VERAISON

https://github.com/veraison

Agenda

- Introduction
- Veraison Remote Attestation Service Overview
  - Provisioning
  - Verification
  - Attestation schemes & policies
- Extending Veraison to match your remote attestation use case
- Libraries and tooling provided by the Veraison Project

Introduction

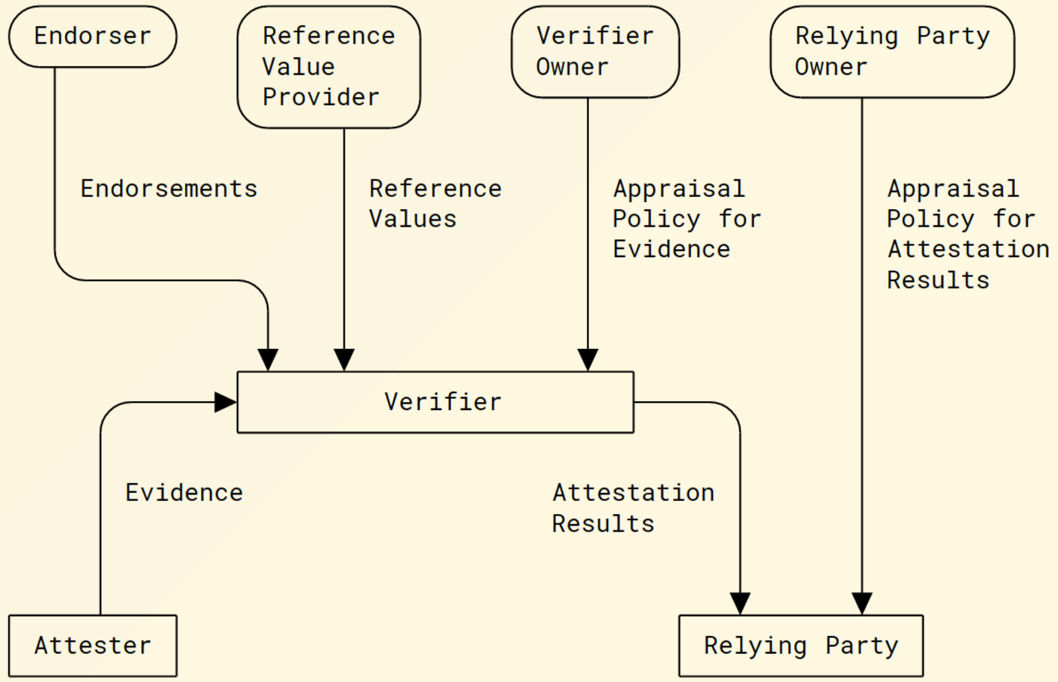
Veraison is

- **VER**ific**A**t**I**on of atte**S**tati**ON**
- A collection of libraries and tools for implementing remote attestation
- A remote attestation verification service
  - RATS Architecture compliant
  - Flexible deployment model
- Open Source (Apache v2.0) & Open Governance
- A Confidential Computing Consortium project

Attestation

- A means to establishing the trustworthiness of a TEE
- Produces signed evidence about an entity
- Attestation report alone is insufficient
  - Must be verified via a trusted service

