

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 28 December 2021

B. E. Carpenter
Univ. of Auckland
L. Ciavaglia
Rakuten Mobile
S. Jiang
Huawei Technologies Co., Ltd
P. Peloso
Nokia
26 June 2021

Guidelines for Autonomic Service Agents
draft-ietf-anima-asa-guidelines-01

Abstract

This document proposes guidelines for the design of Autonomic Service Agents for autonomic networks. Autonomic Service Agents, together with the Autonomic Network Infrastructure, the Autonomic Control Plane and the Generic Autonomic Signaling Protocol constitute base elements of a so-called autonomic networks ecosystem.

Discussion Venue

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

Discussion of this document takes place on the ANIMA mailing list (anima@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/anima/> (<https://mailarchive.ietf.org/arch/browse/anima/>).

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 28 December 2021.

Copyright Notice

Copyright (c) 2021 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 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
2. Logical Structure of an Autonomic Service Agent	4
3. Interaction with the Autonomic Networking Infrastructure . .	6
3.1. Interaction with the security mechanisms	6
3.2. Interaction with the Autonomic Control Plane	6
3.3. Interaction with GRASP and its API	6
3.4. Interaction with policy mechanisms	7
4. Interaction with Non-Autonomic Components	8
5. Design of GRASP Objectives	8
6. Life Cycle	9
6.1. Installation phase	10
6.1.1. Installation phase inputs and outputs	11
6.2. Instantiation phase	11
6.2.1. Operator's goal	12
6.2.2. Instantiation phase inputs and outputs	13
6.2.3. Instantiation phase requirements	13
6.3. Operation phase	14
7. Coordination between Autonomic Functions	15
8. Coordination with Traditional Management Functions	15
9. Data Models	15
10. Robustness	15
11. Security Considerations	17
12. IANA Considerations	18
13. Acknowledgements	18
14. References	18
14.1. Normative References	18
14.2. Informative References	19
Appendix A. Change log	21
Appendix B. Example Logic Flows	22
Authors' Addresses	27

1. Introduction

This document proposes guidelines for the design of Autonomic Service Agents (ASAs) in the context of an Autonomic Network (AN) based on the Autonomic Network Infrastructure (ANI) outlined in the ANIMA reference model [RFC8993]. This infrastructure makes use of the Autonomic Control Plane (ACP) [RFC8994] and the Generic Autonomic Signaling Protocol (GRASP) [RFC8990]. This document is a contribution to the description of an autonomic networks ecosystem, recognizing that a deployable autonomic network needs more than just ACP and GRASP implementations. Such an autonomic network must achieve management tasks that a Network Operations Center (NOC) cannot readily achieve manually, such as continuous resource optimization or automated fault detection and repair. These tasks, and other management automation goals, are described at length in [RFC7575]. The net result should be significant improvement of operational metrics. To achieve this objective, the autonomic networks ecosystem must include at least a library of ASAs and corresponding GRASP objective definitions. There must also be tools to deploy and oversee ASAs, and integration with existing operational mechanisms [RFC8368]. However, this document focuses on the design of ASAs, with some reference to implementation and operational aspects.

There is a considerable literature about autonomic agents with a variety of proposals about how they should be characterized. Some examples are [DeMola06], [Huebscher08], [Movahedi12] and [GANA13]. However, for the present document, the basic definitions and goals for autonomic networking given in [RFC7575] apply. According to RFC 7575, an Autonomic Service Agent is "An agent implemented on an autonomic node that implements an autonomic function, either in part (in the case of a distributed function) or whole."

ASAs must be distinguished from other forms of software component. They are components of network or service management; they do not in themselves provide services to end users. They do however provide management services to network operators and administrators. For example, the services envisaged for network function virtualisation [RFC8568] or for service function chaining [RFC7665] might be managed by an ASA rather than by traditional configuration tools.

Another example is that an existing script for locally monitoring or configuring functions or services on a router could be upgraded as an ASA that could communicate with peer scripts on neighboring or remote routers. A high-level API will allow such upgraded scripts to take full advantage of the secure ACP and the discovery, negotiation and synchronization features of GRASP. Familiar tasks such as configuring an Interior Gateway Protocol (IGP) on neighboring routers

or even exchanging IGP security keys could be performed securely in this way. This document mainly addresses issues affecting quite complex ASAs, but the most useful ones may in fact be rather simple developments from existing scripts.

The reference model [RFC8993] for autonomic networks explains further the functionality of ASAs by adding "[An ASA is] a process that makes use of the features provided by the ANI to achieve its own goals, usually including interaction with other ASAs via the GRASP protocol [RFC8990] or otherwise. Of course it also interacts with the specific targets of its function, using any suitable mechanism. Unless its function is very simple, the ASA will need to handle overlapping asynchronous operations. It may therefore be a quite complex piece of software in its own right, forming part of the application layer above the ANI."

As mentioned, there will certainly be simple ASAs that manage a single objective in a straightforward way and do not need asynchronous operations. In such a case, many aspects of the current document do not apply. However, in the general case, an ASA may be a relatively complex software component that will in many cases control and monitor simpler entities in the same or remote host(s). For example, a device controller that manages tens or hundreds of simple devices might contain a single ASA.

The remainder of this document offers guidance on the design of such ASAs.

2. Logical Structure of an Autonomic Service Agent

As mentioned above, all but the simplest ASAs will need to support asynchronous operations. Not all programming environments explicitly support multi-threading. In that case, an 'event loop' style of implementation could be adopted, in which case each thread would be implemented as an event handler called in turn by the main loop. For this, the GRASP API (Section 3.3) must provide non-blocking calls and possibly support callbacks. When necessary, the GRASP session identifier will be used to distinguish simultaneous operations.

A typical ASA will have a main thread that performs various initial housekeeping actions such as:

- * Obtain authorization credentials, if needed.
- * Register the ASA with GRASP.
- * Acquire relevant policy parameters.

- * Define data structures for relevant GRASP objectives.
- * Register with GRASP those objectives that it will actively manage.
- * Launch a self-monitoring thread.
- * Enter its main loop.

The logic of the main loop will depend on the details of the autonomic function concerned. Whenever asynchronous operations are required, extra threads will be launched, or events added to the event loop. Examples include:

- * Repeatedly flood an objective to the AN, so that any ASA can receive the objective's latest value.
- * Accept incoming synchronization requests for an objective managed by this ASA.
- * Accept incoming negotiation requests for an objective managed by this ASA, and then conduct the resulting negotiation with the counterpart ASA.
- * Manage subsidiary non-autonomic devices directly.

These threads or events should all either exit after their job is done, or enter a wait state for new work, to avoid blocking others unnecessarily.

According to the degree of parallelism needed by the application, some of these threads or events might be launched in multiple instances. In particular, if negotiation sessions with other ASAs are expected to be long or to involve wait states, the ASA designer might allow for multiple simultaneous negotiating threads, with appropriate use of queues and locks to maintain consistency.

The main loop itself could act as the initiator of synchronization requests or negotiation requests, when the ASA needs data or resources from other ASAs. In particular, the main loop should watch for changes in policy parameters that affect its operation. It should also do whatever is required to avoid unnecessary resource consumption, such as including an arbitrary wait time in each cycle of the main loop.

The self-monitoring thread is of considerable importance. Autonomic service agents must never fail. To a large extent this depends on careful coding and testing, with no unhandled error returns or exceptions, but if there is nevertheless some sort of failure, the self-monitoring thread should detect it, fix it if possible, and in the worst case restart the entire ASA.

Appendix B presents some example logic flows in informal pseudocode.

3. Interaction with the Autonomic Networking Infrastructure

3.1. Interaction with the security mechanisms

An ASA by definition runs in an autonomic node. Before any normal ASAs are started, such nodes must be bootstrapped into the autonomic network's secure key infrastructure, typically in accordance with [RFC8995]. This key infrastructure will be used to secure the ACP (next section) and may be used by ASAs to set up additional secure interactions with their peers, if needed.

Note that the secure bootstrap process itself may include special-purpose ASAs that run in a constrained insecure mode.

3.2. Interaction with the Autonomic Control Plane

In a normal autonomic network, ASAs will run as clients of the ACP, which will provide a fully secured network environment for all communication with other ASAs, in most cases mediated by GRASP (next section).

Note that the ACP formation process itself may include special-purpose ASAs that run in a constrained insecure mode.

3.3. Interaction with GRASP and its API

GRASP [RFC8990] is likely to run as a separate process with its API [RFC8991] available in user space. Thus ASAs may operate without special privilege, unless they need it for other reasons. The ASA's view of GRASP is built around GRASP objectives (Section 5), defined as data structures containing administrative information such as the objective's unique name, and its current value. The format and size of the value is not restricted by the protocol, except that it must be possible to serialise it for transmission in CBOR [RFC7049], which is no restriction at all in practice.

The GRASP API should offer the following features:

- * Registration functions, so that an ASA can register itself and the objectives that it manages.
- * A discovery function, by which an ASA can discover other ASAs supporting a given objective.
- * A negotiation request function, by which an ASA can start negotiation of an objective with a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to negotiation requests, and a set of functions to support negotiating steps. Once a negotiation starts, it is a symmetric process with both sides sending successive objective values to each other until agreement is reached (or the negotiation fails).
- * A synchronization function, by which an ASA can request the current value of an objective from a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to synchronization requests. Unlike negotiation, synchronization is an asymmetric process in which the listener sends a single objective value to the requester.
- * A flood function, by which an ASA can cause the current value of an objective to be flooded throughout the AN so that any ASA can receive it.

For further details and some additional housekeeping functions, see [RFC8991].

This API is intended to support the various interactions expected between most ASAs, such as the interactions outlined in Section 2. However, if ASAs require additional communication between themselves, they can do so using any desired protocol, even just a TLS session if that meets their needs. One option is to use GRASP discovery and synchronization as a rendez-vous mechanism between two ASAs, passing communication parameters such as a TCP port number via GRASP. As noted above, the ACP can secure such communications, unless there is a good reason to do otherwise.

3.4. Interaction with policy mechanisms

At the time of writing, the policy mechanisms for the ANI are undefined. In particular, the use of declarative policies (aka Intents) for the definition and management of ASAs behaviors remains a research topic [I-D.irtf-nmrg-ibn-concepts-definitions].

In the cases where ASAs are defined as closed control loops, the specifications defined in [ZSM009-1] regarding imperative and declarative goal statements may be applicable.

In the ANI, policy dissemination is expected to operate by an information distribution mechanism (e.g. via GRASP [RFC8990]) that can reach all autonomic nodes, and therefore every ASA. However, each ASA must be capable of operating "out of the box" in the absence of locally defined policy, so every ASA implementation must include carefully chosen default values and settings for all policy parameters.

4. Interaction with Non-Autonomic Components

An ASA, to have any external effects, must also interact with non-autonomic components of the node where it is installed. For example, an ASA whose purpose is to manage a resource must interact with that resource. An ASA whose purpose is to manage an entity that is already managed by local software must interact with that software. For example, if such management is performed by NETCONF [RFC6241], the ASA must interact directly with the NETCONF server in the same node.

In an environment where systems are virtualized and specialized using techniques such as network function virtualization or network slicing, there will be a design choice whether ASAs are deployed once per physical node or once per virtual context. A related issue is whether the ANI as a whole is deployed once on a physical network, or whether several virtual ANIs are deployed. This aspect needs to be considered by the ASA designer.

5. Design of GRASP Objectives

The general rules for the format of GRASP Objective options, their names, and IANA registration are given in [RFC8990]. Additionally that document discusses various general considerations for the design of objectives, which are not repeated here. However, note that the GRASP protocol, like HTTP, does not provide transactional integrity. In particular, steps in a GRASP negotiation are not idempotent. The design of a GRASP objective and the logic flow of the ASA should take this into account. For example, if an ASA is allocating part of a shared resource to other ASAs, it needs to ensure that the same part of the resource is not allocated twice. The easiest way is to run only one negotiation at a time. If an ASA is capable of overlapping several negotiations, it must avoid interference between these negotiations.

Negotiations will always end, normally because one end or the other declares success or failure. If this does not happen, either a timeout or exhaustion of the loop count will occur. The definition of a GRASP objective should describe a specific negotiation policy if it is not self-evident.

GRASP allows a 'dry run' mode of negotiation, where a negotiation session follows its normal course but is not committed at either end until a subsequent live negotiation session. If 'dry run' mode is defined for the objective, its specification, and every implementation, must consider what state needs to be saved following a dry run negotiation, such that a subsequent live negotiation can be expected to succeed. It must be clear how long this state is kept, and what happens if the live negotiation occurs after this state is deleted. An ASA that requests a dry run negotiation must take account of the possibility that a successful dry run is followed by a failed live negotiation. Because of these complexities, the dry run mechanism should only be supported by objectives and ASAs where there is a significant benefit from it.

The actual value field of an objective is limited by the GRASP protocol definition to any data structure that can be expressed in Concise Binary Object Representation (CBOR) [RFC7049]. For some objectives, a single data item will suffice; for example an integer, a floating point number or a UTF-8 string. For more complex cases, a simple tuple structure such as [item1, item2, item3] could be used. Since CBOR is closely linked to JSON, it is also rather easy to define an objective whose value is a JSON structure. The formats acceptable by the GRASP API will limit the options in practice. A generic solution is for the API to accept and deliver the value field in raw CBOR, with the ASA itself encoding and decoding it via a CBOR library.

A mapping from YANG to CBOR is defined by [I-D.ietf-core-yang-chor]. Subject to the size limit defined for GRASP messages, nothing prevents objectives using YANG in this way.

6. Life Cycle

In simple cases, Autonomic functions could be permanent, in the sense that ASAs are shipped as part of a product and persist throughout the product's life. However, in complex cases, a more likely situation is that ASAs need to be installed or updated dynamically, because of new requirements or bugs. This section describes one approach to the resulting life cycle.

Because continuity of service is fundamental to autonomic networking, the process of seamlessly replacing a running instance of an ASA with a new version needs to be part of the ASA's design. The implication of service continuity on the design of ASAs can be illustrated along the three main phases of the ASA life-cycle, namely Installation, Instantiation and Operation.

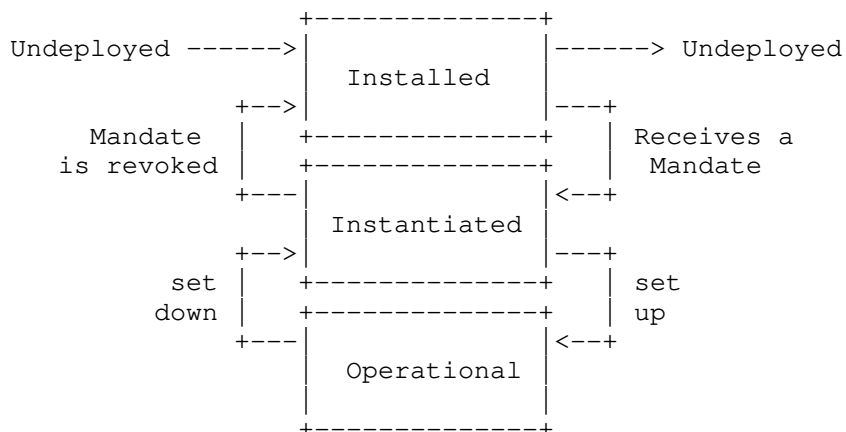


Figure 1: Life cycle of an Autonomic Service Agent

6.1. Installation phase

We define "installation" to mean that a piece of software is loaded into a device, along with any necessary libraries, but is not yet activated.

Before being able to instantiate and run ASAs, the operator will first provision the infrastructure with the sets of ASA software corresponding to its needs and objectives. The provisioning of the infrastructure is realized in the installation phase and consists in installing (or checking the availability of) the pieces of software of the different ASAs in a set of Installation Hosts. Installation Hosts may be nodes of an autonomic network, or servers dedicated to storing the software images of the different ASAs.

There are 3 properties applicable to the installation of ASAs:

The dynamic installation property allows installing an ASA on demand, on any hosts compatible with the ASA.

The decoupling property allows controlling resources of an autonomic

node from a remote ASA, i.e. an ASA installed on a host machine different from the autonomic node resources.

The multiplicity property allows controlling multiple sets of resources from a single ASA.

These three properties are very important in the context of the installation phase as their variations condition how the ASA could be installed on the infrastructure.

6.1.1. Installation phase inputs and outputs

Inputs are:

[ASA of a given type] specifies which ASAs to install,

[Installation_target_Infrastructure] specifies the candidate Installation Hosts,

[ASA placement function, e.g. under which criteria/constraints as defined by the operator] specifies how the installation phase shall meet the operator's needs and objectives for the provision of the infrastructure. In the coupled mode, the placement function is not necessary as in that case, ASA can only be installed together with the autonomic nodes; whereas in the decoupled mode, the placement function is mandatory, even though it can be as simple as an explicit list of Installation Hosts.

The main output of the installation phase is an up-to-date list of installed ASAs which corresponds to [list of ASAs] installed on [list of Installation Hosts]. This output is also useful for the coordination function and corresponds to the static interaction map (see Section 7).

The condition to validate in order to pass to next phase is to ensure that [list of ASAs] are well installed on [list of Installation Hosts]. The state of the ASAs at the end of the installation phase is: installed. (not instantiated). The following minimum set of primitives to support the installation of ASAs could be: install(list of ASAs, Installation_target_Infrastructure, ASA placement function), and un-install (list of ASAs).

6.2. Instantiation phase

We define "instantiation" as the operation of creating a single ASA instance from the corresponding piece of installed software.

Once the ASAs are installed on the appropriate hosts in the network, these ASAs may start to operate. From the operator viewpoint, an operating ASA means the ASA manages the network resources as per the objectives given. At the ASA local level, operating means executing their control loop/algorithm.

But right before that, there are two things to take into consideration. First, there is a difference between 1. having a piece of code available to run on a host and 2. having an agent based on this piece of code running inside the host. Second, in a coupled case, determining which resources are controlled by an ASA is straightforward (the ASA runs on the same autonomic node and resources it is controlling); in a decoupled mode determining this is a bit more complex: a starting agent will have to either discover the set of resources it ought to control, or such information has to be communicated to the ASA.

The instantiation phase of an ASA covers both these aspects: starting the agent piece of code (when this does not start automatically) and determining which resources have to be controlled (when this is not straightforward).

6.2.1. Operator's goal

Through this phase, the operator wants to control its autonomic network regarding at least two aspects:

- 1 determine the scope of autonomic functions by instructing which of the network resources have to be managed by which autonomic function (and more precisely by which release of the ASA software code, e.g. version number or provider),
- 2 determine how the autonomic functions are organized by instantiating a set of ASA across one or more autonomic nodes and instructing them accordingly about the other ASAs in the set as necessary.

Additionally in this phase, the operator may want to set goals to autonomic functions e.g. by configuring GRASP objectives.

The operator's goal can be summarized in an instruction to the ANIMA ecosystem matching the following format:

```
[instances of ASAs of a given type] ready to control
[Instantiation_target_Infrastructure] with
[Instantiation_target_parameters]
```

6.2.2. Instantiation phase inputs and outputs

Inputs are:

[instances of ASAs of a given type] that specifies which ASAs to instantiate

[Instantiation_target_Infrastructure] that specifies which are the resources to be managed by the autonomic function, this can be the whole network or a subset of it like a domain a technology segment or even a specific list of resources,

[Instantiation_target_parameters] that specifies which are the GRASP objectives to be set to ASAs (e.g. an optimization target)

Outputs are:

[Set of ASAs - Resources relations] describing which resources are managed by which ASA instances, this is not a formal message, but a resulting configuration of a set of ASAs.

6.2.3. Instantiation phase requirements

The instructions described in Section 6.2 could be either:

{sent to a targeted ASA} In which case, the receiving Agent will have to manage the specified list of [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

{broadcast to all ASAs} In which case, the ASAs would collectively determine from the list which Agent(s) would handle which [Instantiation_target_Infrastructure], with the [Instantiation_target_parameters].

These instructions may be grouped in an ASA Instance Mandate as a specific data structure. The specification of such an ASA Instance Mandate is beyond the scope of this document.

The conclusion of this instantiation phase is a set of ASA instances ready to operate. These ASA instances are characterized by the resources they manage, the metrics being monitored and the actions that can be executed (like modifying certain parameters values). The description of the ASA instance may be defined in an ASA Instance Manifest data structure. The specification of such an ASA Instance Manifest is beyond the scope of this document.

The ASA Instance Manifest does not only serve informational purposes such as acknowledgement of successful instantiation to the operator, but is also necessary for further autonomous operations with:

- * the coordination entities (see [I-D.ciavaglia-anima-coordination])
- * collaborative entities with the purpose to e.g. establish knowledge exchanges (some ASAs may produce knowledge or monitor metrics that other ASAs cannot and that would be useful for their execution)

6.3. Operation phase

Note: This section is to be further developed in future revisions of the document, especially the implications on the design of ASAs.

During the Operation phase, the operator can:

Activate/Deactivate ASA: meaning enabling those to execute their autonomic loop or not.

Modify ASAs targets: meaning setting them different objectives.

Modify ASAs managed resources: by updating the instance mandate which would specify different set of resources to manage (only applicable to decouples ASAs).

During the Operation phase, running ASAs can interact the one with the other:

in order to exchange knowledge (e.g. an ASA providing traffic predictions to load balancing ASA)

in order to collaboratively reach an objective (e.g. ASAs pertaining to the same autonomic function targeted to manage a network domain, these ASA will collaborate - in the case of a load balancing one, by modifying the links metrics according to the neighboring resources loads)

During the Operation phase, running ASAs are expected to apply coordination schemes

then execute their control loop under coordination supervision/instructions

The ASA life-cycle is discussed in more detail in "A Day in the Life of an Autonomic Function" [I-D.peloso-anima-autonomic-function].

7. Coordination between Autonomic Functions

Some autonomic functions will be completely independent of each other. However, others are at risk of interfering with each other - for example, two different optimization functions might both attempt to modify the same underlying parameter in different ways. In a complete system, a method is needed of identifying ASAs that might interfere with each other and coordinating their actions when necessary. This issue is considered in "Autonomic Functions Coordination" [I-D.ciavaglia-anima-coordination].

8. Coordination with Traditional Management Functions

Some ASAs will have functions that overlap with existing configuration tools and network management mechanisms such as command line interfaces, DHCP, DHCPv6, SNMP, NETCONF, and RESTCONF. This is of course an existing problem whenever multiple configuration tools are in use by the NOC. Each ASA designer will need to consider this issue and how to avoid clashes and inconsistencies. Some specific considerations for interaction with OAM tools are given in [RFC8368]. As another example, [RFC8992] describes how autonomic management of IPv6 prefixes can interact with prefix delegation via DHCPv6. The description of a GRASP objective and of an ASA using it should include a discussion of any such interactions.

9. Data Models

Management functions often include a data model, quite likely to be expressed in a formal notation such as YANG. This aspect should not be an afterthought in the design of an ASA. To the contrary, the design of the ASA and of its GRASP objectives should match the data model; as noted in Section 5, YANG serialized as CBOR may be used directly as the value of a GRASP objective.

10. Robustness

It is of great importance that all components of an autonomic system are highly robust. In principle they must never fail. This section lists various aspects of robustness that ASA designers should consider.

1. If despite all precautions, an ASA does encounter a fatal error, it should in any case restart automatically and try again. To mitigate a hard loop in case of persistent failure, a suitable pause should be inserted before such a restart. The length of the pause depends on the use case.

