Remote ATtestation ProcedureS
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

P. Howard
Arm
T. Fossati
Linaro
H. Birkholz
Fraunhofer SIT
S. Kamal
Fujitsu
G. Mandyam
AMD
D. Ma
Alibaba Cloud
2 March 2026

Concise Selector for Endorsements and Reference Values
draft-ietf-rats-coserv-05

Abstract

In the Remote Attestation Procedures (RATS) architecture, Verifiers require Endorsements and Reference Values to assess the trustworthiness of Attesters. This document specifies the Concise Selector for Endorsements and Reference Values (CoSERV), a structured query/result format designed to facilitate the discovery and retrieval of these artifacts from various providers. CoSERV defines a query language and corresponding result structure using CDDL, which can be serialized in CBOR format, enabling efficient interoperability across diverse systems.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-rats-wg.github.io/draft-ietf-rats-coserv/draft-ietf-rats-coserv.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-rats-coserv/.

Discussion of this document takes place on the Remote ATtestation ProcedureS Working Group mailing list (mailto:rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/. Subscribe at https://www.ietf.org/mailman/listinfo/rats/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-rats-wg/draft-ietf-rats-coserv.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 September 2026.

Table of Contents

1. Introduction

Remote Attestation Procedures (RATS) enable Relying Parties to evaluate the trustworthiness of remote Attesters by appraising Evidence. This appraisal necessitates access to Endorsements and Reference Values, which are often distributed across multiple providers, including hardware manufacturers, firmware developers, and software vendors. The lack of standardized methods for querying and

retrieving these artifacts poses challenges in achieving seamless interoperability.

The Concise Selector for Endorsements and Reference Values (CoSERV) addresses this challenge by defining a query language and a corresponding result structure for the transaction of artifacts between a provider and a consumer. The query language format provides Verifiers with a standard way to specify the environment characteristics of Attesters, such that the relevant artifacts can be obtained from Endorsers and Reference Value Providers. In turn, the result format allows those Endorsers and Reference Value Providers to package the artifacts within a standard structure. This facilitates the efficient discovery and retrieval of relevant Endorsements and Reference Values from providers, maximising the re-use of common software tools and libraries within the transactions.

The CoSERV query language is intended to form the input data type for tools and services that provide access to Endorsements and Reference Values. The CoSERV result set is intended to form the corresponding output data type from those tools and services.

The environment characteristics of Endorsements and Reference Values are derived from the equivalent concepts in CoRIM [I-D.ietf-rats-corim]. CoSERV therefore borrows heavily from CoRIM, and shares some data types for its fields. And, like CoRIM, the CoSERV schema is defined using CDDL [RFC8610]. A CoSERV query can be serialized in CBOR [STD94] format.

In addition to the CBOR-based data formats for CoSERV queries and responses, this specification also defines API bindings and behaviours for the exchange of CoSERV queries and responses. This is to facilitate standard interactions between CoSERV producers and consumers. Standard API endpoints and behaviours will encourage the growth of interoperable software tools and modules, not only for parsing and emitting CoSERV-compliant data, but also for implementing the clients and services that need to exchange such data when acting in the capacity of the relevant RATS roles. This will be of greater benefit to the software ecosystem than the CoSERV data format alone. See Section 6 for the API binding specifications.

1.1. Terminology and Requirements Language

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

This document uses terms and concepts defined by the RATS architecture. For a complete glossary, see Section 4 of [RFC9334].

This document uses terms and concepts defined by the CoRIM specification. For a complete glossary, see Section 1.1.1 of [I-D.ietf-rats-corim].

This document uses the terms _"actual state"_ and _"reference state"_ as defined in Section 2 of [I-D.ietf-rats-endorsements].

The terminology from CBOR [STD94], CDDL [RFC8610] and COSE [STD96] applies; in particular, CBOR diagnostic notation is defined in Section 8 of [STD94] and Appendix G of [RFC8610]. Terms and concepts are always referenced as proper nouns, i.e., with Capital Letters.

2. Aggregation and Trust Models

The roles of Endorser or Reference Value Provider might sometimes be fulfilled by aggregators, which collect from multiple supply chain sources, or even from other aggregators, in order to project a holistic view of the endorsed system. The notion of such an aggregator is not explicit in the RATS architecture. In practice, however, supply chains are complex and multi-layered. Supply chain sources can include silicon manufacturers, device manufacturers, firmware houses, system integrators, service providers and more. In practical terms, an Attester is likely to be a complex entity, formed of components from across such supply chains. Evidence would be likewise structured, with contributions from different segments of the Attester’s overall anatomy. A Verifier for such Evidence may find it convenient to contact an aggregator as a single source of truth for Endorsements and Reference Values. An aggregator would have intelligence about the Attester’s complete anatomy and supply chain. It would have the ability to contact all contributing supply chain actors for their individual Endorsements and Reference Values, before collecting them into a cohesive set, and delivering them to the Verifier as a single, ergonomic package. The contributing supply chain actors might themselves be CoSERV-enabled, in which case the aggregator would send one or more separate CoSERV queries to those actors as part of the aggregation process. Alternatively, it might be necessary to use vendor-specific protocols to gather these artifacts and convert them into the aggregated CoSERV response. The choice between these approaches is deployment-dependent, and is considered to be an implementation detail of the aggregator. In pure RATS terms, an aggregator is still an Endorser or a Reference Value Provider - or, more likely, both. It is not a distinct role, and so there is no distinctly-modeled conveyance between an aggregator and a Verifier. However, when consuming from an aggregator, the Verifier may need visibility of the aggregation process, possibly to the

extent of needing to audit the results by inspecting the individual inputs that came from the original supply chain actors. CoSERV addresses this need, catering equally for both aggregating and non-aggregating supply chain sources.

To support deployments with aggregators, CoSERV allows for flexible trust models as follows.

* *Shallow Trust*: in this model, the consumer trusts the aggregator, solely and completely, to provide authentic descriptions of the endorsed system. The consumer does not need to audit the results of the aggregation process.

* *Deep Trust*: in this model, the consumer has a trust relationship with the aggregator, but does not deem this to be sufficient. The consumer can still use the collected results from the aggregation process, where it is convenient to do so, but also needs to audit those results.

Any given CoSERV transaction can operate according to either model. The consumer decides which model to use when it forms a query. The CoSERV result payload can convey both the aggregated result and the audit trail as needed. The payload size may be smaller when the shallow model is used, but the choice between the two models is a question for implementations and deployments.

Although CoSERV is designed to support aggregation, it is not a requirement. When aggregation is not used, CoSERV still fulfills the need for a standard conveyance mechanism between Verifiers and Endorsers or Reference Value Providers.

3. CoSERV Information Model

3.1. Overview

CoSERV is designed to facilitate query-response transactions between a producer and a consumer. In the RATS model, the producer is either an Endorser or a Reference Value Provider, and the consumer is a Verifier. CoSERV defines a single top-level data type that can be used for both queries and result sets. Queries are authored by the consumer (Verifier), while result sets are authored by the producer (Endorser or Reference Value Provider) in response to the query. A CoSERV data object always contains a query. When CoSERV is used to express a result set, the query is retained alongside the result set that was yielded by that query. This allows consumers to verify a match between the query that was sent to the producer, and the query that was subsequently returned with the result set. Such verification is useful because it mitigates security threats arising

from any untrusted infrastructure or intermediaries that might reside between the producer and the consumer. An example of this is caching in HTTP [STD98] and CoAP [RFC7252]. It might be expensive to compute the result set for a query, which would make caching desirable. However, if caching is managed by an untrusted intermediary, then there is a risk that such an untrusted intermediary might return incorrect results, either accidentally or maliciously. Pairing the original query with each result set provides an end-to-end contract between the consumer and producer, mitigating such risks. The transactional pattern between the producer and the consumer would be that the consumer begins the transaction by authoring a query and sending it to the producer as a CoSERV object. The producer receives the query, computes results, and returns a new CoSERV object formed from the results along with the original query. Notionally, the producer is "adding" the results to the query before sending it back to the consumer.

3.2. Artifacts

Artifacts are what the consumer (Verifier) needs in order to verify and appraise Evidence from the Attester, and therefore they form the bulk of the response payload in a CoSERV transaction. The common CoSERV query language recognises three artifact types. These correspond to the three categories of endorsement artifact that can be identified natively in the RATS architecture:

* *Trust Anchor*: A trust anchor is as defined in [RFC6024]. An example of a trust anchor would be the public part of the asymmetric signing key that is used by the Attester to sign Evidence, such that the Verifier can verify the cryptographic signature. This is just one example. Other forms of trust anchor are possible. CoSERV does not place any additional requirements or constraints on the conveyance or use of trust anchors, beyond what is already described in [RFC9334] and [I-D.ietf-rats-endorsements]. Section 4 of [I-D.ietf-rats-endorsements] sets out the applicable patterns for the endorsement of verification keys, all of which apply equally here.

* *Endorsed Value*: An endorsed value is as defined in Section 1.1.1 of [I-D.ietf-rats-corim]. This represents a characteristic of the Attester that is not directly presented in the Evidence, such as certification data related to a hardware or firmware module.

* *Reference Value*: A reference value is as defined in Section 1.1.1 of [I-D.ietf-rats-corim]. A reference value specifies an individual aspect of the Attester’s desired state. Reference values are sometimes informally called "golden values".

An example of a reference value would be the expected hash or checksum of a binary firmware or software image running in the Attester's environment. Evidence from the Attester would then include claims about the Attester's actual state, which the Verifier can then compare with the reference values at Evidence appraisal time.

When artifacts are produced by an aggregator (see Section 2), the following additional classifications apply:

* *Collected Artifacts*: these refer to artifacts that were derived by the aggregator by collecting and presenting data from original supply chain sources, or from other aggregators. Collected artifacts form a single holistic package, and provide the most ergonomic consumption experience for the Verifier.

* *Source Arfifacts*: these refer to artifacts that were obtained directly from the original supply chain sources, and used as inputs into the aggregation process, allowing the aggregator to derive the collected artifacts.

In the shallow trust model of aggregation, only the collected artifacts are used by the consumer. In the deep trust model, both the collected artifacts and the source artifacts are used. The source artifacts allow the consumer to audit the collected artifacts and operate the trust-but-verify principle.

3.3. Environments

The environment defines the scope (or scopes) in which the endorsement artifacts are applicable. Given that the consumer of these artifacts is likely to be a Verifier in the RATS model, the typical interpretation of the environment would be that of an Attester that either has produced evidence, or is expected to produce evidence, that the Verifier needs to appraise. The Verifier consequently needs to query the Endorser or Reference Value Provider for artifacts that are applicable in that environment. There are three mutually-exclusive methods for defining the environment within a CoSERV query. Exactly one of these three methods MUST be used for the query to be valid. All three methods correspond to environments that are also defined within CoRIM [I-D.ietf-rats-corim].

* *Class*: A class is an environment that is expected to be common to a group of similarly-constructed Attesters, who might therefore share the same set of endorsed characteristics. An example of this might be a fleet of computing devices of the same model and manufacturer.

* *Instance*: An instance is an environment that is unique to an individual and identifiable Attester, such as a single computing device or component.

* *Group*: A group is a collection of common Attester instances that are collected together based on some defined semantics. For example, Attesters may be put into groups for the purpose of anonymity.

3.3.1. Stateful Environments

In addition to specifying the Attester environment by class, instance, or group, it is sometimes necessary to constrain the target environment further by specifying aspects of its state. This is because the applicability of Endorsements and Reference Values might vary, depending on these stateful properties. Consider, for example, an Attester instance who signs Evidence using a derived attestation key, where the derivation algorithm is dependent on one or more aspects of the Attester's current state, such as the version number of an upgradable firmware component. This example Attester would, at different points in its lifecycle, sign Evidence with different attestation keys, since the keys would change upon any firmware update. To provide the correct public key to use as the trust anchor for verification, the Endorser would need to know the configured state of the Attester at the time the Evidence was produced. Specifying such an Attester solely by its instance identifier is therefore insufficient for the Endorser to supply the correct artifact. The environment specification would need to include these critical stateful aspects as well. In CoRIM [I-D.ietf-rats-corim], stateful environments are modeled as an environment identifier plus a collection of measurements, and CoSERV takes the same approach. Therefore, any environment selector in a CoSERV query can optionally be enhanced with a collection of one or more measurements, which specify aspects of the target environment state that might materially impact the selection of artifacts.

3.4. Queries

The purpose of a query is to allow the consumer (Verifier) to specify the artifacts that it needs. The information that is conveyed in a CoSERV query includes the following:

* A specification of the required artifact type: Reference Value, Endorsed Value or Trust Anchor. See Section 3.2 for definitions of artifact types. A single CoSERV query can only specify a single artifact type.

* A specification of the Attester’s environment. Environments can be selected according to Attester instance, group or class. Additional properties of the environment state can be specified by adding one or more measurements to the selector. See Section 3.3 for full definitions. To facilitate efficient transactions, a single query can specify either multiple instances, multiple groups or multiple classes. However, it is not possible to mix instance-based selectors, group-based selectors and class-based selectors in a single query.