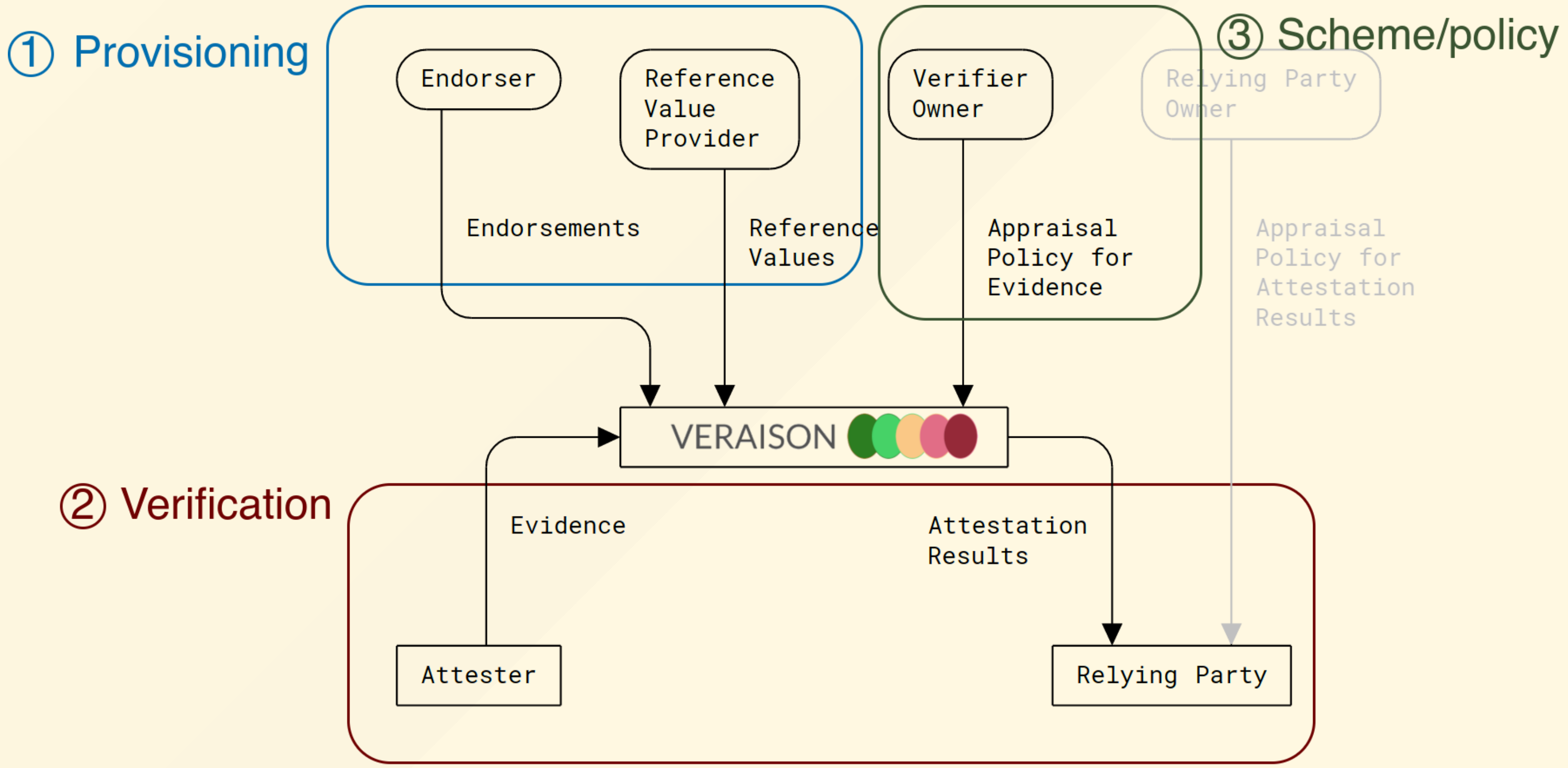
RATS Architecture



https://www.ietf.org/rfc/rfc9334.html

Introduction

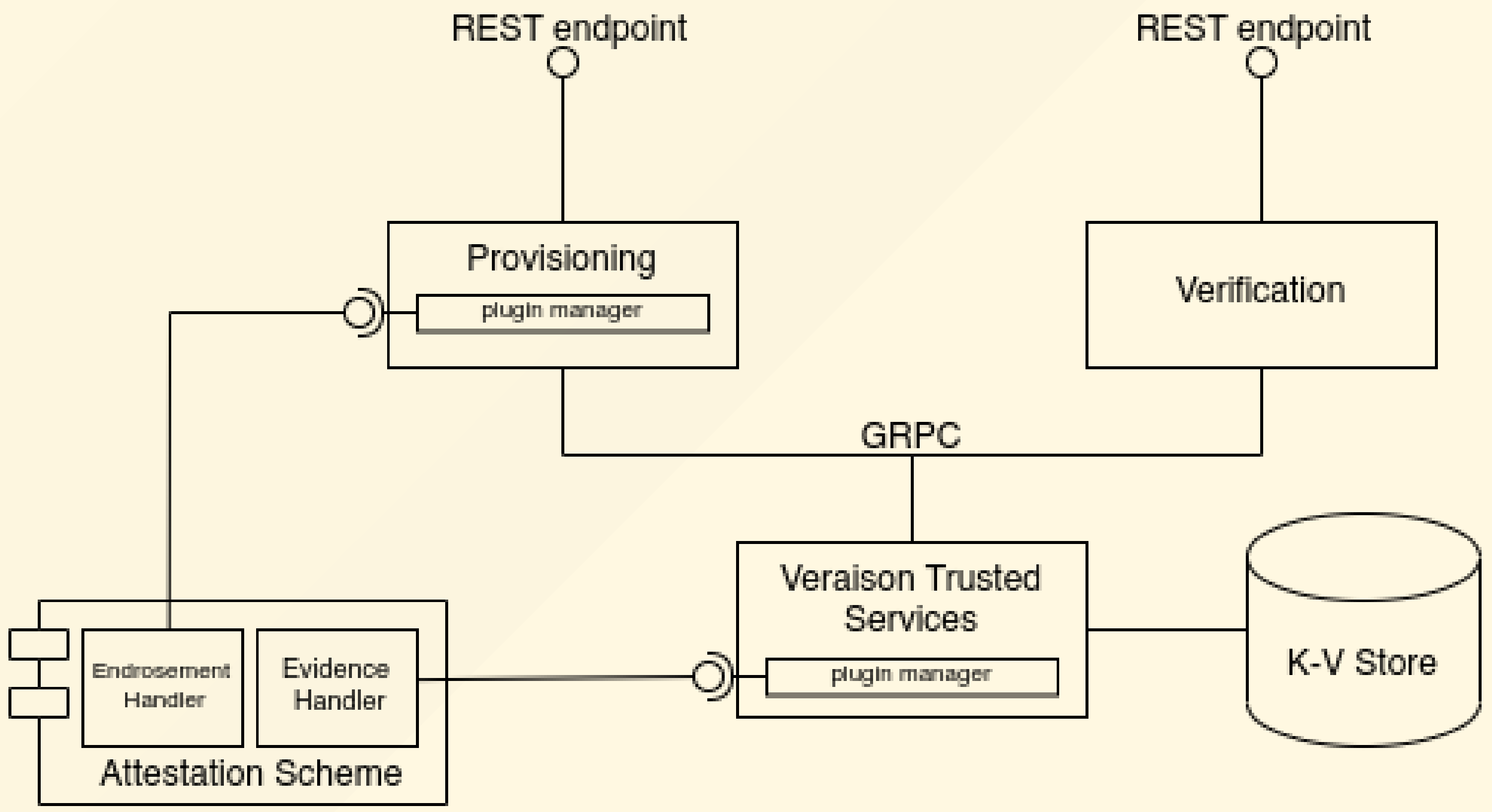
RATS Architecture



