CoSERV: Concise Selector for Endorsements and Reference Values

Problem Statement

Problem Statement

- Conveyance of Endorsements and RVs to the Verifier is within the scope of RFC9334

Problem Statement

- Conveyance of Endorsements and RVs to the Verifier is within the scope of RFC9334
- It is also a proven industry requirement, evidenced by services such as NVIDIA RIM and AMD KDS

Problem Statement

- Conveyance of Endorsements and RVs to the Verifier is within the scope of RFC9334
- It is also a proven industry requirement, evidenced by services such as NVIDIA RIM and AMD KDS
- Existing solutions exhibit fragmentation - proprietary APIs and disjoint data formats

Problem Statement

- Conveyance of Endorsements and RVs to the Verifier is within the scope of RFC9334
- It is also a proven industry requirement, evidenced by services such as NVIDIA RIM and AMD KDS
- Existing solutions exhibit fragmentation - proprietary APIs and disjoint data formats
- Fragmentation worsens as such services become more numerous - makes life hard for Verifiers

Problem Statement

- Conveyance of Endorsements and RVs to the Verifier is within the scope of RFC9334
- It is also a proven industry requirement, evidenced by services such as NVIDIA RIM and AMD KDS
- Existing solutions exhibit fragmentation - proprietary APIs and disjoint data formats
- Fragmentation worsens as such services become more numerous - makes life hard for Verifiers
- How does RATS offer better industry guidance?

Why CoSERV?

Why CoSERV?

- We could define more reference interaction models, or even include concrete API definitions (OpenAPI specs).

Why CoSERV?

- We could define more reference interaction models, or even include concrete API definitions (OpenAPI specs).
- Any such API would need a data format to transact - a query/response language tailored towards RATS artefacts

Why CoSERV?

- We could define more reference interaction models, or even include concrete API definitions (OpenAPI specs).
- Any such API would need a data format to transact - a query/response language tailored towards RATS artefacts
- This is a good, self-contained thing to define in a draft and validate through prototyping

Why CoSERV?

- We could define more reference interaction models, or even include concrete API definitions (OpenAPI specs).
- Any such API would need a data format to transact - a query/response language tailored towards RATS artefacts
- This is a good, self-contained thing to define in a draft and validate through prototyping
- Enter CoSERV - the Concise Selector for Endorsements and Reference Values

Requirements and Guiding Principles

Requirements and Guiding Principles

- Decoupling of message and transport (*e.g.* could transact over HTTP/REST or CoAP)

Requirements and Guiding Principles

- Decoupling of message and transport (*e.g.* could transact over HTTP/REST or CoAP)
- Adaptable to different interaction models (*e.g.* fetch everything, fetch deltas, pub/sub)

Requirements and Guiding Principles

- Decoupling of message and transport (*e.g.* could transact over HTTP/REST or CoAP)
- Adaptable to different interaction models (*e.g.* fetch everything, fetch deltas, pub/sub)
- Efficient use of the transport (*e.g.* amenable to client-side or server-side caching)

Requirements and Guiding Principles

- Decoupling of message and transport (*e.g.* could transact over HTTP/REST or CoAP)
- Adaptable to different interaction models (*e.g.* fetch everything, fetch deltas, pub/sub)
- Efficient use of the transport (*e.g.* amenable to client-side or server-side caching)
- Support constrained consumers - optimise data volume and minimise client-side processing burden

Requirements and Guiding Principles (*cont.*)

Requirements and Guiding Principles (*cont.*)

- Specialise for "RATS-native" artefacts (endorsed values, reference values, trust anchors), but allow some extensibility