2. If a newly received or calculated value for a parameter falls out of bounds, the corresponding parameter should be either left unchanged or restored to a safe value.
3. If a GRASP synchronization or negotiation session fails for any reason, it may be repeated after a suitable pause. The length of the pause depends on the use case.
4. If a session fails repeatedly, the ASA should consider that its peer has failed, and cause GRASP to flush its discovery cache and repeat peer discovery.
5. In any case, it may be prudent to repeat discovery periodically, depending on the use case.
6. Any received GRASP message should be checked. If it is wrongly formatted, it should be ignored. Within a unicast session, an Invalid message (M_INVALID) may be sent. This function may be provided by the GRASP implementation itself.
7. Any received GRASP objective should be checked. Basic formatting errors like invalid CBOR will likely be detected by GRASP itself, but the ASA is responsible for checking the precise syntax and semantics of a received objective. If it is wrongly formatted, it should be ignored. Within a negotiation session, a Negotiation End message (M_END) with a Decline option (O_DECLINE) should be sent. An ASA may log such events for diagnostic purposes.
8. On the other hand, the definitions of GRASP objectives are very likely to be extended, using the flexibility of CBOR or JSON. Therefore, ASAs should be able to deal gracefully with unknown components within the values of objectives.
9. If an ASA receives either an Invalid message (M_INVALID) or a Negotiation End message (M_END) with a Decline option (O_DECLINE), one possible reason is that the peer ASA does not support a new feature of either GRASP or of the objective in question. In such a case the ASA may choose to repeat the operation concerned without using that new feature.
10. All other possible exceptions should be handled in an orderly way. There should be no such thing as an unhandled exception (but see point 1 above).

At a slightly more general level, ASAs are not services in themselves, but they automate services. This has a fundamental impact on how to design robust ASAs. In general, when an ASA

observes a particular state [1] of operations of the services/resources it controls, it typically aims to improve this state to a better state, say [2]. Ideally, the ASA is built so that it can ensure that any error encountered can still lead to returning to [1] instead of a state [3] which is worse than [1]. One example instance of this principle is "make-before-break" used in reconfiguration of routing protocols in manual operations. This principle of operations can accordingly be coded into the operation of an ASA. The GRASP dry run option mentioned in Section 5 is another tool helpful for this ASA design goal of "test-before-make".

11. Security Considerations

ASAs are intended to run in an environment that is protected by the Autonomic Control Plane [RFC8994], admission to which depends on an initial secure bootstrap process such as [RFC8995]. Such an ACP can provide keying material for mutual authentication between ASAs as well as confidential communication channels for messages between ASAs. In some deployments, a secure partition of the link layer might be used instead. However, this does not relieve ASAs of responsibility for security. When ASAs configure or manage network elements outside the ACP, potentially in a different physical node, they must interact with other non-autonomic software components to perform their management functions. The details are specific to each case, but this has an important security implication. An ASA might act as a loophole by which the managed entity could penetrate the security boundary of the ANI. Thus, ASAs must be designed to avoid such loopholes, and should if possible operate in an unprivileged mode. In particular, they must use secure techniques and carefully validate any incoming information. This will apply in particular when an ASA interacts with a management component such as a NETCONF server.

As appropriate to their specific functions, ASAs should take account of relevant privacy considerations [RFC6973].

The initial version of the autonomic infrastructure assumes that all autonomic nodes are trusted by virtue of their admission to the ACP. ASAs are therefore trusted to manipulate any GRASP objective, simply because they are installed on a node that has successfully joined the ACP. In the general case, a node may have multiple roles and a role may use multiple ASAs, each using multiple GRASP objectives. Additional mechanisms for the fine-grained authorization of nodes and ASAs to manipulate specific GRASP objectives could be designed. Independently of this, interfaces between ASAs and the router configuration/monitoring services of the node can be subject to authentication that provides more fine grained authorization for specific services. These additional authentication parameters could be passed to an ASA during its instantiation phase.

12. IANA Considerations

This document makes no request of the IANA.

13. Acknowledgements

Useful comments were received from Michael Behringer Toerless Eckert, Alex Galis, Bing Liu, Michael Richardson, and other members of the ANIMA WG.

14. References

14.1. Normative References

- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <<https://www.rfc-editor.org/info/rfc8990>>.
- [RFC8994] Eckert, T., Ed., Behringer, M., Ed., and S. Bjarnason, "An Autonomic Control Plane (ACP)", RFC 8994, DOI 10.17487/RFC8994, May 2021, <<https://www.rfc-editor.org/info/rfc8994>>.
- [RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.

14.2. Informative References

- [DeMola06] De Mola, F. and R. Quitadamo, "An Agent Model for Future Autonomic Communications", Proceedings of the 7th WOA 2006 Workshop From Objects to Agents 51-59, September 2006.
- [GANA13] "Autonomic network engineering for the self-managing Future Internet (AFI): GANA Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management.", April 2013,
<http://www.etsi.org/deliver/etsi_gs/AFI/001_099/002/01.01.01_60/gs_afi002v010101p.pdf>.
- [Huebscher08]
Huebscher, M. C. and J. A. McCann, "A survey of autonomic computing--degrees, models, and applications", ACM Computing Surveys (CSUR) Volume 40 Issue 3 DOI: 10.1145/1380584.1380585, August 2008.
- [I-D.ciavaglia-anima-coordination]
Ciavaglia, L. and P. Pierre, "Autonomic Functions Coordination", Work in Progress, Internet-Draft, draft-ciavaglia-anima-coordination-01, 21 March 2016,
<<https://datatracker.ietf.org/doc/html/draft-ciavaglia-anima-coordination-01>>.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-15, 25 January 2021,
<<https://datatracker.ietf.org/doc/html/draft-ietf-core-yang-cbor-15>>.
- [I-D.irtf-nmrg-ibn-concepts-definitions]
Clemm, A., Ciavaglia, L., Granville, L. Z., and J. Tantsura, "Intent-Based Networking - Concepts and Definitions", Work in Progress, Internet-Draft, draft-irtf-nmrg-ibn-concepts-definitions-03, 22 February 2021,
<<https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-03>>.
- [I-D.peloso-anima-autonomic-function]
Pierre, P. and L. Ciavaglia, "A Day in the Life of an Autonomic Function", Work in Progress, Internet-Draft, draft-peloso-anima-autonomic-function-01, 21 March 2016,
<<https://datatracker.ietf.org/doc/html/draft-peloso-anima-autonomic-function-01>>.

[Movahedi12]

Movahedi, Z., Ayari, M., Langar, R., and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria", IEEE Communications Surveys & Tutorials Volume: 14 , Issue: 2 DOI: 10.1109/SURV.2011.042711.00078, Page(s): 464 - 490, 2012.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <<https://www.rfc-editor.org/info/rfc7575>>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.

[RFC8368] Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <<https://www.rfc-editor.org/info/rfc8368>>.

[RFC8568] Bernardos, CJ., Rahman, A., Zuniga, JC., Contreras, LM., Aranda, P., and P. Lynch, "Network Virtualization Research Challenges", RFC 8568, DOI 10.17487/RFC8568, April 2019, <<https://www.rfc-editor.org/info/rfc8568>>.

[RFC8991] Carpenter, B., Liu, B., Ed., Wang, W., and X. Gong, "GeneRic Autonomic Signaling Protocol Application Program Interface (GRASP API)", RFC 8991, DOI 10.17487/RFC8991, May 2021, <<https://www.rfc-editor.org/info/rfc8991>>.

- [RFC8992] Jiang, S., Ed., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-Scale Networks", RFC 8992, DOI 10.17487/RFC8992, May 2021, <<https://www.rfc-editor.org/info/rfc8992>>.
- [RFC8993] Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <<https://www.rfc-editor.org/info/rfc8993>>.
- [ZSM009-1] "Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers", June 2021, <https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/00901/01.01.01_60/gs_ZSM00901v010101p.pdf>.

Appendix A. Change log

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

draft-ietf-anima-asa-guidelines-01, 2021-06-27:

- * Incorporated shepherd's review comments
- * Editorial fixes

draft-ietf-anima-asa-guidelines-00, 2020-11-14:

- * Adopted by WG
- * Editorial fixes

draft-carpenter-anima-asa-guidelines-09, 2020-07-25:

- * Additional text on future authorization.
- * Editorial fixes

draft-carpenter-anima-asa-guidelines-08, 2020-01-10:

- * Introduced notion of autonomic ecosystem.
- * Minor technical clarifications.
- * Converted to v3 format.

draft-carpenter-anima-asa-guidelines-07, 2019-07-17:

- * Improved explanation of threading vs event-loop
- * Other editorial improvements.

draft-carpenter-anima-asa-guidelines-06, 2018-01-07:

- * Expanded and improved example logic flow.

- * Editorial corrections.

draft-carpenter-anima-asa-guidelines-05, 2018-06-30:

- * Added section on relationship with non-autonomic components.
- * Editorial corrections.

draft-carpenter-anima-asa-guidelines-04, 2018-03-03:

- * Added note about simple ASAs.
- * Added note about NFV/SFC services.
- * Improved text about threading v event loop model
- * Added section about coordination with traditional tools.
- * Added appendix with example logic flow.

draft-carpenter-anima-asa-guidelines-03, 2017-10-25:

- * Added details on life cycle.
- * Added details on robustness.
- * Added co-authors.

draft-carpenter-anima-asa-guidelines-02, 2017-07-01:

- * Expanded description of event-loop case.
- * Added note about 'dry run' mode.

draft-carpenter-anima-asa-guidelines-01, 2017-01-06:

- * More sections filled in.

draft-carpenter-anima-asa-guidelines-00, 2016-09-30:

- * Initial version

Appendix B. Example Logic Flows

This appendix describes generic logic flows for an Autonomic Service Agent (ASA) for resource management. Note that these are illustrative examples, and in no sense requirements. As long as the rules of GRASP are followed, a real implementation could be different. The reader is assumed to be familiar with GRASP [RFC8990] and its conceptual API [RFC8991].

A complete autonomic function for a resource would consist of a number of instances of the ASA placed at relevant points in a network. Specific details will of course depend on the resource concerned. One example is IP address prefix management, as specified in [RFC8992]. In this case, an instance of the ASA would exist in each delegating router.

An underlying assumption is that there is an initial source of the resource in question, referred to here as an origin ASA. The other ASAs, known as delegators, obtain supplies of the resource from the origin, and then delegate quantities of the resource to consumers that request it, and recover it when no longer needed.

Another assumption is there is a set of network wide policy parameters, which the origin will provide to the delegators. These parameters will control how the delegators decide how much resource to provide to consumers. Thus the ASA logic has two operating modes: origin and delegator. When running as an origin, it starts by obtaining a quantity of the resource from the NOC, and it acts as a source of policy parameters, via both GRASP flooding and GRASP synchronization. (In some scenarios, flooding or synchronization alone might be sufficient, but this example includes both.)

When running as a delegator, it starts with an empty resource pool, it acquires the policy parameters by GRASP synchronization, and it delegates quantities of the resource to consumers that request it. Both as an origin and as a delegator, when its pool is low it seeks quantities of the resource by requesting GRASP negotiation with peer ASAs. When its pool is sufficient, it hands out resource to peer ASAs in response to negotiation requests. Thus, over time, the initial resource pool held by the origin will be shared among all the delegators according to demand.

In theory a network could include any number of origins and any number of delegators, with the only condition being that each origin's initial resource pool is unique. A realistic scenario is to have exactly one origin and as many delegators as you like. A scenario with no origin is useless.

An implementation requirement is that resource pools are kept in stable storage. Otherwise, if a delegator exits for any reason, all the resources it has obtained or delegated are lost. If an origin exits, its entire spare pool is lost. The logic for using stable storage and for crash recovery is not included in the pseudocode below.

The description below does not implement GRASP's 'dry run' function. That would require temporarily marking any resource handed out in a dry run negotiation as reserved, until either the peer obtains it in a live run, or a suitable timeout expires.

The main data structures used in each instance of the ASA are:

- * The `resource_pool`, for example an ordered list of available resources. Depending on the nature of the resource, units of resource are split when appropriate, and a background garbage collector recombines split resources if they are returned to the pool.
- * The `delegated_list`, where a delegator stores the resources it has given to consumers routers.

Possible main logic flows are below, using a threaded implementation model. The transformation to an event loop model should be apparent - each thread would correspond to one event in the event loop.

The GRASP objectives are as follows:

- * `["EX1.Resource", flags, loop_count, value]` where the value depends on the resource concerned, but will typically include its size and identification.
- * `["EX1.Params", flags, loop_count, value]` where the value will be, for example, a JSON object defining the applicable parameters.

In the outline logic flows below, these objectives are represented simply by their names.

<CODE BEGINS>

MAIN PROGRAM:

```
Create empty resource_pool (and an associated lock)
Create empty delegated_list
Determine whether to act as origin
if origin:
    Obtain initial resource_pool contents from NOC
    Obtain value of EX1.Params from NOC
Register ASA with GRASP
Register GRASP objectives EX1.Resource and EX1.Params
if origin:
    Start FLOODER thread to flood EX1.Params
    Start SYNCHRONIZER listener for EX1.Params
Start MAIN_NEGOTIATOR thread for EX1.Resource
if not origin:
    Obtain value of EX1.Params from GRASP flood or synchronization
    Start DELEGATOR thread
Start GARBAGE_COLLECTOR thread
do forever:
    good_peer = none
    if resource_pool is low:
        Calculate amount A of resource needed
        Discover peers using GRASP M_DISCOVER / M_RESPONSE
        if good_peer in peers:
            peer = good_peer
        else:
            peer = #any choice among peers
            grasp.request_negotiate("EX1.Resource", peer)
            i.e., send M_REQ_NEG
            Wait for response (M_NEGOTIATE, M_END or M_WAIT)
            if OK:
                if offered amount of resource sufficient:
                    Send M_END + O_ACCEPT #negotiation succeeded
                    Add resource to pool
                    good_peer = peer
                else:
                    Send M_END + O_DECLINE #negotiation failed
            sleep() #sleep time depends on application scenario
```

MAIN_NEGOTIATOR thread:

```
do forever:
    grasp.listen_negotiate("EX1.Resource")
    i.e., wait for M_REQ_NEG
    Start a separate new NEGOTIATOR thread for requested amount A
```

NEGOTIATOR thread:

```
Request resource amount A from resource_pool
if not OK:
    while not OK and A > Amin:
        A = A-1
        Request resource amount A from resource_pool
if OK:
    Offer resource amount A to peer by GRASP M_NEGOTIATE
    if received M_END + O_ACCEPT:
        #negotiation succeeded
    elif received M_END + O_DECLINE or other error:
        #negotiation failed
else:
    Send M_END + O_DECLINE #negotiation failed
```

DELEGATOR thread:

```
do forever:
    Wait for request or release for resource amount A
    if request:
        Get resource amount A from resource_pool
        if OK:
            Delegate resource to consumer
            Record in delegated_list
        else:
            Signal failure to consumer
            Signal main thread that resource_pool is low
    else:
        Delete resource from delegated_list
        Return resource amount A to resource_pool
```

SYNCHRONIZER thread:

```
do forever:
    Wait for M_REQ_SYN message for EX1.Params
    Reply with M_SYNCH message for EX1.Params
```

FLOODER thread:

```
do forever:
    Send M_FLOOD message for EX1.Params
    sleep() #sleep time depends on application scenario
```

GARBAGE_COLLECTOR thread:

```
do forever:
    Search resource_pool for adjacent resources
    Merge adjacent resources
    sleep() #sleep time depends on application scenario
```

<CODE ENDS>

Authors' Addresses

Brian Carpenter
School of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Laurent Ciavaglia
Rakuten Mobile
Paris
France

Email: laurent.ciavaglia@rakuten.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14 Huawei Campus
156 Beiqing Road
Hai-Dian District
Beijing
100095
China

Email: jiangsheng@huawei.com

Pierre Peloso
Nokia
Villarceaux
91460 Nozay
France

Email: pierre.peloso@nokia.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 5 August 2022

B. E. Carpenter
Univ. of Auckland
L. Ciavaglia
Rakuten Mobile
S. Jiang
Huawei Technologies Co., Ltd
P. Peloso
Nokia
1 February 2022

Guidelines for Autonomic Service Agents
draft-ietf-anima-asa-guidelines-07

Abstract

This document proposes guidelines for the design of Autonomic Service Agents for autonomic networks. Autonomic Service Agents, together with the Autonomic Network Infrastructure, the Autonomic Control Plane and the Generic Autonomic Signaling Protocol constitute base elements of an autonomic networking ecosystem.

Discussion Venue

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

Discussion of this document takes place on the ANIMA mailing list (anima@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/anima/> (<https://mailarchive.ietf.org/arch/browse/anima/>).

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 5 August 2022.

Copyright Notice

Copyright (c) 2022 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 extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Logical Structure of an Autonomic Service Agent	5
3. Interaction with the Autonomic Networking Infrastructure . .	6
3.1. Interaction with the security mechanisms	6
3.2. Interaction with the Autonomic Control Plane	6
3.3. Interaction with GRASP and its API	7
3.4. Interaction with policy mechanisms	8
4. Interaction with Non-Autonomic Components and Systems	9
5. Design of GRASP Objectives	9
6. Life Cycle	10
6.1. Installation phase	11
6.1.1. Installation phase inputs and outputs	12
6.2. Instantiation phase	13
6.2.1. Operator's goal	13
6.2.2. Instantiation phase inputs and outputs	14
6.2.3. Instantiation phase requirements	14
6.3. Operation phase	15
6.4. Removal phase	16
7. Coordination and Data Models	16
7.1. Coordination between Autonomic Functions	16
7.2. Coordination with Traditional Management Functions . . .	16
7.3. Data Models	16
8. Robustness	17
9. Security Considerations	18
10. IANA Considerations	19
11. Acknowledgements	19
12. References	20
12.1. Normative References	20
12.2. Informative References	20
Appendix A. Change log	22
Appendix B. Terminology	25
Appendix C. Example Logic Flows	25

Authors' Addresses	30
------------------------------	----

1. Introduction

This document proposes guidelines for the design of Autonomic Service Agents (ASAs) in the context of an Autonomic Network (AN) based on the Autonomic Network Infrastructure (ANI) outlined in the autonomic networking reference model [RFC8993]. This infrastructure makes use of the Autonomic Control Plane (ACP) [RFC8994] and the Generic Autonomic Signaling Protocol (GRASP) [RFC8990]. A general introduction to this environment may be found at [IPJ], which also includes explanatory diagrams, and a summary of terminology is in Appendix B.

This document is a contribution to the description of an autonomic networking ecosystem, recognizing that a deployable autonomic network needs more than just ACP and GRASP implementations. Such an autonomic network must achieve management tasks that a Network Operations Center (NOC) cannot readily achieve manually, such as continuous resource optimization or automated fault detection and repair. These tasks, and other management automation goals, are described at length in [RFC7575]. The net result should be significant operational improvement. To achieve this, the autonomic networking ecosystem must include at least a library of ASAs and corresponding GRASP technical objective definitions. A GRASP objective [RFC8990] is a data structure whose main contents are a name and a value. The value consists of a single configurable parameter or a set of parameters of some kind.

There must also be tools to deploy and oversee ASAs, and integration with existing operational mechanisms [RFC8368]. However, this document focuses on the design of ASAs, with some reference to implementation and operational aspects.

There is a considerable literature about autonomic agents with a variety of proposals about how they should be characterized. Some examples are [DeMola06], [Huebscher08], [Movahedi12] and [GANA13]. However, for the present document, the basic definitions and goals for autonomic networking given in [RFC7575] apply. According to RFC 7575, an Autonomic Service Agent is "An agent implemented on an autonomic node that implements an autonomic function, either in part (in the case of a distributed function) or whole."

ASAs must be distinguished from other forms of software component. They are components of network or service management; they do not in themselves provide services to end users. They do however provide management services to network operators and administrators. For example, the services envisaged for network function virtualisation [NFV] or for service function chaining [RFC7665] might be managed by an ASA rather than by traditional configuration tools.

Another example is that an existing script running within a router to locally monitor or configure functions or services could be upgraded to an ASA that could communicate with peer scripts on neighboring or remote routers. A high-level API will allow such upgraded scripts to take full advantage of the secure ACP and the discovery, negotiation and synchronization features of GRASP. Familiar tasks such as configuring an Interior Gateway Protocol (IGP) on neighboring routers or even exchanging IGP security keys could be performed securely in this way. This document mainly addresses issues affecting quite complex ASAs, but initially the most useful ASAs may in fact be rather simple evolutions of existing scripts.

The reference model [RFC8993] for autonomic networks explains further the functionality of ASAs by adding "[An ASA is] a process that makes use of the features provided by the ANI to achieve its own goals, usually including interaction with other ASAs via the GRASP protocol [RFC8990] or otherwise. Of course, it also interacts with the specific targets of its function, using any suitable mechanism. Unless its function is very simple, the ASA will need to handle overlapping asynchronous operations. It may therefore be a quite complex piece of software in its own right, forming part of the application layer above the ANI."

As mentioned, there will certainly be simple ASAs that manage a single objective in a straightforward way and do not need asynchronous operations. In nodes where computing power and memory space are limited, ASAs should run at a much lower frequency than the primary workload, so CPU load should not be a big issue, but memory footprint in a constrained node is certainly a concern. ASAs installed in constrained devices will have limited functionality. In such cases, many aspects of the current document do not apply. However, in the general case, an ASA may be a relatively complex software component that will in many cases control and monitor simpler entities in the same or remote host(s). For example, a device controller that manages tens or hundreds of simple devices might contain a single ASA.

The remainder of this document offers guidance on the design of complex ASAs. Some of the material may be familiar to those experienced in distributed fault-tolerant and real-time control systems. Robustness and security are of particular importance in autonomic networks and are discussed in Section 8 and Section 9.

2. Logical Structure of an Autonomic Service Agent

As mentioned above, all but the simplest ASAs will need to support asynchronous operations. Different programming environments support asynchronicity in different ways. In this document, we use an explicit multi-threading model to describe operations. This is illustrative, and alternatives to multi-threading are discussed in detail in connection with the GRASP API (see Section 3.3).

A typical ASA will have a main thread that performs various initial housekeeping actions such as:

- * Obtain authorization credentials, if needed.
- * Register the ASA with GRASP.
- * Acquire relevant policy parameters.
- * Declare data structures for relevant GRASP objectives.
- * Register with GRASP those objectives that it will actively manage.
- * Launch a self-monitoring thread.
- * Enter its main loop.

The logic of the main loop will depend on the details of the autonomic function concerned. Whenever asynchronous operations are required, extra threads may be launched. Examples of such threads include:

- * Repeatedly flood an objective to the AN, so that any ASA can receive the objective's latest value.
- * Accept incoming synchronization requests for an objective managed by this ASA.
- * Accept incoming negotiation requests for an objective managed by this ASA, and then conduct the resulting negotiation with the counterpart ASA.
- * Manage subsidiary non-autonomic devices directly.

These threads should all either exit after their job is done, or enter a wait state for new work, to avoid wasting system resources.

According to the degree of parallelism needed by the application, some of these threads might be launched in multiple instances. In particular, if negotiation sessions with other ASAs are expected to be long or to involve wait states, the ASA designer might allow for multiple simultaneous negotiating threads, with appropriate use of queues and synchronization primitives to maintain consistency.

The main loop itself could act as the initiator of synchronization requests or negotiation requests, when the ASA needs data or resources from other ASAs. In particular, the main loop should watch for changes in policy parameters that affect its operation, and if appropriate, occasionally refresh authorization credentials. It should also do whatever is required to avoid unnecessary resource consumption, for example by limiting its frequency of execution.

The self-monitoring thread is of considerable importance. Failure of autonomic service agents is highly undesirable. To a large extent this depends on careful coding and testing, with no unhandled error returns or exceptions, but if there is nevertheless some sort of failure, the self-monitoring thread should detect it, fix it if possible, and in the worst case restart the entire ASA.