https://www.ietf.org/rfc/rfc9334.html

Veraison Overview

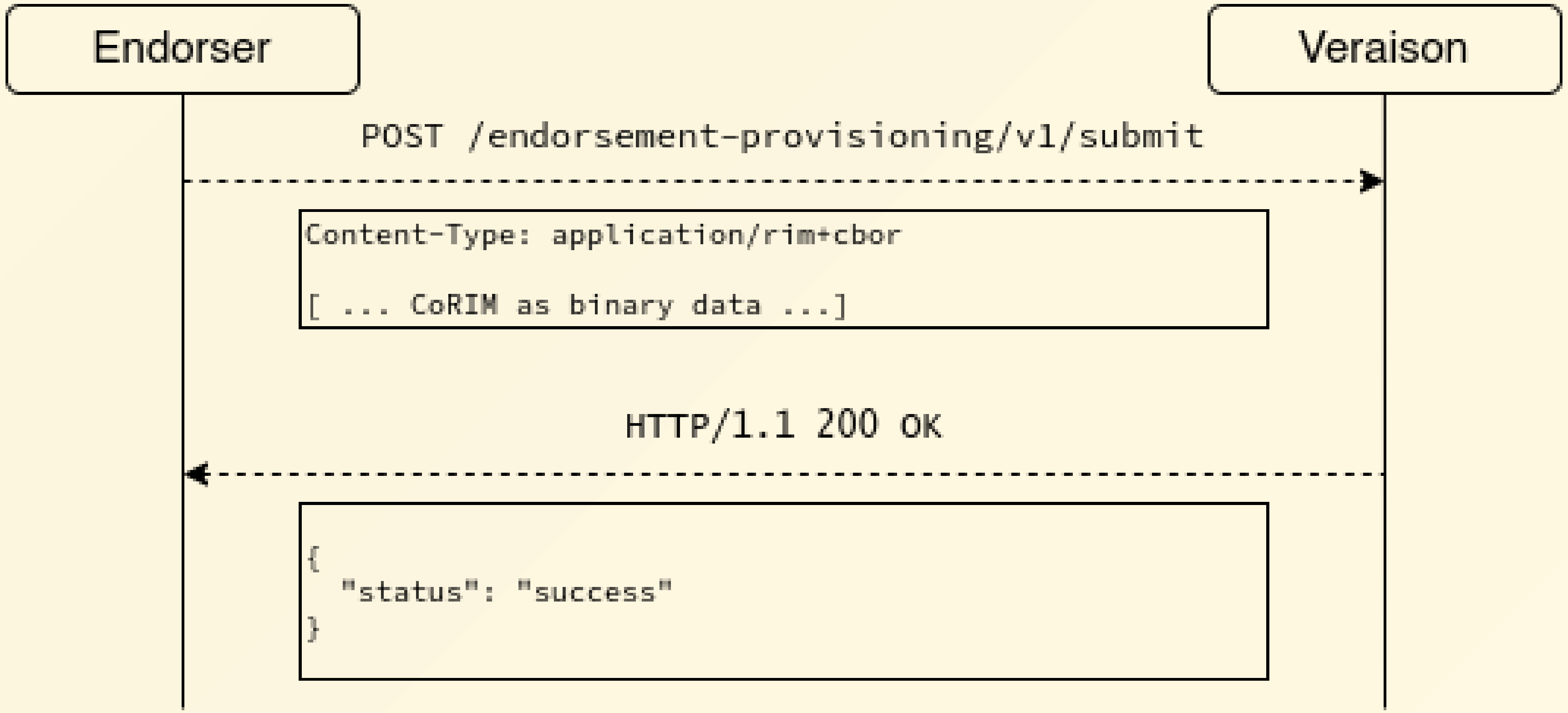
Veraison Architecture



Provisioning

- Supply **endorsements** and **trust anchors** to the Veraison service
- Data is packaged inside **CoRIM** tokens
- https://github.com/veraison/docs/blob/main/api/endorsement-provisioning/README.md