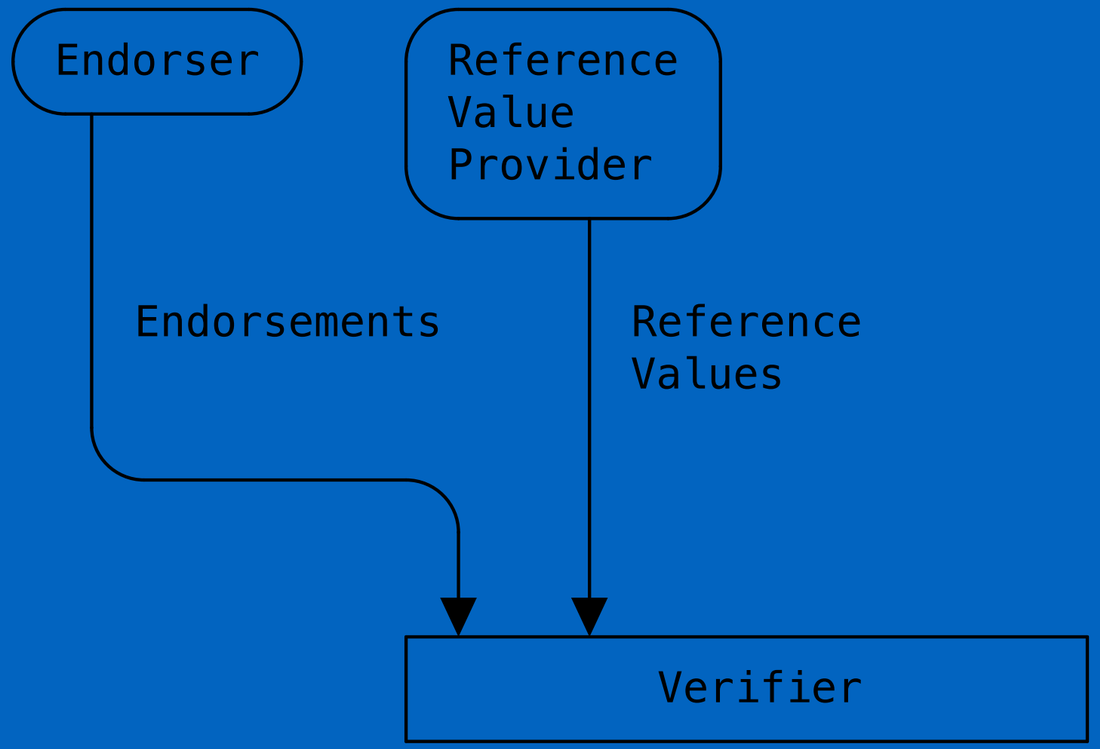
Requirements and Guiding Principles (*cont.*)

- Specialise for "RATS-native" artefacts (endorsed values, reference values, trust anchors), but allow some extensibility
- Re-use existing RATS designs where possible, and align to CDDL data model

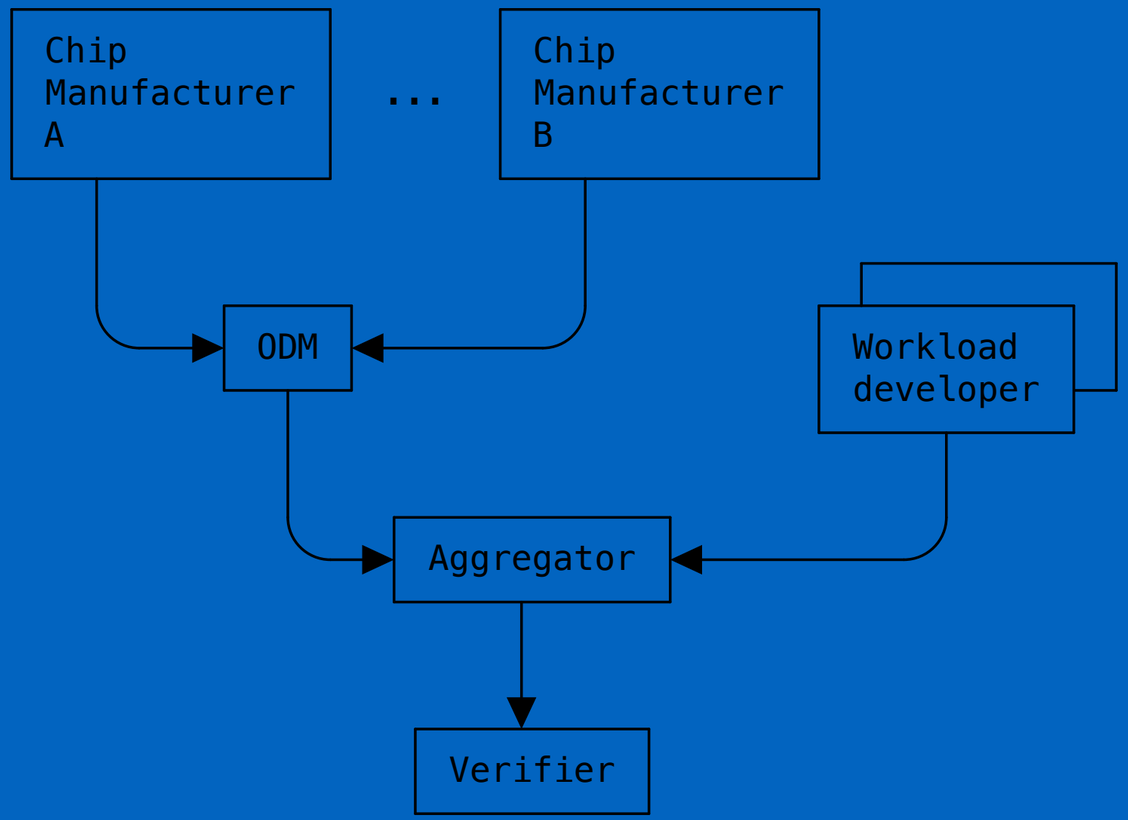
Requirements and Guiding Principles (*cont.*)