Appendix C presents some example logic flows in informal pseudocode.

3. Interaction with the Autonomic Networking Infrastructure

3.1. Interaction with the security mechanisms

An ASA by definition runs in an autonomic node. Before any normal ASAs are started, such nodes must be bootstrapped into the autonomic network's secure key infrastructure, typically in accordance with [RFC8995]. This key infrastructure will be used to secure the ACP (next section) and may be used by ASAs to set up additional secure interactions with their peers, if needed.

Note that the secure bootstrap process itself incorporates simple special-purpose ASAs that use a restricted mode of GRASP (Section 4 of [RFC8995]).

3.2. Interaction with the Autonomic Control Plane

In a normal autonomic network, ASAs will run as clients of the ACP, which will provide a fully secured network environment for all communication with other ASAs, in most cases mediated by GRASP (next section).

Note that the ACP formation process itself incorporates simple special-purpose ASAs that use a restricted mode of GRASP (Section 6.4 of [RFC8994]).

3.3. Interaction with GRASP and its API

In a node where a significant number of ASAs are installed, GRASP [RFC8990] is likely to run as a separate process with its API [RFC8991] available in user space. Thus, ASAs may operate without special privilege, unless they need it for other reasons. The ASA's view of GRASP is built around GRASP objectives (Section 5), defined as data structures containing administrative information such as the objective's unique name, and its current value. The format and size of the value is not restricted by the protocol, except that it must be possible to serialise it for transmission in Concise Binary Object Representation (CBOR) [RFC8949], subject only to GRASP's maximum message size as discussed in Section 5.

As discussed in Section 2, GRASP is an asynchronous protocol, and this document uses a multi-threading model to describe operations. In many programming environments, an 'event loop' model is used instead, in which case each thread would be implemented as an event handler called in turn by the main loop. For this case, the GRASP API must provide non-blocking calls and possibly support callbacks. This topic is discussed in more detail in [RFC8991], and other asynchronicity models are also possible. Whenever necessary, the GRASP session identifier will be used to distinguish simultaneous operations.

The GRASP API should offer the following features:

- * Registration functions, so that an ASA can register itself and the objectives that it manages.
- * A discovery function, by which an ASA can discover other ASAs supporting a given objective.
- * A negotiation request function, by which an ASA can start negotiation of an objective with a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to negotiation requests, and a set of functions to support negotiating steps. Once a negotiation starts, it is a symmetric process with both sides sending successive objective values to each other until agreement is reached (or the negotiation fails).

- * A synchronization function, by which an ASA can request the current value of an objective from a counterpart ASA. With this, there is a corresponding listening function for an ASA that wishes to respond to synchronization requests. Unlike negotiation, synchronization is an asymmetric process in which the listener sends a single objective value to the requester.
- * A flood function, by which an ASA can cause the current value of an objective to be flooded throughout the AN so that any ASA can receive it.

For further details and some additional housekeeping functions, see [RFC8991].

The GRASP API is intended to support the various interactions expected between most ASAs, such as the interactions outlined in Section 2. However, if ASAs require additional communication between themselves, they can do so directly over the ACP to benefit from its security. One option is to use GRASP discovery and synchronization as a rendez-vous mechanism between two ASAs, passing communication parameters such as a TCP port number via GRASP. The use of TLS over the ACP for such communications is advisable, as described in Section 6.9.2 of [RFC8994].

3.4. Interaction with policy mechanisms

At the time of writing, the policy mechanisms for the ANI are undefined. In particular, the use of declarative policies (aka Intents) for the definition and management of ASA's behaviors remains a research topic [I-D.irtf-nmrg-ibn-concepts-definitions].

In the cases where ASAs are defined as closed control loops, the specifications defined in [ZSM009-1] regarding imperative and declarative goal statements may be applicable.

In the ANI, policy dissemination is expected to operate by an information distribution mechanism (e.g. via GRASP [RFC8990]) that can reach all autonomic nodes, and therefore every ASA. However, each ASA must be capable of operating "out of the box" in the absence of locally defined policy, so every ASA implementation must include carefully chosen default values and settings for all policy parameters.

4. Interaction with Non-Autonomic Components and Systems

An ASA, to have any external effects, must also interact with non-autonomic components of the node where it is installed. For example, an ASA whose purpose is to manage a resource must interact with that resource. An ASA managing an entity that is also managed by local software must interact with that software. For example, if such management is performed by NETCONF [RFC6241], the ASA must interact with the NETCONF server as an independent NETCONF client in the same node to avoid any inconsistency between configuration changes delivered via NETCONF and configuration changes made by the ASA.

In an environment where systems are virtualized and specialized using techniques such as network function virtualization or network slicing, there will be a design choice whether ASAs are deployed once per physical node or once per virtual context. A related issue is whether the ANI as a whole is deployed once on a physical network, or whether several virtual ANIs are deployed. This aspect needs to be considered by the ASA designer.

5. Design of GRASP Objectives

The design of an ASA will often require the design of a new GRASP objective. The general rules for the format of GRASP objectives, their names, and IANA registration are given in [RFC8990]. Additionally, that document discusses various general considerations for the design of objectives, which are not repeated here. However, note that the GRASP protocol, like HTTP, does not provide transactional integrity. In particular, steps in a GRASP negotiation are not idempotent. The design of a GRASP objective and the logic flow of the ASA should take this into account. One approach, which should be used when possible, is to design objectives with idempotent semantics. If this is not possible, typically if an ASA is allocating part of a shared resource to other ASAs, it needs to ensure that the same part of the resource is not allocated twice. The easiest way is to run only one negotiation at a time. If an ASA is capable of overlapping several negotiations, it must avoid interference between these negotiations.

Negotiations will always end, normally because one end or the other declares success or failure. If this does not happen, either a timeout or exhaustion of the loop count will occur. The definition of a GRASP objective should describe a specific negotiation policy if it is not self-evident.

GRASP allows a 'dry run' mode of negotiation, where a negotiation session follows its normal course but is not committed at either end until a subsequent live negotiation session. If 'dry run' mode is

defined for the objective, its specification, and every implementation, must consider what state needs to be saved following a dry run negotiation, such that a subsequent live negotiation can be expected to succeed. It must be clear how long this state is kept, and what happens if the live negotiation occurs after this state is deleted. An ASA that requests a dry run negotiation must take account of the possibility that a successful dry run is followed by a failed live negotiation. Because of these complexities, the dry run mechanism should only be supported by objectives and ASAs where there is a significant benefit from it.

The actual value field of an objective is limited by the GRASP protocol definition to any data structure that can be expressed in Concise Binary Object Representation (CBOR) [RFC8949]. For some objectives, a single data item will suffice; for example an integer, a floating point number, a UTF-8 string or an arbitrary byte string. For more complex cases, a simple tuple structure such as [item1, item2, item3] could be used. Since CBOR is closely linked to JSON, it is also rather easy to define an objective whose value is a JSON structure. The formats acceptable by the GRASP API will limit the options in practice. A generic solution is for the API to accept and deliver the value field in raw CBOR, with the ASA itself encoding and decoding it via a CBOR library (section 2.3.2.4 of [RFC8991]).

The maximum size of the value field of an objective is limited by the GRASP maximum message size. If the default maximum size specified as GRASP_DEF_MAX_SIZE by [RFC8990] is not enough, the specification of the objective must indicate the required maximum message size, both for unicast and multicast messages.

A mapping from YANG to CBOR is defined by [I-D.ietf-core-yang-cbor]. Subject to the size limit defined for GRASP messages, nothing prevents objectives transporting YANG in this way.

The flexibility of CBOR implies that the value field of many objectives can be extended in service, to add additional information or alternative content, especially if JSON-like structures are used. This has consequences for the robustness of ASAs, as discussed in Section 8.

6. Life Cycle

The ASA life cycle was discussed in [I-D.peloso-anima-autonomic-function], from which the following text was derived. It does not cover all details, and some of the terms used would require precise definitions in a given implementation.

In simple cases, Autonomic functions could be permanent, in the sense that ASAs are shipped as part of a product and persist throughout the product's life. However, in complex cases, a more likely situation is that ASAs need to be installed or updated dynamically, because of new requirements or bugs. This section describes one approach to the resulting life cycle of individual ASAs. It does not consider wider issues such as updates of shared libraries.

Because continuity of service is fundamental to autonomic networking, the process of seamlessly replacing a running instance of an ASA with a new version needs to be part of the ASA's design. The implication of service continuity on the design of ASAs can be illustrated along the three main phases of the ASA life cycle, namely Installation, Instantiation and Operation.

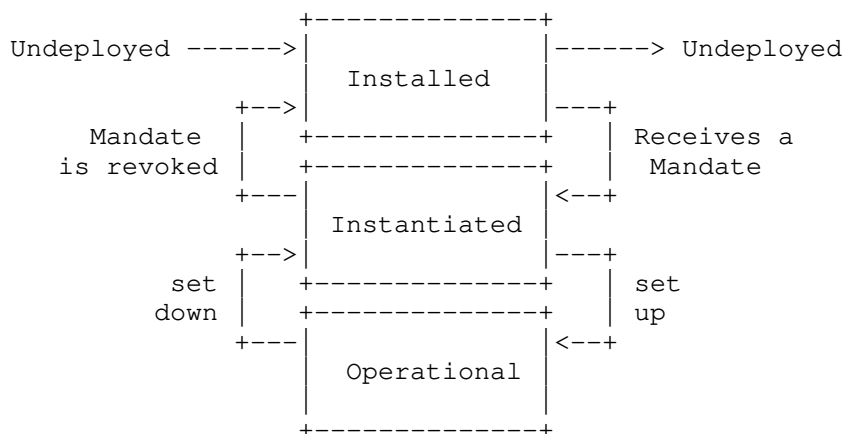


Figure 1: Life Cycle of an Autonomic Service Agent

6.1. Installation phase

We define "installation" to mean that a piece of software is loaded into a device, along with any necessary libraries, but is not yet activated.

Before being able to instantiate and run ASAs, the operator will first provision the infrastructure with the sets of ASA software corresponding to its needs and objectives. Such software must be checked for integrity and authenticity before installation. The provisioning of the infrastructure is realized in the installation phase and consists of installing (or checking the availability of) the pieces of software of the different ASAs in a set of Installation Hosts within the autonomic network.

There are 3 properties applicable to the installation of ASAs:

- * The dynamic installation property allows installing an ASA on demand, on any hosts compatible with the ASA.
- * The decoupling property allows an ASA on one machine to control resources in another machine (known as "decoupled mode").
- * The multiplicity property allows controlling multiple sets of resources from a single ASA.

These three properties are very important in the context of the installation phase as their variations condition how the ASA could be installed on the infrastructure.

6.1.1. Installation phase inputs and outputs

Inputs are:

- * [ASA_type] specifies which ASA to install.
- * [Installation_target_infrastructure] specifies the candidate installation Hosts.
- * [ASA_placement_function] specifies how the installation phase will meet the operator's needs and objectives for the provision of the infrastructure. This function is only useful in the decoupled mode. It can be as simple as an explicit list of hosts on which the ASAs are to be installed, or it could consist of operator-defined criteria and constraints.

The main output of the installation phase is a [List_of_ASAs] installed on [List_of_hosts]. This output is also useful for the coordination function where it acts as a static interaction map (see Section 7.1).

The condition to validate in order to pass to next phase is to ensure that [List_of_ASAs] are correctly installed on [List_of_hosts]. A minimum set of primitives to support the installation of ASAs could be: `install(List_of_ASAs, Installation_target_infrastructure, ASA_placement_function)`, and `uninstall (List_of_ASAs)`.

6.2. Instantiation phase

We define "instantiation" as the operation of creating a single ASA instance from the corresponding piece of installed software.

Once the ASAs are installed on the appropriate hosts in the network, these ASAs may start to operate. From the operator viewpoint, an operating ASA means the ASA manages the network resources as per the objectives given. At the ASA local level, operating means executing their control loop algorithm.

There are two aspects to take into consideration. First, having a piece of code installed and available to run on a host is not the same as having an agent based on this piece of code running inside the host. Second, in a coupled case, determining which resources are controlled by an ASA is straightforward (the ASA runs on the same autonomic node as the resources it is controlling); in a decoupled mode determining this is a bit more complex: a starting agent will have to either discover the set of resources it ought to control, or such information has to be communicated to the ASA.

The instantiation phase of an ASA covers both these aspects: starting the agent code (when this does not start automatically) and determining which resources have to be controlled (when this is not straightforward).

6.2.1. Operator's goal

Through this phase, the operator wants to control its autonomic network regarding at least two aspects:

- 1 determine the scope of autonomic functions by instructing which network resources have to be managed by which autonomic function (and more precisely by which release of the ASA software code, e.g., version number or provider),
- 2 determine how the autonomic functions are organized by instantiating a set of ASAs across one or more autonomic nodes and instructing them accordingly about the other ASAs in the set as necessary.

In this phase, the operator may also want to set goals for autonomic functions, e.g., by configuring GRASP objectives.

The operator's goal can be summarized in an instruction to the autonomic ecosystem matching the following format, explained in detail in the next sub-section:

```
[Instances_of_ASA_type] ready to control
[Instantiation_target_infrastructure] with
[Instantiation_target_parameters]
```

6.2.2. Instantiation phase inputs and outputs

Inputs are:

- * [Instances_of_ASA_type] that specifies which ASAs to instantiate
- * [Instantiation_target_infrastructure] that specifies which are the resources to be managed by the autonomic function; this can be the whole network or a subset of it like a domain, a physical segment or even a specific list of resources,
- * [Instantiation_target_parameters] that specifies which are the GRASP objectives to be sent to ASAs (e.g., an optimization target)

Outputs are:

- * [Set_of_ASA_resources_relations] describing which resources are managed by which ASA instances; this is not a formal message, but a resulting configuration log for a set of ASAs.

6.2.3. Instantiation phase requirements

The instructions described in Section 6.2 could be either:

- * Sent to a targeted ASA. In the case, the receiving Agent will have to manage the specified list of [Instantiation_target_infrastructure], with the [Instantiation_target_parameters].
- * Broadcast to all ASAs. In this case, the ASAs would determine from the list which ASAs would handle which [Instantiation_target_infrastructure], with the [Instantiation_target_parameters].

These instructions may be grouped as a specific data structure, referred to as an ASA Instance Mandate. The specification of such an ASA Instance Mandate is beyond the scope of this document.

The conclusion of this instantiation phase is a set of ASA instances ready to operate. These ASA instances are characterized by the resources they manage, the metrics being monitored and the actions that can be executed (like modifying certain parameters values). The description of the ASA instance may be defined in an ASA Instance Manifest data structure. The specification of such an ASA Instance Manifest is beyond the scope of this document.

The ASA Instance Manifest does not only serve informational purposes such as acknowledgement of successful instantiation to the operator, but is also necessary for further autonomic operations with:

- * coordinated entities (see Section 7.1)
- * collaborative entities with purposes such as to establish knowledge exchange (some ASAs may produce knowledge or monitor metrics that would be useful for other ASAs)

6.3. Operation phase

During the Operation phase, the operator can:

- * Activate/Deactivate ASAs: enable/disable their autonomic loops.
- * Modify ASAs targets: set different technical objectives.
- * Modify ASAs managed resources: update the instance mandate to specify a different set of resources to manage (only applicable to decoupled ASAs).

During the Operation phase, running ASAs can interact with other ASAs:

- * in order to exchange knowledge (e.g. an ASA providing traffic predictions to a load balancing ASA)
- * in order to collaboratively reach an objective (e.g. ASAs pertaining to the same autonomic function will collaborate, e.g., in the case of a load balancing function, by modifying link metrics according to neighboring resource loads)

During the Operation phase, running ASAs are expected to apply coordination schemes as per Section 7.1.

6.4. Removal phase

When an ASA is removed from service and uninstalled, the above steps are reversed. It is important that its data, especially any security key material, is purged.

7. Coordination and Data Models

7.1. Coordination between Autonomic Functions

Some autonomic functions will be completely independent of each other. However, others are at risk of interfering with each other - for example, two different optimization functions might both attempt to modify the same underlying parameter in different ways. In a complete system, a method is needed of identifying ASAs that might interfere with each other and coordinating their actions when necessary.

7.2. Coordination with Traditional Management Functions

Some ASAs will have functions that overlap with existing configuration tools and network management mechanisms such as command line interfaces, DHCP, DHCPv6, SNMP, NETCONF, and RESTCONF. This is of course an existing problem whenever multiple configuration tools are in use by the NOC. Each ASA designer will need to consider this issue and how to avoid clashes and inconsistencies in various deployment scenarios. Some specific considerations for interaction with OAM tools are given in [RFC8368]. As another example, [RFC8992] describes how autonomic management of IPv6 prefixes can interact with prefix delegation via DHCPv6. The description of a GRASP objective and of an ASA using it should include a discussion of any such interactions.

7.3. Data Models

Management functions often include a shared data model, quite likely to be expressed in a formal notation such as YANG. This aspect should not be an afterthought in the design of an ASA. To the contrary, the design of the ASA and of its GRASP objectives should match the data model; as noted in Section 5, YANG serialized as CBOR may be used directly as the value of a GRASP objective.

8. Robustness

It is of great importance that all components of an autonomic system are highly robust. Although ASA designers should aim for their component to never fail, it is more important to design the ASA to assume that failures will happen and to gracefully recover from those failures when they occur. Hence, this section lists various aspects of robustness that ASA designers should consider:

1. If despite all precautions, an ASA does encounter a fatal error, it should in any case restart automatically and try again. To mitigate a loop in case of persistent failure, a suitable pause should be inserted before such a restart. The length of the pause depends on the use case; randomization and exponential backoff should be considered.
2. If a newly received or calculated value for a parameter falls out of bounds, the corresponding parameter should be either left unchanged or restored to a value known to be safe in all configurations.
3. If a GRASP synchronization or negotiation session fails for any reason, it may be repeated after a suitable pause. The length of the pause depends on the use case; randomization and exponential backoff should be considered.
4. If a session fails repeatedly, the ASA should consider that its peer has failed, and cause GRASP to flush its discovery cache and repeat peer discovery.
5. In any case, it may be prudent to repeat discovery periodically, depending on the use case.
6. Any received GRASP message should be checked. If it is wrongly formatted, it should be ignored. Within a unicast session, an Invalid message (M_INVALID) may be sent. This function may be provided by the GRASP implementation itself.
7. Any received GRASP objective should be checked. Basic formatting errors like invalid CBOR will likely be detected by GRASP itself, but the ASA is responsible for checking the precise syntax and semantics of a received objective. If it is wrongly formatted, it should be ignored. Within a negotiation session, a Negotiation End message (M_END) with a Decline option (O_DECLINE) should be sent. An ASA may log such events for diagnostic purposes.

8. On the other hand, the definitions of GRASP objectives are very likely to be extended, using the flexibility of CBOR or JSON. Therefore, ASAs should be able to deal gracefully with unknown components within the values of objectives. The specification of an objective should describe how unknown components are to be handled (ignored, logged and ignored, or rejected as an error).
9. If an ASA receives either an Invalid message (M_INVALID) or a Negotiation End message (M_END) with a Decline option (O_DECLINE), one possible reason is that the peer ASA does not support a new feature of either GRASP or of the objective in question. In such a case the ASA may choose to repeat the operation concerned without using that new feature.
10. All other possible exceptions should be handled in an orderly way. There should be no such thing as an unhandled exception (but see point 1 above).

At a slightly more general level, ASAs are not services in themselves, but they automate services. This has a fundamental impact on how to design robust ASAs. In general, when an ASA observes a particular state (1) of operations of the services/resources it controls, it typically aims to improve this state to a better state, say (2). Ideally, the ASA is built so that it can ensure that any error encountered can still lead to returning to (1) instead of a state (3) which is worse than (1). One example instance of this principle is "make-before-break" used in reconfiguration of routing protocols in manual operations. This principle of operations can accordingly be coded into the operation of an ASA. The GRASP dry run option mentioned in Section 5 is another tool helpful for this ASA design goal of "test-before-make".

9. Security Considerations

ASAs are intended to run in an environment that is protected by the Autonomic Control Plane [RFC8994], admission to which depends on an initial secure bootstrap process such as BRSKI [RFC8995]. Those documents describe security considerations relating to the use of and properties provided by the ACP and BRSKI, respectively. Such an ACP can provide keying material for mutual authentication between ASAs as well as confidential communication channels for messages between ASAs. In some deployments, a secure partition of the link layer might be used instead. GRASP itself has significant security considerations [RFC8990]. However, this does not relieve ASAs of responsibility for security. When ASAs configure or manage network elements outside the ACP, potentially in a different physical node, they must interact with other non-autonomic software components to perform their management functions. The details are specific to each

case, but this has an important security implication. An ASA might act as a loophole by which the managed entity could penetrate the security boundary of the ANI. Thus, ASAs must be designed to avoid loopholes such as passing on executable code or proxying unverified commands, and should if possible operate in an unprivileged mode. In particular, they must use secure coding practices, e.g., carefully validate all incoming information and avoid unnecessary elevation of privilege. This will apply in particular when an ASA interacts with a management component such as a NETCONF server.

A similar situation will arise if an ASA acts as a gateway between two separate autonomic networks, i.e. it has access to two separate ACPs. Such an ASA must also be designed to avoid loopholes and to validate incoming information from both sides.

As a reminder, GRASP does not intrinsically provide transactional integrity (Section 5).

As appropriate to their specific functions, ASAs should take account of relevant privacy considerations [RFC6973].

The initial version of the autonomic infrastructure assumes that all autonomic nodes are trusted by virtue of their admission to the ACP. ASAs are therefore trusted to manipulate any GRASP objective, simply because they are installed on a node that has successfully joined the ACP. In the general case, a node may have multiple roles and a role may use multiple ASAs, each using multiple GRASP objectives. Additional mechanisms for the fine-grained authorization of nodes and ASAs to manipulate specific GRASP objectives could be designed. Meanwhile, we repeat that ASAs should run without special privilege if possible. Independently of this, interfaces between ASAs and the router configuration and monitoring services of the node can be subject to authentication that provides more fine-grained authorization for specific services. These additional authentication parameters could be passed to an ASA during its instantiation phase.

10. IANA Considerations

This document makes no request of the IANA.

11. Acknowledgements

Valuable comments were received from Michael Behringer, Menachem Dodge, Martin Dürst, Toerless Eckert, Thomas Fossati, Alex Galis, Bing Liu, Benno Overeinder, Michael Richardson, Rob Wilton and other IESG members.

12. References

12.1. Normative References

- [RFC8949] 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/info/rfc8949>>.
- [RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <<https://www.rfc-editor.org/info/rfc8990>>.
- [RFC8994] Eckert, T., Ed., Behringer, M., Ed., and S. Bjarnason, "An Autonomic Control Plane (ACP)", RFC 8994, DOI 10.17487/RFC8994, May 2021, <<https://www.rfc-editor.org/info/rfc8994>>.
- [RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.

12.2. Informative References

- [DeMola06] De Mola, F. and R. Quitadamo, "Towards an Agent Model for Future Autonomic Communications", Proceedings of the 7th WOA 2006 Workshop From Objects to Agents 51-59, September 2006.
- [GANA13] "Autonomic network engineering for the self-managing Future Internet (AFI): GANA Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management.", April 2013, <http://www.etsi.org/deliver/etsi_gs/AFI/001_099/002/01.01.01_60/gs_afi002v010101p.pdf>.
- [Huebscher08] Huebscher, M. C. and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications", ACM Computing Surveys (CSUR) Volume 40 Issue 3 DOI: 10.1145/1380584.1380585, August 2008.
- [I-D.ietf-core-yang-cbor] Veillette, M., Petrov, I., Pelov, A., Bormann, C., and M. Richardson, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-

cbor-18, 19 December 2021,
<<https://datatracker.ietf.org/doc/html/draft-ietf-core-yang-cbor-18>>.