* A switch to select the desired supply chain depth. A CoSERV query can request collected artifacts, source artifacts, or both. This switch is especially relevant when the CoSERV query is fulfilled by an aggregator. The collected artifacts are intended for convenient consumption (according to the shallow trust model), while the source artifacts are principally useful for auditing (according to the deep trust model). It is possible for a query to select for source artifacts only, without the collected artifacts. This might happen when the consumer needs to inspect or audit artifacts from across the deep supply chain, while not requiring the convenience of the aggregated view. It could also happen when the consumer is acting as an intermediate broker, gathering artifacts for delivery to another aggregator. See Section 2 for details on aggregation, auditing and trust models.

CoSERV queries do not contain timestamps or any similarly volatile or unpredictable fields. This is to ensure that any set of materially-identical queries will always yield the same encoded sequence of CBOR bytes, regardless of the time when they were issued, or of other volatile factors. Along with the other encoding rules set out in Section 4.5, it means that a query can be used as a stable and canonical identifier of artifacts. This property of queries is an important enabler of efficient CoSERV transactions. See, for example, the HTTP caching design described in Section 6.1.3.

A CoSERV implementation MAY log the time at which a query was received and fulfilled. This might sometimes be desirable for transparency or audit purposes. Implementations are free to define their own transparency events, which can then include timestamps or other suitable information.

3.5. Result Sets

The result set contains the artifacts that the producer collected in response to the query. The top-level structure of the result set consists of the following three items:

* A collection of one or more result entries. This will be a collection of either reference values, endorsed values or trust anchors. See Section 3.2 for definitions of artifact types. Artifact types are never mixed in any single CoSERV result set. The artifacts in the result collection therefore MUST match the single artifact type specified in the original CoSERV query.

* A timestamp indicating the expiry time of the entire result set. Consumers MUST NOT consider any part of the result set to be valid after this expiry time.

* A collection of the original source materials from which the producer derived the correct artifacts to include in the result set. These source materials are optional, and their intended purpose is auditing. They are included only when requested by the original CoSERV query. Source materials would typically be requested in cases where the consumer is not willing to place sole trust in the producer, and therefore needs an audit trail to enable additional verifications.

Each individual result entry combines a CoMID triple with an authority delegation chain. CoMID triples are exactly as defined in Section 5.1.4 of [I-D.ietf-rats-corim]. Each CoMID triple will demonstrate the association between an environment matching that of the CoSERV query, and a single artifact such as a reference value, trust anchor or endorsed value. The authority delegation chain is composed of one or more authority delegates. Each authority delegate is represented by a public key or key identifier, which the consumer can check against its own set of trusted authorities. The authority delegation chain serves to establish the provenance of the result entry, and enables the Verifier to evaluate the trustworthiness of the associated artifact. The purpose of the authority delegation chain is to allow CoSERV responses to support decentralized trust models, where Verifiers may apply their own policy to determine which authorities are acceptable for different classes of artifact.

Because each result entry combines a CoMID triple with an authority delegation chain, the entries are consequently known as quadruples (or "quads" for short).

4. CoSERV Data Model

This section specifies the CBOR data model for CoSERV queries and result sets.

CDDL is used to express rules and constraints of the data model for CBOR. These rules must be strictly followed when creating or validating CoSERV data objects.

The top-level CoSERV data structure is given by the following CDDL:

```
;# import comid-autogen

coserv = {
  &(profile: 0) => profile
  &(query: 1) => query
  ? &(results: 2) => results
}

profile = comid.oid-type / ~uri
```

4.1. Common Data Types

CoSERV inherits the following types from the CoRIM data model class-map, $class-id-type-choice, $instance-id-type-choice and $group-id-type-choice.

The collated CDDL is in Appendix A.1.

4.2. Profile

In common with EAT and CoRIM, CoSERV supports the notion of profiles. As with EAT and CoRIM, profiles are a way to extend or specialize the structure of a generic CoSERV query in order to cater for a specific use case or environment.

In a CoSERV query, the profile can be identified by either a Uniform Resource Identifier (URI) or an Object Identifier (OID). This convention is identical to how EAT profiles are identified using the eat_profile claim as described in Section 4.3.2 of [RFC9711].

4.3. Query Structure

The CoSERV query language enables Verifiers to specify the desired characteristics of Endorsements and Reference Values based on the environment in which they are applicable.

The top-level structure of a CoSERV query is given by the following CDDL:

```
query = {
  &(artifact-type: 0) => artifact-type
  &(environment-selector: 1) => environment-selector-map
  &(result-type: 2) => result-type
}

artifact-type = &(endorsed-values: 0)
                / &(trust-anchors: 1)
                / &(reference-values: 2)

result-type = &(collected-artifacts: 0)
              / &(source-artifacts: 1)
              / &(both: 2)
```

The meanings of these fields are detailed in the following subsections.

4.3.1. Artifact Type

The artifact-type field is the foremost discriminator of the query. It is a top-level category selector. Its three permissible values are trust-anchors (codepoint 1), endorsed-values (codepoint 0) and reference-values (codepoint 2).

See Section 3.2 for full definitions of artifact types.

It is expected that implementations might choose to store these different categories of artifacts in different top-level stores or database tables. Where this is the case, the artifact-type field serves to narrow the query down to the correct store or table. Even where this is not the case, the discriminator is useful as a filter for the consumer, resulting in an efficiency gain by avoiding the transfer of unwanted data items.

4.3.2. Environment Selector

The environment selector forms the main body of the query, and its CDDL is given below:

```
;# import comid-autogen

environment-selector-map = { selector }

stateful-class = [
  class: comid.class-map
  ? measurements: [ + comid.measurement-map ]
]

selector //= ( &(class: 0) => [
  + stateful-class
] )

stateful-instance = [
  instance: comid.$instance-id-type-choice
  ? measurements: [ + comid.measurement-map ]
]

selector //= ( &(instance: 1) => [
  + stateful-instance
] )

stateful-group = [
  group: comid.$group-id-type-choice
  ? measurements: [ + comid.measurement-map ]
]

selector //= ( &(group: 2) => [
  + stateful-group
] )
```

Environments can be specified according to instance, group or class. See Section 3.3 for details.

Although these three environment definitions are mutually-exclusive in a CoSERV query, all three support multiple entries. This is to gain efficiency by allowing the consumer (Verifier) to query for multiple artifacts in a single transaction. For example, where artifacts are being indexed by instance, it would be possible to specify an arbitrary number of instances in a single query, and therefore obtain the artifacts for all of them in a single transaction. Likewise for classes and groups. However, it would not be possible for a single query to specify more than one kind of environment. For example, it would not be possible to query for both class-level and instance-level artifacts in a single CoSERV transaction.

All three environment selector types can optionally be enhanced with one or more measurement-map entries, which are used to express aspects of the environment state. See Section 3.3.1 for a description of stateful environments.

4.3.2.1. Selector Semantics

When multiple environment selectors are present in a single query, such as multiple instances or multiple groups, the recipient of the query MUST consider these to be alternatives, and hence use a logical OR operation when applying the query to its internal data stores.

Below is an illustrative example of how a CoSERV query for endorsed values, selecting for multiple Attester instances, might be transformed into a semantically-equivalent SQL database query:

```
SELECT *
  FROM endorsed_values
 WHERE ( instance-id = "At6tvu/erQ==" ) OR
       ( instance-id = "iZl4ZVY=" )‘
```

The same applies for class-based selectors; however, since class selectors are themselves composed of multiple inner fields, the recipient of the query MUST use a logical AND operation in consideration of the inner fields for each class.

Also, for class-based selectors, any unset fields in the class are assumed to be wildcard (*), and therefore match any value.

Below is an illustrative example of how a CoSERV query for reference values, selecting for multiple Attester classes, might be transformed into a semantically-equivalent SQL database query:

```
SELECT *
  FROM reference_values
 WHERE ( class-id = "iZl4ZVY=" AND class-vendor = "ACME Inc." ) OR
       ( class-id = "31fb5abf-023e-4992-aa4e-95f9c1503bfa" )
```

4.3.3. Result Type

The result-type field selects for the desired type(s) of results: collected-artifacts (codepoint 0), source-artifacts (codepoint 1) or both (codepoint 2). See Section 2 for definitions of source and collected artifacts.

4.4. Result Set Structure

The result set structure is given by the following CDDL:

```
;# import cmw-autogen
;# import comid-autogen

results = {
  result-set
  &(expiry: 10) => tdate ; RFC3339 date
  ? &(source-artifacts: 11) => [ + cmw.cbor-record ]
}

result-set //= reference-values
result-set //= endorsed-values
result-set //= trust-anchors

refval-quad = {
  &(authorities: 1) => [ + comid.$crypto-key-type-choice ]
  &(rv-triple: 2) => comid.reference-triple-record
}

reference-values = (
  &(rvq: 0) => [ * refval-quad ]
)

endval-quad = {
  &(authorities: 1) => [ + comid.$crypto-key-type-choice ]
  &(ev-triple: 2) => comid.endorsed-triple-record
}

cond-endval-quad = {
  &(authorities: 1) => [ + comid.$crypto-key-type-choice ]
  &(ce-triple: 2) => comid.conditional-endorsement-triple-record
}

endorsed-values = (
  &(evq: 1) => [ * endval-quad ]
  &(ceq: 2) => [ * cond-endval-quad ]
)

ak-quad = {
  &(authorities: 1) => [ + comid.$crypto-key-type-choice ]
  &(ak-triple: 2) => comid.attest-key-triple-record
}

cots-stmt = {
  &(authorities: 1) => [ + comid.$crypto-key-type-choice ]
```

```
    &(cots: 2) => cots
  }

  trust-anchors = (
    &(akq: 3) => [ * ak-quad ]
    &(tas: 4) => [ * cots-stmt ]
  )

  ;
  ; import CoTS
  ;
  cots = "TODO COTS"
```

4.5. Encoding Requirements

Implementations may wish to use serialized CoSERV queries as canonical identifiers for artifact collections. For example, a Reference Value Provider service may wish the cache the results of a CoSERV query to gain efficiency when responding to a future identical query. For these use cases to be effective, it is essential that any given CoSERV query is always serialized to the same fixed sequence of CBOR bytes. Therefore, CoSERV queries MUST always use CBOR deterministic encoding as specified in Section 4.2 of [STD94]. Further, CoSERV queries MUST use CBOR definite-length encoding.

4.6. Cryptographic Binding Between Query and Result Set

CoSERV is designed to ensure that any result set passed from a producer to a consumer is precisely the result set that corresponds to the consumer’s original query. This is the reason why the original query is always included along with the result set in the data model. However, this measure is only sufficient in cases where the conveyance protocol guarantees that CoSERV result sets are always transacted over a secure channel without any untrusted intermediaries. Wherever this is not the case, producers MUST create an additional cryptographic binding between the query and the result. This is achieved by transacting the result set within a cryptographic envelope, with a signature added by the producer, which is verified by the consumer. A CoSERV data object can be signed using COSE [STD96]. A signed-coserv is a COSE_Sign1 with the following layout:

```
signed-coserv = #6.18([
  protected: bytes .cbor signed-coserv-protected-hdr
  unprotected: signed-coserv-unprotected-hdr
  payload: bytes .cbor coserv
  signature: bytes
])
```

The payload MUST be the CBOR-encoded CoSERV.

(Artwork only available as CDDL: see https://www.ietf.org/archive/id/draft-ietf-rats-coserv-05.html)

The protected header MUST include the signature algorithm identifier. The protected header MUST include either the content type application/coserv+cbor or the CoAP Content-Format TBD1. Other header parameters MAY be added to the header buckets, for example a kid that identifies the signing key.

5. Examples

5.1. Query Data Examples

This section provides some illustrative examples of valid CoSERV query objects.

The following example shows a query for Reference Values scoped by a single class. The artifact-type is set to 2 (reference-values), indicating a query for Reference Values. The profile is given the example value of tag:example.com,2025:cc-platform#1.0.0. Finally, the environment-selector uses the key 0 to select for class, and the value contains a single entry with illustrative settings for the identifier, vendor and model.

```
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query /   1: {
    / artifact-type /         0: 2, / reference-values /
    / environment-selector /  1: {
      / class / 0: [ [
        {
          / class-id /  0: 560(h'00112233'),  / tagged-bytes /
          / vendor /    1: "Example Vendor",
          / model /     2: "Example Model"
        }
      ] ]
    },
    / result-type / 2: 1 / source-artifacts /
  }
}
```

The next example is similar, but adds a second entry to the set of classes in the environment-map, showing how multiple classes can be queried at the same time.

```
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query /   1: {
    / artifact-type /         0: 2, / reference-values /
    / environment-selector /  1: {
      / class / 0: [
        [ {
          / class-id /  0: 560(h'8999786556'),  / tagged-bytes /
          / vendor /    1: "Example Vendor",
          / model /     2: "Example Model"
        } ],
        [ {
          / class-id /  0:
            37(h'31FB5ABF023E4992AA4E95F9C1503BFA')  / UUID /
        } ]
      ]
    },
    / result-type / 2: 2 / both collected and source artifacts /
  }
}
```

The following example shows a query for Reference Values scoped by instance. Again, the artifact-type is set to 2, and profile is given a demonstration value. The environment-selector now uses the key 1 to select for instances, and the value contains two entries with example instance identifiers.

```
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query /   1: {
    / artifact-type / 0: 2, / reference-values /
    / environment-selector /  1: {
      / instance / 1: [
        [ 550(h'02DEADBEEFDEAD') ], / UEID /
        [ 560(h'8999786556') ]      / tagged-bytes /
      ]
    },
    / result-type / 2: 0 / collected artifacts /
  }
}
```

5.2. Result Data Examples

This section provides some illustrative examples of valid CoSERV queries with their corresponding result sets.

