nil Barua (Logius)

Stas Mironov (Logius)

Authors:

Maikel Hofman (VNG Realisatie)

Guus van der Meer (Vecozo)

Michiel Trimpe (Mondiality)

Participate:

GitHub Logius-standaarden/authorization-decision-log

File an issue

Commit history

Pull requests

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

Status of This Document

This is a proposed recommendation approved by TO. Comments regarding this document may be sent to api@logius.nl

Table of Contents

§ 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*, *RECOMMENDED*, and *SHOULD* 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.

Abstract

This document defines a method for logging authorization decisions to allow or deny API requests, providing accountability within and across organisations. The standard adopts [*trace-context*] for tracing across organisation boundaries and defines an OpenTelemetry-shaped log record based on the [*AuthZEN*] information model, supporting reconstruction, analysis and replay of historical decisions.

1. Introduction

This standard defines a structure for logging authorization decisions together with the context that produced them: the tracing context, the request, the response, the active policies, the information sources consulted, and the configuration of the evaluation engine. These categories correspond to the components of the *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* architecture, also known as the ***PxP*** architecture.

Additionally, it includes a non-normative outline introducing the concerns, principles and requirements for developing and maintaining such a log in concordance with legislation.

1.1 Purpose of this standard

This standard defines a uniform approach for logging authorization decisions, enabling organizations to provide effective accountability for historical decisions.

The standard provides a structured format for all contextual and environmental parameters that affect decisions. A full implementation of the standard allows historical decisions to be replayed for analysis.

It also defines participation in a distributed [*trace-context*] and binds log records to it, allowing correlation of related logs, such as [*LDV*] and [*FSC-Logging*], in complex inter-organisational data processing chains.

1.2 Terminology

The following list defines terminology used throughout this document.

Authorization

Authorization is the process of deciding whether to, fully or partially, allow or deny requests for processing (API requests) and enforcing these decisions.

This is materially different from the noun "authorization" which typically refer to an access token, OAuth scope or other permission grant. These appear as inputs for the decision in the form of attributes of an authorization decision request.

Authorization decision

A decision produced in the authorization process, determining whether a particular request is allowed or denied, fully or partially. Authorization decisions are the units recorded in an Authorization Decision Log.

Externalized Authorization Management

Externalized Authorization Management (EAM) is an architectural pattern in which authorization decisions are made in a different component than the component that enforces the decision.

Log

A list of log records, generated by a service. Logs are intended for operational monitoring, auditing, and troubleshooting.

Log record

A single unit of information within a log, representing one recorded event. A log record is immutable.

Trace

A collection of related spans representing an end-to-end transaction across components, as defined in *Trace Context*.

Span

A single operation within a trace, with a start time and end time, as defined in *Trace Context*.

Trace context

The propagation format defined by *Trace Context* for carrying trace and span identifiers across components.

Policy

A set of one or more rules that determine whether a request should be allowed or denied.

Source

A source of one or more pieces of information, such as attributes or policies, which affected an authorization decision and which come from a single source. A source must be identifiable by a single identifier, for example a version, hash or timestamp.

Reconstruct

The act of reproducing the inputs that affected an authorization decision — including the request, applicable policies, supporting information and engine configuration — using the data stored in or referenced from the log record.

Replay

The act of re-evaluating an authorization decision using its reconstructed inputs in a PDP to verify the original outcome. Replay requires that the PDP behaves identically to the original or can be manually configured to do so.

2. Architecture

The goal of the standard is to enable the reconstruction of the environment in which historical authorization decisions were made, allowing those decisions to be analysed and replayed while preventing unnecessary data duplication.

The inputs for records in the Authorization Decision Log come from the following standard EAM or PxP components as introduced in *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* and adopt the information model introduced in [*AuthZEN*].

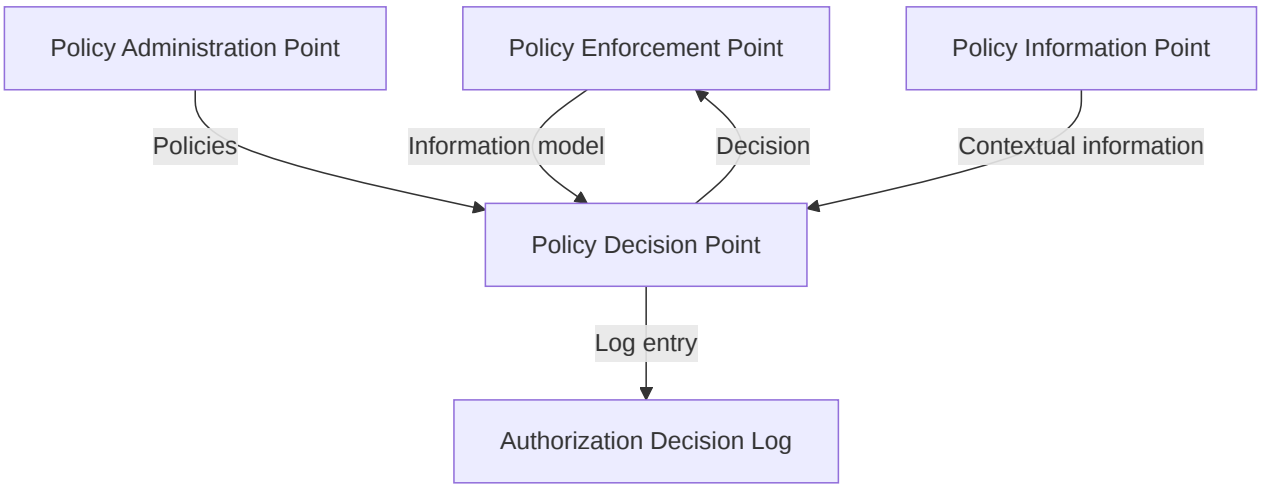

Figure 1 EAM or PxP Architecture

In a federated context, such as introduced by [*FSC-Core*], both the consumer outway and provider inway function as a PEP for incoming and outgoing requests. Both the consumer and the provider ask an internal PDP to decide on allowing the request. Both of these decisions can be logged using this standard.

When combined with tracing headers per [*trace-context*] (also adopted by [*LDV*]) and the FSC Transaction ID used in [*FSC-Logging*], this enables full traceability across complex multi-organizational processing chains.

See the sequence diagram below for an example of such a flow.

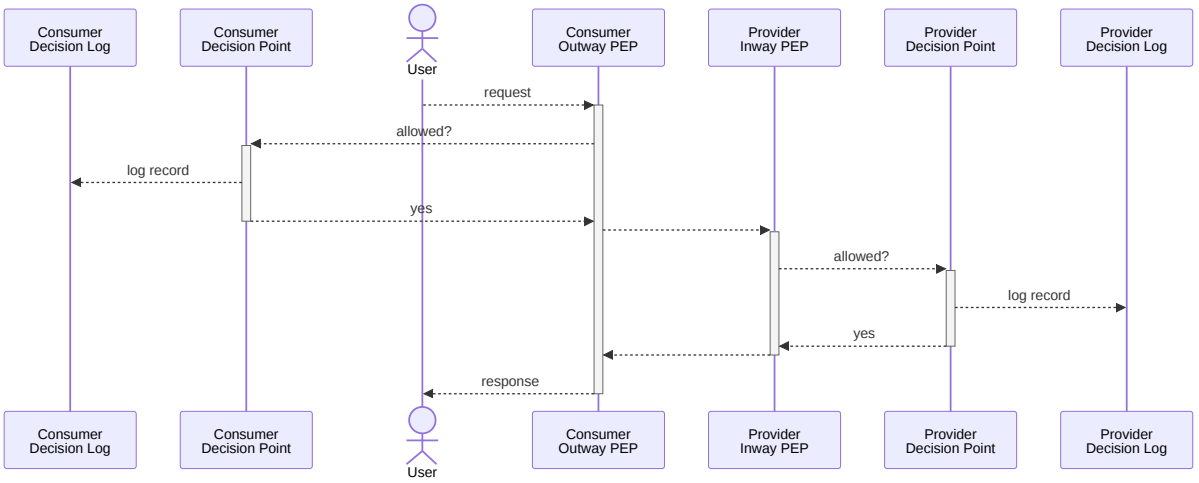

Figure 2 Decision logging in federated context

§ 2.1 Components

The standard EAM architecture has the following conceptual components. These can be deployed as standalone applications, combined in various configurations, or even implemented within a single monolithic application.