[I-D.irtf-nmrg-ibn-concepts-definitions]

Clemm, A., Ciavaglia, L., Granville, L. Z., and J. Tantsura, "Intent-Based Networking - Concepts and Definitions", Work in Progress, Internet-Draft, draft-irtf-nmrg-ibn-concepts-definitions-06, 15 December 2021, <<https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-06>>.

[I-D.peloso-anima-autonomic-function]

Pierre, P. and L. Ciavaglia, "A Day in the Life of an Autonomic Function", Work in Progress, Internet-Draft, draft-peloso-anima-autonomic-function-01, 21 March 2016, <<https://datatracker.ietf.org/doc/html/draft-peloso-anima-autonomic-function-01>>.

[IPJ]

Behringer, M., Bormann, C., Carpenter, B. E., Eckert, T., Campos Nobre, J., Jiang, S., Li, Y., and M. C. Richardson, "Autonomic Networking Gets Serious", The Internet Protocol Journal Volume: 24 , Issue: 3, ISSN 1944-1134, Page(s): 2 - 18, October 2021, <<https://ipj.dreamhosters.com/wp-content/uploads/2021/10/243-ipj.pdf>>.

[Movahedi12]

Movahedi, Z., Ayari, M., Langar, R., and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria", IEEE Communications Surveys & Tutorials Volume: 14 , Issue: 2 DOI: 10.1109/SURV.2011.042711.00078, Page(s): 464 - 490, 2012.

[NFV]

"Network Functions Virtualisation - Introductory White Paper", SDN and OpenFlow World Congress, Darmstadt, Germany 1-16, October 2012, <http://portal.etsi.org/NFV/NFV_White_Paper.pdf>.

[RFC6241]

Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

[RFC6973]

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.

- [RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <<https://www.rfc-editor.org/info/rfc7575>>.
- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RFC8368] Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <<https://www.rfc-editor.org/info/rfc8368>>.
- [RFC8991] Carpenter, B., Liu, B., Ed., Wang, W., and X. Gong, "GeneRic Autonomic Signaling Protocol Application Program Interface (GRASP API)", RFC 8991, DOI 10.17487/RFC8991, May 2021, <<https://www.rfc-editor.org/info/rfc8991>>.
- [RFC8992] Jiang, S., Ed., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-Scale Networks", RFC 8992, DOI 10.17487/RFC8992, May 2021, <<https://www.rfc-editor.org/info/rfc8992>>.
- [RFC8993] Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <<https://www.rfc-editor.org/info/rfc8993>>.
- [ZSM009-1] "Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers", June 2021, <https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/00901/01.01.01_60/gs_ZSM00901v010101p.pdf>.

Appendix A. Change log

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

draft-ietf-anima-asa-guidelines-07, 2022-02-01:

* Editorial

draft-ietf-anima-asa-guidelines-06, 2022-01-27:

- * Clarified two sentences about special-purpose ASAs (Section 3.1, Section 3.2).
- * Fixed indentation bug and added one statement to MAIN PROGRAM pseudocode.
- * Removed mention of software image servers in section 6.1, which confused the reader.
- * Other improvements from IESG reviews.

draft-ietf-anima-asa-guidelines-05, 2021-12-20:

- * Clarified NETCONF wording.
- * Removed <CODE BEGINS> on advice from IETF Trust
- * Noted resource limits in constrained nodes
- * Strengthened text on data integrity in resource management example
- * Strengthen discussion of extensibility of GRASP objectives.
- * Other editorial improvements from IETF Last Call reviews

draft-ietf-anima-asa-guidelines-04, 2021-11-20:

- * Added terminology appendix
- * Further clarified discussion of asynch operations
- * Other editorial improvements from AD review

draft-ietf-anima-asa-guidelines-03, 2021-11-07:

- * Added security consideration for gateway ASAs
- * Cite IPJ article

draft-ietf-anima-asa-guidelines-02, 2021-09-13:

- * Added note on maximum message size.
- * Editorial fixes

draft-ietf-anima-asa-guidelines-01, 2021-06-27:

- * Incorporated shepherd's review comments
- * Editorial fixes

draft-ietf-anima-asa-guidelines-00, 2020-11-14:

- * Adopted by WG
- * Editorial fixes

draft-carpenter-anima-asa-guidelines-09, 2020-07-25:

- * Additional text on future authorization.
- * Editorial fixes

draft-carpenter-anima-asa-guidelines-08, 2020-01-10:

- * Introduced notion of autonomic ecosystem.
- * Minor technical clarifications.
- * Converted to v3 format.

draft-carpenter-anima-asa-guidelines-07, 2019-07-17:

- * Improved explanation of threading vs event-loop
- * Other editorial improvements.

draft-carpenter-anima-asa-guidelines-06, 2018-01-07:

- * Expanded and improved example logic flow.
- * Editorial corrections.

draft-carpenter-anima-asa-guidelines-05, 2018-06-30:

- * Added section on relationship with non-autonomic components.
- * Editorial corrections.

draft-carpenter-anima-asa-guidelines-04, 2018-03-03:

- * Added note about simple ASAs.
- * Added note about NFV/SFC services.
- * Improved text about threading v event loop model
- * Added section about coordination with traditional tools.
- * Added appendix with example logic flow.

draft-carpenter-anima-asa-guidelines-03, 2017-10-25:

- * Added details on life cycle.
- * Added details on robustness.
- * Added co-authors.

draft-carpenter-anima-asa-guidelines-02, 2017-07-01:

- * Expanded description of event-loop case.
- * Added note about 'dry run' mode.

draft-carpenter-anima-asa-guidelines-01, 2017-01-06:

- * More sections filled in.

draft-carpenter-anima-asa-guidelines-00, 2016-09-30:

- * Initial version

Appendix B. Terminology

This appendix summarises various acronyms and terminology used in the document. Where no other reference is given, please consult [RFC8993] or [RFC7575].

- * **Autonomic:** Self-managing (self-configuring, self-protecting, self-healing, self-optimizing), but allowing high-level guidance by a central entity such as a NOC.
- * **Autonomic Function:** A function that adapts on its own to a changing environment.
- * **Autonomic Node:** A node that employs autonomic functions.
- * **ACP:** Autonomic Control Plane [RFC8994].
- * **AN:** Autonomic Network: A network of autonomic nodes, which interact directly with each other.
- * **ANI:** Autonomic Network Infrastructure.
- * **ASA:** Autonomic Service Agent. An agent installed on an autonomic node that implements an autonomic function, either partially (in the case of a distributed function) or completely.
- * **BRSKI:** Bootstrapping Remote Secure Key Infrastructure [RFC8995].
- * **CBOR:** Concise Binary Object Representation [RFC8949].
- * **GRASP:** Generic Autonomic Signaling Protocol [RFC8990].
- * **GRASP API:** GRASP Application Programming Interface [RFC8991].
- * **NOC:** Network Operations Center [RFC8368].
- * **Objective:** A GRASP technical objective is a data structure whose main contents are a name and a value. The value consists of a single configurable parameter or a set of parameters of some kind. [RFC8990].

Appendix C. Example Logic Flows

This appendix describes generic logic flows that combine to act as an Autonomic Service Agent (ASA) for resource management. Note that these are illustrative examples, and in no sense requirements. As long as the rules of GRASP are followed, a real implementation could be different. The reader is assumed to be familiar with GRASP [RFC8990] and its conceptual API [RFC8991].

A complete autonomic function for a distributed resource will consist of a number of instances of the ASA placed at relevant points in a network. Specific details will of course depend on the resource concerned. One example is IP address prefix management, as specified in [RFC8992]. In this case, an instance of the ASA will exist in each delegating router.

An underlying assumption is that there is an initial source of the resource in question, referred to here as an origin ASA. The other ASAs, known as delegators, obtain supplies of the resource from the origin, and then delegate quantities of the resource to consumers that request it, and recover it when no longer needed.

Another assumption is there is a set of network wide policy parameters, which the origin will provide to the delegators. These parameters will control how the delegators decide how much resource to provide to consumers. Thus, the ASA logic has two operating modes: origin and delegator. When running as an origin, it starts by obtaining a quantity of the resource from the NOC, and it acts as a source of policy parameters, via both GRASP flooding and GRASP synchronization. (In some scenarios, flooding or synchronization alone might be sufficient, but this example includes both.)

When running as a delegator, it starts with an empty resource pool, it acquires the policy parameters by GRASP synchronization, and it delegates quantities of the resource to consumers that request it. Both as an origin and as a delegator, when its pool is low it seeks quantities of the resource by requesting GRASP negotiation with peer ASAs. When its pool is sufficient, it hands out resource to peer ASAs in response to negotiation requests. Thus, over time, the initial resource pool held by the origin will be shared among all the delegators according to demand.

In theory a network could include any number of origins and any number of delegators, with the only condition being that each origin's initial resource pool is unique. A realistic scenario is to have exactly one origin and as many delegators as you like. A scenario with no origin is useless.

An implementation requirement is that resource pools are kept in stable storage. Otherwise, if a delegator exits for any reason, all the resources it has obtained or delegated are lost. If an origin exits, its entire spare pool is lost. The logic for using stable storage and for crash recovery is not included in the pseudocode below, which focuses on communication between ASAs. Since GRASP operations are not intrinsically idempotent, data integrity during failure scenarios is the responsibility of the ASA designer. This is a complex topic in its own right that is not discussed in the present document.

The description below does not implement GRASP's 'dry run' function. That would require temporarily marking any resource handed out in a dry run negotiation as reserved, until either the peer obtains it in a live run, or a suitable timeout occurs.

The main data structures used in each instance of the ASA are:

- * The `resource_pool`, for example an ordered list of available resources. Depending on the nature of the resource, units of resource are split when appropriate, and a background garbage collector recombines split resources if they are returned to the pool.
- * The `delegated_list`, where a delegator stores the resources it has given to subsidiary devices.

Possible main logic flows are below, using a threaded implementation model. As noted above, alternative approaches to asynchronous operations are possible. The transformation to an event loop model should be apparent - each thread would correspond to one event in the event loop.

The GRASP objectives are as follows:

- * `["EX1.Resource", flags, loop_count, value]` where the value depends on the resource concerned, but will typically include its size and identification.
- * `["EX1.Params", flags, loop_count, value]` where the value will be, for example, a JSON object defining the applicable parameters.

In the outline logic flows below, these objectives are represented simply by their names.

MAIN PROGRAM:

```
Create empty resource_pool (and an associated lock)
Create empty delegated_list
Determine whether to act as origin
if origin:
    Obtain initial resource_pool contents from NOC
    Obtain value of EX1.Params from NOC
Register ASA with GRASP
Register GRASP objectives EX1.Resource and EX1.Params
if origin:
    Start FLOODER thread to flood EX1.Params
    Start SYNCHRONIZER listener for EX1.Params
Start MAIN_NEGOTIATOR thread for EX1.Resource
if not origin:
    Obtain value of EX1.Params from GRASP flood or synchronization
    Start DELEGATOR thread
Start GARBAGE_COLLECTOR thread
good_peer = none
do forever:
    if resource_pool is low:
        Calculate amount A of resource needed
        Discover peers using GRASP M_DISCOVER / M_RESPONSE
        if good_peer in peers:
            peer = good_peer
        else:
            peer = #any choice among peers
        grasp.request_negotiate("EX1.Resource", peer)
        #i.e., send negotiation request
        Wait for response (M_NEGOTIATE, M_END or M_WAIT)
        if OK:
            if offered amount of resource sufficient:
                Send M_END + O_ACCEPT #negotiation succeeded
                Add resource to pool
                good_peer = peer      #remember this choice
            else:
                Send M_END + O_DECLINE #negotiation failed
                good_peer = none      #forget this choice
        sleep() #periodic timer suitable for application scenario
```

MAIN_NEGOTIATOR thread:

```
do forever:
    grasp.listen_negotiate("EX1.Resource")
    #i.e., wait for negotiation request
    Start a separate new NEGOTIATOR thread for requested amount A
```

NEGOTIATOR thread:

```
Request resource amount A from resource_pool
if not OK:
    while not OK and A > Amin:
        A = A-1
        Request resource amount A from resource_pool
if OK:
    Offer resource amount A to peer by GRASP M_NEGOTIATE
    if received M_END + O_ACCEPT:
        #negotiation succeeded
    elif received M_END + O_DECLINE or other error:
        #negotiation failed
    Return resource to resource_pool
else:
    Send M_END + O_DECLINE #negotiation failed
#thread exits
```

DELEGATOR thread:

```
do forever:
    Wait for request or release for resource amount A
    if request:
        Get resource amount A from resource_pool
        if OK:
            Delegate resource to consumer #atomic
            Record in delegated_list      #operation
        else:
            Signal failure to consumer
            Signal main thread that resource_pool is low
    else:
        Delete resource from delegated_list
        Return resource amount A to resource_pool
```

SYNCHRONIZER thread:

```
do forever:
    Wait for M_REQ_SYN message for EX1.Params
    Reply with M_SYNCH message for EX1.Params
```

FLOODER thread:

```
do forever:
    Send M_FLOOD message for EX1.Params
    sleep() #periodic timer suitable for application scenario
```


GARBAGE_COLLECTOR thread:

```
do forever:
    Search resource_pool for adjacent resources
    Merge adjacent resources
    sleep() #periodic timer suitable for application scenario
```

Authors' Addresses

Brian Carpenter
School of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Laurent Ciavaglia
Rakuten Mobile
Paris
France

Email: laurent.ciavaglia@rakuten.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14 Huawei Campus
156 Beiqing Road
Hai-Dian District
Beijing
100095
China

Email: jiangsheng@huawei.com

Pierre Peloso
Nokia
Villardeaux
91460 Nozay
France

Email: pierre.peloso@nokia.com

ANIMA WG
Internet-Draft
Intended status: Standards Track
Expires: July 11, 2021

S. Fries
H. Brockhaus
Siemens
E. Lear
Cisco Systems
T. Werner
Siemens
January 7, 2021

Support of asynchronous Enrollment in BRSKI (BRSKI-AE)
draft-ietf-anima-brski-async-enroll-01

Abstract

This document describes enhancements of bootstrapping a remote secure key infrastructure (BRSKI) to also operate in domains featuring no or only timely limited connectivity between involved components. Moreover, newly introduced are methods to perform the BRSKI approach in environments, in which the role of the pledge changes to a server instead of the client. This changes the interaction model as the pledge is pushed to interact with the registrar instead of pulling information from the registrar. To support both, BRSKI-AE relies on the exchange of it authenticated self-contained objects (signature-wrapped objects) also for requesting and distributing of domain specific device certificates. The defined approach is agnostic regarding the utilized enrollment protocol allowing the application of existing and potentially new certificate management protocols.

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 July 11, 2021.

Copyright Notice

Copyright (c) 2021 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 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
2. Terminology	6
3. Scope of solution	7
3.1. Supported environment	7
3.2. Application Examples	7
3.2.1. Rolling stock	7
3.2.2. Building automation	8
3.2.3. Substation automation	8
3.2.4. Electric vehicle charging infrastructure	9
3.2.5. Infrastructure isolation policy	9
3.2.6. Less operational security in the target domain	9
4. Requirement discussion and mapping to solution elements	9
5. Architectural Overview and Communication Exchanges	12
5.1. Use Case 1 (PULL): Support of off-site PKI service	12
5.1.1. Behavior of a pledge	15
5.1.2. Pledge - Registrar discovery and voucher exchange	16
5.1.3. Registrar - MASA voucher exchange	16
5.1.4. Pledge - Registrar - RA/CA certificate enrollment	16
5.1.5. Addressing Scheme Enhancements	19
5.2. Use Case 2 (PUSH): pledge-agent	19
5.2.1. Behavior of a pledge	23
5.2.2. Behavior of a pledge-agent	24
5.2.3. Protocol Details (Pledge-Agent - Pledge)	25
5.2.4. Protocol flow	29
5.3. Domain registrar support of different enrollment options	31
6. Example for signature-wrapping using existing enrollment protocols	32
6.1. EST Handling	33
6.2. Lightweight CMP Handling	33
7. IANA Considerations	34
8. Privacy Considerations	34

9. Security Considerations	34
9.1. Exhaustion attack on pledge	34
9.2. PSK usage in TLS establishment	34
9.3. Misuse of acquired voucher and enrollment responses	35
10. Acknowledgments	35
11. References	35
11.1. Normative References	35
11.2. Informative References	36
Appendix A. History of changes [RFC Editor: please delete]	38
Authors' Addresses	40

1. Introduction

BRSKI as defined in [I-D.ietf-anima-bootstrapping-keyinfra] specifies a solution for secure zero-touch (automated) bootstrapping of devices (pledges) in a deployment domain. This includes the discovery of network elements in the target domain, time synchronization, and the exchange of security information necessary to establish trust between a pledge and the domain and to adopt a pledge as new network and application element. Security information about the target domain, specifically the target domain certificate, is exchanged utilizing voucher objects as defined in [RFC8366]. These vouchers are authenticated self-contained (signed) objects, which may be provided online (synchronous) or offline (asynchronous) via the domain registrar to the pledge and originate from a Manufacturer's Authorized Signing Authority (MASA). The MASA signed voucher contains the target domain certificate and can be verified by the pledge due to the possession of a manufacturer root certificate. It facilitates the enrollment of the pledge in the target domain and is used to establish trust from the pledge to the domain.

For the enrollment of devices BRSKI relies on EST [RFC7030] to request and distribute target domain specific device certificates. EST in turn relies on a binding of the certification request to an underlying TLS connection between the EST client and the EST server. According to BRSKI the domain registrar acts as EST server and is also acting as registration authority (RA) or local registration authority (LRA). The binding to TLS is used to protect the exchange of a certification request (for an LDevID certificate) and to provide data origin authentication to support the authorization decision for processing the certification request. The TLS connection is mutually authenticated and the client side authentication utilizes the pledge's manufacturer issued device certificate (IDevID certificate). This approach requires an on-site availability of a local asset or inventory management system performing the authorization decision based on tuple of the certification request and the pledge authentication using the IDevID certificate, to issue a domain specific certificate to the pledge. The EST server (the domain

registrar) terminates the security association with the pledge and thus the binding between the certification request and the authentication of the pledge via TLS. This type of enrollment utilizing an online connection to the PKI is considered as synchronous enrollment.

For certain use cases on-site support of a RA/CA component and/or an asset management is not available and rather provided by an operator's backend and may be provided timely limited or completely through offline interactions. This may be due to higher security requirements for operating the certification authority or for optimization of operation for smaller deployments to avoid the always on-site operation. The authorization of a certification request based on an asset management in this case will not / can not be performed on-site at enrollment time. Enrollment, which cannot be performed in a (timely) consistent fashion is considered as asynchronous enrollment in this document. It requires the support of a store and forward functionality of certification request together with the requester authentication information. This enables processing of the request at a later point in time. A similar situation may occur through network segmentation, which is utilized in industrial systems to separate domains with different security needs. Here, a similar requirement arises if the communication channel carrying the requester authentication is terminated before the RA/CA authorization handling of the certification request. If a second communication channel is opened to forward the certification request to the issuing RA/ CA, the requester authentication information needs to be retained and ideally bound to the certification request. This use case is independent from timely limitations of the first use case. For both cases, it is assumed that the requester authentication information is utilized in the process of authorization of a certification request. There are different options to perform store and forward of certification requests including the requester authentication information:

- o Providing a trusted component (e.g., an LRA) in the target domain, which stores the certification request combined with the requester authentication information (based on the IDevID) and potentially the information about a successful proof of possession (of the corresponding private key) in a way prohibiting changes to the combined information. Note that the assumption is that the information elements may not be cryptographically bound together. Once connectivity to the backend is available, the trusted component forwards the certification request together with the requester information (authentication and proof of possession) to the off-site PKI for further processing. It is assumed that the off-site PKI in this case relies on the local pledge authentication result and thus performs the authorization and

issues the requested certificate. In BRSKI the trusted component may be the EST server residing co-located with the registrar in the target domain.

- o Utilization of authenticated self-contained objects for the enrollment, binding the certification request and the requester authentication in a cryptographic way. This approach reduces the necessary trust in a domain component to storage and delivery. Unauthorized modifications of the requester information (request and authentication) can be detected during the verification of the authenticated self-contained object.

This targets environments, in which connectivity to a PKI is only temporary or not directly available, by specifying support for handling authenticated self-contained objects for enrollment. As it is intended to enhance BRSKI it is named BRSKI-AE, where AE stands for asynchronous enrollment. As BRSKI, BRSKI-AE results in the pledge storing an X.509 root certificate and sufficient for verifying the domain registrar / proxy identity (LDevID CA Certificate) as well as a domain specific X.509 device certificate (LDevID EE certificate).

Based on the proposed approach, a second set of scenarios can be addressed, in which the pledge has either no direct communication path to the domain registrar, e.g., due to missing network connectivity or a different technology stack. In such scenarios the pledge is likely to act as a server rather than a client. It will be pushed (triggered) by the registrar or an intermediate component to generate request objects to be onboarded in the registrar's domain. For this, an additional component is introduced acting as an agent for the pledge towards the domain registrar, e.g., a commissioning tool. In contrast to BRSKI here the objects to trigger a request generation and the responses are pushed to the pledge instead of being pulled as done in BRSKI.

The goal is to enhance BRSKI to either allow other existing certificate management protocols supporting authenticated self-contained objects to be applied or to allow other types of encoding for the certificate management information exchange. This is addressed by

- o enhancing the well-known URI approach with an additional path for the utilized enrollment protocol.
- o defining a certificate waiting indication and handling, if the certifying component is (temporarily) not available.

- o allowing to utilize credentials different from the pledge's IDevID to establish a TLS connection to the domain registrar, which is necessary in case of using a pledge-agent.

Note that in contrast to BRSKI, BRSKI-AE assumes support of multiple enrollment protocols on the infrastructure side, allowing the pledge manufacturer to select the most appropriate. Thus, BRSKI-AE can be applied for both, asynchronous and synchronous enrollment.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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. The following terms are defined additionally:

CA: Certification authority, issues certificates.

RA: Registration authority, an optional system component to which a CA delegates certificate management functions such as authorization checks.

LRA: Local registration authority, an optional RA system component with proximity to end entities.

IED: Intelligent Electronic Device (in essence a pledge).

on-site: Describes a component or service or functionality available in the target deployment domain.

off-site: Describes a component or service or functionality available in an operator domain different from the target deployment domain. This may be a central site, to which only a temporary connection is available, or which is in a different administrative domain.

asynchronous communication: Describes a timely interrupted communication between an end entity and a PKI component.

synchronous communication: Describes a timely uninterrupted communication between an end entity and a PKI component.

authenticated self-contained object: Describes an object, which is cryptographically bound to the IDevID EE certificate of a pledge. The binding is assumed to be provided through a digital signature of the actual object using the corresponding private key of the IDevID.

3. Scope of solution

3.1. Supported environment

This solution is intended to be used in domains with limited support of on-site PKI services and comprises use cases in which:

- o there is no registration authority available in the target domain. The connectivity to an off-site RA in an operator's network may only be available temporarily. A local store and forward device is used for the communication with the off-site services.
- o authoritative actions of a LRA are limited and may not comprise authorization of certification requests or pledges. Final authorization is done at the RA residing in the operator domain.
- o the target deployment domain already has an established certificate management approach that shall be reused to (e.g., in brownfield installations).