In this next example, the query is a reference value query based on class.

The top-level structure is a map with three entries: profile (codepoint 0), query (codepoint 1) and results (codepoint 2).

The profile and query structures are the same as in the previous examples. The result structure is a map with two entries: expiry (codepoint 10) and rvq (codepoint 0). The rvq (reference value quad) entry comprises the asserting authority and the asserted triples. A single reference-value triple is shown in this example. Its environment-map, as expected, is the same as the environment-map that was supplied in the query. The rest of the structure is the measurement-map as defined in CoRIM [I-D.ietf-rats-corim].

```
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query / 1: {
    0: 2,
    1: {
      0: [ [
        {
          0: 560(h'8999786556')
        }
      ] ]
    },
    2: 0
  },
  / results / 2: {
    0: [
      {
        1: [ 560(h'abcdef') ],
        2: [
          {
            0: {
              0: 560(h'8999786556')
            }
          },
          [
            {
              0: 37(h'31FB5ABF023E4992AA4E95F9C1503BFA'),
              1: {
                / version / 0: {
                  0: "1.2.3",
                  1: 16384
                },
                / svn / 1: 553(2)
              }
            }
          ]
        ]
      }
    ],
    10: 0("2030-12-13T18:30:02Z")
  }
}
```

The following example is for a query that requested the results be provided in the "source artifacts" format. This means one or more original signed manifests containing information that satisfies the query criteria.

Compared with the previous example, the rvq entry is empty, while the source-artifacts (codepoint 11) contain two CMW records [I-D.ietf-rats-msg-wrap], each of which contains a (made up) manifest with the type "application/vnd.example.refvals".

```
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query /   1: {
    / artifact-type /         0: 2, / reference-values /
    / environment-selector /  1: {
      / class / 0: [ [
        {
          / class-id /  0: 560(h'00112233'),  / tagged-bytes /
          / vendor /    1: "Example Vendor",
          / model /     2: "Example Model"
        }
      ] ]
    },
    / result-type / 2: 1 / source-artifacts /
  },
  / results / 2: {
    / rvq / 0: [ ],
    / expiry / 10: 0("2030-12-13T18:30:02Z"),
    / source artifacts / 11: [
      [ "application/vnd.example.refvals", h'afaeadac' ],
      [ "application/vnd.example.refvals", h'adacabaa' ]
    ]
  }
}
```

6. API Bindings

This section sets out the ways in which CoSERV queries and responses can be exchanged between software components and services using APIs. The CoSERV data format itself is agnostic of any particular API model or transport. The API bindings provided here are intended to complement the data format. They will allow implementations to build the complete functionality of a CoSERV producer or consumer, in a way that is well-suited to any transport or interaction model that is needed.

It is intended that these API definitions carry minimal additional semantics, since these are largely the preserve of the CoSERV query language itself. The API definitions are merely vehicles for the exchange of CoSERV queries and responses. Their purpose is to facilitate standard interactions that make the most effective use of available transports and protocols.

The only API binding that is specified in this document is a request-response protocol that uses HTTP for transport. This is a simple pattern, and likely to be a commonly occurring one for a variety of use cases. Future specifications may define other API bindings. Such future bindings may introduce further HTTP-based protocols. Alternatively, they may define protocols for use with other transports, such as CoAP [RFC7252].

6.1. Request Response over HTTP

This section defines and mandates the API endpoint behaviours for CoSERV request-response transactions over HTTP. Implementations MUST provide all parts of the API as specified in this section. The API is a simple protocol for the execution of CoSERV queries. It takes a single CoSERV query as input, and produces a corresponding single CoSERV result set as the output. It is a RESTful API because the CoSERV query serves as a unique and stable identifier of the target resource, where that resource is the set of artifacts being selected for by the query. The encoding rules for CoSERV are deterministic as set out in Section 4.5. This means that any given CoSERV query will always encode to the same sequence of bytes. The Base64Url encoding (Section 2 of [RFC7515]) of the byte sequence becomes the rightmost path segment of the URI used to identify the target resource. The HTTP GET verb is then used with this URI to execute the query. Further details are provided in the subsections below.

Authentication is out of scope for this document. Implementations MAY authenticate clients, for example for authorization or for preventing denial of service attacks.

6.1.1. Discovery

Clients discover CoSERV HTTP API endpoints by means of a well-known URI that is formed using the /.well-known/ path prefix as defined in [RFC8615]. This URI supplies a single discovery document that clients can use to locate the URIs of other API endpoints, in addition to finding out other relevant information about the configuration and capabilities of the service.

Implementations that provide CoSERV HTTP API endpoints MUST also provide the discovery endpoint at the path /.well-known/coserv-configuration. This endpoint MUST be available via an HTTP GET method with no additional query parameters.

The response body can be formatted using either JSON or CBOR, governed by standard HTTP content-type negotiation. The media types defined for this purpose are application/coserv-discovery+json (for JSON-formatted documents) or application/coserv-discovery+cbor (for

CBOR-formatted documents). If the client presents any media type other than these two options in its HTTP Accept header, the implementation SHOULD respond with an HTTP 406 (Not Acceptable) status code. If the client presents one of the two valid media types, then the implementation MUST respond with an HTTP 200 (OK) status code, unless it is prevented from doing so by an error condition beyond the scope of this specification. When the 200 (OK) status code is returned, the response body MUST contain exactly one discovery document in the requested format (JSON or CBOR). The contents of the discovery document MUST conform to the CDDL data model given below, which is common to both the JSON and CBOR encodings.

```
;# import rfc9711 as eat
;# import cmw-autogen as cmw
;# import rfc9052 as cose
;# import jwk-autogen as jwk

coserv-well-known-info = {
  version-label => version,
  capabilities-label => [ + capability ],
  api-endpoints-label => { + tstr => tstr },
  ? result-verification-key-label => eat.JC<jwk.JWK_Set, cose.COSE_KeySet>
}

version-label = eat.JC<"version", 1>
capabilities-label = eat.JC<"capabilities", 2>
api-endpoints-label = eat.JC<"api-endpoints", 3>
result-verification-key-label = eat.JC<"result-verification-key", 4>

version = tstr

capability = {
  media-type-label => cmw.media-type,
  artifact-support-label => artifact-support
}

media-type-label = eat.JC<"media-type", 1>
artifact-support-label = eat.JC<"artifact-support", 2>

non-empty-array<M> = (M) .and ([ + any ])
artifact-support = non-empty-array<[ ? "source", ? "collected" ]>
```

6.1.1.1. Discovery Document Contents

This section defines how to populate and interpret the data fields in the discovery document.

The collated CDDL is in Appendix A.2.

6.1.1.1.1. Version

The version field is denoted by the label "version" in JSON documents and by the codepoint 1 in CBOR documents. It is a Semantic Versioning (semver) string, which denotes the version and patch level of the service that is providing the API endpoints described by the document. The semver string MUST conform to the ABNF defined in [SEMVER]. Version numbers and patch levels are otherwise implementation-defined.

6.1.1.1.2. Capabilities

The capabilities field is denoted by the label "capabilities" in JSON documents and by the codepoint 2 in CBOR documents. This field allows clients to discover the profiled variants of CoSERV for which the service implementation can satisfy queries and provide artifacts. This field is structured as an array, which allows for service implementations that support more than one profile. Each supported profile is indicated according to its parameterized media type, along with the categories of artifact that can be provided for the profile. The artifact categories are source and collected, as described in Section 3.2. Each profile is paired with a non-empty set of artifact categories, allowing the service implementation to indicate whether it supports the retrieval of source artifacts, collected artifacts, or both. This pairing caters for situations where the service implementation might support different combinations of artifact category for different profiles.

6.1.1.1.3. API Endpoints

The API endpoints field is denoted by the label "api-endpoints" in JSON documents and by the codepoint 3 in CBOR documents. This field allows clients to derive the correct URL for making HTTP API calls. The field is a map whose keys are the symbolic names of the APIs, and whose values are the URL path for the API endpoint.

The symbolic name CoSERVRequestResponse is defined for services that offer the transactional API described in Section 6.1.2. Service implementations that offer this API MUST include a key with this name in the endpoints map field, and the corresponding endpoint URL path MUST end with /{query}. This allows the consumer to form a valid CoSERV query URI using variable expansion as per [RFC6570], replacing the {query} variable with the Base64Url-encoded CoSERV query object. There MUST NOT be any other variables that require substitution.

6.1.1.1.4. Result Verification Key

The result verification key is denoted by the label "result-verification-key" in JSON documents and by the codepoint 4 in CBOR documents. This field provides one or more public keys that can be used for the cryptographic verification of CoSERV query results that are returned by the service implementation. In JSON-formatted discovery documents, each key is a JSON Web Key (JWK) as defined in [RFC7517]. In CBOR-formatted discovery documents, each key is a COSE Key as defined in [STD96].

This field is optional. As described in Section 4.6, there are situations where it is permissible for CoSERV result sets to be unsigned, namely when they are transacted over an end-to-end secure channel without any untrusted intermediaries. CoSERV service implementations MAY publish discovery documents without result-verification keys in cases where they exclusively produce unsigned CoSERV result sets. Unsigned CoSERV result sets are characterized by use of the application/coserv+cbor media type (as opposed to the application/coserv+cose media type). The supported media types, along with their profile parameters, are published in the capabilities field of the discovery document. If all supported media types are variants of application/coserv+cbor, indicating unsigned results only, then there is no need for the verification key set to be included in the discovery document. If one or more of the supported media types are variants of application/coserv+cose, indicating signed results, then the verification key set MUST be included.

6.1.1.2. Discovery Document CBOR Encoding

When the discovery document is encoded as CBOR, it is exempt from the encoding rules specified in Section 4.5. These encoding rules are designed to ensure that CoSERV queries can be used as canonical and stable identifiers. The discovery document is an independent structure, and not part of the CoSERV data model itself. Therefore, these encoding rules do not apply.

6.1.1.3. Discovery Document Examples

In the following examples, the contents of bodies are informative examples only.

Example HTTP request for retrieving the discovery document in JSON format:

```
GET /.well-known/coserv-configuration HTTP/1.1
Host: endorsements-distributor.example
Accept: application/coserv-discovery+json
```

Corresponding HTTP response:

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

Body (JSON)

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  "version": "1.2.3-beta",
  "capabilities": [
    {
      "media-type": "application/coserv+cose; profile=\"tag:vendor.\
                                          com,2025:cc_platform#1.0.0\"",
      "artifact-support": [
        "source",
        "collected"
      ]
    }
  ],
  "api-endpoints": {
    "CoSERVRequestResponse": "/endorsement-distribution/v1/coserv/{\
                                                              query}"
  },
  "result-verification-key": [
    {
      "alg": "ES256",
      "crv": "P-256",
      "kty": "EC",
      "x": "usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8",
      "y": "IBOL-C3BttVivg-lSreASjpkttcsz-1rb7btKLv8EX4",
      "kid": "key1"
    }
  ]
}
```

Example HTTP request for retrieving the discovery document in CBOR format:

```
GET /.well-known/coserv-configuration HTTP/1.1
Host: endorsements-distributor.example
Accept: application/coserv-discovery+cbor
```

Corresponding HTTP response:

```
HTTP/1.1 200 OK
Content-Type: application/coserv-discovery+cbor
```

Body (in CBOR Extended Diagnostic Notation (EDN))

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  / version / 1: "1.2.3-beta",
  / capabilities / 2: [
    {
      / media-type / 1: "application/coserv+cose; profile=\"tag:\
                                 vendor.com,2025:cc_platform#1.0.0\"",
      / artifact-support / 2: [
        "source",
        "collected"
      ]
    }
  ],
  / api-endpoints / 3: {
    "CoSERVRequestResponse": "/endorsement-distribution/v1/coserv/{\
                                                              query}"
  },
  / result-verification-key / 4: [
    {
      / kty / 1: 2,
      / alg / 3: -7,
      / crv / -1: 1,
      / x / -2: h'1A2B3C4D',
      / y / -3: h'5E6F7A8B',
      / kid / 2: h'ABCDEF1234'
    }
  ]
}
```

6.1.2. Execute Query

This endpoint executes a single CoSERV query and returns a CoSERV result set.

The HTTP method is GET.

The URL path is formed of the discovered coserv endpoint (as set out in Section 6.1.1), where the {query} template variable is substituted with the CoSERV query to be executed, which is represented as a Base64Url encoding of the query's serialized CBOR byte sequence.

There are no additional URL query parameters.

Clients MUST set the HTTP Accept header to a suitably-profiled application/coserv+cose or application/coserv+cbor media type.