Authorization Decision Log

The Authorization Decision Log contains all information that was used in the authorization decision. Using the Authorization Decision Log it *SHOULD* be possible to accurately reconstruct environmental factors that affected historical authorization decisions.

Policy Enforcement Point

A Policy Enforcement Point (PEP) intercepts a user's request, sends it to the PDP for evaluation, and then enforces the resulting "permit" or "deny" decision. A PEP can be implemented as an API gateway, as application middleware or as a component within the application itself.

Policy Administration Point

A Policy Administration Point (PAP) is where access policies are authored, managed, and stored. It is responsible for distributing current policies to the PDP and archiving previous versions for traceability. This can be a dedicated commercial or open-source tool or a version control system like a Git repository.

Policy Information Point

A Policy Information Point (PIP) enriches access requests with additional attributes needed to make a decision. For example, it might retrieve a user's role or the sensitivity level of a data record from an external source. PIPs are often integrated directly into the PDP.

Policy Decision Point

A Policy Decision Point (PDP) evaluates incoming requests from the PEP against the relevant policies (from the PAP) and contextual data (from the PIP) to make a "permit" or "deny" decision. The PDP is often a separate application or sidecar container.

> NOTE: EAM components within a monolithic application
>
> These are architectural roles, not deployment boundaries. A single application can fulfil all four: the authorization interceptor as the PEP, the authorization handler as the PDP, the services it invokes as PIPs, and the application's own Git repository as the PAP.

§ 2.2 Scope

The specification defines an interface for persisting log records. This is the component that *MUST* be consistent across organizations to ensure interoperability.

Any authorization decision representable in the [*AuthZEN*] information model is in scope of this standard, regardless of the wire protocol by which the decision is delivered.

The management of a log, however, is left to the discretion of individual implementations. Consequently, the specification does NOT define behavior or interfaces for:

- deleting or modifying log records
- managing access to the log
- ensuring long-term accessibility
- handling archival and retention periods
- ensuring integrity and non-repudiation

- maintaining time synchronisation

See Information management for an overview of various aspects which *MAY* be required for legal and regulatory compliance.

§ 2.3 Flows

§ 2.3.1 Evaluating an authorization decision

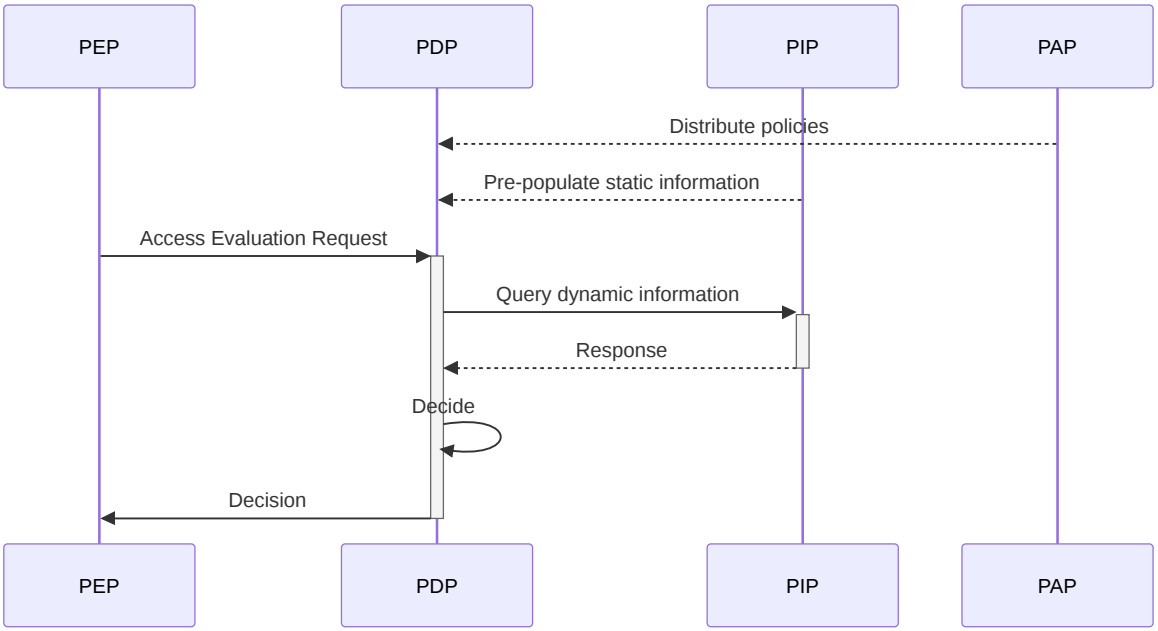

Figure 3 Evaluating an authorization decision

When a PEP needs an authorization decision it sends an evaluation request to the PDP. The PDP evaluates the request against the policies received from the PAP and pre-populated information from PIPs, *MAY* query PIPs for additional dynamic information, and returns the decision.

The PAP distributes policies and PIPs pre-populate static information into the PDP on their own schedule, independently of any individual authorization decision.

See Tracing for how this flow maps onto a trace and its spans.

§ 2.3.2 Writing a log record

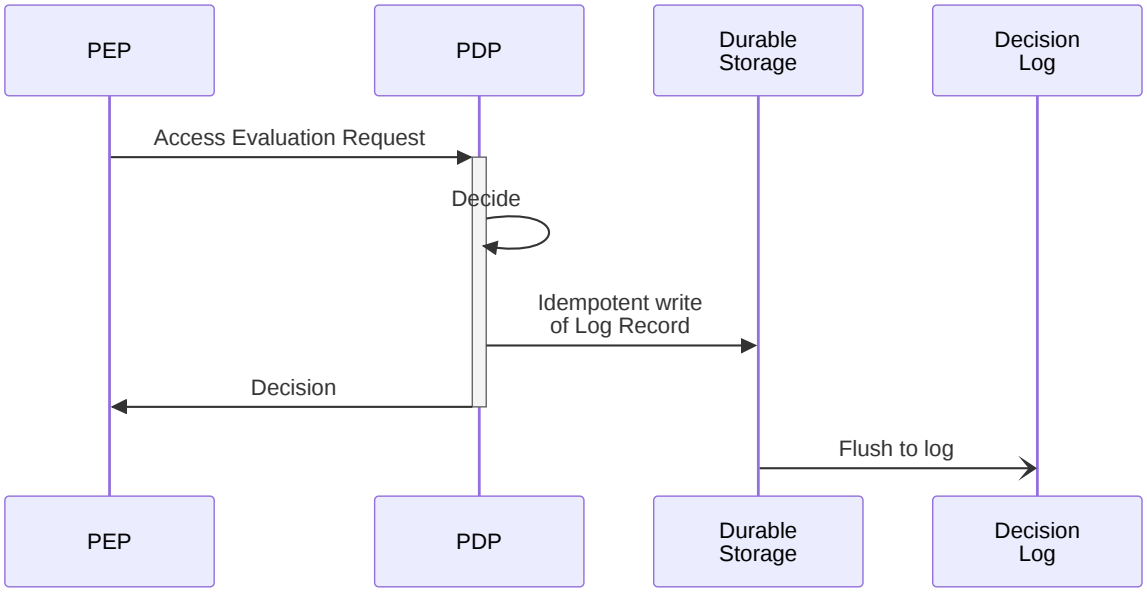

Figure 4 Writing a log record after an authorization decision

To provide accountability for historical authorization decisions it needs to be possible to reconstruct the information and environment that affected the decision. The PDP provides the information required for this to the Authorization Decision Log in the form of a log record.