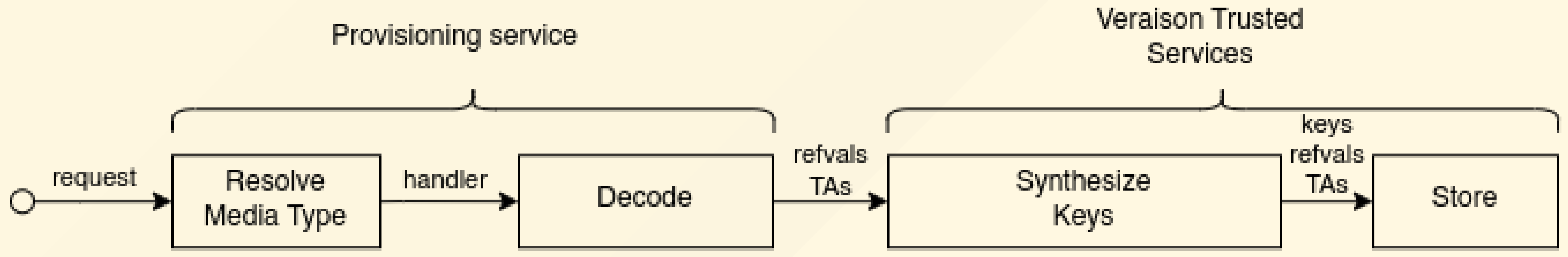
Provisioning



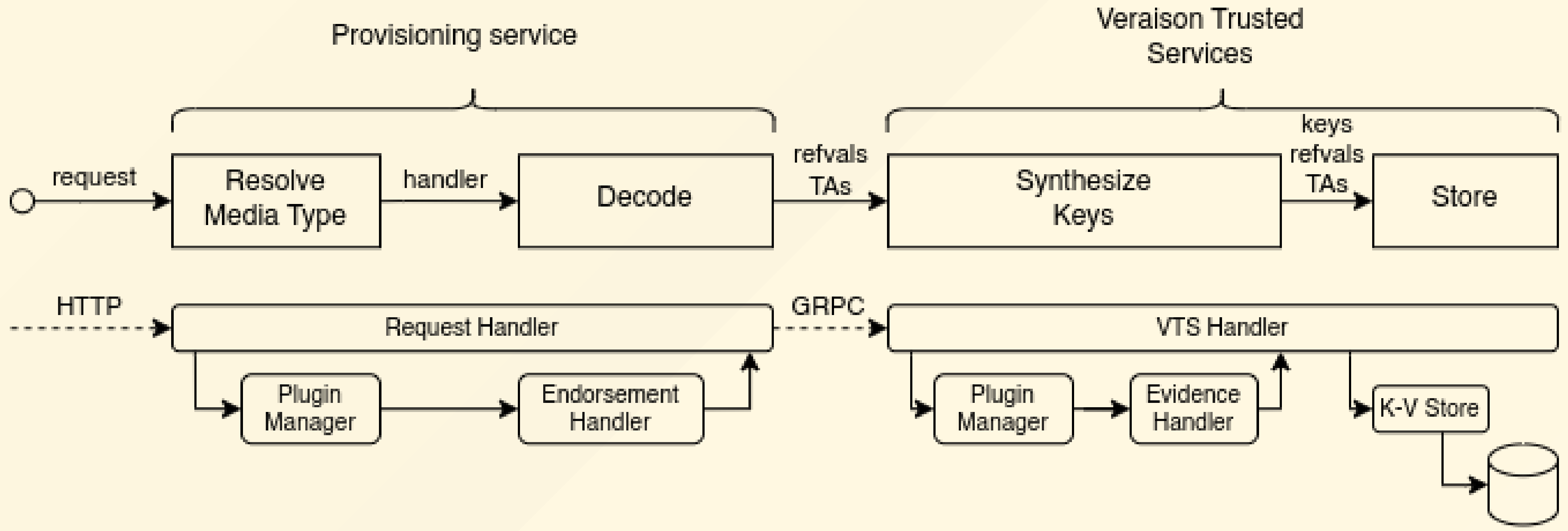
```
Content-Type: application/rim+cbor

[ ... CoRIM as binary data ...]
```

```
{
  "status": "success"
}
```

Provisioning Pipeline



Provisioning Pipeline



CoRIM

- **Co**ncise **R**eference **I**ntegrity **M**anifest
- A signed, **CBOR**-formatted document (**COSE**)
- Data are represented as subject-verb-object "triples", e.g.

```
component "X" -  has reference values - [list of values]
```

- Also contains metadata (provisioner identity, versioning, etc.)
- Adopted by IETF RATS and TCG working groups
- https://github.com/veraison/corim