Endpoint implementations MUST respond with an HTTP status code and response body according to one of the subheadings below.

6.1.2.1. Responses

6.1.2.1.1. Successful Transaction (200)

This response indicates that the CoSERV query was executed successfully.

Example HTTP request:

```
# NOTE: '\' line wrapping per RFC 8792

GET /coserv/ogB4I3R... HTTP/1.1
Host: endorsements-distributor.example
Accept: application/coserv+cose; \
        profile="tag:vendor.com,2025:cc_platform#1.0.0"
```

Example HTTP response:

```
# NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/coserv+cose; \
              profile="tag:vendor.com,2025:cc_platform#1.0.0"
```

Body (in CBOR Extended Diagnostic Notation (EDN))

```
/ signed-coserv / 18([
  / protected / << {
    / alg / 1: -7 / ECDSA 256 /,
    / cty / 2 : "application/coserv+cbor"
  } >>,
  / unprotected / {},
  / payload / <<
{
  / profile / 0: "tag:example.com,2025:cc-platform#1.0.0",
  / query /   1: {
    / artifact-type /         0: 2, / reference-values /
    / environment-selector /  1: {
      / class / 0: [ [
        {
```

```
            / class-id /  0: 560(h'00112233'),  / tagged-bytes /
            / vendor /    1: "Example Vendor",
            / model /     2: "Example Model"
          }
        ] ]
      },
      / result-type / 2: 0 / collected-artifacts /
    },
    / results / 2: {
      / rvq / 0: [
        {
          / authorities / 1: [ 560(h'abcdef') ],
          / reference-triple / 2: [
            / environment-map / {
              / class / 0: {
                / class-id /  0: 560(h'00112233'),
                / vendor /    1: "Example Vendor",
                / model /     2: "Example Model"
              }
            },
            [
              / measurement-map / {
                / mval / 1: {
                  / digests / 2: [
                    [ 1, h'aa' ],
                    [ 2, h'bb' ]
                  ],
                  / name / 11: "Component A"
                }
              },
              / measurement-map / {
                / mval / 1: {
                  / digests / 2: [
                    [ 1, h'cc' ],
                    [ 2, h'dd' ]
                  ],
                  / name / 11: "Component B"
                }
              }
            ]
          ]
        }
      ],
      / expiry / 10: 0("2030-12-13T18:30:02Z")
    }
  }
    >>,
    / signature / h'face5190'
```

```
   ])
```

6.1.2.1.2. Failure to Validate Query (400)

This response indicates that the supplied query is badly formed.

Example HTTP request:

```
# NOTE: '\' line wrapping per RFC 8792

GET /coserv/badquery... HTTP/1.1
Host: endorsements-distributor.example
Accept: application/coserv+cose; \
        profile="tag:vendor.com,2025:cc_platform#1.0.0"
```

Example HTTP response:

```
# NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 400 Bad Request
Content-Type: application/concise-problem-details+cbor
```

Body (in CBOR Extended Diagnostic Notation (EDN))

```
{
  / title /  -1: "Query validation failed",
  / detail / -2: "The query payload is not in CBOR format"
}
```

6.1.2.1.3. Failure to Negotiate Profile (406)

This response indicates that the client has specified a CoSERV profile that is not understood or serviceable by the receiving endpoint implementation.

Example HTTP request:

```
# NOTE: '\' line wrapping per RFC 8792

GET /coserv/ogB4I3R... HTTP/1.1
Host: endorsements-distributor.example
Accept: application/coserv+cose; \
        profile="tag:vendor.com,2025:cc_platform#2.0.0"
```

Example HTTP response:

```
# NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 406 Not Acceptable
Content-Type: application/concise-problem-details+cbor

Body (in CBOR Extended Diagnostic Notation (EDN))

{
  / title /  -1: "Unsupported profile",
  / detail / -2: "Profile tag:vendor.com,2025:cc_platform#2.0.0 \
                  not supported",
}
```

6.1.3. Caching

In practical usage, the artifacts transacted via CoSERV queries (such as trust anchors and reference values) may change significantly less often than they are used. For example, a Verifier needs to use the artifacts whenever it needs to verify or appraise Evidence from an Attester. This might be a very frequent operation, for which a low latency is desirable. By contrast, the artifacts themselves would vary only as a consequence of impactful changes to the Attester's desired state or environment. One example of such an impactful change might be the roll-out of a firmware update, which would result in a new reference value for the impacted firmware component(s). Such changes would tend to be relatively infrequent. The caching of CoSERV artifacts is therefore beneficial for overall system performance.

CoSERV is designed to facilitate both client-side and server-side caching by use of the standard HTTP caching mechanisms specified in [STD98]. This includes use of the HTTP Cache-Control header and its associated directives. It also includes the use of entity-tags (ETags). The following features of CoSERV and its HTTP binding are specifically designed to favor caching implementations:

* CoSERV queries form stable URL paths. As specified in Section 4.5, any given CoSERV query will always serialize to the same fixed sequence of bytes. This allows queries to be used as canonical and stable resource identifiers, which in turn allows them to function effectively as cache keys.

* The result set is cryptographically bound to the query. As specified in Section 4.6, the origin server is required to return a signed response that combines the result set with the client's original query, in any deployment where untrusted intermediaries might exist. This means that the client can always verify the integrity of the result on an end-to-end basis, even in the presence of caching infrastructure.

* The use of safe HTTP methods. CoSERV queries are executed as read-only operations using HTTP GET. The execution of a query does not modify any state on the server, which creates more opportunities for the re-use of cached results.

6.1.3.1. HTTP Caching and Result Set Expiry

CoSERV's data model includes a mandatory expiration timestamp on every result set. This is an authoritative marker of the point in time at which the entire result set becomes invalid, and the query must be re-executed to obtain fresh results. This timestamp is established by the origin server.

In the presence of HTTP caching infrastructure, the origin server MUST NOT set HTTP cache directives (e.g. Cache-Control: max-age, Expires) such that the freshness lifetime of the HTTP response exceeds the result set expiry timestamp contained within the CoSERV result set payload.

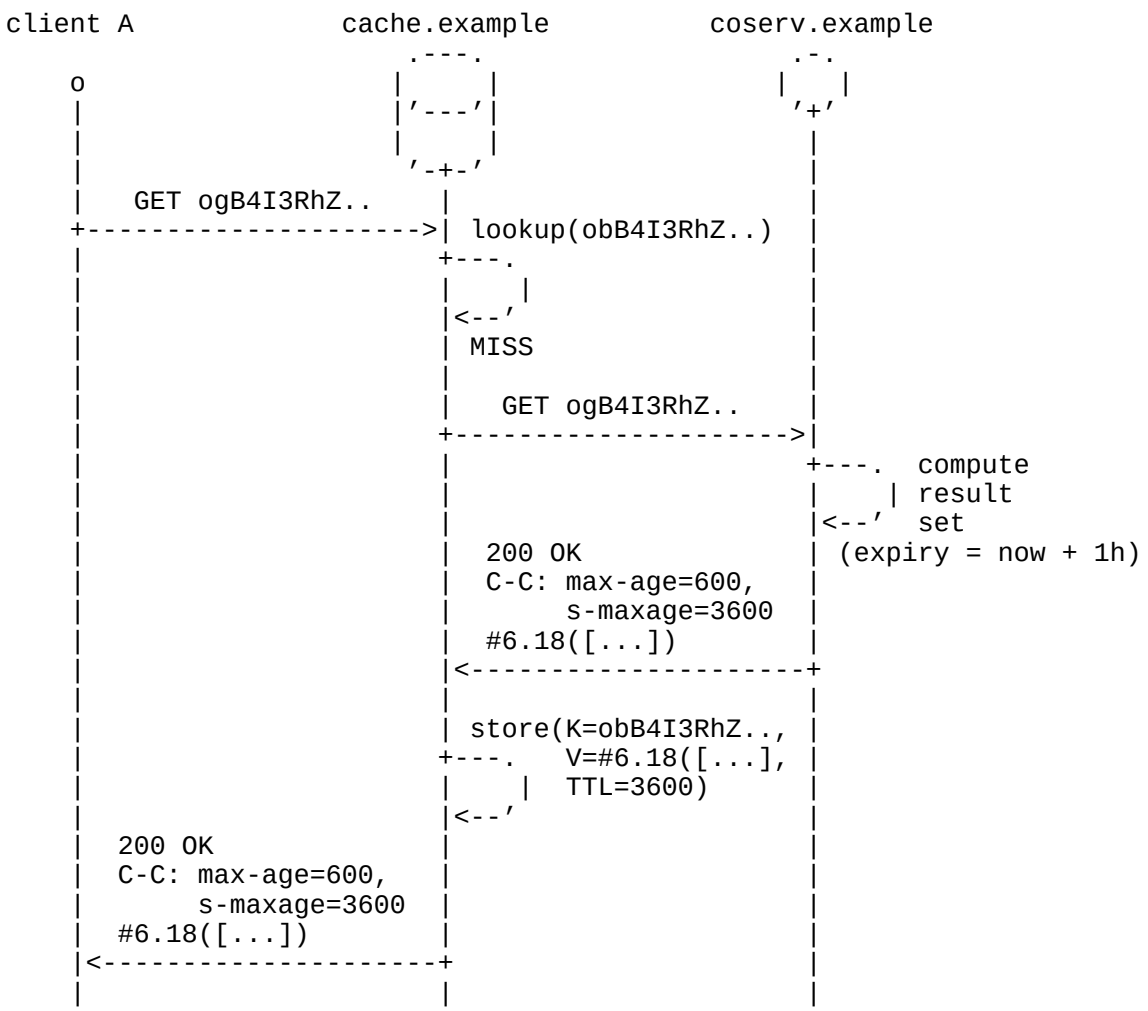
6.1.3.2. Example HTTP Messages with Caching

This section illustrates a caching scenario.

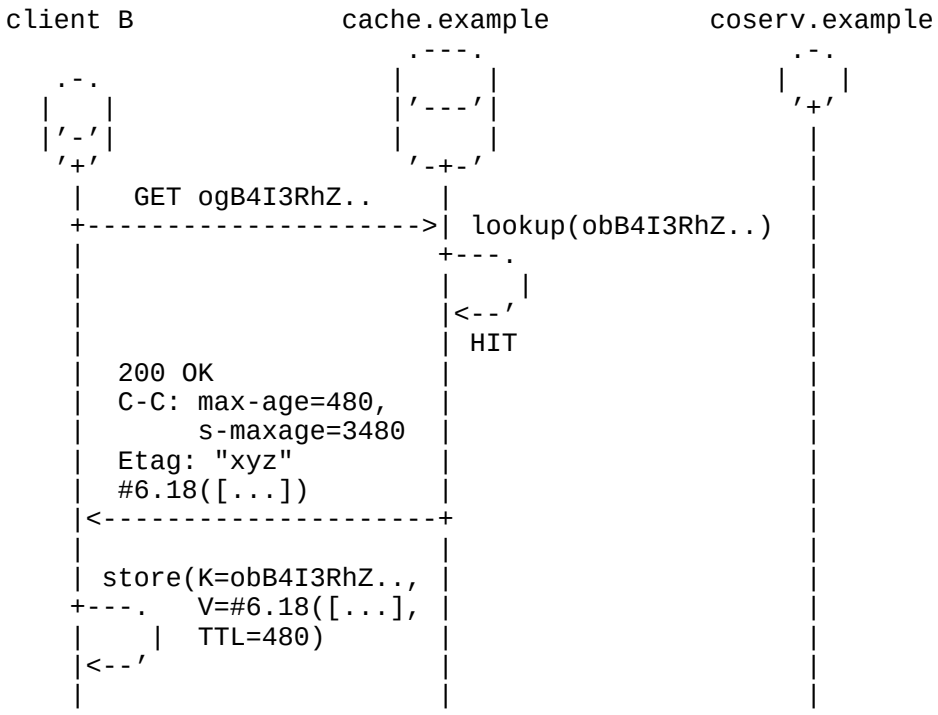
In this example, the CoSERV HTTP API server endpoint is hosted by an HTTP origin (coserv.example), while a reverse proxy (cache.example) operates a public cache in front of the origin.

Client A sends a request using a specific CoSERV query. As the reverse proxy has a "cache miss" for the resource, it forwards the request to the origin. The origin then constructs the response and returns it to the proxy. The response includes cache-control headers that are compatible with the time-to-live associated with the computed result set. For the purposes of this example, the HTTP response max-age has been set to 10 minutes and the s-maxage to 1 hour. This means that the origin allows intermediaries (e.g., its CDN) to cache this resource for longer than the client. The result is different caching behaviours between clients and intermediaries, which reduces the load on the origin by enabling CDNs to cache content for longer, while ensuring that clients receive fresher content. Before forwarding it to the client, the proxy stores the response in its cache using the request URI as the cache key alongside the entry’s time-to-live value.

