

RATS
Internet-Draft
Intended status: Informational
Expires: 14 December 2020

A. Shaw
H. Tschofenig
S. Trofimov
S. Frost
T. Fossati
arm
12 June 2020

Restful Attested Resources
draft-shaw-rats-rear-00

Abstract

This memo describes a REST interface based on the RATS architecture that can be used to retrieve attested system state, for example the reading of a security critical sensor. The objective is to present a common vocabulary of data formats and basic protocol transactions that can be pieced together into a cohesive interface that is capable of serving different attestation workflows.

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 14 December 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

Internet-Draft

REAR

June 2020

extracted from this document must include Simplified BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Use Cases	3
1.2.	Document Organisation	4
1.3.	Conventions used in this document	4
2.	Abstract Mechanism	4
2.1.	Attester Interface	4
2.1.1.	Resource Validation	5
2.2.	Verifier Interface	5
2.2.1.	Attestation Result Validation	6
2.3.	Example Compositions	6
2.3.1.	Background Check with Nonce-based Freshness	6
2.3.2.	Background Check with Timestamp-based Freshness	7
2.3.3.	Passport with Timestamp-based Freshness	7
2.3.4.	Timestamp-based Uni-directional	8
3.	REST Instantiation	9
3.1.	Basic Data Formats	9
3.1.1.	Resource	9
3.1.2.	Nonce	10
3.1.3.	Timestamp	10
3.1.4.	Evidence	10
3.1.5.	Attestation Result	10
3.2.	Request and Response Payloads	10
3.2.1.	Requesting an Attested Resource	10
3.2.2.	Attested Resource	10
3.2.3.	Request for Attestation Result	11
3.2.4.	Verifier Response	11
3.3.	Interaction Model	12
3.3.1.	Channel Security Considerations	12
3.3.2.	URLs	12
3.3.3.	Methods	12
3.3.4.	Multicast Support	13
3.3.5.	Examples	13
4.	Discovery	17
4.1.	Resource Directory	17
4.1.1.	Attested Resource Registration	17
4.1.2.	Verifier Resource Registration	19
5.	IANA Considerations	19

6.	Privacy Considerations	20
7.	Security Considerations	20
7.1.	Model Architecture for the Origin	20
	Acknowledgments	20
	References	20

	Normative References	20
	Informative References	22
	Authors' Addresses	22

[1.](#) Introduction

This memo describes a REST [[Fielding](#)] interface based on the RATS architecture [[I-D.ietf-rats-architecture](#)] that can be used to retrieve attested system state, for example the reading of a security critical sensor.

We present a simple vocabulary of data formats and basic protocol transactions that can be pieced together into a cohesive interface capable of serving different attestation workflows. At a minimum, we want to cater for the "background check" and "passport" topological models, and for freshness of attestation based on nonces as well as timestamps.

The obvious advantage of sharing a uniform interface across different actors is it creates an ecosystem in which variability is minimised and so is the need to add complex and often fragile logics into the deployed components, e.g., data format and protocol translation. Besides, using the familiar REST toolbox provides additional benefits in terms of developer friendliness as well as code base and infrastructure reuse (e.g., web caching).

[1.1.](#) Use Cases

The primary use case is that of a device that needs to provide application state to third parties with strong authenticity.

This is a common situation in critical infrastructure systems where an actuator device needs some assurance that the sensing equipment is in pristine state before acting on its signals. Here, the sensor would expose its safety critical samples via an attested resource whose authenticity can be verified by the actuator.

Another potential application is a fleet controller that needs to know the current state of its dependent devices to inform its next actions (e.g., scheduling a firmware update campaign). Here, the dependent devices uniformly expose the same resource (e.g., the list of currently installed software components) to the controller, which can decide, based on the information provided, which devices need a certain security patch.

Many more use cases exist.