- Specialise for "RATS-native" artefacts (endorsed values, reference values, trust anchors), but allow some extensibility
- Re-use existing RATS designs where possible, and align to CDDL data model
- Allow distributors to aggregate multiple sources, with flexible trust models

Idealistic picture



Slightly more realistic



Specification

In a nutshell

Self-contained CBOR data item to model an "endorsement" query and the optional result set

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

Basic building block for an "endorsement distribution API"

CoSERV Queries

```
query = {
  &(artifact-type: 0) => artifact-type
  &(environment-selector: 1) => environment-selector-map
  &(timestamp: 2) => tdate ; RFC3339 date
  &(result-type: 3) => result-type
}
```

What artifact you are interested in (RVs, TAs, etc.)?

The attesters you are interested in?

What format would you like the results to be in (collected, source or both)?

Query Selectors

```
environment-selector-map = { selector }

selector //= ( &(class: 0) => [ + class-map ] )
selector //= ( &(instance: 1) => [ + $instance-id-type-choice ] )
selector //= ( &(group: 2) => [ + $group-id-type-choice ] )
```

Query Selectors Semantics

Class selectors:

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

Instance selectors:

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

Result Set

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

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

Expressed as CoRIM triples + authority(ies) = quads

Reference Values' Result Set

For example:

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

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

Prototyping

CoSERV Veraison Prototype

CoSERV Veraison Prototype

- Veraison has a rich history as a proving ground for RATS designs

CoSERV Veraison Prototype

- Veraison has a rich history as a proving ground for RATS designs
- Was built as a Verifier, but can be re-purposed as an Endorser or Reference Value Provider with new API endpoints, transacting CoSERV

CoSERV Veraison Prototype

- Veraison has a rich history as a proving ground for RATS designs
- Was built as a Verifier, but can be re-purposed as an Endorser or Reference Value Provider with new API endpoints, transacting CoSERV
- PoC exercise is currently in flight

Prototype API

Simple request-response over HTTP using `GET`

CoSERV query is in the outer-most path segment (allow caching)

Parameterised media type (`profile` is a CoRIM profile)

Request

```
GET /endorsement-distribution/v1/coserv/ogB4I3R... HTTP/1.1
Host: localhost:11443
Accept: application/coserv+cbor; \
        profile="tag:arm.com,2023:cca_platform#1.0.0"
```

Response

```
HTTP/1.1 200
Content-Type: application/coserv+cbor
Content-Length: 702
Date: Tue, 15 Apr 2025 10:37:28 GMT

{
  / profile / 0: "tag:arm.com,2023:cca_platform#1.0.0",
  / query / 1: {
    / artifact / 0: 2, / reference-values /
    / environment-selector / 1: {
      / class / 0: [
        { / class-id / 0: 600(h'7f45...') / impl-id / }
      ]
    }
  },
  / results / 2: { ...  }
}
```