In addition, the solution is intended to be applicable in domains in which pledges have no direct connection to the domain registrar, but are expected to be managed by the registrar. This can be motivated by pledges featuring a different technology stack or by pledges without an existing connection to the domain registrar during onboarding. These pledges are likely to act in a server role. Therefore, the pledge needs to provide endpoints on which it can be triggered for requesting the generation of voucher-request objects and certification objects as well as to provide the response objects to the pledge. Here the pledge is not expected to start a communication with the domain registrar for onboarding, but is expected to be pushed for the interaction.

3.2. Application Examples

The following examples are intended to motivate the support of different enrollment approaches in general and asynchronous enrollment specifically, by introducing industrial applications cases, which could leverage BRSKI as such but also require support of asynchronous operation as intended with BRSKI-AE.

3.2.1. Rolling stock

Rolling stock or railroad cars contain a variety of sensors, actuators, and controllers, which communicate within the railroad car but also exchange information between railroad cars building a train, or with a backend. These devices are typically unaware of backend connectivity. Managing certificates may be done during maintenance

cycles of the railroad car, but can already be prepared during operation. The preparation may comprise the generation of certification requests by the components which are collected and forwarded for processing, once the railroad car is connected to the operator backend. The authorization of the certification request is then done based on the operator's asset/inventory information in the backend.

3.2.2. Building automation

In building automation, a use case can be described by a detached building or the basement of a building equipped with sensor, actuators, and controllers connected, but with only limited or no connection to the centralized building management system. This limited connectivity may be during the installation time but also during operation time. During the installation in the basement, a service technician collects the necessary information from the basement network and provides them to the central building management system, e.g., using a laptop or even a mobile phone to transport the information. This information may comprise parameters and settings required in the operational phase of the sensors/actuators, like a certificate issued by the operator to authenticate against other components and services.

The collected information may be provided by a domain registrar already existing in the installation network. In this case connectivity to the backend PKI may be facilitated by the service technician's laptop. Contrary, the information can also be collected from the pledges directly and provided to a domain registrar deployed in a different network. In this cases connectivity to the domain registrar may be facilitated by the service technician's laptop.

3.2.3. Substation automation

In electrical substation automation a control center typically hosts PKI services to issue certificates for Intelligent Electronic Devices (IED)s operated in a substation. Communication between the substation and control center is done through a proxy/gateway/DMZ, which terminates protocol flows. Note that [NERC-CIP-005-5] requires inspection of protocols at the boundary of a security perimeter (the substation in this case). In addition, security management in substation automation assumes central support of different enrollment protocols to facilitate the capabilities of IEDs from different vendors. The IEC standard IEC62351-9 [IEC-62351-9] specifies the mandatory support of two enrollment protocols, SCEP [RFC8894] and EST [RFC7030] for the infrastructure side, while the IED must only support one of the two.

3.2.4. Electric vehicle charging infrastructure

For the electric vehicle charging infrastructure protocols have been defined for the interaction between the electric vehicle (EV) and the charging point (e.g., ISO 15118-2 [ISO-IEC-15118-2]) as well as between the charging point and the charging point operator (e.g. OCPP [OCPP]). Depending on the authentication model, unilateral or mutual authentication is required. In both cases the charging point uses an X.509 certificate to authenticate itself in the context of a TLS connection between the EV and the charging point. The management of this certificate depends (beyond others) on the selected backend connectivity protocol. Specifically, in case of OCPP it is intended as single communication protocol between the charging point and the backend carrying all information to control the charging operations and maintain the charging point itself. This means that the certificate management is intended to be handled in-band of OCPP. This requires to be able to encapsulate the certificate management exchanges in a transport independent way. Authenticated self-containment will ease this by allowing the transport without a separate enrollment protocol.

3.2.5. Infrastructure isolation policy

This refers to any case in which network infrastructure is normally isolated from the Internet as a matter of policy, most likely for security reasons. In such a case, limited access to external PKI resources will be allowed in carefully controlled short periods of time, for example when a batch of new devices are deployed, but impossible at other times.

3.2.6. Less operational security in the target domain

The registration point performing the authorization of a certificate request is a critical PKI component and therefore implicates higher operational security than other components utilizing the issued certificates for their security features. CAs may also demand higher security in the registration procedures. Especially the CA/Browser forum currently increases the security requirements in the certificate issuance procedures for publicly trusted certificates. There may be the situation that the target domain does not offer enough security to operate a registration point and therefore wants to transfer this service to a backend.

4. Requirement discussion and mapping to solution elements

For the requirements discussion it is assumed that the domain registrar receiving a certification request as authenticated self-contained object is not the authorization point for this

certification request. If the domain registrar is the authorization point, BRSKI can be used directly. Note that BRSKI-AE could also be used in this case.

Based on the intended target environment described in Section 3.1 and the motivated application examples described in Section 3.2 the following base requirements are derived to support authenticated self-contained objects as container carrying the certification request and further information to support asynchronous operation.

At least the following properties are required:

- o Proof of Possession: proves to possess and control the private key corresponding to the public key contained in the certification request, typically by adding a signature using the private key.
- o Proof of Identity: provides data-origin authentication of a data object, e.g., a certificate request, utilizing an existing IDevID. Certificate updates may utilize the certificate that is to be updated.

Solution examples (not complete) based on existing technology are provided with the focus on existing IETF documents:

- o Certification request objects: Certification requests are structures protecting only the integrity of the contained data providing a proof-of-private-key-possession for locally generated key pairs. Examples for certification requests are:
 - * PKCS#10 [RFC2986]: Defines a structure for a certification request. The structure is signed to ensure integrity protection and proof of possession of the private key of the requester that corresponds to the contained public key.
 - * CRMF [RFC4211]: Defines a structure for the certification request message. The structure supports integrity protection and proof of possession, through a signature generated over parts of the structure by using the private key corresponding to the contained public key.

Note that the integrity of the certification request is bound to the public key contained in the certification request by performing the signature operation with the corresponding private key. In the considered application examples, this is not sufficient and needs to be bound to the existing credential of the pledge (IDevID) additionally. This binding supports the authorization decision for the certification request through the provisioning of a proof of identity. The binding of data origin

authentication to the certification request may be delegated to the protocol used for certificate management.

- o Proof of Identity options: The certification request should be bound to an existing credential (here IDevID) to enable a proof of identity and based on it an authorization of the certification request. The binding may be realized through security options in an underlying transport protocol if the authorization of the certification request is done at the next communication hop. Alternatively, this binding can be done by a wrapping signature employing an existing credential (initial: IDevID, renewal: LDevID). This requirement is addressed by existing enrollment protocols in different ways, for instance:
 - * EST [RFC7030]: Utilizes PKCS#10 to encode the certification request. The Certificate Signing Request (CSR) may contain a binding to the underlying TLS by including the tls-unique value in the self-signed CSR structure. The tls-unique value is one result of the TLS handshake. As the TLS handshake is performed mutually authenticated and the pledge utilized its IDevID for it, the proof of identity can be provided by the binding to the TLS session. This is supported in EST using simpleenroll. To avoid the binding to the underlying authentication in the transport layer, EST offers the support of a wrapping the CSR with an existing certificate by using Full PKI Request messages.
 - * SCEP [RFC8894]: Provides the option to utilize either an existing secret (password) or an existing certificate to protect the CSR based on SCEP Secure Message Objects using CMS wrapping ([RFC5652]). Note that the wrapping using an existing IDevID credential in SCEP is referred to as renewal. SCEP therefore does not rely on the security of an underlying transport.
 - * CMP [RFC4210] Provides the option to utilize either an existing secret (password) or an existing certificate to protect the PKIMessage containing the certification request. The certification request is encoded utilizing CRMF. PKCS#10 is optionally supported. The proof of identity of the PKIMessage containing the certification request can be achieved by using IDevID credentials to a PKIProtection carrying the actual signature value. CMP therefore does not rely on the security of an underlying transport protocol.
 - * CMC [RFC5272] Provides the option to utilize either an existing secret (password) or an existing certificate to protect the certification request (either in CRMF or PKCS#10) based on CMS

[RFC5652])). Here a FullCMCRequest can be used, which allows signing with an existing IDevID credential to provide a proof of identity. CMC therefore does not rely on the security of an underlying transport.

Note that besides the already existing enrollment protocols there is ongoing work in the ACE WG to define an encapsulation of EST messages in OSCORE to result in a TLS independent way of protecting EST. This approach [I-D.selander-ace-coap-est-oscure] may be considered as further variant.

5. Architectural Overview and Communication Exchanges

To support asynchronous enrollment, the base system architecture defined in BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] is enhanced to facilitate the two target use cases.

- o Use case 1 (PULL case): the pledge requests certificates from a PKI operated off-site via the domain registrar.
- o Use case 2 (PUSH case): allows delayed (delegated) onboarding using a pledge-agent instead a direct connection to the domain registrar. The communication model between pledge-agent and pledge is intended to use a PUSH approach in which the pledge acts as a server.

Note that the terminology PUSH and PULL relates to the pledge behavior. In PULL the pledge requests data objects as in BRSKI, while in the PUSH case the pledge is in the server role and will be provided with the data objects. The pledge-agent, as it represents the pledge, is expected to act in a PULL mode towards the domain registrar. Both use cases are described in the next subsections. They utilize the existing BRSKI architecture elements as much as possible. Necessary enhancements to support authenticated self-contained objects for certificate enrollment are kept on a minimum to ensure reuse of already defined architecture elements and interactions.

For the authenticated self-contained objects used for the certification request, BRSKI-AE relies on the defined message wrapping mechanisms of the enrollment protocols stated in Section 4 above.

5.1. Use Case 1 (PULL): Support of off-site PKI service

One assumption of BRSKI-AE is that the authorization of a certification request is performed based on an authenticated self-contained object, binding the certification request to the

authentication using the IDevID. This supports interaction with off-site or off-line PKI (RA/CA) components. In addition, the authorization of the certification request may not be done by the domain registrar but by a PKI residing in the backend of the domain operator (off-site) as described in Section 3.1. This leads to changes in the placement or enhancements of the logical elements as shown in Figure 1.

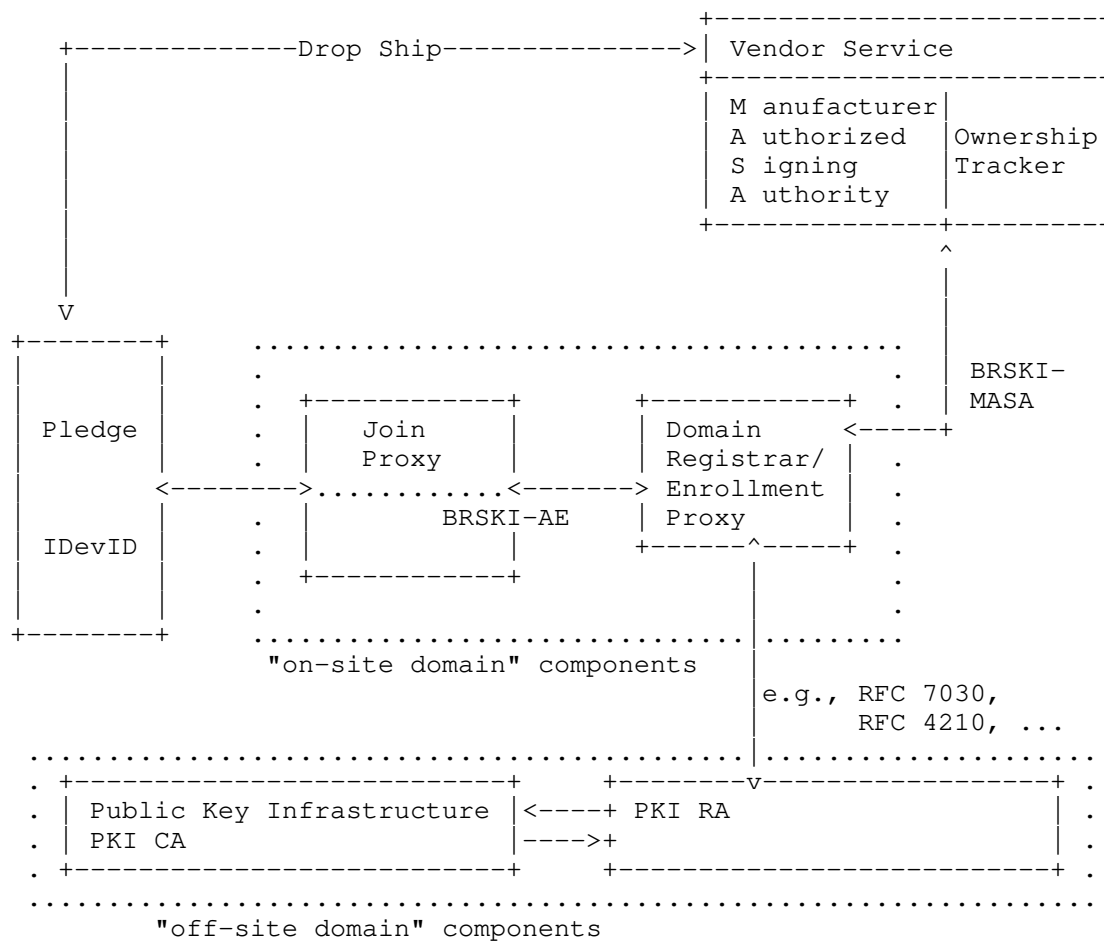


Figure 1: Architecture overview using off-site PKI components

The architecture overview in Figure 1 utilizes the same logical elements as BRSKI but with a different placement in the deployment architecture for some of the elements. The main difference is the placement of the PKI RA/CA component, which is performing the authorization decision for the certification request message. It is

placed in the off-site domain of the operator (not the deployment site directly), which may have no or only temporary connectivity to the deployment or on-site domain of the pledge. This is to underline the authorization decision for the certification request in the backend rather than on-site. The following list describes the components in the target domain:

- o Join Proxy: same functionality as described in BRSKI.
- o Domain Registrar / Enrollment Proxy: In general the domain registrar proxy has a similar functionality regarding the imprinting of the pledge in the deployment domain to facilitate the communication of the pledge with the MASA and the PKI. Different is the authorization of the certification request. BRSKI-AE allows to perform this in the operator's backend (off-site), and not directly at the domain registrar.
- * Voucher exchange: The voucher exchange with the MASA via the domain registrar is performed as described in BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] .
- * Certificate enrollment: For the pledge enrollment the domain registrar in the deployment domain supports the adoption of the pledge in the domain based on the voucher request. Nevertheless, it may not have sufficient information for authorizing the certification request. If the authorization of the certification request is done in the off-site domain, the domain registrar forwards the certification request to the RA to perform the authorization. Note that this requires, that the certification request object is enhanced with a proof-of-identity to allow the authorization based on the bound identity information of the pledge. As stated above, this can be done by an additional signature using the IDevID. The domain registrar here acts as an enrollment proxy or local registration authority. It is also able to handle the case having no connection temporarily to an off-site PKI, by storing the authenticated certification request and forwarding it to the RA upon reestablished connectivity. As authenticated self-contained objects are used, it requires an enhancement of the domain registrar. This is done by supporting alternative enrollment approaches (protocol options, protocols, encoding) by enhancing the addressing scheme to communicate with the domain registrar (see Section 5.1.5).

The following list describes the vendor related components/service outside the deployment domain:

- o MASA: general functionality as described in [I-D.ietf-anima-bootstrapping-keyinfra]. Assumption is that the interaction with the MASA may be synchronous (voucher request with nonce) or asynchronous (voucher request without nonce).
- o Ownership tracker: as defined in [I-D.ietf-anima-bootstrapping-keyinfra].

The following list describes the operator related components/service operated in the backend:

- o PKI RA: Performs certificate management functions (validation of certification requests, interaction with inventory/asset management for authorization of certification requests, etc.) for issuing, updating, and revoking certificates for a domain as a centralized infrastructure for the domain operator. The inventory (asset) management may be a separate component or integrated into the RA directly.
- o PKI CA: Performs certificate generation by signing the certificate structure provided in the certification request.

Based on BRSKI and the architectural changes the original protocol flow is divided into three phases showing commonalities and differences to the original approach as depicted in the following.

- o Discovery phase (same as BRSKI)
- o Voucher exchange with deployment domain registrar (same as BRSKI).
- o Enrollment phase (changed to support the application of authenticated self-contained objects).

5.1.1. Behavior of a pledge

The behavior of a pledge as described in [I-D.ietf-anima-bootstrapping-keyinfra] is kept with one exception. After finishing the imprinting phase (4) the enrollment phase (5) is performed with a method supporting authenticated self-contained objects. Using EST with simpleenroll cannot be applied here, as it binds the pledge authentication with the existing IDevID to the transport channel (TLS) rather than to the certification request object directly. This authentication in the transport layer is not visible / verifiable at the authorization point in the off-site domain. Section 6 discusses potential enrollment protocols and options applicable.

5.1.2. Pledge - Registrar discovery and voucher exchange

The discovery phase is applied as specified in [I-D.ietf-anima-bootstrapping-keyinfra].

5.1.3. Registrar - MASA voucher exchange

The voucher exchange is performed as specified in [I-D.ietf-anima-bootstrapping-keyinfra].

5.1.4. Pledge - Registrar - RA/CA certificate enrollment

As stated in Section 4 the enrollment shall be performed using an authenticated self-contained object providing proof of possession and proof of identity.

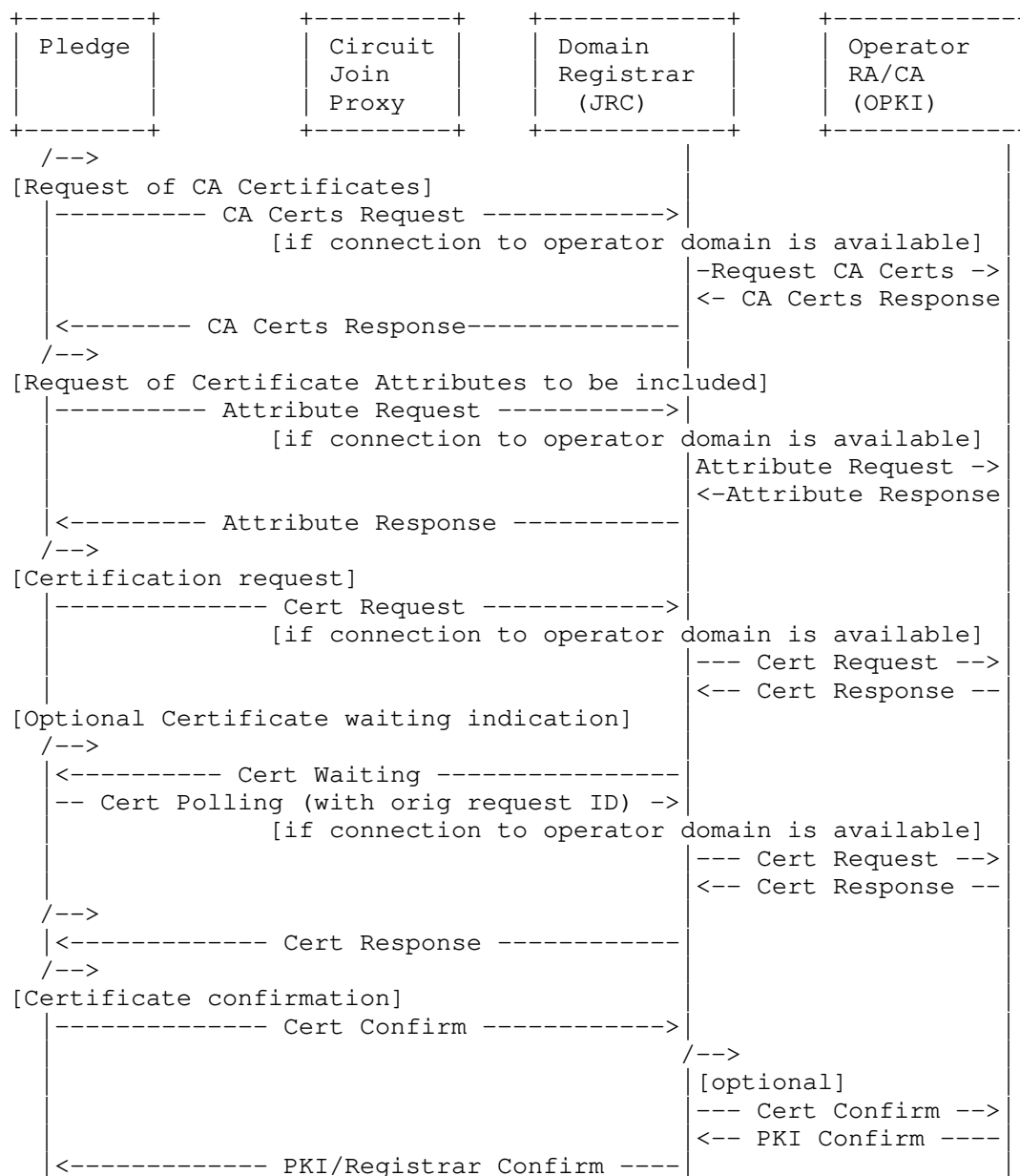


Figure 2: Certificate enrollment

The following list provides an abstract description of the flow depicted in Figure 2.

- o CA Cert Request: The pledge SHOULD request the full distribution of CA Certificates. This ensures that the pledge has the complete set of current CA certificates beyond the pinned-domain-cert (which may be the domain registrar certificate contained in the voucher).
- o CA Cert Response: Contains at least one CA certificate of the issuing CA.
- o Attribute Request: Typically, the automated bootstrapping occurs without local administrative configuration of the pledge. Nevertheless, there are cases, in which the pledge may also include additional attributes specific to the deployment domain into the certification request. To get these attributes in advance, the attribute request SHOULD be used.
- o Attribute Response: Contains the attributes to be included in the certification request message.
- o Cert Request: Depending on the utilized enrollment protocol, this certification request contains the authenticated self-contained object ensuring both, proof-of-possession of the corresponding private key and proof-of-identity of the requester.
- o Cert Response: certification response message containing the requested certificate and potentially further information like certificates of intermediary CAs on the certification path.
- o Cert Waiting: waiting indication for the pledge to retry after a given time. For this a request identifier is necessary. This request identifier may be either part of the enrollment protocol or build based on the certification request.
- o Cert Polling: querying the registrar, if the certificate request was already processed; can be answered either with another Cert Waiting, or a Cert Response.
- o Cert Confirm: confirmation message from pledge after receiving and verifying the certificate.
- o PKI/Registrar Confirm: confirmation message from PKI/registrar about reception of the pledge's certificate confirmation.

[RFC Editor: please delete] /*

Open Issues:

- o Description of certificate waiting and retries.

- o Message exchange description is expected to be done by the utilized enrollment protocol based on the addressing scheme (see also Section 6).
- o Handling of certificate/PKI confirmation message between pledge and domain registrar and PKI (treated optional?).

*/

5.1.5. Addressing Scheme Enhancements

BRSKI-AE requires enhancements to the addressing scheme defined in [I-D.ietf-anima-bootstrapping-keyinfra] to accommodate the additional handling of authenticated self-contained objects for the certification request. As this is supported by different enrollment protocols, they can be directly employed (see also Section 6). For the support of different enrollment options at the domain registrar, the addressing approach of BRSKI using a "/.well-known" tree from [RFC8615] is enhanced.