[1.2.](#) Document Organisation

The remainder of this document describes:

- * An abstract protocol that allows a device to expose arbitrary attested system state, which can be consumed by third parties ([Section 2](#));
- * An instantiation of said abstract protocol as a set of uniform data formats and interaction primitives based on the REST paradigm for both HTTP [[RFC7230](#)] and CoAP [[RFC7252](#)] ([Section 3](#));
- * A way to advertise and discover said capability ([Section 4](#)).

[1.3.](#) Conventions used in this document

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.

[2.](#) Abstract Mechanism

The protocol principals are the three RATS actors: the attester (A), the relying party (RP) and the verifier (V).

It is assumed that A either directly owns a resource, *r*, or has a direct trust relationship with the resource owner.

In the following, "*n*" and "*t*" are freshness indicators: "*n*" is an

initiator provided nonce, "t" is a timestamp sourced by the responder. When using timestamp based freshness, producers' and consumers' clocks MUST be synchronised.

2.1. Attester Interface

The interface to the Attester is illustrated in Figure 1.

X is any entity interacting with the Attester, typically a Relying Party, which wants to retrieve an attested resource.

A function "E(n_X, r, t_A)" is used by A to compute an evidence report binding the device status to the resource ("r") together with the freshness indicators "n_X" and "t_A". Typically, only one of "n_X" or "t_A" will be present.

"E()" outputs an EAT token [[I-D.ietf-rats-eat](#)], "E", carrying a "nonce" claim that is used as described in the following.

The binding between "n_X", "t_A" and "r" is obtained by hashing their concatenation, "H(n_X || r || t_A)", and storing the result in the "nonce" claim which is then cryptographically signed by the Attester as part of the produced evidence, "E". The presence of any freshness indicator (i.e., "n_X" or "t_A") is optional. For the purpose of computing "E", a nil freshness indicator is replaced by the zero-length string, "". If "t_A != nil", then its value needs to be sent back to the requester as an additional explicit protocol entity.

Optionally, an attestation result "R" computed on evidence "E" MAY be returned by an Attester that acts as a forwarder for a Verifier.

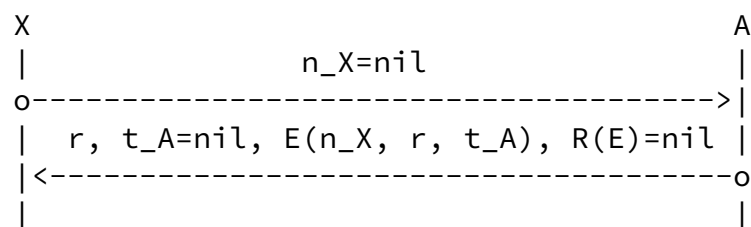


Figure 1: Attester Interface

2.1.1. Resource Validation

Given an Appraisal Policy for Evidence "APE" and an Appraisal Policy for Attestation Result "APR", X accepts "r" if and only if:

- * "E | APE => true"
- * "E.nonce == H(n_X || r || t_A)"

If "R(E) != nil", two further conditions MUST hold:

- * "R(E) | APR => true"
- * "R.nonce == H("" || E || "")"

Note that not all the appraisal operations are computed directly by X. For example, "E | APE" is typically delegated to a trusted Verifier.

[2.2.](#) Verifier Interface

The interface to the Verifier is illustrated in Figure 2.

Y is any entity interacting with the Verifier, e.g., a Relying Party or an Attester, which supplies an evidence and receives an attestation result.

The function "R(n_Y, E, t_V)" is used by V to compute the attestation result over "E" using an implicit Appraisal Policy for Evidence "APE". The result is cryptographically signed by V and bound to any available freshness indicator.

"R()" outputs an EAT token [[I-D.ietf-rats-eat](#)], "R", carrying at a minimum:

- * a "result" claim carrying a boolean value that reflects the validity of the submitted evidence given the Appraisal Policy for Evidence used by the Verifier;
- * a "nonce" claim that is used as described in the following.

The token MAY contain further information associated with the evidence validation process.