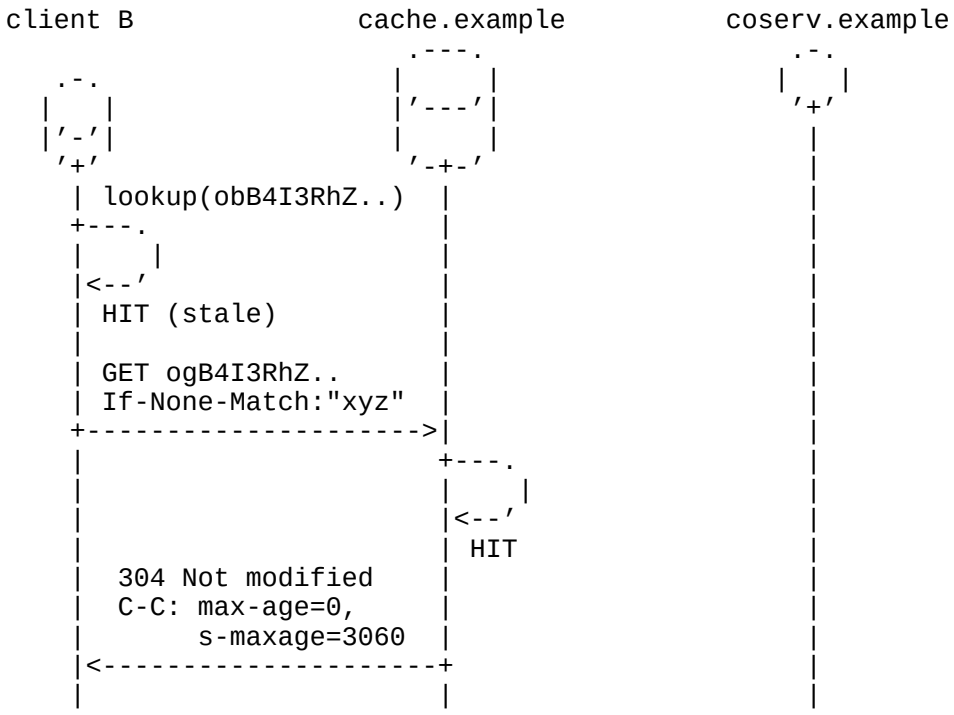
| This "differential caching" strategy could be useful if the origin and its CDN have control plane APIs that the origin owner can use to instruct the CDN operator to purge certain cached entries [RFC8007]. For instance, in CoSERV, this feature could be used in case of an unexpected revocation.



At a later point, after 2 minutes, a different client B makes the same request. This time, the request generates a "cache hit" event on the proxy. The response is therefore served from the public cache, bypassing the origin. This reduces the load on the origin, where computing the result set is generally costly, as well as reducing the overall latency of the transaction. Client B operates a local cache, where it stores a copy of the response.



After 9 more minutes, B is instructed to make the same request again. The request generates a "cache hit" event on the local cache. However, the cached resource is become stale and needs to be revalidated. Therefore, B sends a conditional request to the proxy. The request generates a "cache hit" event on the proxy where the resource is still fresh due to the differential caching behaviour dictated by the original response from the origin. The proxy returns a 304 (Not modified) status code, which instructs the client to reuse its local copy of the response.



7. Implementation Status

// RFC Editor: please remove this section prior to publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

7.1. Veraison

Responsible Organisation: Veraison (open source project within the Confidential Computing Consortium).

Location: https://github.com/veraison

Description: Veraison provides components that can be used to build a Verifier, and also exemplifies adjacent RATS roles such as the Relying Party. There is an active effort to extend Veraison so that it can act in the capacity of an Endorser or Reference Value Provider, showing how CoSERV can be used as a query language for such services. This includes library code to assist with the creation, parsing and manipulation of CoSERV queries.

Level of Maturity: This is a proof-of-concept prototype implementation.

License: Apache-2.0.

Coverage: This implementation covers all aspects of the CoSERV query language.

Contact: Thomas Fossati, Thomas.Fossati@linaro.org

8. Security Considerations

CoSERV implements a conveyance protocol for specific categories of Conceptual Message in [RFC9334], namely Endorsements and Reference Values. Consequently, it is used only between the Endorser and Verifier roles, or between the Reference Value Provider and Verifier roles of the RATS architecture. The relevant security considerations are therefore the ones associated with those roles and their interactions.

Certain security characteristics are desirable for interactions between the Verifier and the Endorser or Reference Value Provider. However, these characteristics would be the province of the specific implementations of these roles, and of the transport protocols in between them. They would not be the province of the CoSERV data object itself. Examples of such desirable characteristics might be:

* The Endorser or Reference Value Provider is available to the Verifier when needed.

* The Verifier is authorised to query data from the Endorser or Reference Value Provider.

* Queries cannot be intercepted or undetectably modified by an entity that is interposed between the Verifier and the Endorser or Reference Value Provider.

8.1. In Relation to CoRIM

CoSERV's data model inherits heavily from that of [I-D.ietf-rats-corim]. CoSERV responses can contain one or more complete CoRIM artifacts. They can also contain aggregated views that are composed of multiple CoRIM fragments. The security and privacy considerations set out in Section 11 of [I-D.ietf-rats-corim] therefore apply equally to CoSERV.

8.2. Forming Native Database Queries from CoSERV

Implementations should take care when transforming CoSERV queries into native query types that are compatible with their underlying storage technology (such as SQL queries). There is a risk of injection attacks arising from poorly-formed or maliciously-formed CoSERV queries. Implementations must ensure that suitable sanitization procedures are in place when performing such translations.

8.3. Aggregators

Aggregation (see Section 2) is the process of combining artifacts from multiple Endorser or Reference Value Provider sources, which is a necessary step in some supply chains. CoSERV supports aggregation explicitly at the protocol level, but is agnostic with regards to how (or whether) such support is used. However, there are specific security considerations for deployments that make use of aggregators.

When used, aggregators feed Endorsements and Reference Values to the Verifier (possibly via further aggregators). This means that they act in the Endorser and/or Reference Value Provider roles of RATS, both of which are trusted roles. Aggregators are therefore trusted components. Further, since the purpose of an aggregator is to provide a consolidated point of consumption for the Verifier, there is a risk of its becoming a single point of failure or vulnerability. An aggregator that is unavailable, malfunctioning, or malicious, has the potential to impact the security of the overall deployment. For example, a malicious aggregator might attempt to impersonate or otherwise subvert the authority of other actors in the supply chain, such as hardware or firmware vendors.

The intent of CoSERV is for aggregators to provide an optional convenience layer for the Verifier, rather than to be a subversive authority. The design features of CoSERV can be used alongside judicious deployment practices to mitigate the risks. An informative and non-exhaustive list of mitigations follows:-

* *Use independent chains of trust.* It is established above that aggregators are trusted components. This does not mean that they necessarily become a sole or replacement trust authority. CoSERV’s aggregation model allows for deference to other authorities that exist in the supply chain. This is true even when an aggregator is acting in the Endorser role. A hardware Endorsement, for example, might be delivered to the Verifier via an aggregator (along with multiple other artifacts, such as Reference Values). But the authority of that Endorsement can still be chained back to the hardware provider, and this authority can be checked by the Verifier using an independent trust anchor.

* *Inspect the authority delegation chains.* The "quads" feature of the CoSERV data model provides explicit tracking of supply chain sources. Each inner CoMID triple of an aggregated CoSERV response is annotated with an authority delegation chain. This is a sequence of delegated trust authorities, each of which might be either a further upstream aggregator or a primary supply chain actor. This information allows the consumer (Verifier) to inspect the provenance of each aggregated result, which can be checked against its own independent record of trustworthy sources.

* *Use source artifacts.* CoSERV's aggregation model supports the pass-through of artifacts from upstream supply-chain actors, known as "source artifacts" in the data model. Source artifacts are passed verbatim in CoSERV, meaning that they retain any original signatures. This provides another means of checking the provenance and integrity of such artifacts, independently of any signature that is applied to the CoSERV result by the aggregator (see Section 4.6).

* *Mutual trust between aggregators and primary supply chain actors.* The default assumption of CoSERV is that trust flows in one direction only: CoSERV consumers trust CoSERV producers, but not the reverse. When a Verifier sends a CoSERV query to an aggregator, the Verifier is trusting that aggregator, but not the reverse. Likewise, when an aggregator calls a primary supply chain source (whether using CoSERV or some proprietary mechanism), then the aggregator is trusting that primary supply chain source, but not the reverse. Indeed, a primary supply chain source might not even be aware of the existence of any aggregator that is consuming artifacts from it, let alone place any trust in such an aggregator. However, it is perfectly valid for a deployment to deviate from this baseline, provided that the suitable technical and contractual enablers are put in place. There could exist an aggregator that is trusted - and vouched for - by the primary supply chain actor(s) from which it consumes. Supply chain actors might even actively provision their artifacts into the aggregator for onward distribution. The clients of such an aggregator might then be able to make more use of the "shallow trust" model described in Section 2, with a greater reliance on collected artifacts rather than source artifacts.

9. Privacy Considerations

A CoSERV query can potentially contain privacy-sensitive information. Specifically, the environment-selector field of the query may reference identifiable Attester instances in some cases. This concern naturally also extends to the data objects that might be returned to the consumer in response to the query, although the specifications of such data objects are beyond the scope of this document. Implementations should ensure that appropriate attention is paid to this. Suitable mitigations include the following:

* The use of authenticated secure channels between the producers and the consumers of CoSERV queries and returned artifacts.

* Collating Attester instances into anonymity groups, and referencing the groups rather than the individual instances.

9.1. Aggregators

Aggregators (as described in Section 2) can pose a specific privacy risk. This is because they necessarily correlate inputs from multiple supply-chain actors, and can observe repeated CoSERV queries over time. In doing so, an aggregator might be able to infer details about the composition of an Attester's hardware, firmware or software components. Such details would not be accessible to individual supply-chain actors implementing the Endorser or Reference Value Provider roles. There is consequently a risk that such inferred details could be misused to create a covert channel.

10. IANA Considerations

// RFC Editor: replace "RFCthis" with the RFC number assigned to this document.

10.1. Media Types Registrations

IANA is requested to add the following media types to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
coserv+cbor	application/coserv+cbor	Section 4 of RFCthis
coserv+cose	application/coserv+cose	Section 4.6 of RFCthis
coserv-discovery+cbor	application/coserv-discovery+cbor	Section 6.1.1 of RFCthis
coserv-discovery+json	application/coserv-discovery+json	Section 6.1.1 of RFCthis

Table 1: CoSERV Media Types

10.1.1. application/coserv+cbor

Type name: application
Subtype name: coserv+cbor
Required parameters: n/a
Optional parameters: "profile" (CoSERV profile in string format. OIDs must use the dotted-decimal notation.)

Encoding considerations: binary (CBOR)
Security considerations: Section 8 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Verifiers, Endorsers, Reference Value Providers
Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/cbor". (No fragment identification syntax is currently defined for "application/cbor".)
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration: no

10.1.2. application/coserv+cose

Type name: application
Subtype name: coserv+cose
Required parameters: n/a (cose-type is explicitly not supported, as it is understood to be "cose-sign1")
Optional parameters: "profile" CoSERV profile in string format. OIDs must use the dotted-decimal notation. Note that the cose-type parameter is explicitly not supported, as it is understood to be "cose-sign1".
Encoding considerations: binary
Security considerations: Section 8 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Verifiers, Endorsers, Reference Value Providers
Fragment identifier considerations: n/a
Person and email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration? no

10.1.3. application/coserv-discovery+cbor

Type name: application
Subtype name: coserv-discovery+cbor
Required parameters: n/a
Optional parameters: n/a
Encoding considerations: binary (CBOR)

Security considerations: Section 8 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Verifiers, Endorsers, Reference Value Providers
Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/cbor". (No fragment identification syntax is currently defined for "application/cbor".)
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration: no

10.1.4. application/coserv-discovery+json

Type name: application
Subtype name: coserv-discovery+json
Required parameters: n/a
Optional parameters: n/a
Encoding considerations: binary (JSON is UTF-8-encoded text)
Security considerations: Section 8 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Verifiers, Endorsers, Reference Value Providers
Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/json". (No fragment identification syntax is currently defined for "application/json".)
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration: no

10.2. CoAP Content-Formats

IANA is requested to register the following Content-Format IDs in the "CoAP Content-Formats" registry, within the "Constrained RESTful Environments (CoRE) Parameters" registry group [IANA.core-parameters]:

Content-Type	Content Coding	ID	Reference
application/coserv+cbor	-	TBD1	Section 4 of RFCthis
application/coserv+cose	-	TBD2	Section 4.6 of RFCthis

Table 2: New CoAP Content Formats

If possible, TBD1 and TBD2 should be assigned in the 256..9999 range.

10.3. Well-Known URI for CoSERV Configuration

IANA is requested to register the following in the "Well-Known URIs" registry [IANA.well-known-uris]:

URI suffix: coserv-configuration

Change controller: IETF

Specification document: RFCthis

Related information: N/A

11. References

11.1. Normative References

[I-D.ietf-rats-corim]
Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-ietf-rats-corim-09, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-corim-09>.

[I-D.ietf-rats-msg-wrap]
Birkholz, H., Smith, N., Fossati, T., Tschofenig, H., and D. Glaze, "RATS Conceptual Messages Wrapper (CMW)", Work in Progress, Internet-Draft, draft-ietf-rats-msg-wrap-23, 11 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-msg-wrap-23>.

[IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types]
IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA.well-known-uris]
IANA, "Well-Known URIs", <https://www.iana.org/assignments/well-known-uris>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[SEMVER] "Semantic Versioning 2.0.0", 2013, <https://semver.org/spec/v2.0.0.html>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[STD96] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

11.2. Informative References

[I-D.ietf-iotops-mud-rats] Birkholz, H., Richardson, M., and P. C. Liu, "MUD-Based RATS Resources Discovery", Work in Progress, Internet-Draft, draft-ietf-iotops-mud-rats-02, 28 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-mud-rats-02>.

[I-D.ietf-rats-endorsements] Thaler, D., Birkholz, H., and T. Fossati, "RATS Endorsements", Work in Progress, Internet-Draft, draft-ietf-rats-endorsements-09, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-endorsements-09>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/rfc/rfc8007>.

[RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

[STD98] Internet Standard 98, <https://www.rfc-editor.org/info/std98>. At the time of writing, this STD comprises the following:

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

Appendix A. Collated CDDL

A.1. CoSERV Data Model

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

signed-coserv = #6.18([
    protected: bytes .cbor signed-coserv-protected-hdr,
    unprotected: signed-coserv-unprotected-hdr,
    payload: bytes .cbor coserv,
    signature: bytes,
])
signed-coserv-protected-hdr = {
  1 => int,
  2 => "application/coserv+cbor" / 10000,
  * cose.label => cose.values,
}
signed-coserv-unprotected-hdr = {* cose.label => cose.values}
cose.label = int / text
```

```
   cose.values = any
   coserv = {
     &(profile: 0) => profile,
     &(query: 1) => query,
     ? &(results: 2) => results,
   }
   profile = comid.oid-type / ~uri
   query = {
     &(artifact-type: 0) => artifact-type,
     &(environment-selector: 1) => environment-selector-map,
     &(result-type: 2) => result-type,
   }
   artifact-type = &(endorsed-values: 0) / &(trust-anchors: 1) / &(\
                                                   reference-values: 2)
   result-type = &(collected-artifacts: 0) / &(source-artifacts: 1) / &\
                                                                (both: 2)
   results = {
     result-set,
     &(expiry: 10) => tdate,
     ? &(source-artifacts: 11) => [+ cmw.cbor-record],
   }
   result-set //= (reference-values // endorsed-values // trust-anchors)
   refval-quad = {
     &(authorities: 1) => [+ comid.$crypto-key-type-choice],
     &(rv-triple: 2) => comid.reference-triple-record,
   }
   reference-values = (&(rvq: 0) => [* refval-quad])
   endval-quad = {
     &(authorities: 1) => [+ comid.$crypto-key-type-choice],
     &(ev-triple: 2) => comid.endorsed-triple-record,
   }
   cond-endval-quad = {
     &(authorities: 1) => [+ comid.$crypto-key-type-choice],
     &(ce-triple: 2) => comid.conditional-endorsement-triple-record,
   }
   endorsed-values = (
     &(evq: 1) => [* endval-quad],
     &(ceq: 2) => [* cond-endval-quad],
     )
   ak-quad = {
     &(authorities: 1) => [+ comid.$crypto-key-type-choice],
     &(ak-triple: 2) => comid.attest-key-triple-record,
   }
   cots-stmt = {
     &(authorities: 1) => [+ comid.$crypto-key-type-choice],
     &(cots: 2) => cots,
   }
   trust-anchors = (
```

```
    &(akq: 3) => [* ak-quad],
    &(tas: 4) => [* cots-stmt],
    )
  cots = "TODO COTS"
  environment-selector-map = {selector}
  stateful-class = [
    class: comid.class-map,
    ? measurements: [+ comid.measurement-map],
  ]
  selector //= (&(class: 0) => [+ stateful-class] // &(instance: 1) =\
          > [+ stateful-instance] // &(group: 2) => [+ stateful-group])
  stateful-instance = [
    instance: comid.$instance-id-type-choice,
    ? measurements: [+ comid.measurement-map],
  ]
  stateful-group = [
    group: comid.$group-id-type-choice,
    ? measurements: [+ comid.measurement-map],
  ]
  cmw.start = cmw.cmw
  cmw.cmw = cmw.json-cmw / cmw.cbor-cmw
  cmw.json-cmw = cmw.json-record / cmw.json-collection
  cmw.cbor-cmw = cmw.cbor-record / cmw.cbor-collection / cmw.$cbor-tag
  cmw.json-record = [
    type: cmw.media-type,
    value: cmw.base64url-string,
    ? ind: uint .bits cmw.cm-type,
  ]
  cmw.cbor-record = [
    type: cmw.coap-content-format-type / cmw.media-type,
    value: bytes,
    ? ind: uint .bits cmw.cm-type,
  ]
  cmw.tag-cm-cbor<tn, fmt> = #6.<tn>(bytes .cbor fmt)
  cmw.tag-cm-data<tn> = #6.<tn>(bytes)
  cmw.json-collection = {
    ? __cmwc_t: ~uri / cmw.oid,
    + &(label: text) => cmw.json-cmw,
  }
  cmw.cbor-collection = {
    ? __cmwc_t: ~uri / cmw.oid,
    + &(label: int / text) => cmw.cbor-cmw,
  }
  cmw.media-type = text .abnf ("Content-Type" .cat cmw.Content-Type-\
                                                                  ABNF)
  cmw.base64url-string = text .regexp "[A-Za-z0-9_-]+"
  cmw.coap-content-format-type = uint .size 2
  cmw.oid = text .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"
```

```
   cmw.cm-type = &(
     reference-values: 0,
     endorsements: 1,
     evidence: 2,
     attestation-results: 3,
     appraisal-policy: 4,
   )
   cmw.Content-Type-ABNF = '

   Content-Type   = Media-Type-Name *( *SP ";" *SP parameter )
   parameter      = token "=" ( token / quoted-string )

   token          = 1*tchar
   tchar          = "!" / "#" / "$" / "%" / "&" / "\'" / "*"
                  / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
                  / DIGIT / ALPHA
   quoted-string  = %x22 *( qdtext / quoted-pair ) %x22
   qdtext         = SP / %x21 / %x23-5B / %x5D-7E
   quoted-pair    = "\\" ( SP / VCHAR )

   Media-Type-Name = type-name "/" subtype-name

   type-name = restricted-name
   subtype-name = restricted-name

   restricted-name = restricted-name-first *126restricted-name-chars
   restricted-name-first  = ALPHA / DIGIT
   restricted-name-chars  = ALPHA / DIGIT / "!" / "#" /
                            "$" / "&" / "-" / "^" / "_"
   restricted-name-chars =/ "." ; Characters before first dot always
                                ; specify a facet name
   restricted-name-chars =/ "+" ; Characters after last plus always
                                ; specify a structured syntax suffix

   DIGIT     =  %x30-39           ; 0 - 9
   POS-DIGIT =  %x31-39           ; 1 - 9
   ALPHA     =  %x41-5A / %x61-7A ; A - Z / a - z
   SP        =  %x20
   VCHAR     =  %x21-7E           ; printable ASCII (no SP)
   '
   cmw.$cbor-tag /= cmw.tag-cm-data<1668612070> / cmw.tag-cm-cbor<\
                                          1668612069, cmw.my-evidence>
   cmw.my-evidence = {&(eat_nonce: 10) => bytes .size (8 .. 64)}
   comid.concise-mid-tag = {
     ? &(language: 0) => text,
     &(tag-identity: 1) => comid.tag-identity-map,
     ? &(entities: 2) => [+ comid.comid-entity-map],
     ? &(linked-tags: 3) => [+ comid.linked-tag-map],
```

```
    &(triples: 4) => comid.triples-map,
    * $$concise-mid-tag-extension,
  }
  comid.attest-key-triple-record = [
    environment: comid.environment-map,
    key-list: [+ comid.$crypto-key-type-choice],
    ? conditions: comid.non-empty<{
              ? &(mkey: 0) => comid.$measured-element-type-choice,
              ? &(authorized-by: 1) => [+ comid.$crypto-key-type-\
                                                          choice],
  }>,
  ]
  comid.$class-id-type-choice /= comid.tagged-oid-type / comid.tagged-\
                                      uuid-type / comid.tagged-bytes
  comid.class-map = comid.non-empty<{
      ? &(class-id: 0) => comid.$class-id-type-choice,
      ? &(vendor: 1) => tstr,
      ? &(model: 2) => tstr,
      ? &(layer: 3) => uint,
      ? &(index: 4) => uint,
  }>
  comid.comid-entity-map = comid.entity-map<comid.$comid-role-type-\
                                 choice, $$comid-entity-map-extension>
  comid.$comid-role-type-choice /= &(tag-creator: 0) / &(creator: 1) \
                                                    / &(maintainer: 2)
  comid.conditional-endorsement-series-triple-record = [
    condition: comid.stateful-environment-record,
    series: [+ comid.conditional-series-record],
  ]
  comid.conditional-series-record = [
    selection: [+ comid.measurement-map],
    addition: [+ comid.measurement-map],
  ]
  comid.COSE_Key = {
    1 => tstr / int,
    ? 2 => bstr,
    ? 3 => tstr / int,
    ? 4 => [+ tstr / int],
    ? 5 => bstr,
    * comid.cose-label => comid.cose-value,
  }
  comid.cose-label = int / tstr
  comid.cose-value = any
  comid.coswid-triple-record = [
    comid.environment-map,
    [+ comid.concise-swid-tag-id],
  ]
  comid.concise-swid-tag-id = text / bstr .size 16
```

```
   comid.$crypto-key-type-choice /= comid.tagged-pkix-base64-key-type \
   / comid.tagged-pkix-base64-cert-type / comid.tagged-pkix-base64-cert\
   -path-type / comid.tagged-cose-key-type / comid.tagged-pkix-asn1der-\
   cert-type / comid.tagged-key-thumbprint-type / comid.tagged-cert-\
   thumbprint-type / comid.tagged-cert-path-thumbprint-type / comid.\
                                                          tagged-bytes
   comid.tagged-pkix-base64-key-type = #6.554(tstr)
   comid.tagged-pkix-base64-cert-type = #6.555(tstr)
   comid.tagged-pkix-base64-cert-path-type = #6.556(tstr)
   comid.tagged-key-thumbprint-type = #6.557(comid.digest)
   comid.tagged-cose-key-type = #6.558(comid.COSE_Key)
   comid.tagged-cert-thumbprint-type = #6.559(comid.digest)
   comid.tagged-cert-path-thumbprint-type = #6.561(comid.digest)
   comid.tagged-pkix-asn1der-cert-type = #6.562(bstr)
   comid.domain-dependency-triple-record = [
     comid.domain-type,
     [+ comid.domain-type],
   ]
   comid.domain-membership-triple-record = [
     domain-id: comid.domain-type,
     members: [+ comid.domain-type],
   ]
   comid.conditional-endorsement-triple-record = [
     conditions: [+ comid.stateful-environment-record],
     endorsements: [+ comid.endorsed-triple-record],
   ]
   comid.domain-type = comid.environment-map
   comid.endorsed-triple-record = [
     condition: comid.environment-map,
     endorsement: [+ comid.measurement-map],
   ]
   comid.entity-map<role-type-choice, extension-socket> = {
       &(entity-name: 0) => comid.$entity-name-type-choice,
       ? &(reg-id: 1) => uri,
       &(role: 2) => [+ role-type-choice],
       * extension-socket,
   }
   comid.$entity-name-type-choice /= text
   comid.environment-map = comid.non-empty<{
       ? &(class: 0) => comid.class-map,
       ? &(instance: 1) => comid.$instance-id-type-choice,
       ? &(group: 2) => comid.$group-id-type-choice,
   }>
   comid.flags-map = {
     ? &(is-configured: 0) => bool,
     ? &(is-secure: 1) => bool,
     ? &(is-recovery: 2) => bool,
     ? &(is-debug: 3) => bool,
```