The current addressing scheme in BRSKI for the client certificate request function during the enrollment is using the definition from EST [RFC7030], here on the example on simple enroll: "/.well-known/est/simpleenroll" This approach is generalized to the following notation: "/.well-known/enrollment-protocol/request" in which enrollment-protocol may be an already existing protocol or a newly defined approach. Note that enrollment is considered here as a sequence of at least a certification request and a certification response. In case of existing enrollment protocols the following notation is used proving compatibility to BRSKI:

- o enrollment-protocol: references either EST [RFC7030] as in BRSKI or CMP, CMC, SCEP, or newly defined approaches as alternatives. Note: the IANA registration of the well-known URI is expected to be done by the enrollment protocol. For CMP an update is defined, which provides the definition of the well-known URI in CMP Updates Lightweight CMP Profile [I-D.ietf-lamps-cmp-updates].
- o request: depending on the utilized enrollment protocol, the request describes the required operation at the registrar side. Enrollment protocols are expected to define the request endpoints as done by existing protocols (see also Section 6).

5.2. Use Case 2 (PUSH): pledge-agent

To support mutual trust establishment of pledges, not directly connected to the domain registrar, a similar approach is applied as discussed for the use case 1. It relies on the exchange of

authenticated self-contained objects (the voucher request/response objects as known from BRSKI and the certification request/response objects as introduced by BRSKI-AE). This allows independence from the protection provided by the underlying transport.

In contrast to BRSKI, the exchange of these objects is performed with the help of a pledge-agent, supporting the interaction of the pledge with the domain registrar. It may be an integrated functionality of a commissioning tool. This leads to enhancements of the logical elements in the BRSKI architecture as shown in Figure 3. The pledge-agent interfaces with the pledge to enable creation or consumption of required data objects, which are exchanged with the domain registrar. Moreover, the addition of the pledge-agent also influences the sequences for the data exchange between the pledge and the domain registrar described in [I-D.ietf-anima-bootstrapping-keyinfra]. The general goal for the pledge-agent application is the reuse of already defined endpoints on the domain registrar side. The behavior of the endpoint may need to be adapted.

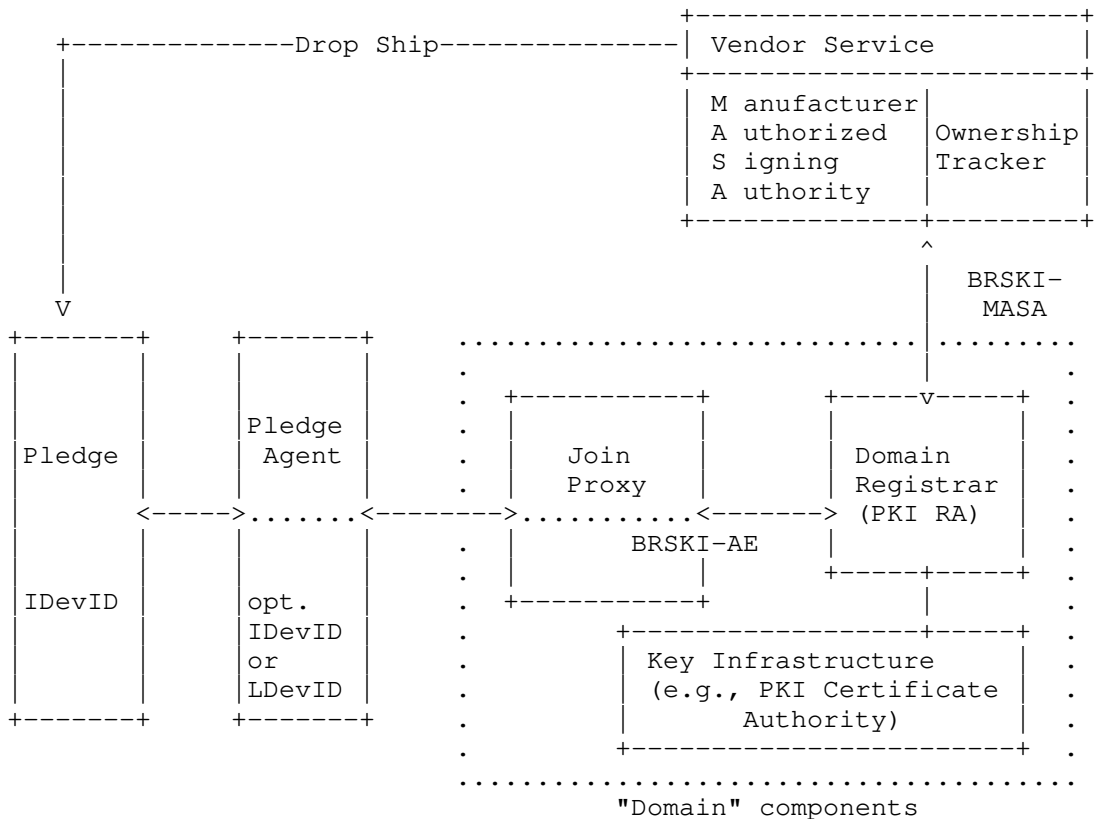


Figure 3: Architecture overview using a pledge-agent

The architecture overview in Figure 3 utilizes the same logical components as BRSKI with the pledge-agent component as additional component.

For authentication towards the domain registrar, the pledge-agent may use the IDevID or LDevID credentials if available, which are verified by the domain registrar as part of the TLS establishment. The provisioning of this credential to the pledge-agent is out of scope for this specification. Alternatively, the domain registrar may authenticate the user operating the pledge-agent to perform authorization of a pledge onboarding action. Examples for such user level authentication are the application of HTTP authentication or the usage of authorization tokens or the application of user related certificates in the TLS handshake or other. If the pledge-agent utilizes a certificate, the domain registrar must be able to verify the certificate by possessing the corresponding root certificate.

The following list describes the components in the deployment domain:

- o Pledge: The pledge is expected to respond the necessary data objects for bootstrapping to the pledge-agent. The transport protocol used between the pledge and the pledge-agent in the context of this document is assumed to be HTTP. Other transport protocols may be used, such as CoAP, but their usage is out of scope for this document. As the pledge is triggered/PUSHED by the pledge-agent, it becomes a callee. This leads to some differences to BRSKI:
 - * Discovery of the domain registrar by the pledge will be omitted as the pledge is expected to be triggered by the pledge-agent.
 - * Discovery of the pledge by the pledge-agent must be possible to enable interaction.
 - * As the pledge-agent must be able to trigger the pledge for bootstrapping, the pledge must offer communication endpoints.
 - * The pledge-agent is expected to provide an option to trigger the bootstrapping by pushing data objects to the pledge.
 - * Order of exchanges in the call flow is different as the pledge-agent collects both voucher request objects and certification request objects at once and provides them to the registrar. This approach may also be used to perform a bulk bootstrapping of several devices.
 - * The data objects utilized for the data exchange between the pledge and the registrar are signature-wrapped objects as in use case 1 Section 5.1.
- o Pledge-Agent: provides a communication path to exchange data objects between the pledge and the domain registrar. The pledge-agent facilitates situations, in which the domain registrar is not directly reachable by the pledge, either due to a different technology stack or due to missing connectivity (e.g., if the domain registrar resides in the cloud and the pledge has no connectivity, yet). The pledge-agent collect the bootstrapping information such as voucher request objects and certification request objects from one or multiple pledges at once and performs a bulk bootstrapping based on the collected data. The pledge-agent triggers the pledge for generating these objects. The pledge-agent may be configured with the domain registrar information or may use the discovery mechanism defined in [I-D.ietf-anima-bootstrapping-keyinfra]. The trust assumptions on the connection between the pledge and the pledge-agent only

require to ensure proximity between both to provide a minimum protection of arbitrary request to generate voucher request objects and/or enrollment request objects. The trust assumptions on the connection between the pledge-agent and the registrar only requires that the pledge-agent enables the exchange of signature wrapped objects between the pledge and the registrar.

- o Join Proxy: same functionality as described in [I-D.ietf-anima-bootstrapping-keyinfra]. Note that it may be used by the pledge-agent instead of the pledge.
- o Domain Registrar: In general the domain registrar fulfills the same functionality regarding the bootstrapping of the pledge in the deployment domain by facilitating the communication of the pledge with the MASA service and the domain PKI service. In contrast to [I-D.ietf-anima-bootstrapping-keyinfra], the domain registrar does not interact with a pledge directly but through the pledge-agent. This prohibits a pledge authentication using its IDevID during TLS establishment towards the registrar. If the pledge-agent has an IDevID or is already possessing a LDevID valid in the deployment domain, it is expected to use this authentication towards the domain registrar.

The manufacturer provided components/services (MASA and Ownership tracker) are used as defined in [I-D.ietf-anima-bootstrapping-keyinfra].

5.2.1. Behavior of a pledge

In contrast to use case 1 Section 5.1 the pledge acts as a server component if data is pushed in the bootstrapping phase. It is triggered by the pledge-agent for the generation of voucher request and enrollment request objects as well as for the processing of the response objects and the generation of status information. Due to the use of the pledge-agent, the interaction with the domain registrar is changed as shown in Figure 4. To enable interaction with the pledge-agent, the pledge provides interfaces using the BRSKI REST interface based on the `"/.well-known/brski"` URI tree. The following endpoints are defined for the pledge:

- o `/.well-known/brski/triggervoucherrequest`: trigger pledge to create voucher request.
- o `/.well-known/brski/triggerenrollrequest`: trigger pledge to create enrollment request.
- o `/.well-known/brski/supplyvoucherresponse`: supply voucher response to pledge.

- o `/.well-known/brski/supplyenrollresponse`: supply enroll response to pledge.
- o `/.well-known/brski/supplyCACerts`: supply CACerts to pledge (optional).

5.2.2. Behavior of a pledge-agent

The pledge-agent is a new component in the BRSKI context. It provides connectivity between the pledge and the domain registrar and utilizes the endpoints on the domain registrar side already specified in [I-D.ietf-anima-bootstrapping-keyinfra]. The pledge-agent is expected to interact with the pledge independent of the domain registrar. As stated before, the data exchange concerns the data objects exchanged between the pledge and the domain registrar, which are the voucher request/response objects, the enrollment request/response objects, as well as status information. As the pledge acts as server, it has to provide endpoints to allow for request/response interaction. For the transport HTTPS is chosen in non-constraint environments, but may also be performed using other transport mechanisms. This changes the general interaction between the pledge and the domain registrar as shown in Figure 4.

The pledge-agent may have an own IDevID or a deployment domain issued LDevID to be utilized in the TLS communication establishment towards the domain registrar. Note that the pledge-agent may also be used without TLS client-side authentication if no suitable credential is available. This is a deviation from BRSKI, in which the pledge's IDevID credential is used to perform TLS client authentication. As BRSKI-AE targets the use of authenticated self-contained data objects between the pledge and the domain registrar, the binding of the pledge identity to the requests can be achieved through the data object signature. Nevertheless, the authentication of the pledge-agent is recommended if the onboarding is only to be performed by an authorized pledge-agent. This authentication may be realized by a device (IDevID or LDevID), and if not available through user related credentials in the context of the HTTP based authentication, SAML tokens or other. Note that having no specific credentials on the pledge-agent allows to employ applications with low trust requirements.

According to [I-D.ietf-anima-bootstrapping-keyinfra] section 5.3, the domain registrar performs the pledge authorization for onboarding within his domain based on the provided voucher request.

5.2.2.1. Registrar discovery

The discovery phase may be applied as specified in [I-D.ietf-anima-bootstrapping-keyinfra] with the deviation that it is done between the pledge-agent and the domain registrar. Alternatively, the pledge-agent may be configured with the address of the domain registrar.

5.2.2.2. Pledge/Pledge-agent discovery

The discovery of the pledge by pledge-agent is done by mDNS. The pledge constructs a local host name based on device local information (device serial number), which results for instance in "serialnumber.brski-pledge._tcp.local.". This can then be discovered by the pledge-agent via mDNS. Note that other mechanisms for discovery may be used.

5.2.3. Protocol Details (Pledge-Agent - Pledge)

The interaction of the pledge with the pledge-agent may be accomplished using different transport means (protocols and or network technologies). For this document the usage of HTTP is targeted as in BRSKI. Alternatives may be CoAP or Bluetooth Low Energy (BLE) or Nearfield Communication (NFC). This requires an independence of the security of the exchanged data objects between the pledge and the registrar from the security provided by the transport. Therefore, signature-wrapped objects build the base for the information exchange between the pledge and the registrar. Note that trigger messages from the pledge-agent may not be signed as the pledge has no means to verify them. Therefore, TLS support is required to ensure a secure transport based on a proximity information shared between the pledge-agent and the pledge. This is done to ensure that a technician had physical contact to the pledge.

5.2.3.1. TLS establishment

The pledge and the pledge-agent establish a TLS secured communication channel. As the IDevID on the pledge cannot be used as TLS server certificate, a pre-shared key (PSK) is applied.

TLS [RFC8446] allows to import externally provided PSKs. The use of an external PSK is defined based on the guidelines provided in [I-D.ietf-tls-external-psk-guidance]

The PSK is derived from information known to the pledge and the pledge-agent, which are

device serial-number: Device serial number stored in the pledge and also part of the X520SerialNumber (defined in [RFC5280]) as part of the IDevID.

randomizer: additional value, which is not exposed on the communication interface of the pledge. This information may be provided through the bill of material or a QR code attached to the device and must have a length of at least 128 bits.

The pledge and the pledge-agent construct the PSK to be used in TLS based on a HMAC-based Extract-and-Expand Key Derivation Function (HKDF, [RFC5869]). The PSK is derived following the external PSK importer [I-D.ietf-tls-external-psk-importer]. The interface takes an EPSK (External PSK) with an external identity and a base key (epsk) as input. The external identity is provided as part of the ImportedIdentity structure containing information:

```
ImportedIdentity.external_identity = "onboarding"
```

```
ImportedIdentity.context = "brski_proximity"
```

```
ImportedIdentity.target_protocol = 0x0304
```

```
ImportedIdentity.target_kdf = 0x0001
```

The target protocol identified is TLS 1.3 (value 0x0304). The target KDF identified is HKDF_SHA256 (value 0x0001).

The base key is determined as following:

```
epsk = serial-number | randomizer
```

```
epskx = HKDF-Extract(0, epsk)
```

```
ipskx = HKDF-Expand-Label(epskx, "derived psk",  
Hash(ImportedIdentity), 32)
```

The length value of the HKDF-Expand function is set to 32 octets corresponds to the output length of the selected KDF.

TLS shall be used with the derived IPSK with

```
TLS key agreement: "psk_dhe_ke"
```

```
TLS cipher suite: TLS_AES_128_GCM_SHA256
```

5.2.3.2. Object exchange

The pledge-agent provides the registrar certificate to the pledge to be put into the "proximity-registrar-cert" leaf in the pledge voucher-request object. This enables the registrar to verify, that it is the target registrar for the request. The registrar certificate may be configured at the pledge-agent or may be fetched by the pledge-agent based on a prior TLS connection establishment with the domain registrar. The pledge-agent triggers the pledge, to generate a pledge voucher-request object (PleVouReq) .

Triggering is done using HTTPS POST with the operation path value of `"/.well-known/brski/triggervoucherrequest"`.

The pledge-agent triggervoucherrequest Content-Type header is:

```
application/json: /* to be defined */(different format used as
response): defines a JSON document to provide the registrar
certificate to the pledge. The pledge shall construct the voucher
request object as defined [I-D.ietf-anima-bootstrapping-keyinfra] and
sign the request using the pledges IDevID credential. The response
is encoded as JSON-in-JWS (or JWS-signed-JSON, tbd).
```

After the voucher request exchange the pledge will be triggered to generate an enrollment request object. As in BRSKI the enrollment request object is a PKCS#10 request, with an additional wrapping signature using the IDevID. As the initial enrollment aims to request a general certificate, no certificate attributes are provided to the pledge.

```
/* Discussion: The pledge-agent may have been pre-configured with the
CSR attributes, that it could provide to the pledge to request a
specific certificate (for a communication endpoint on the pledge.
This is expected to be done later, once the pledge has an IDevID and
can be further configured. */
```

The enrollment request is generated as authenticated self-signed object, which assures proof of possession of the private key corresponding to the contained public key in the enrollment request as well as a proof of identity, based on the IDevID of the pledge. This is done as described for use case 1 Section 5.1.

The pledge-agent enrollment-request Content-Type header is:

```
application/json: to be defined (different format used as response):
defines a JSON document. Optional CSR parameter may be provided to
the pledge. The pledge shall construct the certification request as
PKCS#10 object and sign the request using the pledge's IDevID
```

credential. The response is encoded as PKCS#10-in-JSON/JWS (tbd). If the pledge does not have specific information about the content of the LDevID to be applied for (like device name, etc.) the domain registrar will provide this information to the issuing CA.

/* to be discussed */ The domain registrar may either enhance the PKCS#10 request or generate a structure containing the attributes to be included by the CA and sends both (the original PKCS#10 request and the enhancements) to the domain CA. As enhancing the PKCS#10 request destroys the initial proof of possession of the corresponding private key, the CA would need to accept RA-verified requests.

With the collected voucher request object and the enrollment request object, the pledge-agent starts the interaction with the domain registrar. If the domain registrar is in a different network, the pledge-agent closes the TLS session with the pledge (to be resumed for provisioning of voucher object and certificate).

/* further description necessary at least for */

Consideration of TLS session resumption for the second run

Authentication of pledge-agent to domain registrar

Behavior of registrar when pledge LDevID not used in TLS

Values to be taken from the IDevID into the PKCS#10 (like pledge serial number or subjectName, or certificate template)

PKCS#10-in-JSON/JWS (tbd) handling by domain registrar to request a generic LDevID from the domain CA service.

Order of requests may be different as in BRSKI

Definition of objects and encoding, some existing encoding may change as for the voucher?

Once the pledge-agent has collected the response objects from the domain registrar (voucher and certificate), it will re-start the interaction the pledge. For this the pledge-agent resumes the previous TLS connection with the pledge.

The pledge-agent will provide the information via two distinct endpoints at the pledge:

The voucher response will be provided with a HTTPS POST using the operation path value of `"/.well-known/brski/supplyvoucherresponse"`.

The pledge-agent voucher-response Content-Type header is:

```
application/jose: /* to be discussed, as the current voucher is a
"application/voucher-cms+json" object). */ The pledge will generate a
voucher status object and provides it in the response. The response
is encoded as JSON-in-JWS ("application/jose"), signed by the IDevID
of the pledge (LDevID not available yet).
```

The enrollment response will be provided with a HTTPS POST using the operation path value of `"/.well-known/brski/supplyenrollresponse"`.

The pledge-agent enroll-response Content-Type header is:

```
application/pkcs7-mime: to be defined (contains LDevID + certificate
chain).
```

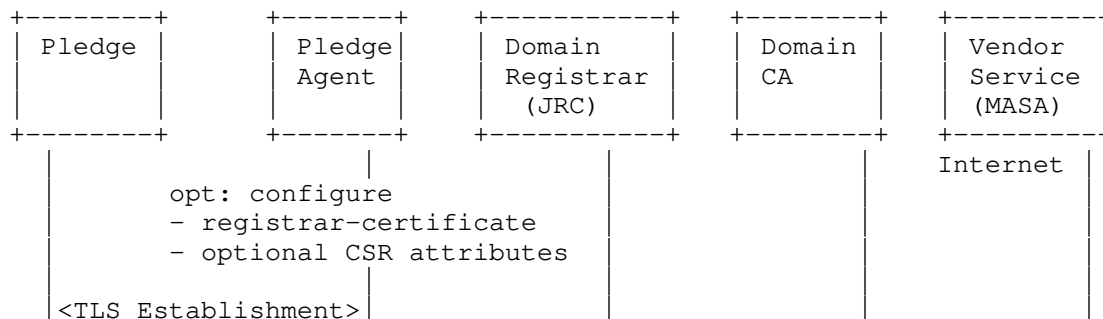
```
/* to be discussed */: the enrollment response object may also be an
application/jose object with a signature of the domain registrar.
This may be used either to transport additional data which is bound
to the LDevID or it may be considered for enrollment status to ensure
that in an error case the registrar providing the certificate can be
identified.
```

The pledge will generate an enrollment status object as confirmation, showing it can apply the certificate and possesses the corresponding private key

The response is encoded as JSON-in-JWS. The signature is created using the LDevID of the pledge.

5.2.4. Protocol flow

The following protocol description assumes an already established TLS channel as described in Section 5.2.3.1 and focuses on the exchange of signature wrapped objects using endpoints defined for the pledge in Section 5.2.3.2



```

[example: trigger voucher and certification request generation ]

<-trigger PleVouReq
  (registrar-cert)
- Voucher Request->

<--trigger ER-----

----Cert Request-->
<----- TLS ---->

---- VouReq -->
    [accept device?]
    [contact vendor]
        ----- Voucher Request ----->
        ----- Pledge ID ----->
        ----- Domain ID ----->
        ----- optional: nonce ----->
            [extract DomainID]
            [update audit log]
        <----- Voucher -----
<-- Voucher --
<----- device audit log ----

[optional retrieve CA certs]
- CACertReq ->
    - CACertReq -->
    <-CACertResp --
< CACertResp -

[certification request handling pledge-agent and infrastructure]
-- CertReq -->
    -- CertReq ---->
    <--CertResp-----
<-- CertResp -

[push voucher and certificate to pledge, optionally push CA certs]

< TLS Resumption >

<--supply Voucher--
- Voucher Status-->

<---supply Cert----
-- Enroll Status-->

```

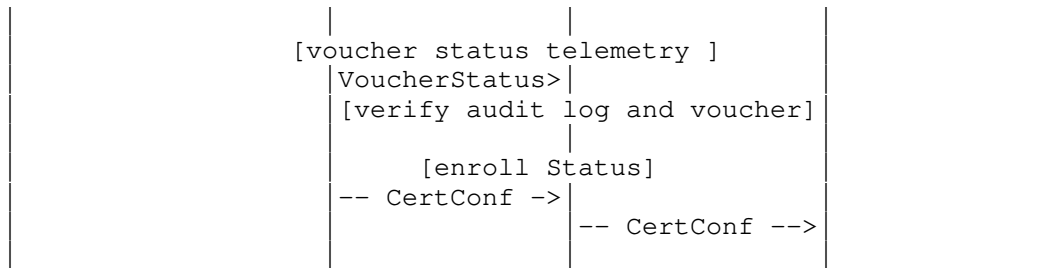


Figure 4: Request handling of the pledge using a pledge-agent

As shown in Figure 4 the pledge-agent collects the voucher request and enrollment request objects from a pledge. As the pledge-agent is intended to work between the pledge and the domain registrar, a collection of requests from multiple pledges is possible, allowing a bulk bootstrapping of multiple pledges using the connection between the pledge-agent and the domain registrar.

The information exchange between the pledge-agent and the domain registrar resembles the exchanges between the pledge and the domain registrar from BRSKI in the PULL case.

[RFC Editor: please delete] /* to be discussed: Description on how the registrar makes the decision if he is connected with pledge directly (as in BRSKI PULL) or with a pledge-agent (PUSH). This may result in a case statement (client-side authentication in TLS, user authentication above TLS, etc.) for the TLS connection establishment in the original BRSKI document in section 5.1 */

Once the pledge-agent has finished the exchanges with the domain registrar to get the voucher object and the enrollment object, it can close the TLS connection to the domain registrar and provide the objects to the pledge(s). The content of the response objects is defined through the voucher [RFC8366] and the certificate [RFC5280].

5.3. Domain registrar support of different enrollment options

Well-known URIs for different endpoints on the domain registrar are already defined as part of the base BRSKI specification. In addition, alternative enrollment endpoints may be supported at the domain registrar. The pledge / pledge-agent will recognize if its supported enrollment option is supported by the domain registrar by sending a request to its preferred enrollment endpoint.