The binding between "n_Y", "t_V" and "E" is obtained by hashing their concatenation, "H(n_Y || E || t_V)", and storing the result in the "nonce" claim which is then cryptographically signed by the Verifier as part of the produced attestation result, "R". The presence of any freshness indicator (i.e., "n_Y" or "t_V") is optional. For the purpose of computing "R", a nil freshness indicator is replaced by the zero-length string, "".

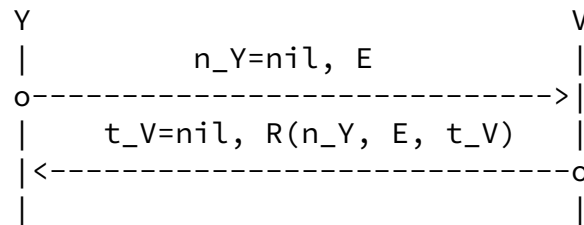


Figure 2: Verifier Interface

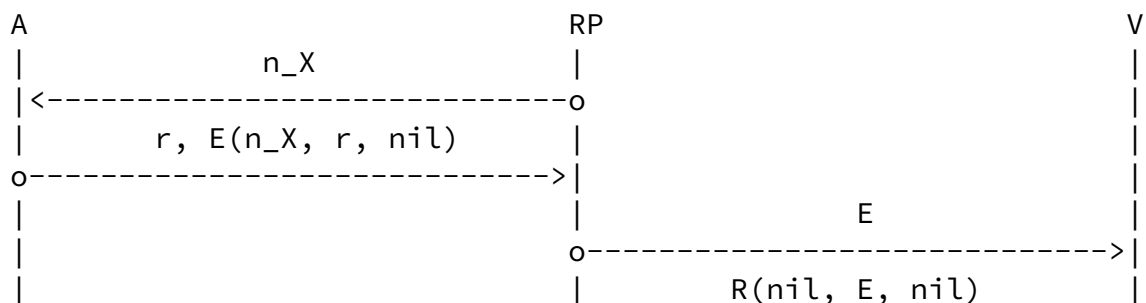
[2.2.1.](#) Attestation Result Validation

Given an Appraisal Policy for Attestation Result "APR", Y accepts "R" if and only if:

- * "R(E) | APR => true"
- * "R.nonce == H(n_Y || E || t_V)"

[2.3.](#) Example Compositions

[2.3.1.](#) Background Check with Nonce-based Freshness



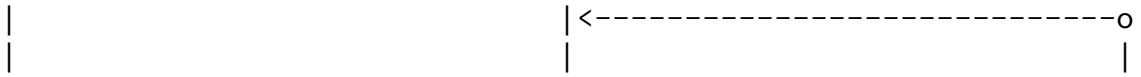


Figure 3: Background Check with Nonce-based Freshness

RP accepts "r" if and only if:

- * "E | APE => true"
- * "E.nonce == H(n_X || r || "")"
- * "R | APR => true", or equivalently "R.result == true"
- * "R.nonce == H("" || E || "")"

2.3.2. Background Check with Timestamp-based Freshness

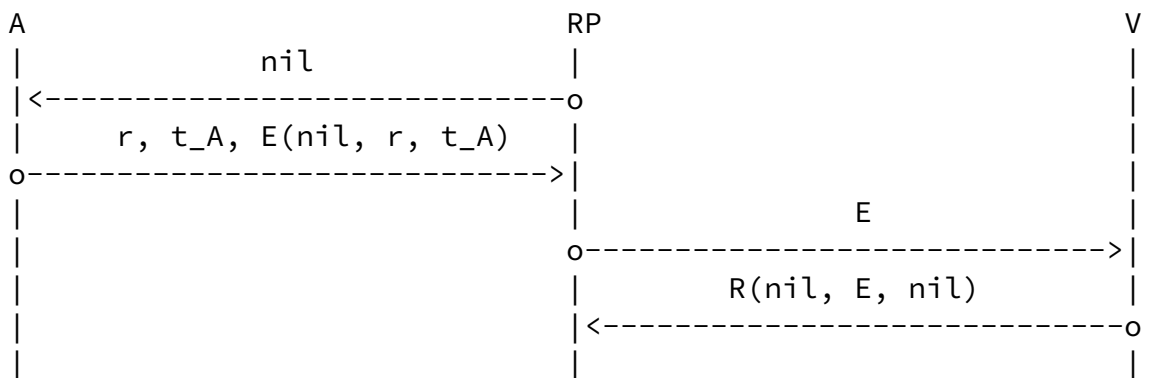


Figure 4: Background Check with Timestamp-based Freshness

RP accepts r if and only if:

- * "R | APR => true", or equivalently "R.result == true"
- * "R.nonce == H("" || E || "")"
- * "E | APE => true"
- * "E.nonce == H("" || r || t_A)"

2.3.3. Passport with Timestamp-based Freshness

The idea is that whenever the state of r changes, the Attester will "self-issue" an evidence for the changed resource using a locally sourced timestamp ("t_A") as the freshness indicator.

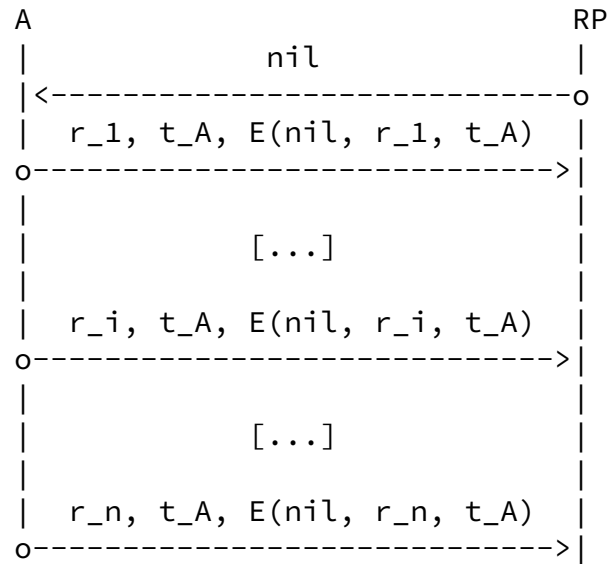


Figure 6: Timestamp-based Uni-directional

3. REST Instantiation

Four new MIME types are defined for the requests and responses among the three actors that have been identified in the abstract mechanism. The MIME types are composed of the basic data types defined in [Section 3.1](#).

3.1. Basic Data Formats

- * The resource to be attested;
- * A caller provided nonce;
- * A locally sourced timestamp;
- * The evidence produced by the Attester, and
- * The attestation result produced by the Verifier.

These basic types are described by the following CDDL rules, which reuse the eat-token definition from [\[I-D.ietf-rats-eat\]](#).

3.1.1. Resource

An "ANY DEFINED BY"-like payload with type set to the original MIME type, either Content-Type (HTTP) or Content-Format (CoAP), of the resource representation.

```

resource-type = (
  typ tstr / uint,
  val any,
)

```

Internet-Draft

REAR

June 2020

[3.1.2.](#) Nonce

nonce-type = bstr

[3.1.3.](#) Timestamp

timestamp-type = tdate / time

[3.1.4.](#) Evidence

An EAT token signed by the attester bound to the relying party request and the attested resource state.

evidence-type = eat-token

[3.1.5.](#) Attestation Result

An EAT token signed by the verifier and bound to an evidence.

attestation-result-type = eat-token

[3.2.](#) Request and Response Payloads

[3.2.1.](#) Requesting an Attested Resource

MIME type "application/rats-attested-resource-request"

CoAP Content-Format: TBD-rats-attested-resource-request-CT

nonce-key = 0 / "n_X"

```
attested-resource-request = {  
    ? nonce-key => nonce-type,  
}
```

This type is used in a POST request to an attested resource.

[3.2.2.](#) Attested Resource

MIME type "application/rats-attested-resource"

CoAP Content-Format: TBD-rats-attested-resource-CT

```
resource-key = 1 / "r"
t-A-key = 2 / "t_A"
evidence-key = 3 / "E"
attestation-result-key = 4 / "R"

attested-resource = {
    resource-key => resource-type,
    ? t-A-key => timestamp-type,
    evidence-key => evidence-type,
    ? attestation-result-key => attestation-result-type,
}
```

This type is used in a successful response to a request to an attested resource endpoint.