New coserv-service

```
const (
  edApiPath = "/endorsement-distribution/v1"
)

func NewRouter(handler Handler) *gin.Engine {
  // ...
  router.GET("/.well-known/veraison/endorsement-distribution", handler.GetEdApiWellKnownInfo)

  coservEndpoint := path.Join(edApiPath, "coserv/:query")
  router.GET(coservEndpoint, handler.CoservRequest)

  // ...
}
```

New VTS Service

```
service VTS {
  // endorsement distribution API
  rpc GetEndorsements(EndorsementQueryIn) returns (EndorsementQueryOut);
  // ...
}

message EndorsementQueryIn {
  string media_type = 1; // media type (including profile)
  string query = 2;      // base64url-encoded CoSERV
}

message EndorsementQueryOut {
  Status status = 1;
  bytes result_set = 2;  // CBOR-encoded result-set CoSERV
}
```

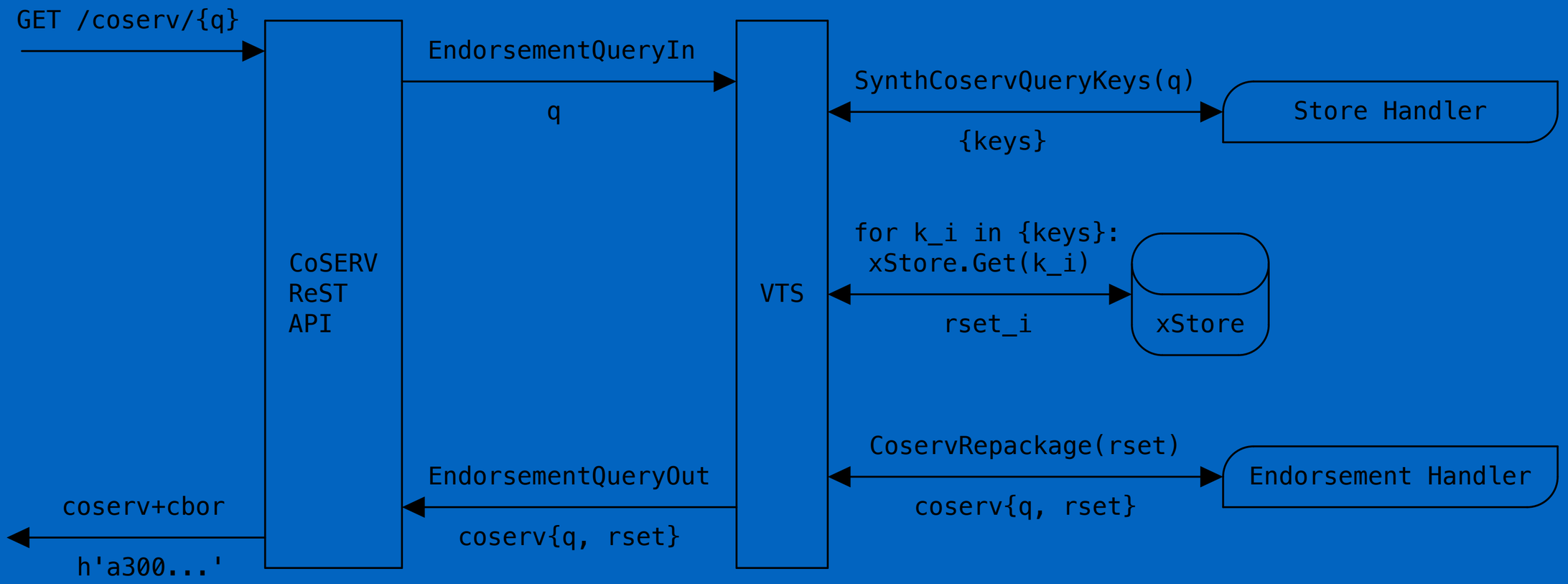
Plugins

Local mode: reuses / expands `Store` and `Endorsement`[1] handlers

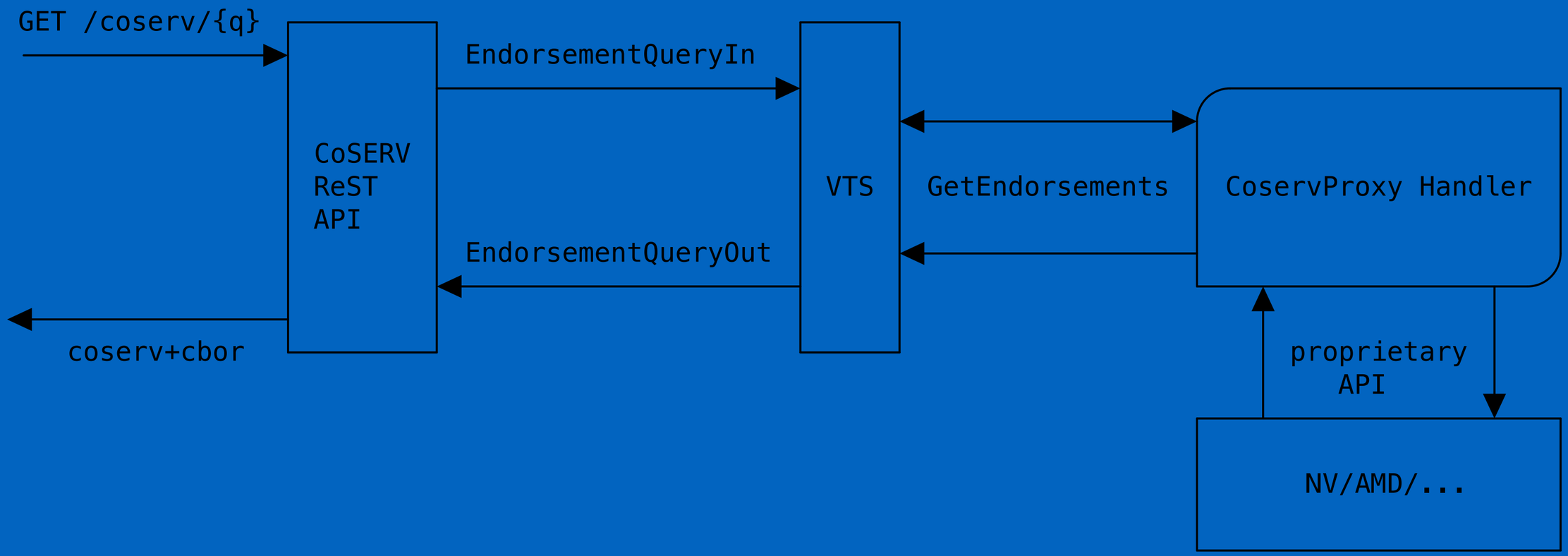
Proxy mode: new `CoservProxy` handler type

[1] Also, small tweak to synthesize a new key for instance queries on ingest.

Local