The PDP *SHOULD* ensure that a log record has been persisted to durable storage before providing the PEP with the decision. Once the log record has reached durable storage, an asynchronous flush to the Authorization Decision Log completes ingestion. The component responsible for this flush is implementation-defined and *MAY* be the PDP, a sidecar (such as the OpenTelemetry Collector or Fluent Bit), or a host-level log collector.

> NOTE: Definition of durable storage
>
> "Durable storage" is intentionally not defined by this specification. Organisations *MUST* define a working definition in their logging policy, taking into account their risk appetite and operational constraints. Common interpretations range from a local `fsync`'d write to a replicated commit on a queue with quorum. See Information management for related considerations.

> NOTE: Common pattern: write-ahead log with asynchronous flush
>
> A widely deployed pattern is to write the log record to a local write-ahead log (durable storage) before the PDP returns the decision, then asynchronously flush from the write-ahead log to the Authorization Decision Log with idempotent ingestion. This pattern bounds the synchronous-path latency while preserving the durability guarantee.

3. Specifications

This section provides the specification for the protocols and interfaces to be used and the expected behavior of the components.

> NOTE: Information model independence
>
> The fields defined in this section form an information model for an authorization-decision log record. The model is independent of any particular telemetry framework; conformant records can be emitted via any transport, provided the record carries the fields specified here. The model is intentionally compatible in shape with [*OpenTelemetry*], so that organisations already operating an OpenTelemetry pipeline can carry log records on existing infrastructure without conversion.

3.1 Protocols

This standard prescribes the shape of the messages exchanged between the PDP and the log (see 3.3 Interface), but not the transport that carries them. Components *MUST* comply with the message interface; the choice of REST, gRPC, messaging, file ingestion, or any other delivery mechanism is left to the implementer.

It is *RECOMMENDED* to use the OpenTelemetry Protocol [*OTLP*] for the interaction between the Application and the log.

When using HTTP-based protocols (e.g., HTTP/1.1 [*RFC9112*] or HTTP/2 [*RFC9113*]), implementations *MUST* use the *Trace Context* HTTP header format (`traceparent`, `tracestate`) for propagation.

For non-HTTP protocols (e.g., gRPC, messaging systems), implementations *MUST* provide equivalent trace context propagation semantics, preserving `trace_id`, `span_id`, and parent relationships across boundaries. Where the protocol has no formally standardised [*trace-context*]

binding, OpenTelemetry context propagation or another widely adopted convention with equivalent semantics is sufficient.

§ 3.2 Behavior

§ 3.2.1 Encryption

The log *MUST* enforce TLS on connections, in accordance with the standard practice established within the organization.

§ 3.2.2 Tracing

All components participating in the evaluation of authorization decisions (including PEPs, PDPs, PAPs, and PIPs) *MUST* propagate trace context as defined by *Trace Context*, and *MUST* preserve trace continuity across component boundaries.

The same rule applies on every request-scoped hop in the evaluation of an authorization decision, whether the sender is a PEP calling a PDP, a PDP calling a PIP or PAP, or any other component:

- On receiving an incoming request, the component *MUST* create a span as a child of the span identified by the incoming `traceparent`, or — if no trace context is present — start a new trace with a root span.
- Before making an outgoing request, the component *MUST* start a new child span representing that outgoing operation. The outgoing `traceparent` carries the active `trace_id` and the `span_id` of this new outgoing span as the `parent-id` from the receiver's perspective, as defined by *Trace Context*.

> NOTE: The W3C Trace Context parent-id field
>
> *Trace Context* names the field `parent-id` because the field's value, from the next hop's perspective, identifies the parent of any span the next hop creates; from the sender's perspective it is the sender's own `span_id` for the outgoing span.

This ensures that every span correctly identifies its immediate parent, and that authorization decisions are consistently correlated with the broader transaction or request lifecycle in which they occur.

When a trace context is received from another organisation, the receiving component *MUST* preserve the `trace_id` unchanged in every log record and outgoing trace context originating from the resulting authorization decision flow. A new `trace_id` *MUST NOT* be allocated within an in-progress authorization decision flow.

Logging of authorization decisions is independent of the `sampled` bit in `traceparent`: that bit governs telemetry sampling only, and log records are accountability records that *MUST* be produced regardless. ADL emitters *MUST NOT* modify the `sampled` flag.

When propagating, components *MUST* follow the propagation rules in section 3.5 (Mutating the tracestate Field) of [*trace-context*] — in particular, if a component does not modify `traceparent`, it *MUST NOT* modify `tracestate`.

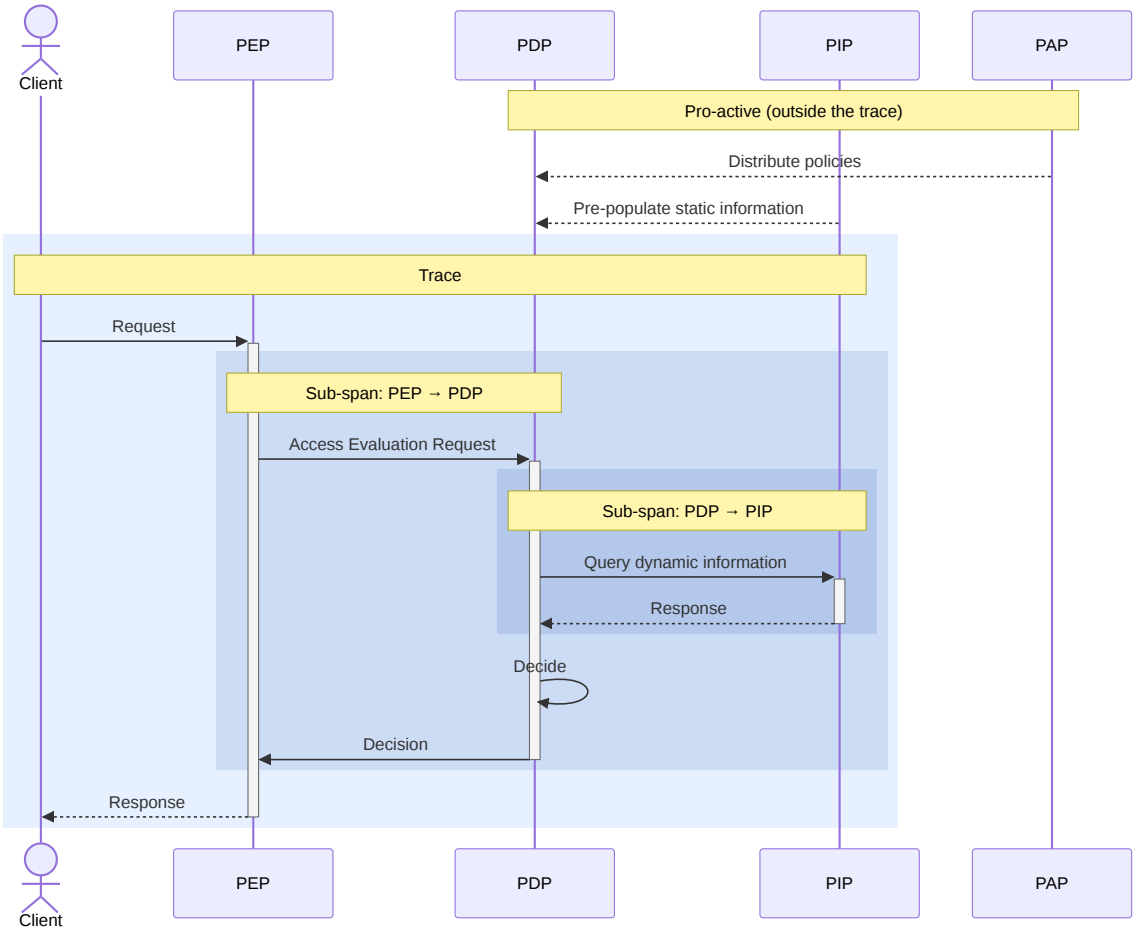

Figure 5 Trace and spans during the evaluation of an authorization decision

> NOTE: Pro-active interactions are outside the trace
>
> Pro-active interactions — for example, a PAP distributing policies to a PDP, or a PIP pre-populating data into a PDP — happen outside the scope of any individual authorization decision and are not part of its trace. They are independent operations with their own lifecycle and, where applicable, their own traces. The policies and information delivered through these pro-active interactions are the same content that subsequent log records reference via `adl.core.policies` and `adl.core.information` source references when an authorization decision is evaluated.