Note that an attestation result is only present when the Passport model is used.

Note also that the fact that the inner resource representation is embedded within the "application/rats-attested-resource" envelope suppresses the ability to do content negotiation on it, i.e., the inner representation format is unilaterally chosen by the origin.

[3.2.3.](#) Request for Attestation Result

MIME type "application/rats-attestation-result-request"

CoAP Content-Format: TBD-rats-attestation-result-request-CT

n-Y-key = 5 / "n_Y"

```
attestation-result-request = {
```

```
    ? n-Y-key => nonce-type,  
    evidence-key => evidence-type,  
}
```

This type is used in a POST request to a verifier endpoint.

[3.2.4.](#) Verifier Response

MIME type "application/rats-attestation-result-response"

CoAP Content-Format: TBD-rats-attestation-result-response-CT

```
t-V-key = 6 / "n_Y"
```

```
attestation-result-response = {  
    ? t-V-key => timestamp-type,  
    attestation-result-key => attestation-result-type,  
}
```

This type is used in a successful response to a POST request to a verifier endpoint.

[3.3.](#) Interaction Model

(For now) we only describe a synchronous, RPC-like transaction model, including the slight variant with a one-off trigger presented in [Section 2.3.4.](#)

This might be not suited for devices that sit behind a NAT/firewall box, or those that have to go through extended sleep cycles in order to save energy. For this kind of devices, we assume in-network support in the form of store-and-forward nodes (e.g., LwM2M queue mode, specialised border routers, etc.).

[3.3.1.](#) Channel Security Considerations

Unless the channel can be considered free from passive and active

attackers at all times, all transactions are to be carried over a secure transport (i.e., HTTPS or COAPS).

[3.3.2.](#) URLs

In the spirit of [[RFC7320](#)], no specific URL format is mandated. An application is free to specify the URL scheme of its liking for the exposed attested resources.

When an origin exposes the same underlying state both as nonce- and timestamp-based resources, these are identified by two separate URIs.

The verifier function is exposed via an URI that accepts evidence in form of "application/rats-attestation-result-request" typed requests and returns attestation results in form of "application/rats-attestation-result-response" typed responses.

[3.3.3.](#) Methods

As per usual REST conventions, the guiding principles are:

- * POST is used for all requests involving a payload;
- * GET is used for requests without a payload.

The only example of the latter is when retrieving an "Attested Resource" using the timestamp-based freshness model. Any other request uses POST.

[3.3.3.1.](#) Response Codes and Caching

The possible status codes are:

- * HTTP
 - 200 (OK) for successful GET. This response is cacheable; origins can use Cache-Control (max-age) and ETag headers in order to instruct on-path caches.
 - 201 (Created) for a successful POST. This response is not cacheable.
- * CoAP
 - 2.05 (Content) for successful GET. This response is cacheable; origins can use Max-Age and ETag Options to instruct on-path caches;

- 2.01 (Created) for successful POST. This response is not cacheable.

Otherwise, a suitable error response (i.e., HTTP 4xx/5xx, CoAP 4.nn/5.nn) is returned.

[3.3.4.](#) Multicast Support

TODO (This is a CoAP only feature.)

[3.3.5.](#) Examples

A few examples are given to illustrate the different interaction models using both CoAP and HTTP transports.