GET /coserv/{q}
CoSERV
ReST
API
EndorsementQueryIn
q
VTS
SynthCoservQueryKeys(q)
{keys}
Store Handler
for k_i in {keys}:
xStore.Get(k_i)
rset_i
xStore
CoservRepackage(rset)
coserv{q, rset}
Endorsement Handler
EndorsementQueryOut
coserv{q, rset}
coserv+cbor
h'a300...'

Proxy


GET /coserv/{q}
CoSERV
ReST
API
EndorsementQueryIn
EndorsementQueryOut
coserv+cbor
VTS
GetEndorsements
CoservProxy Handler
proprietary
API
NV/AMD/...

TODO / in progress

TODO / in progress

- Keep up with latest developments in the spec

TODO / in progress

- Keep up with latest developments in the spec
- HTTP cache control headers

TODO / in progress

- Keep up with latest developments in the spec
- HTTP cache control headers
- COSE signature

TODO / in progress

- Keep up with latest developments in the spec
- HTTP cache control headers
- COSE signature
- Example proxy plugin (NVIDIA and/or AMD)

Pointers

Pointers

- `corim git:(coserv)`

Pointers

- `corim git:(coserv)`
- `services git:(coserv)`

Pointers

- `corim git:(coserv)`
- `services git:(coserv)`
- `I-D.howard-rats-coserv`

FIN