§ 3.2.3 Generation

Each authorization decision evaluated by a PDP — whether the PDP completed its evaluation (`status` of `Unset` or `Ok`, with the decision carried in `body.adl.core.response`) or could not complete it (`status` of `Error`) — *MUST* produce exactly one log record persisted to durable storage.

A single API call to the PDP results in exactly one log record, regardless of whether the call is to the Access Evaluation API (one decision) or the Access Evaluations API (multiple decisions in a single call). Per-sub-decision outcomes are carried in `body.adl.core.response`.

> NOTE: Use of Access Evaluations
>
> The [*AuthZEN*] Access Evaluations API is well-suited to prospective enumeration (e.g. determining which UI elements to enable for a user) and to compound checks where multiple permissions together correspond to a single user action. In both cases the call has a 1:1 correspondence to that user action and to the resulting log record, and cross-request correlation is preserved.
>
> This standard RECOMMENDS against using Access Evaluations as a batching mechanism for otherwise independent decisions, because the single resulting log record cannot uniquely correlate to per-decision records in downstream logs (such as [*LDV*]). Implementations that nevertheless use it that way are responsible for ensuring that those downstream records can still be joined to the corresponding sub-decision.

> NOTE: Cached decisions
>
> Caching authorization decisions is discouraged. A cached decision references the policies and information that were current at cache population; replaying it against the current state may yield a different outcome, and per-cache-use logging — required to preserve accountability — erodes much of the performance benefit caching is intended to provide.
>
> If a cached decision is nevertheless applied to a new request, that application *SHOULD* produce its own log record, carrying the `trace_id` and `parent_span_id` of the new request with a freshly allocated `span_id`. The `trace_id` of the original cache-population request *MUST NOT* be reused. The `attributes.adl.core.policies` and `attributes.adl.core.information` source references in the new log record point to the same sources used by the original decision.

§ 3.2.4 Ingestion

Ingestion of log records into the Authorization Decision Log *MUST* be idempotent: re-submitting the same log record — for example as a result of a queue redelivery in the write-ahead log pattern described in 2. Architecture — *MUST NOT* produce a duplicate persisted record. The idempotency key is implementation-defined; common choices are the pair `(trace_id, span_id)` or a content-hash of the log record.

§ 3.3 Interface

A log record *MUST* contain the following mandatory fields and *MAY* contain the optional fields:

Field	Type	Mandatory?
`trace_id`	16 byte	mandatory
`span_id`	8 byte	mandatory
`parent_span_id`	8 byte	conditional
`event_name`	string	mandatory
`timestamp`	uint64	mandatory
`status`	enum	mandatory
`attributes`	object	optional
`resource`	object	optional

Field	Type	Mandatory?
body	object	optional

§ 3.3.1 `trace_id`

Unique identifier of the trace that this authorization decision belongs to.

When serialised in textual form (e.g., JSON), `trace_id` *MUST* be encoded as 32 lowercase hexadecimal characters, matching the form used in the `traceparent` header defined by [*trace-context*].

`trace_id` values *MUST* be generated using a cryptographically secure random number generator and *MUST NOT* be derived from user-identifiable input, in line with section 6 (Privacy Considerations) of [*trace-context*].

§ 3.3.2 `span_id`

Unique identifier of the span representing this authorization decision.

When serialised in textual form (e.g., JSON), `span_id` *MUST* be encoded as 16 lowercase hexadecimal characters, matching the form used in the `traceparent` header defined by [*trace-context*].

`span_id` values *MUST* be generated using a cryptographically secure random number generator and *MUST NOT* be derived from user-identifiable input, in line with section 6 (Privacy Considerations) of [*trace-context*].

§ 3.3.3 `parent_span_id`

Unique identifier of the span of which this log record's span is a child.

`parent_span_id` *MUST* be set to the `parent-id` of the incoming `traceparent` when one is present. It *MAY* be omitted only when the authorization decision flow has no upstream caller — that is, when the log record's span is the root of the trace.

§ 3.3.4 `event_name`

The `event_name` field identifies the type of authorization decision the log record represents. The five conformant values, each corresponding to one of the [*AuthZEN*] APIs and its information model, are:

AuthZEN API	`event_name`
Access Evaluation API	`adl.access_evaluation`
Access Evaluations API	`adl.access_evaluations`
Subject Search API	`adl.search_subject`
Action Search API	`adl.search_action`
Resource Search API	`adl.search_resource`

The `adl.` prefix avoids collisions with other events in the same backend and allows log records to be filtered or indexed by themselves.

> NOTE: Decisions not delivered via the AuthZEN API
>
> Implementations that do not deliver decisions through the [*AuthZEN*] HTTP API select the `event_name` whose [*AuthZEN*] information model is most closely aligned with the shape of the decision. This applies to in-application authorization checks, XACML PDPs, and OPA-style direct calls. Decisions whose shape does not correspond to any of the five `event_name` values are out of scope of this standard.

§ 3.3.5 `timestamp`

The `timestamp` field represents the exact point in time when the authorization decision was made. The timestamp *MUST* be encoded as the number of milliseconds since the Unix epoch (1970-01-01T00:00:00Z), as an unsigned 64-bit integer.

§ 3.3.6 `status`

The `status` field describes the outcome of the PDP's evaluation attempt. It *MUST* be one of the following values:

- `Unset` — the default value. The PDP completed its evaluation without internal error. The response itself — in [*AuthZEN*] format, e.g. a `decision` boolean for the Evaluation APIs or a `results` array for the Search APIs — is carried in `body.adl.core.response`.
- `Ok` — optional. Used when an implementation explicitly marks a successfully completed evaluation. Functionally equivalent to `Unset`.
- `Error` — the PDP could not produce a decision (for example, engine fault, missing attributes, downstream timeout).

A successful evaluation that resulted in denial (for example, `decision: false` from the Access Evaluation API, or an empty `results` array from a Search API) *MUST NOT* be reported as `Error`. The `Error` value is reserved for failures of the PDP itself to evaluate the request — denial is a valid outcome of evaluation and is therefore `Unset` (or `Ok`).

§ 3.3.7 `attributes`

The `attributes` object contains source references and metadata for the authorization decision. It *MAY* be omitted when no source references or metadata are recorded.

Each `adl.core.*` entry in `attributes` *MUST* be a source reference: a value sufficient to retrieve the referenced data. Raw payloads *MUST NOT* be inlined in `attributes`; they belong in `body`. The patterns in 3.5 Sources and referencing illustrate common forms of source references. The retention obligation that follows from this — keeping the upstream source available for as long as the log record references it — is treated in Data Retention.

The following `adl.*` attributes are defined by this specification. The five conditional attributes are present when the corresponding aspect of the authorization decision is to be made retrievable.

Attribute	Type	Mandatory?
`adl.core.request`	object	conditional
`adl.core.response`	object	conditional
`adl.core.policies`	object	conditional
`adl.core.information`	object	conditional
`adl.core.configuration`	object	conditional
`adl.fsc.transaction_id`	string	conditional

When an `adl.core.*` field is recorded in the log record, the data *MUST* be carried in exactly one of two locations:

- as raw data in `body`, or

- via a source reference in `attributes`.

A field *MUST NOT* appear in both `body` and `attributes`. Implementations choose one location per field per record.