```
    ? &(is-replay-protected: 4) => bool,
    ? &(is-integrity-protected: 5) => bool,
    ? &(is-runtime-meas: 6) => bool,
    ? &(is-immutable: 7) => bool,
    ? &(is-tcb: 8) => bool,
    ? &(is-confidentiality-protected: 9) => bool,
    * $$flags-map-extension,
  }
  comid.$group-id-type-choice /= comid.tagged-uuid-type / comid.tagged\
                                                              -bytes
  comid.identity-triple-record = [
    environment: comid.environment-map,
    key-list: [+ comid.$crypto-key-type-choice],
    ? conditions: comid.non-empty<{
              ? &(mkey: 0) => comid.$measured-element-type-choice,
              ? &(authorized-by: 1) => [+ comid.$crypto-key-type-\
                                                        choice],
  }>,
  ]
  comid.$instance-id-type-choice /= comid.tagged-ueid-type / comid.\
  tagged-uuid-type / comid.tagged-bytes / comid.tagged-pkix-base64-key\
  -type / comid.tagged-pkix-base64-cert-type / comid.tagged-cose-key-\
  type / comid.tagged-key-thumbprint-type / comid.tagged-cert-\
                 thumbprint-type / comid.tagged-pkix-asn1der-cert-type
  comid.ip-addr-type-choice = comid.ip4-addr-type / comid.ip6-addr-type
  comid.ip4-addr-type = bytes .size 4
  comid.ip6-addr-type = bytes .size 16
  comid.int-range-type-choice = int / comid.tagged-int-range
  comid.int-range = [
    min: int / comid.negative-inf,
    max: int / comid.positive-inf,
  ]
  comid.tagged-int-range = #6.564(comid.int-range)
  comid.positive-inf = null
  comid.negative-inf = null
  comid.linked-tag-map = {
    &(linked-tag-id: 0) => comid.$tag-id-type-choice,
    &(tag-rel: 1) => comid.$tag-rel-type-choice,
  }
  comid.mac-addr-type-choice = comid.eui48-addr-type / comid.eui64-\
                                                            addr-type
  comid.eui48-addr-type = bytes .size 6
  comid.eui64-addr-type = bytes .size 8
  comid.$measured-element-type-choice /= comid.tagged-oid-type / comid\
                                      .tagged-uuid-type / uint / tstr
  comid.measurement-map = {
    ? &(mkey: 0) => comid.$measured-element-type-choice,
    &(mval: 1) => comid.measurement-values-map,
```

```
    ? &(authorized-by: 2) => [+ comid.$crypto-key-type-choice],
  }
  comid.measurement-values-map = comid.non-empty<{
      ? &(version: 0) => comid.version-map,
      ? &(svn: 1) => comid.svn-type-choice,
      ? &(digests: 2) => comid.digests-type,
      ? &(flags: 3) => comid.flags-map,
      ? (
                    &(raw-value: 4) => comid.$raw-value-type-choice,
                    ? &(raw-value-mask-DEPRECATED: 5) => comid.raw-\
                                                         value-mask-type,
          ),
      ? &(mac-addr: 6) => comid.mac-addr-type-choice,
      ? &(ip-addr: 7) => comid.ip-addr-type-choice,
      ? &(serial-number: 8) => text,
      ? &(ueid: 9) => comid.ueid-type,
      ? &(uuid: 10) => comid.uuid-type,
      ? &(name: 11) => text,
      ? &(cryptokeys: 13) => [+ comid.$crypto-key-type-choice],
      ? &(integrity-registers: 14) => comid.integrity-registers,
      ? &(int-range: 15) => comid.int-range-type-choice,
      * $$measurement-values-map-extension,
  }>
  comid.non-empty<M> = M .and ({+ any => any})
  comid.oid-type = bytes
  comid.tagged-oid-type = #6.111(comid.oid-type)
  comid.$raw-value-type-choice /= comid.tagged-bytes / comid.tagged-\
                                                       masked-raw-value
  comid.raw-value-mask-type = bytes
  comid.tagged-masked-raw-value = #6.563([
      value: bytes,
      mask: bytes,
  ])
  comid.reference-triple-record = [
    ref-env: comid.environment-map,
    ref-claims: [+ comid.measurement-map],
  ]
  comid.stateful-environment-record = [
    environment: comid.environment-map,
    claims-list: [+ comid.measurement-map],
  ]
  comid.svn-type = uint
  comid.svn = comid.svn-type
  comid.min-svn = comid.svn-type
  comid.tagged-svn = #6.552(comid.svn)
  comid.tagged-min-svn = #6.553(comid.min-svn)
  comid.svn-type-choice = comid.svn / comid.tagged-svn / comid.tagged-\
                                                         min-svn
```

```
   comid.$tag-id-type-choice /= tstr / comid.uuid-type
   comid.tag-identity-map = {
     &(tag-id: 0) => comid.$tag-id-type-choice,
     ? &(tag-version: 1) => comid.tag-version-type,
   }
   comid.$tag-rel-type-choice /= &(supplements: 0) / &(replaces: 1)
   comid.tag-version-type = uint .default 0
   comid.tagged-bytes = #6.560(bytes)
   comid.triples-map = comid.non-empty<{
       ? &(reference-triples: 0) => [+ comid.reference-triple-record],
       ? &(endorsed-triples: 1) => [+ comid.endorsed-triple-record],
       ? &(identity-triples: 2) => [+ comid.identity-triple-record],
       ? &(attest-key-triples: 3) => [+ comid.attest-key-triple-record],
       ? &(dependency-triples: 4) => [+ comid.domain-dependency-triple-\
                                                              record],
       ? &(membership-triples: 5) => [+ comid.domain-membership-triple-\
                                                              record],
       ? &(coswid-triples: 6) => [+ comid.coswid-triple-record],
       ? &(conditional-endorsement-series-triples: 8) => [+ comid.\
                           conditional-endorsement-series-triple-record],
       ? &(conditional-endorsement-triples: 10) => [+ comid.conditional\
                                            -endorsement-triple-record],
       * $$triples-map-extension,
   }>
   comid.ueid-type = bytes .size (7 .. 33)
   comid.tagged-ueid-type = #6.550(comid.ueid-type)
   comid.uuid-type = bytes .size 16
   comid.tagged-uuid-type = #6.37(comid.uuid-type)
   comid.version-map = {
     &(version: 0) => text,
     ? &(version-scheme: 1) => comid.$version-scheme,
   }
   comid.digest = [
     alg: int / text,
     val: bytes,
   ]
   comid.digests-type = [+ comid.digest]
   comid.integrity-register-id-type-choice = uint / text
   comid.integrity-registers = {+ comid.integrity-register-id-type-\
                                          choice => comid.digests-type}
   comid.concise-swid-tag = {
     comid.tag-id => text / bstr .size 16,
     comid.tag-version => integer,
     ? comid.corpus => bool,
     ? comid.patch => bool,
     ? comid.supplemental => bool,
     comid.software-name => text,
     ? comid.software-version => text,
```

```
    ? comid.version-scheme => comid.$version-scheme,
    ? comid.media => text,
    ? comid.software-meta => comid.one-or-more<comid.software-meta-\
                                                          entry>,
    comid.entity => comid.one-or-more<comid.entity-entry>,
    ? comid.link => comid.one-or-more<comid.link-entry>,
    ? comid.payload-or-evidence,
    * $$coswid-extension,
    comid.global-attributes,
  }
  comid.payload-or-evidence //= (comid.payload => comid.payload-entry \
                           // comid.evidence => comid.evidence-entry)
  comid.any-uri = uri
  comid.label = text / int
  comid.$version-scheme /= comid.multipartnumeric / comid.\
  multipartnumeric-suffix / comid.alphanumeric / comid.decimal / comid\
                                                    .semver / int / text
  comid.any-attribute = (comid.label => comid.one-or-more<text> / \
                                              comid.one-or-more<int>)
  comid.one-or-more<T> = T / [2*T]
  comid.global-attributes = (
    ? comid.lang => text,
    * comid.any-attribute,
    )
  comid.hash-entry = [
    hash-alg-id: int,
    hash-value: bytes,
  ]
  comid.entity-entry = {
    comid.entity-name => text,
    ? comid.reg-id => comid.any-uri,
    comid.role => comid.one-or-more<comid.$role>,
    ? comid.thumbprint => comid.hash-entry,
    * $$entity-extension,
    comid.global-attributes,
  }
  comid.$role /= comid.tag-creator / comid.software-creator / comid.\
  aggregator / comid.distributor / comid.licensor / comid.maintainer \
                                                        / int / text
  comid.link-entry = {
    ? comid.artifact => text,
    comid.href => comid.any-uri,
    ? comid.media => text,
    ? comid.ownership => comid.$ownership,
    comid.rel => comid.$rel,
    ? comid.media-type => text,
    ? comid.use => comid.$use,
    * $$link-extension,
```

```
    comid.global-attributes,
  }
  comid.$ownership /= comid.shared / comid.private / comid.abandon / \
                                                          int / text
  comid.$rel /= comid.ancestor / comid.component / comid.feature / \
  comid.installationmedia / comid.packageinstaller / comid.parent / \
  comid.patches / comid.requires / comid.see-also / comid.supersedes \
                          / comid.supplemental / -256 .. 64436 / text
  comid.$use /= comid.optional / comid.required / comid.recommended / \
                                                          int / text
  comid.software-meta-entry = {
    ? comid.activation-status => text,
    ? comid.channel-type => text,
    ? comid.colloquial-version => text,
    ? comid.description => text,
    ? comid.edition => text,
    ? comid.entitlement-data-required => bool,
    ? comid.entitlement-key => text,
    ? comid.generator => text / bstr .size 16,
    ? comid.persistent-id => text,
    ? comid.product => text,
    ? comid.product-family => text,
    ? comid.revision => text,
    ? comid.summary => text,
    ? comid.unspsc-code => text,
    ? comid.unspsc-version => text,
    * $$software-meta-extension,
    comid.global-attributes,
  }
  comid.path-elements-group = (
    ? comid.directory => comid.one-or-more<comid.directory-entry>,
    ? comid.file => comid.one-or-more<comid.file-entry>,
    )
  comid.resource-collection = (
    comid.path-elements-group,
    ? comid.process => comid.one-or-more<comid.process-entry>,
    ? comid.resource => comid.one-or-more<comid.resource-entry>,
    * $$resource-collection-extension,
    )
  comid.file-entry = {
    comid.filesystem-item,
    ? comid.size => uint,
    ? comid.file-version => text,
    ? comid.hash => comid.hash-entry,
    * $$file-extension,
    comid.global-attributes,
  }
  comid.directory-entry = {
```

```
    comid.filesystem-item,
    ? comid.path-elements => {comid.path-elements-group},
    * $$directory-extension,
    comid.global-attributes,
  }
  comid.process-entry = {
    comid.process-name => text,
    ? comid.pid => integer,
    * $$process-extension,
    comid.global-attributes,
  }
  comid.resource-entry = {
    comid.type => text,
    * $$resource-extension,
    comid.global-attributes,
  }
  comid.filesystem-item = (
    ? comid.key => bool,
    ? comid.location => text,
    comid.fs-name => text,
    ? comid.root => text,
    )
  comid.payload-entry = {
    comid.resource-collection,
    * $$payload-extension,
    comid.global-attributes,
  }
  comid.evidence-entry = {
    comid.resource-collection,
    ? comid.date => comid.integer-time,
    ? comid.device-id => text,
    ? comid.location => text,
    * $$evidence-extension,
    comid.global-attributes,
  }
  comid.integer-time = #6.1(int)
  comid.tag-id = 0
  comid.software-name = 1
  comid.entity = 2
  comid.evidence = 3
  comid.link = 4
  comid.software-meta = 5
  comid.payload = 6
  comid.hash = 7
  comid.corpus = 8
  comid.patch = 9
  comid.media = 10
  comid.supplemental = 11
```

```
   comid.tag-version = 12
   comid.software-version = 13
   comid.version-scheme = 14
   comid.lang = 15
   comid.directory = 16
   comid.file = 17
   comid.process = 18
   comid.resource = 19
   comid.size = 20
   comid.file-version = 21
   comid.key = 22
   comid.location = 23
   comid.fs-name = 24
   comid.root = 25
   comid.path-elements = 26
   comid.process-name = 27
   comid.pid = 28
   comid.type = 29
   comid.entity-name = 31
   comid.reg-id = 32
   comid.role = 33
   comid.thumbprint = 34
   comid.date = 35
   comid.device-id = 36
   comid.artifact = 37
   comid.href = 38
   comid.ownership = 39
   comid.rel = 40
   comid.media-type = 41
   comid.use = 42
   comid.activation-status = 43
   comid.channel-type = 44
   comid.colloquial-version = 45
   comid.description = 46
   comid.edition = 47
   comid.entitlement-data-required = 48
   comid.entitlement-key = 49
   comid.generator = 50
   comid.persistent-id = 51
   comid.product = 52
   comid.product-family = 53
   comid.revision = 54
   comid.summary = 55
   comid.unspsc-code = 56
   comid.unspsc-version = 57
   comid.multipartnumeric = 1
   comid.multipartnumeric-suffix = 2
   comid.alphanumeric = 3
```