[3.3.5.1.](#) Background Check with Nonce Based Freshness

- * RP - Attester (CoAP)

```
>> Request:
POST coap://device.example/my-attested-resource
Content-Format: TBD-application/rats-attested-resource-request-CT
Accept: application/rats-attested-resource
Payload:
{
  "n_X": "bm9uY2Uh"
}

<< Response:
2.01 Created
```

```
ETag: "xyzzzy"
Content-format: TBD-application/rats-attested-resource-CT
Payload:
{
  "r" : {
    "typ": "text/plain",
    "val": "foobar"
  },
  "E": "eyJhbGciOi0...RfrKmTWk"
}
```

* RP - Verifier (HTTP)

```
>> Request:
POST /my-verify
Host: verifier.example
Content-Type: application/rats-attestation-result-request
Accept: application/rats-attestation-result-response
```

```
{
  "E": "eyJhbGciOi0...RfrKmTWk"
}
```

```
<< Response:
HTTP/1.1 201 Created
ETag: "abccb"
Content-format: application/rats-attestation-result-response
Payload:
{
  "R": "eyJhbGciOi0...8j5EDGYc"
}
```

[3.3.5.2.](#) Background Check with Timestamp Based Freshness

* RP - Attester (CoAP) with POST

```
>> Request:
POST coap://device.example/my-attested-resource
Content-Format: TBD-application/rats-attested-resource-request-CT
Accept: TBD-application/rats-attested-resource-CT
```


Payload:
{ }

```
<< Response:
2.01 Created
ETag: "xyzzzy"
Content-format: TBD-application/rats-attested-resource-CT
Payload:
{
  "r" : {
    "typ": "text/plain",
    "val": "foobar"
  },
  "t_A": "2020-04-01T21:02:31Z",
  "E": "eyJhbGciOi00Lz0iZ09AaA"
}
```

* RP - Attester (CoAP) with GET

```
>> Request:
GET coap://device.example/my-attested-resource
Accept: TBD-application/rats-attested-resource-CT
```

```
<< Response:
2.05 Content
ETag: "xyzzzy"
Max-Age: 3600
Content-format: TBD-application/rats-attested-resource-CT
Payload:
{
  "r" : {
    "typ": "text/plain",
    "val": "foobar"
  },
  "t_A": "2020-04-01T21:02:31Z",
  "E": "eyJhbGciOi00Lz0iZ09AaA"
}
```

* RP - Verifier (HTTP) is the same as [Section 3.3.5.1](#).

[3.3.5.3](#). Passport Model

* Attester - Verifier (CoAP)

>> Request:

```
POST coap://verifier.example/my-verify
Content-Format: application/rats-attestation-result-request
Accept: application/rats-attestation-result-response
Payload:
{
  "E": "eyJhbGciOi0...RfrKmTWk"
}
```

<< Response:

```
2.01 Created
ETag: "jklk"
Content-format: application/rats-attestation-result-response
Payload:
{
  "R": "eyJhbGciOi0...Z0IKW9aA"
}
```

* Relying Party - Attester (CoAP) with POST

>> Request:

```
POST coap://device.example/my-attested-resource
Content-Format: TBD-application/rats-attested-resource-request-CT
Accept: TBD-application/rats-attested-resource-CT
Payload:
{ }
```

<< Response:

```
2.01 Created
ETag: "qwerty"
Content-format: TBD-application/rats-attested-resource-CT
Payload:
{
  "r": {
    "type": "text/plain",
    "val": "foobar"
  },
  "t_A": "2020-04-01T21:02:31Z",
  "E": "eyJhbGciOi0...RfrKmTWk",
  "R": "eyJhbGciOi0...Z0IKW9aA"
}
```

* Relying Party - Attester (CoAP) with GET

Internet-Draft

REAR

June 2020

```
>> Request:
  GET coap://device.example/my-attested-resource
  Accept: TBD-application/rats-attested-resource-CT

<< Response:
  2.05 Content
  ETag: "qwerty"
  Max-Age: 3600
  Content-format: TBD-application/rats-attested-resource-CT
  Payload:
  {
    "r": {
      "type": "text/plain",
      "val": "foobar"
    },
    "t_A": "2020-04-01T21:02:31Z",
    "E": "eyJhbGciOi0uLi0RfrKmTWk",
    "R": "eyJhbGciOi0uLi0Z0IKW9aA"
  }
```

[4.](#) Discovery

[4.1.](#) Resource Directory

The following describes the new link format attribute values needed for registering attested resources as well as verification endpoints to a Resource Directory [[I-D.ietf-core-resource-directory](#)].