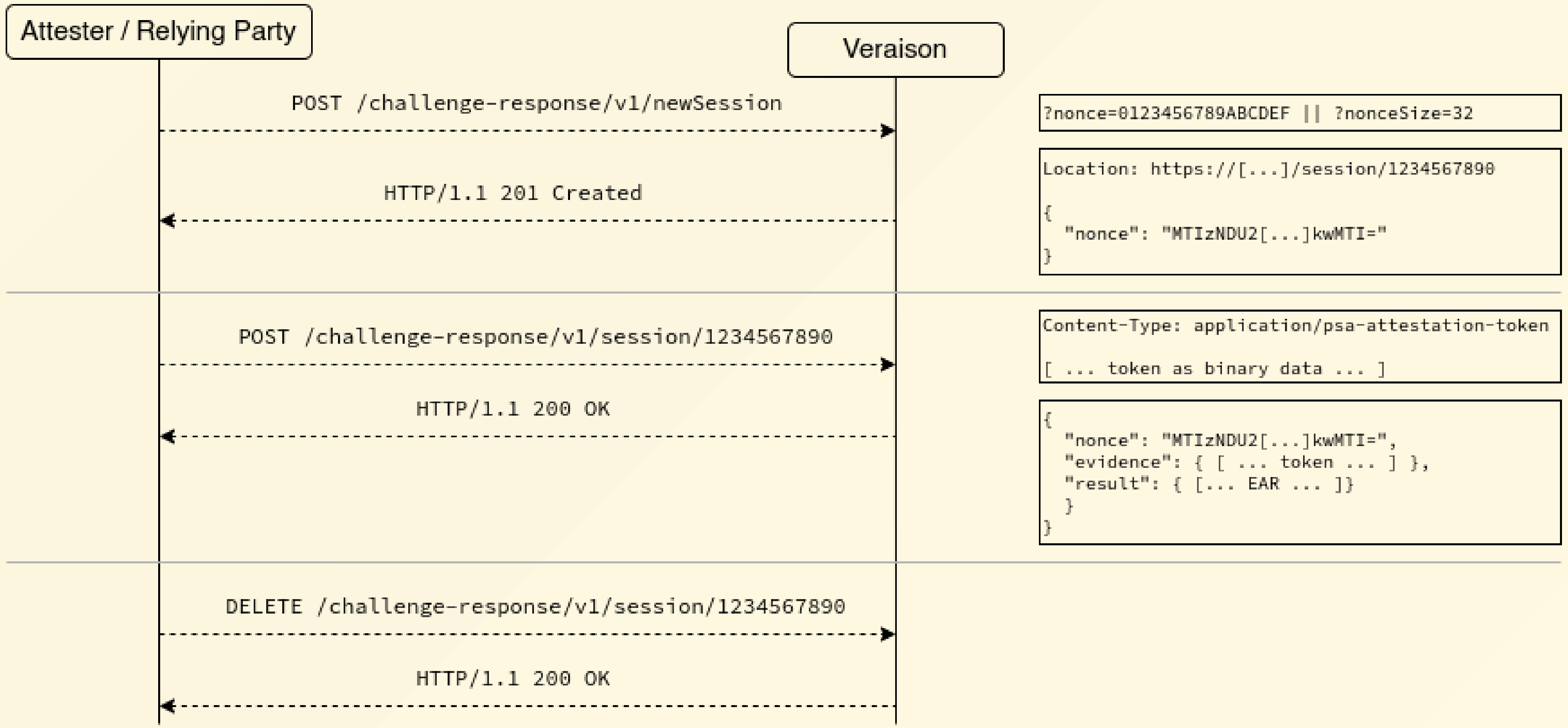
CoRIM Template Excerpt

```
  "entities": [{
      "name": "ACME Corp.",
      "regid": "https://acme.com",
      "roles": [ "tagCreator", "creator", "maintainer"]
  }],
  "triples": {
    "reference-values": [
      {
        "environment": { "instance": { "type": "uuid", "value": "7d<...>f1" }},
        "measurements": [
          { "value": { "digests": [ "sha-256:h0KPxS<...>MTPJcc=" ] } }
        ]
     }
 }
```

Verification

- A session is established with an agreed upon **nonce**
- Attester/Relying Party submits **evidence** to the verification service
- Gets back a signed **attestation result** as an **EAR** document
- https://github.com/veraison/docs/blob/main/api/challenge-response/README.md

Verification



```
POST /challenge-response/v1/newSession
```

```
?nonce=0123456789ABCDEF || ?nonceSize=32
```

```
HTTP/1.1 201 Created
```

```
Location: https://[...]/session/1234567890

{
  "nonce": "MTIzNDU2[...]kwMTI="
}
```

```
POST /challenge-response/v1/session/1234567890
```

```
Content-Type: application/psa-attestation-token

[ ... token as binary data ... ]
```

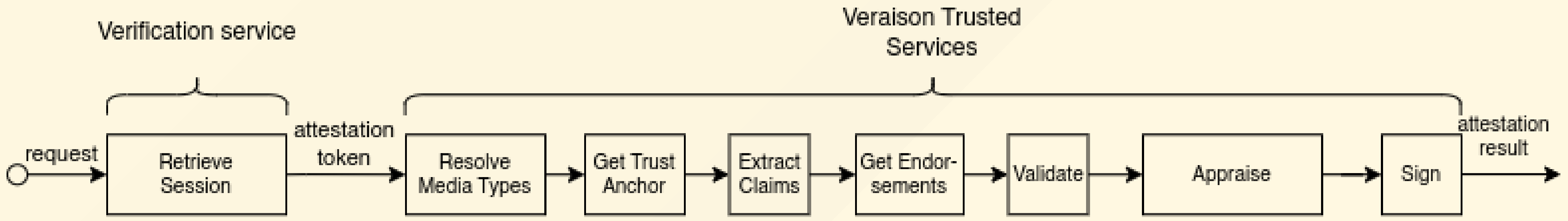
```
HTTP/1.1 200 OK
```

```
{
  "nonce": "MTIzNDU2[...]kwMTI=",
  "evidence": { [ ... token ... ] },
  "result": { [... EAR ... ]}
  }
}
```