Implementations *MAY* include additional attribute keys outside the `adl.*` namespace. Such keys *SHOULD* follow lowercase, dot-separated, namespaced convention (`<vendor>.<area>.<name>`). A consumer that does not recognise an attribute key *MUST* ignore it without error.

§ *3.3.7.1 `adl.core.request`*

The `adl.core.request` attribute is a source reference to the input of the decision when the raw request is not carried in `body`. The reference *SHOULD* resolve to the request in [*AuthZEN*] format, unless privacy considerations require portions to be omitted.

§ *3.3.7.2 `adl.core.response`*

The `adl.core.response` attribute is a source reference to the output of the decision when the raw response is not carried in `body`. The reference *SHOULD* resolve to the response in [*AuthZEN*] format, unless privacy considerations require portions to be omitted.

The response *MUST* be retrievable when `status` is `Unset` or `Ok` and *MAY* be omitted when `status` is `Error`.

> NOTE: Search responses and data minimization
>
> Responses to the Search APIs — in particular `adl.search_subject` and `adl.search_resource` — typically enumerate sets of subjects or resources for which a permission applies. Logging these enumerations in full can itself constitute a privacy exposure beyond the original request. Implementers *SHOULD* pay particular attention to data minimization when logging Search API responses.

§ *3.3.7.3 `adl.core.policies`*

The `adl.core.policies` attribute references the policies that the PDP used to evaluate the request. In a PxP architecture, this represents the information that would come from the PAP.

A PDP can have one or more sources of policies which can be individually versioned. To accommodate that, `adl.core.policies` is an object in which each key identifies a specific policy source.

All policy sources that affected the decision *SHOULD* be referenced. Each value *MUST* be a source reference (see 3.5 Sources and referencing) sufficient to retrieve the policies from that source.

> EXAMPLE 1
> A reference value could be:
>
> - A timestamp
> - A unique identifier
> - A semantic version
> - A Git hash

§ *3.3.7.4 `adl.core.information`*

The `adl.core.information` attribute references the supporting information used in evaluating the access decision. In a PxP architecture, this represents the information that would come from PIPs.

It is an object in which each key identifies an information source. All information sources that affected the decision *SHOULD* be referenced. Each value *MUST* be a source reference (see 3.5 Sources and referencing) sufficient to retrieve the information.

§ *3.3.7.5 `adl.core.configuration`*

The `adl.core.configuration` attribute references the configuration required to reconstruct the software environment that evaluated the original decision. In a PxP architecture, this primarily represents the configuration of the PDP, but *MAY* also include configuration of PIPs and PAPs.

It is an object in which each key identifies a configuration source. All configuration sources that affected the decision *SHOULD* be referenced. Each value *MUST* be a source reference (see 3.5 Sources and referencing) sufficient to retrieve the configuration.

A high-confidence replay claim typically requires capturing all of the following: the engine identifier and version (for example, a container digest), the runtime configuration of the engine, the configuration of every PIP and PAP that contributed to the decision, and any external sources (for example time, randomness) on which the decision depended.

> EXAMPLE 2
> A configuration source could reference:
>
> - The configuration of the policy engine (PDP)
> - The version of the policy language
> - The identifier or hostname of the PDP in case multiple PDPs are used
> - The configuration of PIPs, such as API endpoints
> - The Git hash of an IaaS definition, such as a Terraform repository

§ *3.3.7.6* `adl.fsc.transaction_id`

Unique identifier of the FSC transaction this request belongs to. This attribute *MUST* be set when the corresponding request crosses an FSC inway or outway and is therefore expected to be present in [*FSC-Logging*]; it *MUST NOT* be set otherwise.

> NOTE
>
> The Authorization Decision Log and the FSC Log have the same granularity and can thus be combined into a single physical log. This specification ensures that no fields are defined that conflict with those defined in [*FSC-Logging*].

§ **3.3.8 body**

The `body` field carries the raw `adl.core.*` payloads. It is an object that *MAY* contain any of the following keys:

- `adl.core.request` - full request in [*AuthZEN*] format
- `adl.core.response` - full response in [*AuthZEN*] format
- `adl.core.policies` - full policies that affected the decision

- `adl.core.information` - full information that affected the decision
- `adl.core.configuration` - full configuration that affected the decision

The shape of `body.adl.core.policies`, `body.adl.core.information` and `body.adl.core.configuration` mirrors the shape of the corresponding `attributes.adl.core.*` reference: an object keyed by source name, with the raw payload as the value in place of the source reference.

A `body` *MAY* contain any subset of these keys, and *MAY* be omitted entirely.

3.3.9 `resource`

The `resource` field identifies the producer of the log record: the system, application, or environment in which the PDP evaluated the authorization decision. It is an open object of key/value attribute pairs.

The vocabulary of `resource` keys is intentionally not prescribed by this standard. Organisations select keys appropriate to their operational practice — for example a service identifier, an environment label, or a controller identifier — and document the chosen vocabulary in their logging policy.

When log records are aggregated outside the producing organisation, the producing organisation *MUST* set `resource` such that records can be unambiguously attributed to their producer.

3.4 Span attributes

The log record interface defined in 3.3 Interface does not require OpenTelemetry; it may be emitted via any transport. When ADL *is* implemented on an OpenTelemetry pipeline, however, the same PDP evaluation is typically also represented by a span. The span and the log record describe the same operation and are correlated via the shared `trace_id` and `span_id`. To let observability tooling and the Authorization Decision Log navigate to each other:

- The span's `name` *SHOULD* equal the log record's `event_name` (e.g. `adl.access_evaluation`), so authorization decisions can be identified in tracing tools without joining to the log record.
- Implementations *MAY* additionally mirror selected `adl.core.*` attributes from the log record onto the span's attributes (for example, the `adl.core.policies` reference), to enable filtering of decisions in tracing tools.

§ 3.5 Sources and referencing

A ***source reference*** is a compact value that identifies a source. A source reference *MUST* be sufficient to retrieve, on demand, the data from that source that affected the authorization decision; this lets a log record omit the raw payload from `body` without loss of accountability.

The patterns described below illustrate common reference formats. Implementations are free to use any other format with these properties.

§ 3.5.1 Versioned sources

This section is non-normative.

Some sources offer the ability to 'time-travel' by providing a version at which to query. In such cases, the data itself may be omitted and the version can be stored instead.

The version identifier can be a simple value, such as a string or number, or a complex object, such as an array or object containing multiple version identifiers.

The following example shows a reference to a specific semantic version of a policy source.

EXAMPLE 3: Policy source reference using semantic versioning

```
{
    "traffic-policy": "gmb-2025-94604@1.1.0"
}
```

In a complex case, such as limiting requests for open data per IP per minute across a large number of servers, the version could also consist of an array of partition offsets in a Kafka stream of HTTP requests.

EXAMPLE 4: Complex versioned information sources using Kafka partition offsets

```
{
    "nginx-requests": [ 8376912, 8368118, 8377785, 8386285, 8383526 ]
}
```

> NOTE: Coordinating retention of versioned sources
>
> Organisations relying on Versioned source references *MUST* coordinate the retention of the upstream source with the retention of the log records that reference it, and *MUST* document this coordination in their logging policy. Reference targets that have become unavailable invalidate the accountability claim of the referencing log record.

§ 3.5.2 Temporal sources

This section is non-normative.

In case a source offers the ability to 'time-travel' by providing a timestamp at which to query, then the data itself may be omitted.

It is *RECOMMENDED* to use the timestamp defined in the `time` field in the `context` of the `request` as the base time. In that case the source *MAY* be omitted fully.

If a different timestamp is used, then it *SHOULD* be included in [*RFC3339*] format.

> NOTE: Inter-system clock inconsistencies
>
> When system clocks are not aligned properly, a system may be asked to provide data for a timestamp that lies in the future. This can be mitigated by requesting data of a few seconds or minutes ago at the expense of reducing the speed with which changes can be deployed. The risk of inter-system clock inconsistency can be reduced further by synchronising all participating systems to a trusted time source (for example NTP or PTP); see Information management for related guidance.