```
comid.decimal = 4
comid.semver = 16384
comid.tag-creator = 1
comid.software-creator = 2
comid.aggregator = 3
comid.distributor = 4
comid.licensor = 5
comid.maintainer = 6
comid.abandon = 1
comid.private = 2
comid.shared = 3
comid.ancestor = 1
comid.component = 2
comid.feature = 3
comid.installationmedia = 4
comid.packageinstaller = 5
comid.parent = 6
comid.patches = 7
comid.requires = 8
comid.see-also = 9
comid.supersedes = 10
comid.optional = 1
comid.required = 2
comid.recommended = 3
```

A.2. API Discovery Data Model

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

coserv-well-known-info = {
  version-label => version,
  capabilities-label => [+ capability],
  api-endpoints-label => {+ tstr => tstr},
  ? result-verification-key-label => eat.JC<jwk.JWK_Set, cose.\
                                                        COSE_KeySet>,
}
version-label = eat.JC<"version", 1>
capabilities-label = eat.JC<"capabilities", 2>
api-endpoints-label = eat.JC<"api-endpoints", 3>
result-verification-key-label = eat.JC<"result-verification-key", 4>
version = tstr
capability = {
  media-type-label => cmw.media-type,
  artifact-support-label => artifact-support,
}
media-type-label = eat.JC<"media-type", 1>
artifact-support-label = eat.JC<"artifact-support", 2>
non-empty-array<M> = M .and ([+ any])
```

```
   artifact-support = non-empty-array<[
       ? "source",
       ? "collected",
   ]>
   cmw.start = cmw.cmw
   cmw.cmw = cmw.json-cmw / cmw.cbor-cmw
   cmw.json-cmw = cmw.json-record / cmw.json-collection
   cmw.cbor-cmw = cmw.cbor-record / cmw.cbor-collection / cmw.$cbor-tag
   cmw.json-record = [
     type: cmw.media-type,
     value: cmw.base64url-string,
     ? ind: uint .bits cmw.cm-type,
   ]
   cmw.cbor-record = [
     type: cmw.coap-content-format-type / cmw.media-type,
     value: bytes,
     ? ind: uint .bits cmw.cm-type,
   ]
   cmw.tag-cm-cbor<tn, fmt> = #6.<tn>(bytes .cbor fmt)
   cmw.tag-cm-data<tn> = #6.<tn>(bytes)
   cmw.json-collection = {
     ? __cmwc_t: ~uri / cmw.oid,
     + &(label: text) => cmw.json-cmw,
   }
   cmw.cbor-collection = {
     ? __cmwc_t: ~uri / cmw.oid,
     + &(label: int / text) => cmw.cbor-cmw,
   }
   cmw.media-type = text .abnf ("Content-Type" .cat cmw.Content-Type-\
                                                                  ABNF)
   cmw.base64url-string = text .regexp "[A-Za-z0-9_-]+"
   cmw.coap-content-format-type = uint .size 2
   cmw.oid = text .regexp "([0-2])((\\.0)|(\\.[1-9][0-9]*))*"
   cmw.cm-type = &(
     reference-values: 0,
     endorsements: 1,
     evidence: 2,
     attestation-results: 3,
     appraisal-policy: 4,
   )
   cmw.Content-Type-ABNF = ’

   Content-Type   = Media-Type-Name *( *SP ";" *SP parameter )
   parameter      = token "=" ( token / quoted-string )

   token          = 1*tchar
   tchar          = "!" / "#" / "$" / "%" / "&" / "\’" / "*"
                  / "+" / "-" / "." / "^" / "_" / "‘" / "|" / "~"
```

```
                / DIGIT / ALPHA
   quoted-string  = %x22 *( qdtext / quoted-pair ) %x22
   qdtext         = SP / %x21 / %x23-5B / %x5D-7E
   quoted-pair    = "\\" ( SP / VCHAR )

   Media-Type-Name = type-name "/" subtype-name

   type-name = restricted-name
   subtype-name = restricted-name

   restricted-name = restricted-name-first *126restricted-name-chars
   restricted-name-first  = ALPHA / DIGIT
   restricted-name-chars  = ALPHA / DIGIT / "!" / "#" /
                            "$" / "&" / "-" / "^" / "_"
   restricted-name-chars =/ "." ; Characters before first dot always
                                ; specify a facet name
   restricted-name-chars =/ "+" ; Characters after last plus always
                                ; specify a structured syntax suffix

   DIGIT     =  %x30-39           ; 0 - 9
   POS-DIGIT =  %x31-39           ; 1 - 9
   ALPHA     =  %x41-5A / %x61-7A ; A - Z / a - z
   SP        =  %x20
   VCHAR     =  %x21-7E           ; printable ASCII (no SP)
   '
   cmw.$cbor-tag /= cmw.tag-cm-data<1668612070> / cmw.tag-cm-cbor<\
                                          1668612069, cmw.my-evidence>
   cmw.my-evidence = {&(eat_nonce: 10) => bytes .size (8 .. 64)}
   jwk.JWK = {
     "kty" => tstr,
     ? "use" => tstr,
     ? "key_ops" => [* tstr],
     ? "alg" => tstr,
     ? "kid" => tstr,
     ? "x5u" => tstr,
     ? "x5c" => [* jwk.bytes-b64u],
     ? "x5t" => jwk.bytes-b64u,
     ? "x5t#S256" => jwk.bytes-b64u,
     ? jwk.RSA-Key-Fields,
     ? jwk.EC-Key-Fields,
     ? jwk.Symmetric-Key-Fields,
   }
   jwk.JWK_Set = [+ jwk.JWK]
   jwk.RSA-Key-Fields = (
     "n" => jwk.bytes-b64u,
     "e" => jwk.bytes-b64u,
     ? "d" => jwk.bytes-b64u,
     ? "p" => jwk.bytes-b64u,
```

```
    ? "q" => jwk.bytes-b64u,
    ? "dp" => jwk.bytes-b64u,
    ? "dq" => jwk.bytes-b64u,
    ? "qi" => jwk.bytes-b64u,
    )
  jwk.EC-Key-Fields = (
    "crv" => tstr,
    "x" => jwk.bytes-b64u,
    "y" => jwk.bytes-b64u,
    ? "d" => jwk.bytes-b64u,
    )
  jwk.Symmetric-Key-Fields = ("k" => jwk.bytes-b64u)
  jwk.bytes-b64u = tstr .b64u bytes
  eat.JC<J, C> = eat.JSON-ONLY<J> / eat.CBOR-ONLY<C>
  eat.JSON-ONLY<J> = J .feature "json"
  eat.CBOR-ONLY<C> = C .feature "cbor"
  cose.COSE_KeySet = [+ cose.COSE_Key]
  cose.COSE_Key = {
    1 => tstr / int,
    ? 2 => bstr,
    ? 3 => tstr / int,
    ? 4 => [+ tstr / int],
    ? 5 => bstr,
    * cose.label => cose.values,
  }
  cose.label = int / tstr
  cose.values = any
```

Appendix B. OpenAPI Schema

The OpenAPI schema for the request/response HTTP API described in Section 6.1 is provided below.

```
openapi: '3.0.0'

info:
  title: CoSERV HTTP API Binding
  description: CoSERV HTTP API Binding, including request-response and discovery
  version: '1.0.0alpha'
paths:
  /coserv/{query}:
    get:
      summary: Query the CoSERV endpoint.
      parameters:
        - in: path
          name: query
          schema:
            type: string
```

```
            format: base64url
          required: true
          description: base64url-encoded CoSERV query
      responses:
        '200':
          description: >
            A CoSERV result set has been successfully computed.
          content:
            application/coserv+cose:
              schema:
                type: string
                format: binary
                description: >
                  A CoSERV result set enveloped in a COSE Sign1 object.
        '400':
          description: >
            The request was malformed; e.g., the query was not valid base64url,
            or the CoSERV data item could not be successfully parsed.
          content:
            application/concise-problem-details+cbor:
              schema:
                type: string
                format: binary
                description: >
                  A CBOR-encoded problem details data item.
        '406':
          description: >
            The server cannot produce a response matching the list of acceptable
            values defined in the request's 'Accept' header field.  In particular,
            the client may have specified a CoSERV profile that is not understood or
            serviceable by the server.
          content:
            application/concise-problem-details+cbor:
              schema:
                type: string
                format: binary
                description: >
                  A CBOR-encoded problem details data item.
  /.well-known/coserv-configuration:
    get:
      summary: Retrieve the CoSERV configuration document.
      responses:
        '200':
          description: >
            The CoSERV configuration document has been successfully retrieved.
          content:
            application/coserv-discovery+json:
              schema:
```

```
                type: string
                format: binary
                description: >
                  A JSON-encoded CoSERV configuration document.
            application/coserv-discovery+cbor:
              schema:
                type: string
                format: binary
                description: >
                  A CBOR-encoded CoSERV configuration document.
```

Appendix C. Locating CoSERV Services

CoSERV facilitates the conveyance of Endorsements and Reference Values to the Verifier. The question of how the Verifier locates the CoSERV-enabled service(s) that it needs is beyond the scope of this specification. But it is an important consideration for successful deployments. When aggregators are used (see Section 2), those might also need to locate upstream CoSERV-enabled services. This non-normative appendix sets out some illustrative examples of how services might be located. This list is neither exhaustive nor prescriptive. Deployments are free to use whatever logistics are sensible. Note that the goal here is solely one of bootstrapping. Once the base URL of a suitable service is known, CoSERV provides in-protocol discovery mechanisms, such as the one described in Section 6.1.1, which cater for the discovery of more specific API endpoints and capabilities.

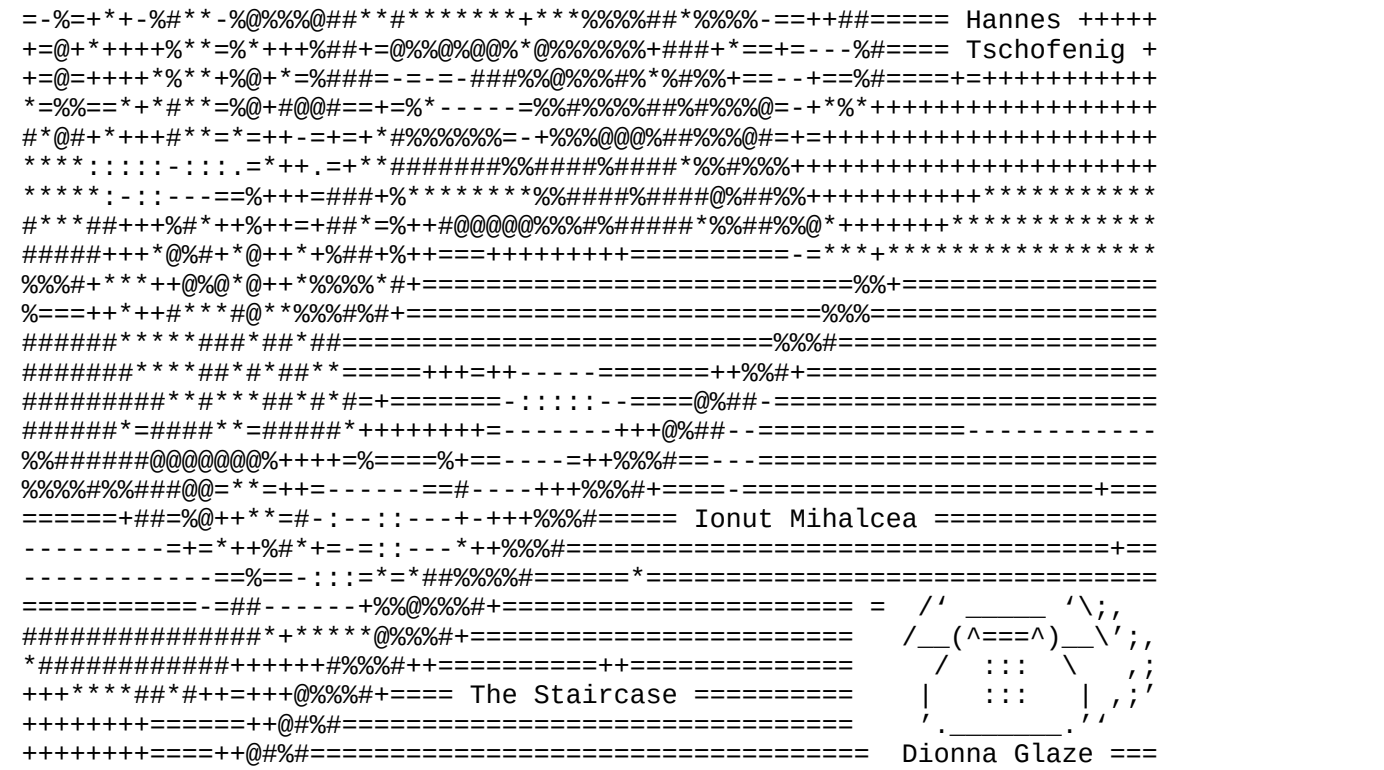
* Some CoSERV-enabled services might exist in locations that are documented publicly by supply chain actors. A hardware vendor, for example, might document the base URL for the service that endorses their products. In such a case, the location would be prior knowledge within the Verifier or aggregator that needs to consume the service. It could be hard-coded, or made available via a configuration file.

* The locations of suitable services might be carried within the Evidence produced by an Attester. An example would be a specific claim within an attestation report that is reserved and documented for this purpose. As part of the verification process, the Verifier would process this claim and use it to locate the required service(s).

* Services could be located via Manufacturer Usage Description (MUD) files as per [I-D.ietf-iotops-mud-rats].

Acknowledgments

The participants in the "Staircase meeting" at FOSDEM ’25:



Henk Birkholz and Jag Raman are puppeteering in the shadows.

The authors would like to thank Carl Wallace for their reviews and suggestions.

Authors' Addresses

Paul Howard
Arm
Email: paul.howard@arm.com

Thomas Fossati
Linaro
Email: Thomas.Fossati@linaro.org

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@ietf.contact

Shefali Kamal
Fujitsu
Email: Shefali.Kamal@fujitsu.com

Giridhar Mandyam
AMD
Email: gmandyam@amd.com

Ding Ma
Alibaba Cloud
Email: xynnn@linux.alibaba.com