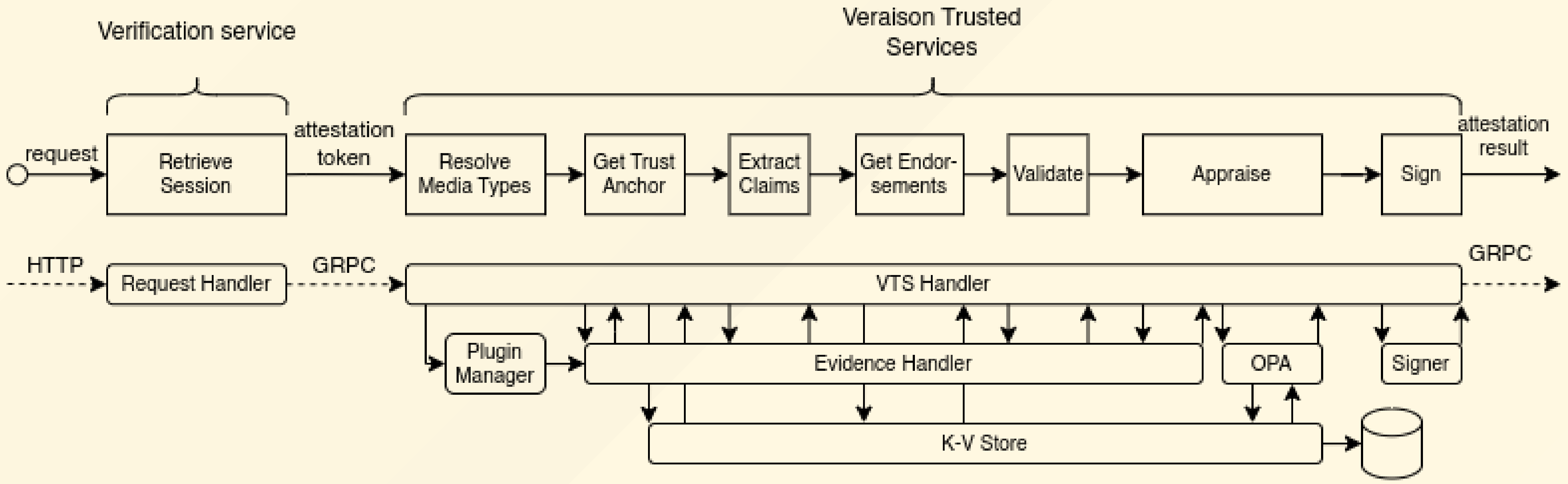
```
DELETE /challenge-response/v1/session/1234567890
```

```
HTTP/1.1 200 OK
```

Verification Pipeline



Verification Pipeline



EAR

- **E**AT **A**ttestation **R**esults
- A signed JSON document (**JWT**) containing
  - An overall status and an **AR4SI** trust vector
  - Annotated evidence*
  - Policy claims*
- https://datatracker.ietf.org/doc/draft-fv-rats-ear/
- https://datatracker.ietf.org/doc/draft-ietf-rats-ar4si/

*Veraison extension

EAR Example

```
{
    "ear.status": "affirming",
    "ear.trustworthiness-vector": {
        "configuration": 0,
        "executables": 2,
        [ ... ]
    },
    "ear.veraison.annotated-evidence": {
            "firmware-version": 7,
            "pcr-selection": [1, 2, 3, 4],
            "pcr-digest": "h0KPxSKAPTEGXnvOPPA/5HUJZjHl4Hu9eg/eYMTPJcc=",
            [ ... ]
    }
}
```

Attestation Scheme

- Defines
  - Evidence token structure
  - What endorsements and trust anchors are expected
  - How the evidence is appraised
- Implemented via pluggable interfaces
- May be augmented via deployment-specific policies

Policies

- Allow "post-processing" results generated by the scheme
    - Override appraisal decisions
    - Insert additional claims
- Implemented using **OPA** Engine
- Written in **Rego** language
- https://www.openpolicyagent.org/docs/latest/policy-language/

Policy Example

```
# This sets executables trust vector value to AFFIRMING iff BL version is
# 3.5 or greater, and to failure otherwise.
executables = "AFFIRMING" {
  # there exisists some i such that...
  some i
  # ...the i'th software component has type "BL", and...
  evidence["psa-software-components"][i]["measurement-type"] == "BL"

  # ...the version of this component is greater or equal to 3.5.
  # (semver_cmp is defined by the policy package. It returns 1 if the first
  # parameter is greater than the second, -1 if it is less than the second,
  # and 0 if they are equal.)
  semver_cmp(evidence["psa-software-components"][i].version, "3.5") >= 0
} else = "CONTRAINDICATED" # unless the above condition is met, return "CONTRAINDICATED"
```