The following provides an illustrative example for a domain registrar supporting different options for EST as well as CMP to be used in BRSKI-AE. The listing contains the supported endpoints for the

bootstrapping, to which the pledge may connect. This includes the voucher handling as well as the enrollment endpoints.

```
</brski/voucherrequest>,ct=voucher-cms+json
</brski/voucher_status>,ct=json
</brski/enrollstatus>,ct=json
</est/cacerts>;ct=pkcs7-mime
</est/simpleenroll>;ct=pkcs7-mime
</est/simplereenroll>;ct=pkcs7-mime
</est/fullcmc>;ct=pkcs7-mime
</est/serverkeygen>;ct=pkcs7-mime
</est/csrattrs>;ct=pkcs7-mime
</cmp/initialization>;ct=pkixcmp
</cmp/certification>;ct=pkixcmp
</cmp/keyupdate>;ct=pkixcmp
</cmp/pl0>;ct=pkixcmp
</cmp/getCAcert>;ct=pkixcmp
</cmp/getCSRparam>;ct=pkixcmp
```

[RFC Editor: please delete] /*

Open Issues:

- o Clarify, if /.well-known discovery can be performed as discussed in the design team (usage of GET /.well-known/brski to collect information about enrollment specific endpoint support, to be specified in a separate draft). Also, is a discovery option necessary at all, as the pledge will most likely implement only one enrollment option? It can be helpful in the pledge-agent use case, when the pledge-agent has no information about the supported enrollment options (less likely).
- o In addition to the current content types, we may specify that the response provide information about different content types as multiple values. This would allow to further adopt the encoding of the objects exchanges (ASN.1, JSON, CBOR, ...). -> dependent on the utilized protocol.

*/

6. Example for signature-wrapping using existing enrollment protocols

This sections map the requirements to support proof of possession and proof of identity to selected existing enrollment protocols. Note that that the work in the ACE WG described in [I-D.selander-ace-coap-est-oscore] may be considered here as well, as it also addresses the encapsulation of EST in a way to make it

independent from the underlying TLS using OSCORE resulting in an authenticated self-contained object.

6.1. EST Handling

When using EST [RFC7030], the following constraints should be considered:

- o Proof of possession is provided by using the specified PKCS#10 structure in the request.
- o Proof of identity is achieved by signing the certification request object, which is only supported when Full PKI Request (the /fullcmc endpoint) is used. This contains sufficient information for the RA to make an authorization decision on the received certification request. Note: EST references CMC [RFC5272] for the definition of the Full PKI Request. For proof of identity, the signature of the SignedData of the Full PKI Request would be calculated using the IDevID credential of the pledge.
- o [RFC Editor: please delete] /* TBD: in this case the binding to the underlying TLS connection is not be necessary. */
- o When the RA is not available, as per [RFC7030] Section 4.2.3, a 202 return code should be returned by the Registrar. The pledge in this case would retry a simpleenroll with a PKCS#10 request. Note that if the TLS connection is teared down for the waiting time, the PKCS#10 request would need to be rebuilt if it contains the unique identifier (tls_unique) from the underlying TLS connection for the binding.
- o [RFC Editor: please delete] /* TBD: clarification of retry for fullcmc is necessary as not specified in the context of EST */

6.2. Lightweight CMP Handling

Instead of using CMP [RFC4210], this specification refers to the lightweight CMP profile [I-D.ietf-lamps-lightweight-cmp-profile], as it restricts the full featured CMP to the functionality needed here. For this, the following constrains should be observed:

- o For proof of possession, the defined approach in Lightweight CMP section 5.1.1 (based on CRMF) and 5.1.5 based on PCKS#10 should be supported.
- o Proof of identity can be provided by using the signatures to protect the certificate request message as outlined in section 4.2.

- o When the RA/CA is not available, a waiting indication should be returned in the PKIStatus by the Registrar. The pledge in this case would retry using the PollReqContent with a request identifier certReqId provided in the initial CertRequest message as specified in section 6.1.4 with delayed enrollment.

7. IANA Considerations

This document requires the following IANA actions:

IANA is requested to enhance the Registry entitled: "BRSKI well-known URIs" with the following:

URI	document	description
triggervoucherrequest	[THISRFC]	create voucher request
triggerenrollrequest	[THISRFC]	create enrollment request
supplyvoucherresponse	[THISRFC]	supply voucher response
supplyenrollresponse	[THISRFC]	supply enrollment response
supplyCACerts	[THISRFC]	supply CA certs

```
[RFC Editor: please delete] /* to be done: IANA consideration to be
included for the defined namespaces in Section 5.1.5 and Section 5.3
. */
```

8. Privacy Considerations

```
[RFC Editor: please delete] /* to be done: clarification necessary */
```

9. Security Considerations

9.1. Exhaustion attack on pledge

Exhaustion attack on pledge based on DoS attack (connection establishment, etc.)

9.2. PSK usage in TLS establishment

TLS is used to provide a proximity information to the pledge. As the devices in scope may not feature an input or output interface for local interaction, a PSK is use to establish the connection between the pledge and the pledge-agent. Certificate based authentication of the pledge cannot be used, as the device does not have the appropriate information contained in the IDevID. The PSK is build using a KDF, which uses the serial number of the device, potential further device related information and a randomizer value. This information is stored within the pledge and is also part of product information (also an QR code attached to the device). If a potential attacker is able to physically access the device, he may read this

information and is to connect to the pledge. Without physical proximity to the device, to capture the QR code information, the attacker may guess the device' serial number but will not be able to construct the PSK as the randomizer value is not known.

9.3. Misuse of acquired voucher and enrollment responses

Pledge-agent that uses acquired voucher and enrollment response for domain 1 in domain 2: can be detected in Voucher Request processing on domain registrar side. Requires domain registrar to verify the proximity-registrar-cert leaf in the voucher request against his own as well as the association of the pledge to his domain based on the serial number contained in the voucher.

Misbinding of pledge by a faked domain registrar is countered as described in BRSKI security considerations (section 11.4).

10. Acknowledgments

We would like to thank the various reviewers for their input, in particular Brian E. Carpenter, Michael Richardson, Giorgio Romanenghi, Oskar Camenzind, for their input and discussion on use cases and call flows.

11. References

11.1. Normative References

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-45 (work in progress), November 2020.
- [I-D.ietf-tls-external-psk-importer]
Benjamin, D. and C. Wood, "Importing External PSKs for
TLS", draft-ietf-tls-external-psk-importer-06 (work in
progress), December 2020.
- [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>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<https://www.rfc-editor.org/info/rfc7030>>.

- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.

11.2. Informative References

- [I-D.ietf-lamps-cmp-updates]
Brockhaus, H., "CMP Updates", draft-ietf-lamps-cmp-updates-06 (work in progress), November 2020.
- [I-D.ietf-lamps-lightweight-cmp-profile]
Brockhaus, H., Fries, S., and D. Oheimb, "Lightweight CMP Profile", draft-ietf-lamps-lightweight-cmp-profile-04 (work in progress), November 2020.
- [I-D.ietf-tls-external-psk-guidance]
Housley, R., Hoyland, J., Sethi, M., and C. Wood, "Guidance for External PSK Usage in TLS", draft-ietf-tls-external-psk-guidance-01 (work in progress), November 2020.
- [I-D.selander-ace-coap-est-oscore]
Selander, G., Raza, S., Furuheid, M., Vucinic, M., and T. Claeys, "Protecting EST Payloads with OSCORE", draft-selander-ace-coap-est-oscore-04 (work in progress), November 2020.
- [IEC-62351-9]
International Electrotechnical Commission, "IEC 62351 - Power systems management and associated information exchange - Data and communications security - Part 9: Cyber security key management for power system equipment", IEC 62351-9 , May 2017.
- [ISO-IEC-15118-2]
International Standardization Organization / International Electrotechnical Commission, "ISO/IEC 15118-2 Road vehicles - Vehicle-to-Grid Communication Interface - Part 2: Network and application protocol requirements", ISO/IEC 15118-2 , April 2014.

- [NERC-CIP-005-5] North American Reliability Council, "Cyber Security - Electronic Security Perimeter", CIP 005-5, December 2013.
- [OCPP] Open Charge Alliance, "Open Charge Point Protocol 2.0.1 (Draft)", December 2019.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<https://www.rfc-editor.org/info/rfc2986>>.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.
- [RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <<https://www.rfc-editor.org/info/rfc4211>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

- [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/info/rfc8615>>.
- [RFC8894] Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <<https://www.rfc-editor.org/info/rfc8894>>.

Appendix A. History of changes [RFC Editor: please delete]

From IETF draft 00 -> IETF 01:

- o Update of scope in Section 3.1 to include in which the pledge acts as a server. This is one main motivation for use case 2.
- o Rework of use case 2 in Section 5.2 to consider the transport between the pledge and the pledge-agent. Addressed is the TLS channel establishment between the pledge-agent and the pledge as well as the endpoint definition on the pledge.
- o First description of exchanged object types (needs more work)
- o Clarification in discovery options for enrollment endpoints at the domain registrar based on well-known endpoints in Section 5.3 do not result in additional /.well-known URIs. Update of the illustrative example. Note that the change to /brski for the voucher related endpoints has been taken over in the BRSKI main document.
- o Updated references.
- o Included Thomas Werner as additional author for the document.

From individual version 03 -> IETF draft 00:

- o Inclusion of discovery options of enrollment endpoints at the domain registrar based on well-known endpoints in Section 5.3 as replacement of section 5.1.3 in the individual draft. This is intended to support both use cases in the document. An illustrative example is provided.
- o Missing details provided for the description and call flow in pledge-agent use case Section 5.2, e.g. to accommodate distribution of CA certificates.
- o Updated CMP example in Section 6 to use lightweight CMP instead of CMP, as the draft already provides the necessary /.well-known endpoints.

- o Requirements discussion moved to separate section in Section 4. Shortened description of proof of identity binding and mapping to existing protocols.
- o Removal of copied call flows for voucher exchange and registrar discovery flow from [I-D.ietf-anima-bootstrapping-keyinfra] in Section 5.1 to avoid doubling or text or inconsistencies.
- o Reworked abstract and introduction to be more crisp regarding the targeted solution. Several structural changes in the document to have a better distinction between requirements, use case description, and solution description as separate sections. History moved to appendix.

From individual version 02 -> 03:

- o Update of terminology from self-contained to authenticated self-contained object to be consistent in the wording and to underline the protection of the object with an existing credential. Note that the naming of this object may be discussed. An alternative name may be attestation object.
- o Simplification of the architecture approach for the initial use case having an offsite PKI.
- o Introduction of a new use case utilizing authenticated self-contained objects to onboard a pledge using a commissioning tool containing a pledge-agent. This requires additional changes in the BRSKI call flow sequence and led to changes in the introduction, the application example, and also in the related BRSKI-AE call flow.
- o Update of provided examples of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 5.1.5.

From individual version 01 -> 02:

- o Update of introduction text to clearly relate to the usage of IDevID and LDevID.
- o Definition of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 5.1.5. This section also contains a first discussion of an optional discovery mechanism to address situations in which the registrar supports more than one enrollment approach. Discovery should avoid that the pledge performs a trial and error of enrollment protocols.

- o Update of description of architecture elements and changes to BRSKI in Section 5.
- o Enhanced consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 4 and in Section 6.

From individual version 00 -> 01:

- o Update of examples, specifically for building automation as well as two new application use cases in Section 3.2.
- o Deletion of asynchronous interaction with MASA to not complicate the use case. Note that the voucher exchange can already be handled in an asynchronous manner and is therefore not considered further. This resulted in removal of the alternative path the MASA in Figure 1 and the associated description in Section 5.
- o Enhancement of description of architecture elements and changes to BRSKI in Section 5.
- o Consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 4.
- o New section starting Section 6 with the mapping to existing enrollment protocols by collecting boundary conditions.

Authors' Addresses

Steffen Fries
Siemens AG
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: steffen.fries@siemens.com
URI: <https://www.siemens.com/>

Hendrik Brockhaus
Siemens AG
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: hendrik.brockhaus@siemens.com
URI: <https://www.siemens.com/>

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Thomas Werner
Siemens AG
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: thomas-werner@siemens.com
URI: <https://www.siemens.com/>

ANIMA WG
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

D. von Oheimb, Ed.
S. Fries
H. Brockhaus
Siemens
E. Lear
Cisco Systems
7 March 2022

BRSKI-AE: Alternative Enrollment Protocols in BRSKI
draft-ietf-anima-brski-async-enroll-05

Abstract

This document enhances Bootstrapping Remote Secure Key Infrastructure (BRSKI, [RFC8995]) to allow employing alternative enrollment protocols, such as CMP.

Using self-contained signed objects, the origin of enrollment requests and responses can be authenticated independently of message transfer. This supports end-to-end security and asynchronous operation of certificate enrollment and provides flexibility where to authenticate and authorize certification requests.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 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 extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Motivation	3
1.2. Supported environment	5
1.3. List of application examples	6
2. Terminology	6
3. Requirements discussion and mapping to solution elements . .	7
4. Adaptations to BRSKI	10
4.1. Architecture	10
4.2. Message exchange	13
4.2.1. Pledge - Registrar discovery and voucher exchange . .	13
4.2.2. Registrar - MASA voucher exchange	13
4.2.3. Pledge - Registrar - RA/CA certificate enrollment . .	13
4.2.4. Pledge - Registrar - enrollment status telemetry . .	16
4.2.5. Addressing scheme enhancements	16
4.3. Domain registrar support of alternative enrollment protocols	16
5. Examples for signature-wrapping using existing enrollment protocols	17
5.1. Instantiation to EST (informative)	17
5.2. Instantiation to CMP (normative if CMP is chosen)	18
6. IANA Considerations	18
7. Security Considerations	19
8. Acknowledgments	19
9. References	19
9.1. Normative References	19
9.2. Informative References	20
Appendix A. Using EST for certificate enrollment	21
Appendix B. Application examples	22
B.1. Rolling stock	23
B.2. Building automation	23
B.3. Substation automation	24
B.4. Electric vehicle charging infrastructure	24
B.5. Infrastructure isolation policy	24
B.6. Sites with insufficient level of operational security . .	25
Appendix C. History of changes TBD RFC Editor: please delete . .	25
Authors' Addresses	29

1. Introduction

1.1. Motivation

BRSKI, as defined in [RFC8995], specifies a solution for secure automated zero-touch bootstrapping of new devices, so-called pledges. This includes the discovery of the registrar in the target domain, time synchronization, and the exchange of security information necessary to establish mutual trust between pledges and the target domain.

A pledge gains trust in the target domain via the domain registrar as follows. It obtains security information about the domain, specifically a domain certificate to be trusted, by requesting a voucher object defined in [RFC8366]. Such a voucher is a self-contained signed object originating from a Manufacturer Authorized Signing Authority (MASA). Therefore, the voucher may be provided in online mode (synchronously) or offline mode (asynchronously). The pledge can authenticate the voucher because it is shipped with a trust anchor of its manufacturer such that it can validate signatures (including related certificates) by the MASA.

Trust by the target domain in a pledge is established by providing the pledge with a domain-specific LDevID certificate. The certification request of the pledge is signed using its IDevID secret and can be validated by the target domain using the trust anchor of the pledge manufacturer, which needs to be pre-installed in the domain.

For enrolling devices with LDevID certificates, BRSKI typically utilizes Enrollment over Secure Transport (EST) [RFC7030]. EST has its specific characteristics, detailed in Appendix A. In particular, it requires online or on-site availability of the RA for performing the data origin authentication and final authorization decision on the certification request. This type of enrollment can be called 'synchronous enrollment'. For various reasons, it may be preferable to use alternative enrollment protocols such as the Certificate Management Protocol (CMP) [RFC4210] profiled in [I-D.ietf-lamps-lightweight-cmp-profile] or Certificate Management over CMS (CMC) [RFC5272], that are more flexible and independent of the transfer mechanism because they represent certification request messages as authenticated self-contained objects.

Depending on the application scenario, the required RA/CA components may not be part of the registrar. They even may not be available on-site but rather be provided by remote backend systems. The registrar or its deployment site may not have an online connection with them or the connectivity may be intermittent. This may be due to security requirements for operating the backend systems or due to site

deployments where on-site or always-online operation may be not feasible or too costly. In such scenarios, the authentication and authorization of certification requests will not or can not be performed on-site at enrollment time. In this document, enrollment that is not performed in a (time-wise) consistent way is called _asynchronous enrollment_. Asynchronous enrollment requires a store-and-forward transfer of certification requests along with the information needed for authenticating the requester. This allows offline processing the request.

Application scenarios may also involve network segmentation, which is utilized in industrial systems to separate domains with different security needs. Such scenarios lead to similar requirements if the TLS connection carrying the requester authentication is terminated and thus request messages need to be forwarded on further channels before the registrar/RA can authorize the certification request. In order to preserve the requester authentication, authentication information needs to be retained and ideally bound directly to the certification request.

There are basically two approaches for forwarding certification requests along with requester authentication information:

- * A trusted component (e.g., a local RA) in the target domain is needed that forwards the certification request combined with the validated identity of the requester (e.g., its IDevID certificate) and an indication of successful verification of the proof-of-possession (of the corresponding private key) in a way preventing changes to the combined information. When connectivity is available, the trusted component forwards the certification request together with the requester information (authentication and proof-of-possession) for further processing. This approach offers only hop-by-hop security. The backend PKI must rely on the local pledge authentication result provided by the local RA when performing the authorization of the certification request. In BRSKI, the EST server is such a trusted component, being co-located with the registrar in the target domain.
- * Involved components use authenticated self-contained objects for the enrollment, directly binding the certification request and the requester authentication in a cryptographic way. This approach supports end-to-end security, without the need to trust in intermediate domain components. Manipulation of the request and the requester identity information can be detected during the validation of the self-contained signed object.

Focus of this document is the support of alternative enrollment protocols that allow using authenticated self-contained objects for device credential bootstrapping. This enhancement of BRSKI is named BRSKI-AE, where AE stands for alternative enrollment protocols and for asynchronous enrollment. This specification carries over the main characteristics of BRSKI, namely that the pledge obtains trust anchor information for authenticating the domain registrar and other target domain components as well as a domain-specific X.509 device certificate (the LDevID certificate) along with the corresponding private key (the LDevID secret) and certificate chain.

The goals are to enhance BRSKI to

- * support alternative enrollment protocols,
- * support end-to-end security for enrollment, and
- * make it applicable to scenarios involving asynchronous enrollment.

This is achieved by

- * extending the well-known URI approach with an additional path element indicating the enrollment protocol being used, and
- * defining a certificate waiting indication and handling, for the case that the certifying component is (temporarily) not available.

This specification can be applied to both synchronous and asynchronous enrollment.

In contrast to BRSKI, this specification supports offering multiple enrollment protocols on the infrastructure side, which enables pledges and their developers to pick the preferred one.

1.2. Supported environment

BRSKI-AE is intended to be used in domains that may have limited support of on-site PKI services and comprises application scenarios like the following.

- * There are requirements or implementation restrictions that do not allow using EST for enrolling an LDevID certificate.
- * Pledges and/or the target domain already have an established certificate management approach different from EST that shall be reused (e.g., in brownfield installations).

- * There is no registration authority available on site in the target domain. Connectivity to an off-site RA is intermittent or entirely offline. A store-and-forward mechanism is used for communicating with the off-site services.
- * Authoritative actions of a local RA are limited and may not be sufficient for authorizing certification requests by pledges. Final authorization is done by an RA residing in the operator domain.

1.3. List of application examples

Bootstrapping can be handled in various ways, depending on the application domains. The informative Appendix B provides illustrative examples from various industrial control system environments and operational setups. They motivate the support of alternative enrollment protocols, based on the following examples of operational environments:

- * Rolling stock
- * Building automation
- * Electrical substation automation
- * Electric vehicle charging infrastructures
- * Infrastructure isolation policy
- * Sites with insufficient level of operational security

2. Terminology

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 relies on the terminology defined in [RFC8995] and [IEEE.802.1AR_2009]. The following terms are defined in addition:

EE: End entity, in the BRSKI context called pledge. It is the entity that is bootstrapped to the target domain. It holds a public-private key pair, for which it requests a public-key certificate. An identifier for the EE is given as the subject name of the certificate.

RA: Registration authority, an optional system component to which a CA delegates certificate management functions such as authenticating requesters and performing authorization checks on certification requests.

CA: Certification authority, issues certificates and provides certificate status information.

target domain: The set of entities that share a common local trust anchor, independent of where the entities are deployed.

site: Describes the locality where an entity, e.g., pledge, registrar, RA, CA, is deployed. Different sites can belong to the same target domain.

on-site: Describes a component or service or functionality available in the target deployment site.

off-site: Describes a component or service or functionality available in an operator site different from the target deployment site. This may be a central site or a cloud service, to which only a temporary connection is available.

asynchronous communication: Describes a time-wise interrupted communication between a pledge (EE) and a registrar or PKI component.

synchronous communication: Describes a time-wise uninterrupted communication between a pledge (EE) and a registrar or PKI component.

authenticated self-contained object: Describes in this context an object that is cryptographically bound to the IDevID certificate of a pledge. The binding is assumed to be provided through a digital signature of the actual object using the IDevID secret.

3. Requirements discussion and mapping to solution elements

There were two main drivers for the definition of BRSKI-AE:

- * The solution architecture may already use or require a certificate management protocol other than EST. Therefore, this other protocol should be usable for requesting LDevID certificates.
- * The domain registrar may not be the (final) point that authenticates and authorizes certification requests and the pledge may not have a direct connection to it. Therefore, certification requests should be self-contained signed objects.

Based on the intended target environment described in Section 1.2 and the application examples described in Appendix B, the following requirements are derived to support authenticated self-contained objects as containers carrying certification requests.

At least the following properties are required:

- * proof-of-possession: demonstrates access to the private key corresponding to the public key contained in a certification request. This is typically achieved by a self-signature using the corresponding private key.
- * proof-of-identity: provides data origin authentication of the certification request. This typically is achieved by a signature using the IDevID secret of the pledge.

Here is an incomplete list of solution examples, based on existing technology described in IETF documents:

- * Certification request objects: Certification requests are data structures protecting only the integrity of the contained data and providing proof-of-possession for a (locally generated) private key. Examples for certification request data structures are:
 - PKCS#10 [RFC2986]. This certification request structure is self-signed to protect its integrity and prove possession of the private key that corresponds to the public key included in the request.
 - CRMF [RFC4211]. Also this certificate request message format supports integrity protection and proof-of-possession, typically by a self-signature generated over (part of) the structure with the private key corresponding to the included public key. CRMF also supports further proof-of-possession methods for types of keys that do not support any signature algorithm.

The integrity protection of certification request fields includes the public key because it is part of the data signed by the corresponding private key. Yet note that for the above examples this is not sufficient to provide data origin authentication, i.e., proof-of-identity. This extra property can be achieved by an additional binding to the IDevID of the pledge. This binding to source authentication supports the authorization decision for the certification request. The binding of data origin authentication to the certification request may be delegated to the protocol used for certificate management.