> NOTE: Usage of REST API Design Rules
>
> In the context of REST APIs developed by the Dutch government the Temporal extension of the [*ADR*] can be used for this purpose.

§ 3.5.3 Logged sources

This section is non-normative.

For sources that are logged in an external log, a request identifier is needed to look up the corresponding entry in the external log.

It is *RECOMMENDED* to use the `trace_id` and `span_id` carried by [*trace-context*] as the request identifier. The referenced request *SHOULD* have the same `trace_id` as the log record, in which case the source reference can consist of only the value of the `span_id`.

It is *RECOMMENDED* to log requests in the [*WARC*] format as it includes all request and response headers that may be used in the authorization decision.

> NOTE: Where the logged source lives
>
> The external log referenced from a Logged source is typically the producing organisation's own record of the call (for example, the WARC archive of HTTP exchanges originated by PIPs running alongside the PDP) and is managed alongside the Authorization Decision Log itself. When the external log is hosted by another organisation, the producing organisation *SHOULD* coordinate the retention of the external log with the retention of the log records that reference it, and *SHOULD* document this coordination in their logging policy.

The following example shows a log record for a request to find all subjects capable of approving a holiday request, where the call to the HR API is recorded in an external WARC log:

EXAMPLE 5: Log record of a search request for managers with approval rights


```
{
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "17c59821784ee492",
    "parent_span_id": "c4e1d75a3f9b8240",
    "event_name": "adl.search_subject",
    "timestamp": 1757240136089,
    "status": "Unset",
    "attributes": {
        "adl.core.policies": {
            "git": "e4c15a063048367da367d5588d703b5e4a6b760e"
        },
        "adl.core.information": {
            "managers-api": { "span_id": "45deb36022f53afa" }
        }
    },
    "body": {
        "adl.core.request": {
            "subject": {
                "type": "user"
            },
            "action": {
                "name": "approve"
            },
            "resource": {
                "type": "holiday-request",
                "id": "446epbc8y7",
                "properties": {
                    "employee": "bob"
                }
            }
        },
        "adl.core.response": {
            "results": [
                {
                    "type": "user",
                    "id": "carol"
                },
                {
                    "type": "user",
                    "id": "dan"
                }
            ]
```

```
          }
      }
}
```

The `adl.core.information` reference points to a span (`45deb36022f53afa`) within the same trace. That span represents the HR API call. The HTTP exchange of that call is logged as WARC entries indexed by the same `trace_id` and `span_id`:

EXAMPLE 6: WARC entries for REST API call to HR system

```
WARC/1.1
WARC-Type: request
WARC-Date: 2025-09-07T10:15:31Z
WARC-Record-ID: <urn:uuid:48bafbce-8d2a-45c1-9d7a-1a851c36e1c8>
Content-Type: application/http; msgtype=request
Content-Length: 142

GET /users/bob/managers?fields=can_sign HTTP/1.1
Host: hr.example.com
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-45deb36022f53afa-01

WARC/1.1
WARC-Type: response
WARC-Date: 2025-09-07T10:15:32Z
WARC-Record-ID: <urn:uuid:4a381180-21a7-4712-8706-5b321c17e3f8>
WARC-Concurrent-To: <urn:uuid:48bafbce-8d2a-45c1-9d7a-1a851c36e1c8>
Content-Type: application/http; msgtype=response
Content-Length: 175

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 107

{
    "alice": {
        "can_sign": false
    },
    "carol": {
        "can_sign": true
    },
    "dan": {
        "can_sign": true
    }
}
```

4. Data Verifiability and Level of Detail

This section is non-normative.

The ability to provide accountability depends on the log's level of detail. A balance needs to be struck between capturing enough information to accurately replay historical decisions and practical challenges like data duplication and scalability. The appropriate level of detail depends on the organization's specific context and legal requirements, as the highest level is not always necessary.

To ensure historical accuracy while minimizing data storage, referencing external information (e.g., via a timestamp or version number) is preferred over storing copies. This approach keeps logs lean but is contingent on the ability of source systems to provide versioned historical data.

For full replayability, the log also needs to identify the exact version and configuration of the policy engine that evaluated the decision; however, providing reliable versioning for the engine may not always be feasible, depending on the infrastructure.

§ 4.1 Definition of Levels

We have identified four levels of detail, in order from least to most detail. Each level builds on the information from the previous level.

§ 4.1.1 Level 1: Decision Request/Response

At the most basic level only the decision request and the decision response are logged.

> NOTE: Engine boundaries
>
> The decision request and response *MAY* contain all information required for an audit log, as described by [*ISO/IEC 27002:2022*] and [*BIO2*]. Where that is the case, and all auditable actions are decided on by the PDP, the Authorization Decision Log *MAY* be combined with logs of other auditable events and used as an audit log.

At this level of detail log records contain all keys that are described as mandatory in 3.3 Interface.

§ *4.1.1.1 Example*

In the following example a manager called Alice attempts to approve a holiday request for a team member called Bob, but the request is denied because she does not have signing authority.

Her browser submits the following request to the API:

EXAMPLE 7: HTTP request for holiday approval

```
POST /users/bob/holiday-requests/446epbc8y7 HTTP/1.1
Host: hr.example.com
Authorization: Bearer <alice-auth-token>
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-dec5220770f8f4f4-01

{"action":"approve"}
```

The application, acting as the PEP, then submits the following HTTP request to the PDP:

EXAMPLE 8: HTTP request from the PEP to the PDP

```
POST /access/v1/evaluation HTTP/1.1
Host: pdp.example.com
Content-Type: application/json
Authorization: Bearer <pep-auth-token>
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-893e1b2ac52d712f-01

{
    "subject": {
        "type": "user",
        "id": "alice"
    },
    "action": {
        "name": "approve"
    },
    "resource": {
        "type": "holiday-request",
        "id": "446epbc8y7",
        "properties": {
            "employee": "bob"
        }
    },
    "context": {
        "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-dec5220770
    }
}
```

The PDP then determines that Alice can't sign on behalf of the company and thus cannot approve the holiday request. It returns the following response:

EXAMPLE 9: Response from the PDP to the PEP

```
{
    "decision": false,
    "context": {
        "reason": {
            "48": "No signing authority"
        }
    }
}
```

A log record as expressed as a JSON object for this scenario:

EXAMPLE 10: Log record of denied holiday approval


```
{
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "5e3c8a4f9b2d1e07",
    "parent_span_id": "893e1b2ac52d712f",
    "event_name": "adl.access_evaluation",
    "timestamp": 1757240058042,
    "status": "Unset",
    "attributes": {},
    "body": {
        "adl.core.request": {
            "subject": {
                "type": "user",
                "id": "alice"
            },
            "action": {
                "name": "approve"
            },
            "resource": {
                "type": "holiday-request",
                "id": "446epbc8y7",
                "properties": {
                    "employee": "bob"
                }
            },
            "context": {
                "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-de
            }
        },
        "adl.core.response": {
            "decision": false,
            "context": {
                "reason": {
                    "48": "No signing authority"
                }
            }
        }
    }
}
```

§ 4.1.2 Level 2: Decision and Policies

In addition to the request and response, one can refer to the exact version of the policies that were used to evaluate the request. This can be achieved by incorporating the `adl.core.policies` attribute.

§ *4.1.2.1 Example*

We can extend the example of the holiday-approval request by adding a reference to a Git repository in which current HR approval policies are documented. In the example below the git hash of the version currently deployed together with the PDP is `6266d07750c44b4c9b05d0801b752c0ef884e4f6`.

EXAMPLE 11: Git-versioned policy source

```
{
    "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6"
}
```

More complex references can be achieved by using an object as the version identifier. If, for example, the HR application takes part in a federation with predefined policies for different maturity levels. The following example shows how those policies can be referenced using a semantic version combined with a filter for policies relevant to the current maturity level.