Extending Veraison

Options

- Write an OPA Policy
    - Simpler -- a single Rego file
    - Leverages existing functionality (e.g. token validation)
    - Limited to working within the confines on an existing scheme
    - https://github.com/veraison/services/blob/main/policy/README.opa.md
- Implement Attestation Scheme
    - More flexible, but
    - More involved

Implementing an Attestation Scheme

Write a Go executable that

- Implements `IEndorsementHandler` and `IEvidenceHandler` interfaces, and
- Serves them as plugins

Can use existing implementations as examples and/or functionality re-use:

- PSA (profiles 1 & 2), CCA, EnactTrust TPM, Parsec TPM, TCG DICE

Implementing an Attestation Scheme

```go
// Attestation Scheme plugin implementation
package main

import (
        "github.com/veraison/services/handler"
        "github.com/veraison/services/plugin"
)

type MyEndrosementHandler struct {}

// Implementation of IEndrosementHandler for MyEndrosementHandler
// ...

type MyEvidenceHandler struct {}

// Implementation of IEvidenceHandler for MyEvidenceHandler
// ...

func main() {
        handler.RegisterEndorsementHandler(&MyEndorsementHandler{})
        handler.RegisterEvidenceHandler(&MyEvidenceHandler{})

        plugin.Serve()
}
```

Endorsements Handler

```
type EndorsementHandlerParams map[string]interface{}

type IEndorsementHandler interface {
        GetName() string
        GetAttestationScheme() string
        GetSupportedMediaTypes() []string

        Init(params EndorsementHandlerParams) error
        Close() error
        Decode([]byte) (*EndorsementHandlerResponse, error)
}
```

Endorsements & Trust Anchors

```
type EndorsementHandlerResponse struct {
        ReferenceValues []Endorsement
        TrustAnchors    []Endorsement
}

type Endorsement struct {
        Scheme string
        Type   int32
        SubType    string
        Attributes map[string]interface{}
}
```

Evidence Handler

```
type IEvidenceHandler interface {
        GetName() string
        GetAttestationScheme() string
        GetSupportedMediaTypes() []string

        SynthKeysFromRefValue(tenantID string, refVal *Endorsement) ([]string, error)
        SynthKeysFromTrustAnchor(tenantID string, ta *Endorsement) ([]string, error)

        GetTrustAnchorID(token *AttestationToken) (string, error)
        ExtractClaims(token *AttestationToken, trustAnchor string) (*ExtractedClaims, error)
        ValidateEvidenceIntegrity(
                token *AttestationToken,
                trustAnchor string,
                endorsementsStrings []string,
        ) error
        AppraiseEvidence(ec *EvidenceContext, endorsements []string) (*ear.AttestationResult, error)
}
```

Attestation Token & Claims

```
type AttestationToken struct {
        TenantId  string
        Data      []byte
        MediaType string
        Nonce     []byte
}

type ExtractedClaims struct {
        ClaimsSet   map[string]interface{}
        ReferenceID string
}
```

Libraries & Tooling

Tooling

- `cocli`
  - Create CoRIMs from JSON "templates" and send them to Veraison provisioning service.
  - https://github.com/veraison/corim/tree/main/cocli
- `evcli`
  - Manipulate attestation evidence
  - Currently only supports CCA and PSA
  - https://github.com/veraison/evcli
- `polcli`
  - Manage OPA policies
  - https://github.com/veraison/services/tree/main/policy/cmd/polcli
- `arc`
  - Create and verify signed EARs
  - https://github.com/veraison/ear/tree/main/arc

Libraries

- https://github.com/veraison/apiclient
    - Veraison REST API client (Go)
- https://github.com/veraison/rust-apiclient (Rust)
    - Veraison REST API client (Rust)
- https://github.com/veraison/corim
    - Concise Reference Integrity Manifest and Module Identifiers
- https://github.com/veraison/ear
    - Attestation Results for Secure Interactions (Go)
- https://github.com/veraison/c-ear
    - Attestation Results for Secure Interactions (C)
- ...and several others under https://github.com/veraison

Thank You