The same attribute values can be used by RD clients to discover attestation related resources.

[4.1.1.](#) Attested Resource Registration

An attested resource is registered with:

- * an interface description (if=) with value "rats.if.timestamp" or "rats.if.nonce" depending on the supported freshness model, which determines the access method (i.e., POST+nonce vs GET);
- * a content format (ct=) with value "TBD-application/rats-attested-resource-CT";

- * an inner content format (ict=) that reflects the "type" field of the returned "resource";
- * a resource type (rt=) that reflects the nature of the inner resource.

If a resource has both a "plain" and an "attested" variant, then the link value corresponding to the "attested" resource can be associated to its "plain" twin by means of the link relationship "attested-variant".

TBD: Should we have rats.if.timestamp variants for GET and POST? Alternative includes: 1) let the client probe and server return 405/4.05 if the requested variant is not supported; 2) add another attribute that explicitly states which request methods are supported.

[4.1.1.1](#). Examples

The following example shows a registrant endpoint with the name "node1" registering an attested heart rate sensor resource to an RD.

The location /rd is an example RD location discovered in a previous .well-known/core query.

```
>> Request:
POST /rd?ep=node1 HTTP/1.1
Host: rd.example
Content-Type: application/link-format

</sensors/attested-heartrate>;
  if="rats.if.timestamp";
  rt="heart-rate-zoladz";
  ct=TBD-application/rats-attested-resource-CT;
  ict=0

<< Response:
HTTP/1.1 201 Created
Location: /rd/4520
```

The following example shows a registrant endpoint with the name "node1" registering a temperature sensor resource along with its attested twin to an RD.

The "attested-variant" link relation establishes the semantics of the link between /sensors/temp and /sensors/attested-temp: the latter being an attested version of the former. Note, in particular, that the resource type (rt=) of the linked resource is inherited by the attested twin. Missing an explicit inner content format (ict=) the content type of the inner resource representation can be assumed to be that of the linked resource. The interface description (if=) "rats.if.nonce" says that the access to the attested resource happens by supplying a nonce through a POST.

```
>> Request:
POST /rd?ep=node1 HTTP/1.1
Host: rd.example
Content-Type: application/link-format

</sensors/temp>;
  ct=41;
  rt="temperature-c";
  if="sensor",
</sensors/attested-temp>;
  anchor="/sensors/temp";
  rel="attested-variant";
  if="rats.if.nonce";
  ct=TBD-application/rats-attested-resource-CT;
  ict=41

<< Response:
HTTP/1.1 201 Created
Location: /rd/4521
```

[4.1.2.](#) Verifier Resource Registration

A Verifier resource is registered with:

- * An "rt" with value "rats.verifier";
- * A "ct" with value "TBD-application/rats-attestation-result-

response-CT"

[4.1.2.1.](#) Examples

```
>> Request:
  POST /rd?ep=node1 HTTP/1.1
  Host: rd.example
  Content-Type: application/link-format

  </my-verifier>;
    ct=application/rats-attestation-result-response;
    rt="rats.verifier"

<< Response:
  HTTP/1.1 201 Created
  Location: /rd/4522
```

[5.](#) IANA Considerations

TODO

Shaw, et al.

Expires 14 December 2020

[Page 19]

Internet-Draft

REAR

June 2020

[6.](#) Privacy Considerations

TODO

[7.](#) Security Considerations

[7.1.](#) Model Architecture for the Origin

The model architecture for the origin of the attested resource is illustrated in Figure 7. The REST client (an user agent of a relying party or verifier) interfaces directly with a REST front-end (a CoAP or HTTP server stack) running in the Rich Execution Environment (REE), for example a Linux operating system. The REST front-end is paired with a back-end Trusted Application (TA) running in the Trusted Execution Environment (TEE). The TA has exclusive control over some "resource" (e.g., a sensor that feeds back into some kind of critical infrastructure control system) and can talk to the attestation service hosted inside the TEE to request EAT tokens.