EXAMPLE 12: Complex policy source reference

```
{
    "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
    "federation": {
        "version": "2.7.1",
        "filter": "maturity_level <= 3"
    }
}
```

A log record as expressed as a JSON object for this scenario:

EXAMPLE 13: Log record of denied holiday approval including policies reference


```
{
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "5e3c8a4f9b2d1e07",
    "parent_span_id": "893e1b2ac52d712f",
    "event_name": "adl.access_evaluation",
    "timestamp": 1757240058042,
    "status": "Unset",
    "attributes": {
        "adl.core.policies": {
            "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
            "federation": {
                "version": "2.7.1",
                "filter": "maturity_level <= 3"
            }
        }
    },
    "body": {
        "adl.core.request": {
            "subject": {
                "type": "user",
                "id": "alice"
            },
            "action": {
                "name": "approve"
            },
            "resource": {
                "type": "holiday-request",
                "id": "446epbc8y7",
                "properties": {
                    "employee": "bob"
                }
            },
            "context": {
                "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-de
            }
        },
        "adl.core.response": {
            "decision": false,
            "context": {
                "reason": {
                    "48": "No signing authority"
                }
            }
```

```
            }
        }
    }
```

§ 4.1.3 Level 3: All Information Sources

Furthermore, every piece of information used in the evaluation can also be programmatically retrieved, by providing the `adl.core.information` attribute. This allows full replayability, assuming the engine (PDP) behaves identically or can be manually reconstructed in the correct state, which is generally achievable.

§ *4.1.3.1 Example*

In the example of the holiday approval, the ability to sign is accessed through the `can_sign` field of the user. The Policy Information Point called `can-sign-api` requests this via an API from the HR application using the request below:

EXAMPLE 14: PIP's request to the Managers API

```
GET /users/alice?fields=can_sign HTTP/1.1
Host: hr.example.com
traceparent: 00-28dbeec32e77635cc19bc3204ec56c41-836ff5286112f460-01
```

And the API returns the following response:

EXAMPLE 15: Managers API response to the PIP's request

```
{
    "can_sign": false
}
```

A log record as expressed as a JSON object for this scenario:

EXAMPLE 16: Log record of denied holiday approval including policies and information references


```
{
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "5e3c8a4f9b2d1e07",
    "parent_span_id": "893e1b2ac52d712f",
    "event_name": "adl.access_evaluation",
    "timestamp": 1757240058042,
    "status": "Unset",
    "attributes": {
        "adl.core.policies": {
            "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
            "federation": {
                "version": "2.7.1",
                "filter": "maturity_level <= 3"
            }
        },
        "adl.core.information": {
            "can-sign-api": { "span_id": "836ff5286112f460" }
        }
    },
    "body": {
        "adl.core.request": {
            "subject": {
                "type": "user",
                "id": "alice"
            },
            "action": {
                "name": "approve"
            },
            "resource": {
                "type": "holiday-request",
                "id": "446epbc8y7",
                "properties": {
                    "employee": "bob"
                }
            },
            "context": {
                "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-de
            }
        },
        "adl.core.response": {
            "decision": false,
            "context": {
                "reason": {
```

```
                    "48": "No signing authority"
                }
            }
        }
    }
}
```

> NOTE: Logged source for the PIP call
>
> The `adl.core.information` reference for `can-sign-api` is a Logged source: it points to a span (`836ff5286112f460`) within the same trace that represents the PIP call to the HR API. The actual API request/response is retrieved from an external log (for example a WARC archive) keyed by `trace_id` and `span_id`. See 3.5 Sources and referencing for the alternative reference patterns.

§ 4.1.4 Level 4: Full Environment

In addition to all information used in the evaluation, the environment and configuration of the system that evaluates the decision can also be accurately reconstructed through the use of the `adl.core.configuration` attribute. This provides full, guaranteed replayability and maximum accountability.

> NOTE: System boundaries
>
> The configuration of all components that influence the decision should be included. While this may be limited to the configuration of the PDP, it often also requires configuration of the PIP and sometimes the PAP as well.

At this level of detail log records contain the following keys, as defined in 3. Specifications:

§ *4.1.4.1 Example*

In the example below we extend the holiday approval request example by describing the version of the language used by the PDP and the configuration of the `can-sign-api` PIP.

EXAMPLE 17: Log record of denied holiday approval including configuration in body


```
{
    "trace_id": "28dbeec32e77635cc19bc3204ec56c41",
    "span_id": "5e3c8a4f9b2d1e07",
    "parent_span_id": "893e1b2ac52d712f",
    "event_name": "adl.access_evaluation",
    "timestamp": 1757240058042,
    "status": "Unset",
    "attributes": {
        "adl.core.policies": {
            "hr": "6266d07750c44b4c9b05d0801b752c0ef884e4f6",
            "federation": {
                "version": "2.7.1",
                "filter": "maturity_level <= 3"
            }
        },
        "adl.core.information": {
            "can-sign-api": { "span_id": "836ff5286112f460" }
        }
    },
    "body": {
        "adl.core.request": {
            "subject": {
                "type": "user",
                "id": "alice"
            },
            "action": {
                "name": "approve"
            },
            "resource": {
                "type": "holiday-request",
                "id": "446epbc8y7",
                "properties": {
                    "employee": "bob"
                }
            },
            "context": {
                "traceparent": "00-28dbeec32e77635cc19bc3204ec56c41-de
            }
        },
        "adl.core.response": {
            "decision": false,
            "context": {
                "reason": {
                    "48": "No signing authority"
```

```
                }
            }
        },
        "adl.core.configuration": {
            "opa": "1.10.0",
            "can-sign-api": "https://hr.example.com/users/{subject.id}
        }
    }
}
```

NOTE: Configuration as raw data in body

In this example the `configuration` is raw data — a few key/value pairs describing the OPA version and the can-sign-api endpoint URL — rather than a pointer to a coherent external configuration source. It therefore belongs in `body` alongside `request` and `response`.

When configuration *is* available from an external source (e.g., a Git-versioned IaaS definition or a configuration management system), the `attributes.adl.core.configuration` reference pattern (analogous to `adl.core.policies`) *SHOULD* be used instead.

§ 4.2 Implications of levels

The higher the level of detail, the more useful the log is for determining the context of an authorization decision. On the other hand, higher levels of detail also introduce challenges around scalability, technical feasibility, and security.

Conversely, the lowest level of detail may not be sufficient to provide effective accountability. This depends on the data processing which is being authorized and legal requirements for it.

For that reason it's important to decide for different use cases which level of detail is required and appropriate. Aiming for the highest level of detail for all authorization decisions is thus not necessary.

§ 5. Information Management and Compliance

This section is non-normative.

Conformance to this standard does not, by itself, guarantee legal or regulatory compliance. The implementing organization is solely responsible for ensuring its implementation adheres to all applicable frameworks, such as the General Data Protection Regulation (GDPR / AVG) and relevant security baselines, such as [*ISO/IEC 27001:2022*], [*ISO/IEC 27002:2022*], and [*BIO2*]. Each organization is responsible for its own Authorization Decision Log. There is no central log, although logs of several organizations can be aggregated if desired.

The following sections list aspects that should be taken into consideration when creating a compliant and secure logging solution.

§ 5.1 Legal and Privacy Compliance

Logging authorization decisions creates a new processing of data, which is subject to privacy regulations when personal data is involved.

§ 5.1.1 Purpose Limitation

When collecting personal data, the specific purposes for logging (e.g., operational auditing, forensic analysis, citizen accountability) should be defined and documented in a formal policy before implementation, in line with the purpose-limitation principle of [*AVG*] art. 5(1)(b). Data collection should be limited to those defined purposes.

§ 5.1.2 Data Minimization

A core principle is to avoid storing unnecessary information, especially sensitive data. The goal is to make the log precise, compact, and manageable. Instead of duplicating large amounts of (personal) data, prefer storing only the data used in the decision or even just a reference that allows the state at the time of the decision to be reconstructed.

A logging policy that manages what is logged based on risk should be in place. When logging sensitive data for high-risk use cases, this should be explicitly documented and approved.

Implementers should consider pseudonymization and anonymization for personal data, in line with the data-masking control in section 8.11 of [*ISO/IEC 27002:2022*]. For example, personal identifiers can be hashed or aliased and location data can be randomized.

When aggregate statistics, such as decisions per type per time period, are sufficient, implementers should consider using aggregation as a method of data minimization and anonymization.

§ 5.1.3 Data Retention

A data retention policy is required by [*AVG*] art. 5(1)(e) and should be defined and enforced. Retention periods should be based on the defined purposes and legal obligations, and may differ for different types of log records.

As a general guideline, operational logs (for debugging, support) should be retained for a short period (months), while forensic/audit logs may be retained for longer periods (years).

For implementations within Dutch public-sector organisations, retention is also subject to the Archiefwet and the applicable Selectielijst — for example, the Selectielijst gemeenten en intergemeentelijke organen for municipal deployments, or the central-government Selectielijst for ministries. Retention periods chosen for logs should be reconciled with these obligations.

Logs that have exceeded their defined retention period should be automatically and securely purged in line with the secure-deletion control in section 8.10 of [*ISO/IEC 27002:2022*].

When `attributes.adl.core.*` carries source references rather than raw payloads, the upstream sources *MUST* remain retrievable for the full retention period of the referencing log records; otherwise the accountability claim of the referencing log record is invalidated. Coordination of upstream retention is the responsibility of the producing organisation and should be documented in its logging policy. See 3.5 Sources and referencing for the related Versioned, Temporal, and Logged source patterns.

§ 5.1.4 Transparency and Subject Rights

If logs contain personal data, they should be designed to support data subject access requests under [*AVG*] art. 15. To enable subject-access queries that span ADL and [*LDV*], implementations *SHOULD* include the `dpl.core.data_subject_id` attribute defined by [*LDV*] in `attributes` when the request involves personal data. This anchors the log record to the data subject in the same convention used by LDV.

The log's structure, purpose, and retention are to be documented in the organization's Register of Processing Activities, as required by [*AVG*] art. 30.

Implementations whose log content constitutes large-scale or high-risk processing of personal data are subject to the DPIA requirement under [*AVG*] art. 35. This standard does not impose a DPIA

requirement; it does require organisations to assess whether one applies. The DPIA-leidraad published by the Autoriteit Persoonsgegevens can serve as the basis for such an assessment. When a DPIA is performed, the choice of detail level, retention period, and access scope of the Authorization Decision Log should be considered as part of it.

§ 5.2 Access Control

Access to log data should be restricted based on the principle of least privilege.

It is essential to define clear policies for authorizing access to the Authorization Decision Log or parts thereof. These decisions to provide or deny access to the log should also be included in the Authorization Decision Log.

Common and important usage policies include:

- **(Forensic) Audits**: The log is a critical tool for auditing and forensic analysis after a security incident or data breach. It can help determine what actions were permitted at a specific time and on what basis, even if that permission was technically correct but improper in hindsight.
- **Observability in Trust Frameworks**: The log offers a structured method for data users to provide insight into their data usage to data providers when required, as may be required in trust frameworks. It thus offers an implementation standard for "Observability services" as defined for data spaces under the EU Data Act.
- **Debugging and Support**: The log can be a useful tool for determining why the authorization is not working as expected. It can also contain highly sensitive data, however, so it's essential to carefully define if, and under which conditions, the Authorization Decision Log can be used for this purpose.

§ 5.3 Security and Integrity

The log should be protected against unauthorized access, modification, and deletion. The relevant operational baselines for the Dutch government context are [*ISO/IEC 27002:2022*] (mirrored in [*BIO2*]).

Key concerns for ensuring security and integrity include:

- **Transport Security**: It's recommended to use mTLS (Mutual Transport Layer Security) for network connections that transport log data to ensure authenticated and encrypted transport.

- **Encryption at Rest**: All log data should be encrypted at rest. It's recommended to manage this using a Key Management System (KMS).
- **Data Integrity**: The log should be configured as append-only storage (WORM) to prevent undetected modification or deletion, in line with section 5.33 of [*ISO/IEC 27002:2022*]. Mechanisms such as a cryptographic hash chains and periodic cryptographic sealing can be used to ensure integrity and non-repudiation of logs.
- **Time Synchronization**: All systems involved in generating and storing logs should be synchronized to a trusted Network Time Protocol (NTP) source to ensure a reliable and accurate timeline of events, in line with section 8.17 of [*ISO/IEC 27002:2022*].
- *Ingestion*: The logging endpoint should be implemented as idempotent writes to an asynchronous, buffered service (e.g., using a durable queue) to mitigate latency and availability risks in the event of log-ingest failures. See 3.2.4 Ingestion for the normative idempotency rule.

6. List of Figures

A. Index

A.1 Terms defined by this specification

Reconstruct §1.2

Replay §1.2

Source §1.2

source reference §3.5

Span §1.2

Trace §1.2

Trace context §1.2

§ A.2 Terms defined by reference

[*AUTHZEN*] defines the following:

Access Evaluation API

Access Evaluations API

Action Search API

Resource Search API

Subject Search API

[*TRACE-CONTEXT*] defines the following:

section 3.5 (Mutating the tracestate Field)

section 6 (Privacy Considerations)

§ B. References

§ B.1 Normative references

[AuthZEN]

Authorization API 1.0. O. Gazitt; D. Brossard; A. Tulshibagwale. URL: https://openid.net/specs/authorization-api-1_0.html

[FSC-Core]

FSC - Core. Eelco Hotting; Ronald Koster; Henk van Maanen; Niels Dequeker; Edward van Gelderen; Pim Gaemers. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/fsc/core/2.0.0/

[FSC-Logging]

FSC - Logging. Eelco Hotting; Ronald Koster; Henk van Maanen; Niels Dequeker; Edward van Gelderen; Pim Gaemers. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/fsc/logging/1.1.0/

[LDV]

Logboek Dataverwerkingen. Logius. URL: https://logius-standaarden.github.io/logboek-dataverwerkingen/

[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

[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

[RFC9112]

HTTP/1.1. R. Fielding; M. Nottingham; J. Reschke. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9112.html

[RFC9113]

HTTP/2. M. Thomson; C. Benfield. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9113.html

[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/

§ B.2 Informative references

[ADR]

API Design Rules. Jasper Roes; Joost Farla. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/2.1

[AVG]

Algemene Verordening Gegevensbescherming. Verordening (EU) 2016/679 van het Europees Parlement. 27 april 2016. URL: https://eur-lex.europa.eu/legal-content/NL/TXT/?uri=CELEX%3A32016R0679

[BIO2]

Circulaire Baseline Informatiebeveiliging Overheid 2. 5 maart 2026. URL: https://zoek.officielebekendmakingen.nl/stcrt-2026-7416-n1.html

[ISO/IEC 27001:2022]

Information security, cybersecurity and privacy protection — Information security management systems — Requirements. 2022-10. URL: https://www.iso.org/standard/27001

[ISO/IEC 27002:2022]

Information security, cybersecurity and privacy protection — Information security controls. 2022-02. URL: https://www.iso.org/standard/75652.html

[NIST.SP.800-162]

Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Chung Tong Hu; David F. Ferraiolo; David R. Kuhn. February 25, 2019. URL: https://doi.org/10.6028/NIST.SP.800-162

[OpenTelemetry]

OpenTelemetry Specification. Cloud Native Computing Foundation. URL: https://opentelemetry.io/docs/specs/otel/

[OTLP]

OpenTelemetry Protocol (OTLP) Specification. Cloud Native Computing Foundation. URL: https://opentelemetry.io/docs/specs/otlp/

[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

[WARC]

Information and documentation — WARC file format. International Organization for Standardization (ISO). August 2017. Published. URL: https://www.iso.org/standard/68004.html

↥