In this model, it is critical that the attestation service can only be used by the intended TA or, failing that, that the identity of the calling TA can be securely proved to the relying party or verifier. An example of the latter is the Client ID claim used in PSA attestation [[I-D.tschofenig-rats-psa-token](#)].

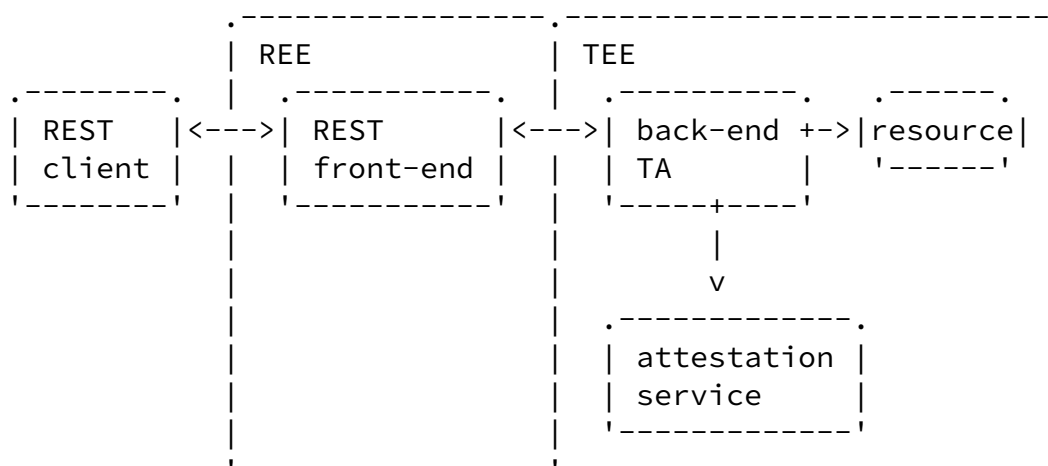


Figure 7: Model Security Architecture

Acknowledgments

TBD

References

Normative References

Shaw, et al.

Expires 14 December 2020

[Page 20]

Internet-Draft

REAR

June 2020

[I-D.ietf-core-resource-directory]

Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", Work in Progress, Internet-Draft, [draft-ietf-core-resource-directory-24](#), 9 March 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-core-resource-directory-24.txt>>.

[I-D.ietf-rats-architecture]

Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, [draft-ietf-rats-architecture-04](#), 21 May 2020, <<http://www.ietf.org/internet-drafts/>

[draft-ietf-rats-architecture-04.txt](#)>.

[I-D.ietf-rats-eat]

Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, [draft-ietf-rats-eat-03](#), 20 February 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-rats-eat-03.txt>>.

[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/info/rfc2119>>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

[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/info/rfc7252>>.

[RFC7320] Nottingham, M., "URI Design and Ownership", [BCP 190](#), [RFC 7320](#), DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", [RFC 7641](#), DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

[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/info/rfc8174>>.

Shaw, et al.

Expires 14 December 2020

[Page 21]

Internet-Draft

REAR

June 2020

Informative References

[Fielding] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000,

http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[I-D.birkholz-rats-tuda]

Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", Work in Progress, Internet-Draft, [draft-birkholz-rats-tuda-02](#), 9 March 2020, <<http://www.ietf.org/internet-drafts/draft-birkholz-rats-tuda-02.txt>>.

[I-D.tschofenig-rats-psa-token]

Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", Work in Progress, Internet-Draft, [draft-tschofenig-rats-psa-token-05](#), 6 March 2020, <<http://www.ietf.org/internet-drafts/draft-tschofenig-rats-psa-token-05.txt>>.

Authors' Addresses

Adrian Shaw
arm

Email: Adrian.Shaw@arm.com

Hannes Tschofenig
arm

Email: Hannes.Tschofenig@arm.com

Sergei Trofimov
arm

Email: Sergei.Trofimov@arm.com

Simon Frost
arm

Email: Simon.Frost@arm.com

Thomas Fossati
arm

Email: Thomas.Fossati@arm.com