- * Solution options for proof-of-identity: The certification request should be bound to an existing authenticated credential (here, the IDevID certificate) to enable a proof of identity and, based on it, an authorization of the certification request. The binding may be achieved through security options in an underlying transport protocol such as TLS if the authorization of the certification request is (completely) done at the next communication hop. This binding can also be done in a transport-independent way by wrapping the certification request with signature employing an existing IDevID. In the BRSKI context, this will be the IDevID. This requirement is addressed by existing enrollment protocols in various ways, such as:
 - EST [RFC7030] utilizes PKCS#10 to encode the certification request. The Certificate Signing Request (CSR) optionally provides a binding to the underlying TLS session by including the tls-unique value in the self-signed PKCS#10 structure. The tls-unique value results from the TLS handshake. Since the TLS handshake includes client authentication and the pledge utilizes its IDevID for it, the proof-of-identity is provided by such a binding to the TLS session. This can be supported using the EST /simpleenroll endpoint. Note that the binding of the TLS handshake to the CSR is optional in EST. As an alternative to binding to the underlying TLS authentication in the transport layer, [RFC7030] sketches wrapping the CSR with a Full PKI Request message using an existing certificate.
 - SCEP [RFC8894] supports using a shared secret (passphrase) or an existing certificate to protect CSRs based on SCEP Secure Message Objects using CMS wrapping ([RFC5652]). Note that the wrapping using an existing IDevID in SCEP is referred to as renewal. Thus SCEP does not rely on the security of the underlying transfer.
 - CMP [RFC4210] supports using a shared secret (passphrase) or an existing certificate, which may be an IDevID credential, to authenticate certification requests via the PKIProtection structure in a PKIMessage. The certification request is typically encoded utilizing CRMF, while PKCS#10 is supported as an alternative. Thus CMP does not rely on the security of the underlying transfer protocol.

- CMC [RFC5272] also supports utilizing a shared secret (passphrase) or an existing certificate to protect certification requests, which can be either in CRMF or PKCS#10 structure. The proof-of-identity can be provided as part of a FullCMCRequest, based on CMS [RFC5652] and signed with an existing IDevID secret. Thus CMC does not rely on the security of the underlying transfer protocol.

4. Adaptations to BRSKI

In order to support alternative enrollment protocols, asynchronous enrollment, and more general system architectures, BRSKI-AE lifts some restrictions of BRSKI [RFC8995]. This way, authenticated self-contained objects such as those described in Section 3 above can be used for certificate enrollment.

The enhancements needed are kept to a minimum in order to ensure reuse of already defined architecture elements and interactions. In general, the communication follows the BRSKI model and utilizes the existing BRSKI architecture elements. In particular, the pledge initiates communication with the domain registrar and interacts with the MASA as usual.

4.1. Architecture

The key element of BRSKI-AE is that the authorization of a certification request **MUST** be performed based on an authenticated self-contained object. The certification request is bound in a self-contained way to a proof-of-origin based on the IDevID. Consequently, the authentication and authorization of the certification request **MAY** be done by the domain registrar and/or by other domain components. These components may be offline or reside in some central backend of the domain operator (off-site) as described in Section 1.2. The registrar and other on-site domain components may have no or only temporary (intermittent) connectivity to them. The certification request **MAY** also be piggybacked on another protocol.

This leads to generalizations in the placement and enhancements of the logical elements as shown in Figure 1.

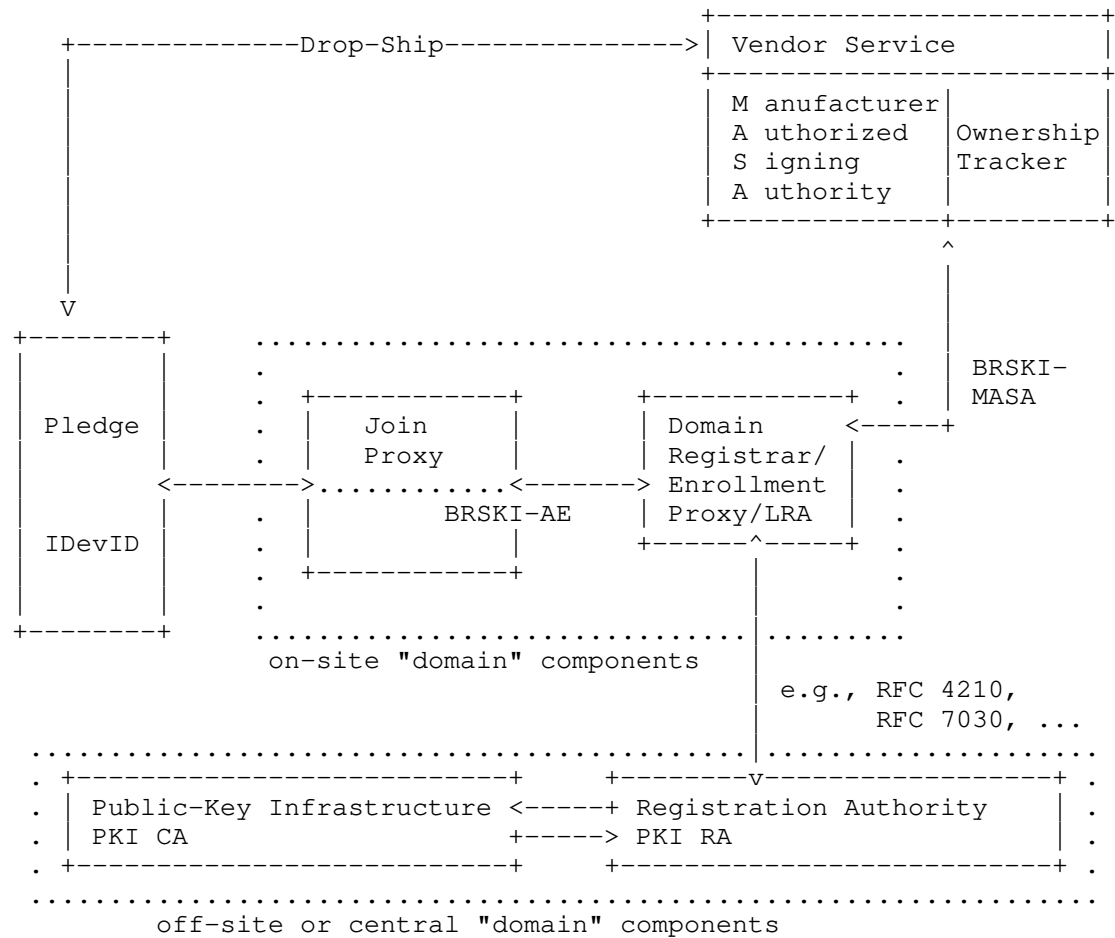


Figure 1: Architecture overview using off-site PKI components

The architecture overview in Figure 1 has the same logical elements as BRSKI, but with more flexible placement of the authentication and authorization checks on certification requests. Depending on the application scenario, the registrar MAY still do all of these checks (as is the case in BRSKI), or part of them, or none of them.

The following list describes the on-site components in the target domain of the pledge shown in Figure 1.

* Join Proxy: same functionality as described in BRSKI [RFC8995].

- * Domain Registrar / Enrollment Proxy / LRA: in BRSKI-AE, the domain registrar has mostly the same functionality as in BRSKI, namely to facilitate the communication of the pledge with the MASA and the PKI. Yet in contrast to BRSKI, the registrar offers different enrollment protocols and MAY act as a local registration authority (LRA) or simply as an enrollment proxy. In such cases, the domain registrar forwards the certification request to some off-site RA component, which performs at least part of the authorization. This also covers the case that the registrar has only intermittent connection and forwards the certification request to the RA upon re-established connectivity.

Note: To support alternative enrollment protocols, the URI scheme for addressing the domain registrar is generalized (see Section 4.2.5).

The following list describes the components provided by the vendor or manufacturer outside the target domain.

- * MASA: general functionality as described in BRSKI [RFC8995]. The voucher exchange with the MASA via the domain registrar is performed as described in BRSKI.

Note: The interaction with the MASA may be synchronous (voucher request with nonce) or asynchronous (voucher request without nonce).

- * Ownership tracker: as defined in BRSKI.

The following list describes the target domain components that can optionally be operated in the off-site backend of the target domain.

- * PKI RA: Performs certificate management functions for the domain as a centralized public-key infrastructure for the domain operator. As far as not already done by the domain registrar, it performs the final validation and authorization of certification requests.
- * PKI CA: Performs certificate generation by signing the certificate structure requested in already authenticated and authorized certification requests.

Based on the diagram in Section 2.1 of BRSKI [RFC8995] and the architectural changes, the original protocol flow is divided into three phases showing commonalities and differences to the original approach as follows.

- * Discovery phase: same as in BRSKI steps (1) and (2)

- * Voucher exchange phase: same as in BRSKI steps (3) and (4).
- * Enrollment phase: step (5) is changed to employing an alternative enrollment protocol that uses authenticated self-contained objects.

4.2. Message exchange

The behavior of a pledge described in Section 2.1 of BRSKI [RFC8995] is kept with one exception. After finishing the Imprint step (4), the Enroll step (5) MUST be performed with an enrollment protocol utilizing authenticated self-contained objects. Section 5 discusses selected suitable enrollment protocols and options applicable.

4.2.1. Pledge - Registrar discovery and voucher exchange

The discovery phase and voucher exchange are applied as specified in [RFC8995].

4.2.2. Registrar - MASA voucher exchange

This voucher exchange is performed as specified in [RFC8995].

4.2.3. Pledge - Registrar - RA/CA certificate enrollment

As stated in Section 3, the enrollment MUST be performed using an authenticated self-contained object providing not only proof-of-possession but also proof-of-identity (source authentication).

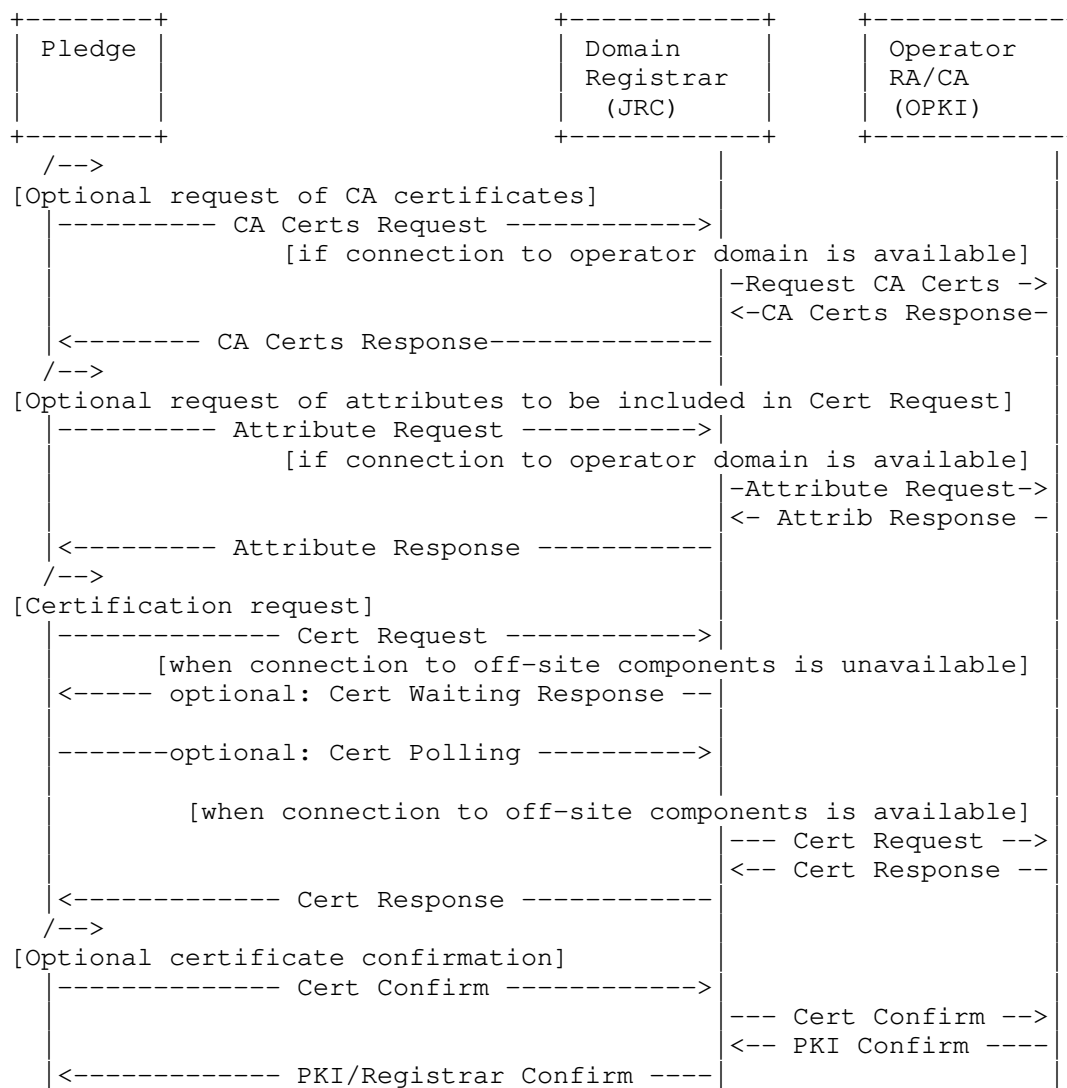


Figure 2: Certificate enrollment

The following list provides an abstract description of the flow depicted in Figure 2.

- * CA Cert Request: The pledge optionally requests the latest relevant CA certificates. This ensures that the pledge has the complete set of current CA certificates beyond the pinned-domain-cert (which is contained in the voucher and may be just the domain registrar certificate).

- * CA Cert Response: It MUST contain the current root CA certificate, which typically is the LDevID trust anchor, and any additional certificates that the pledge may need to validate certificates.
- * Attribute Request: Typically, the automated bootstrapping occurs without local administrative configuration of the pledge. Nevertheless, there are cases in which the pledge may also include additional attributes specific to the target domain into the certification request. To get these attributes in advance, the attribute request can be used.
- * Attribute Response: It MUST contain the attributes to be included in the subsequent certification request.
- * Cert Request: This certification request MUST contain the authenticated self-contained object ensuring both proof-of-possession of the corresponding private key and proof-of-identity of the requester.
- * Cert Response: The certification response message MUST contain on success the requested certificate and MAY include further information, like certificates of intermediate CAs.
- * Cert Waiting Response: Optional waiting indication for the pledge, which SHOULD poll for a Cert Response after a given time. To this end, a request identifier is necessary. The request identifier may be either part of the enrollment protocol or can be derived from the certification request.
- * Cert Polling: This SHOULD be used by the pledge in reaction to a Cert Waiting Response to query the registrar whether the certification request meanwhile has been processed. It MUST be answered either by another Cert Waiting, or the Cert Response.
- * Cert Confirm: An optional confirmation sent after the requested certificate has been received and validated. It contains a positive or negative confirmation by the pledge whether the certificate was successfully enrolled and fits its needs.
- * PKI/Registrar Confirm: An acknowledgment by the PKI or registrar that MUST be sent on reception of the Cert Confirm.

The generic messages described above may be implemented using various enrollment protocols supporting authenticated self-contained objects, as described in Section 3. Examples are available in Section 5.

4.2.4. Pledge - Registrar - enrollment status telemetry

The enrollment status telemetry is performed as specified in [RFC8995]. In BRSKI this is described as part of the enrollment phase, but due to the generalization on the enrollment protocol described in this document it fits better as a separate step here.

4.2.5. Addressing scheme enhancements

BRSKI-AE provides generalizations to the addressing scheme defined in BRSKI [RFC8995] to accommodate alternative enrollment protocols that use authenticated self-contained objects for certification requests. As this is supported by various existing enrollment protocols, they can be directly employed (see also Section 5).

The addressing scheme in BRSKI for certification requests and the related CA certificates and CSR attributes retrieval functions uses the definition from EST [RFC7030]; here on the example of simple enrollment: `"/.well-known/est/simpleenroll"`. This approach is generalized to the following notation: `"/.well-known/<enrollment-protocol>/<request>"` in which `<enrollment-protocol>` refers to a certificate enrollment protocol. Note that enrollment is considered here a message sequence that contains at least a certification request and a certification response. The following conventions are used in order to provide maximal compatibility to BRSKI:

- * `<enrollment-protocol>`: MUST reference the protocol being used, which MAY be CMP, CMC, SCEP, EST [RFC7030] as in BRSKI, or a newly defined approach.

Note: additional endpoints (well-known URIs) at the registrar may need to be defined by the enrollment protocol being used.

- * `<request>`: if present, the `<request>` path component MUST describe, depending on the enrollment protocol being used, the operation requested. Enrollment protocols are expected to define their request endpoints, as done by existing protocols (see also Section 5).

4.3. Domain registrar support of alternative enrollment protocols

Well-known URIs for various endpoints on the domain registrar are already defined as part of the base BRSKI specification or indirectly by EST. In addition, alternative enrollment endpoints MAY be supported at the registrar. The pledge will recognize whether its preferred enrollment option is supported by the domain registrar by sending a request to its preferred enrollment endpoint and evaluating the HTTP response status code.

The following list of endpoints provides an illustrative example for a domain registrar supporting several options for EST as well as for CMP to be used in BRSKI-AE. The listing contains the supported endpoints to which the pledge may connect for bootstrapping. This includes the voucher handling as well as the enrollment endpoints. The CMP related enrollment endpoints are defined as well-known URIs in CMP Updates [I-D.ietf-lamps-cmp-updates] and the Lightweight CMP profile [I-D.ietf-lamps-lightweight-cmp-profile].

```
</brski/voucherrequest>,ct=voucher-cms+json
</brski/voucher_status>,ct=json
</brski/enrollstatus>,ct=json
</est/cacerts>;ct=pkcs7-mime
</est/fullcmc>;ct=pkcs7-mime
</est/csrattrs>;ct=pkcs7-mime
</cmp/initialization>;ct=pkixcmp
</cmp/pl0>;ct=pkixcmp
</cmp/getcacerts>;ct=pkixcmp
</cmp/getcertreqtemplate>;ct=pkixcmp
```

5. Examples for signature-wrapping using existing enrollment protocols

This section maps the requirements to support proof-of-possession and proof-of-identity to selected existing enrollment protocols.

5.1. Instantiation to EST (informative)

When using EST [RFC7030], the following aspects and constraints need to be considered and the given extra requirements need to be fulfilled, which adapt Section 5.9.3 of BRSKI [RFC8995]:

- * proof-of-possession is provided typically by using the specified PKCS#10 structure in the request. Together with Full PKI requests, also CRMF can be used.
- * proof-of-identity needs to be achieved by signing the certification request object using the Full PKI Request option (including the /fullcmc endpoint). This provides sufficient information for the RA to authenticate the pledge as the origin of the request and to make an authorization decision on the received certification request. Note: EST references CMC [RFC5272] for the definition of the Full PKI Request. For proof-of-identity, the signature of the SignedData of the Full PKI Request is performed using the IDevID secret of the pledge.

Note: In this case the binding to the underlying TLS connection is not necessary.

- * When the RA is temporarily not available, as per Section 4.2.3 of [RFC7030], an HTTP status code 202 should be returned by the registrar, and the pledge will repeat the initial Full PKI Request

5.2. Instantiation to CMP (normative if CMP is chosen)

Note: Instead of referring to CMP as specified in [RFC4210] and [I-D.ietf-lamps-cmp-updates], this document refers to the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile] because the subset of CMP defined there is sufficient for the functionality needed here.

When using CMP, the following requirements SHALL be fulfilled:

- * For proof-of-possession, the approach defined in Section 4.1.1 (based on CRMF) or Section 4.1.4 (based on PKCS#10) of the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile] SHALL be applied.
- * proof-of-identity SHALL be provided by using signature-based protection of the certification request message as outlined in Section 3.2. of [I-D.ietf-lamps-lightweight-cmp-profile] using the IDevID secret.
- * When the Cert Response from the RA/CA is not available and if polling is supported, the registrar SHALL a Cert Waiting Response as specified in Sections 4.4 and 5.1.2 of [I-D.ietf-lamps-lightweight-cmp-profile].
- * As far as requesting CA certificates or certificate request attributes is supported, they SHALL be implemented as specified in Sections 4.3.1 and 4.3.3 of [I-D.ietf-lamps-lightweight-cmp-profile].

TBD RFC Editor: please delete /* ToDo: The following aspects need to be further specified: * Whether to use /getcacerts or the caPubs and extraCerts fields to return trust anchor and CA Certificates * Whether to use /getcertreqtemplate or modify the CRMF and use raVerified * Whether to specify the usage of /p10 */

6. IANA Considerations

This document does not require IANA actions.

7. Security Considerations

The security considerations as laid out in BRSKI [RFC8995] apply for the discovery and voucher exchange as well as for the status exchange information.

The security considerations as laid out in the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile] apply as far as CMP is used.

8. Acknowledgments

We would like to thank Brian E. Carpenter, Michael Richardson, and Giorgio Romanenghi for their input and discussion on use cases and call flows.

9. References

9.1. Normative References

[I-D.ietf-lamps-cmp-updates]

Brockhaus, H., Oheimb, D. V., and J. Gray, "Certificate Management Protocol (CMP) Updates", Work in Progress, Internet-Draft, draft-ietf-lamps-cmp-updates-17, 12 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-lamps-cmp-updates-17.txt>>.

[I-D.ietf-lamps-lightweight-cmp-profile]

Brockhaus, H., Oheimb, D. V., and S. Fries, "Lightweight Certificate Management Protocol (CMP) Profile", Work in Progress, Internet-Draft, draft-ietf-lamps-lightweight-cmp-profile-10, 1 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-lamps-lightweight-cmp-profile-10.txt>>.

[IEEE.802.1AR_2009]

IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR-2009, DOI 10.1109/ieeestd.2009.5367679, 28 December 2009, <<http://ieeexplore.ieee.org/servlet/opac?punumber=5367676>>.

[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>>.

- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.

9.2. Informative References

- [IEC-62351-9] International Electrotechnical Commission, "IEC 62351 - Power systems management and associated information exchange - Data and communications security - Part 9: Cyber security key management for power system equipment", IEC 62351-9, May 2017.
- [ISO-IEC-15118-2] International Standardization Organization / International Electrotechnical Commission, "ISO/IEC 15118-2 Road vehicles - Vehicle-to-Grid Communication Interface - Part 2: Network and application protocol requirements", ISO/IEC 15118-2, April 2014.
- [NERC-CIP-005-5] North American Reliability Council, "Cyber Security - Electronic Security Perimeter", CIP 005-5, December 2013.
- [Ocpp] Open Charge Alliance, "Open Charge Point Protocol 2.0.1 (Draft)", December 2019.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<https://www.rfc-editor.org/info/rfc2986>>.

- [RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <<https://www.rfc-editor.org/info/rfc4211>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [RFC8894] Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <<https://www.rfc-editor.org/info/rfc8894>>.
- [UNISIG-Subset-137]
UNISIG, "Subset-137; ERTMS/ETCS On-line Key Management FFFIS; V1.0.0", December 2015, <https://www.era.europa.eu/sites/default/files/filesystem/ertms/ccs_tsi_annex_a_-_mandatory_specifications/set_of_specifications_3_etcs_b3_r2_gsm-r_b1/index083_-_subset-137_v100.pdf>.
<http://www.kmc-subset137.eu/index.php/download/>

Appendix A. Using EST for certificate enrollment

When using EST with BRSKI, pledges interact via TLS with the domain registrar, which acts both as EST server and as registration authority (RA). The TLS connection is mutually authenticated, where the pledge uses its IDevID certificate issued by its manufacturer.

In order to provide a strong proof-of-origin of the certification request, EST has the option to include in the certification request the so-called `tls-unique` value [RFC5929] of the underlying TLS channel. This binding of the proof-of-identity of the TLS client, which is supposed to be the certificate requester, to the proof-of-possession for the private key is conceptually non-trivial and requires specific support by TLS implementations.

The registrar terminates the security association with the pledge at TLS level and thus the binding between the certification request and the authentication of the pledge. The EST server uses the authenticated pledge identity provided by the LDevID for checking the authorization of the pledge for the given certification request before issuing to the pledge a domain-specific certificate (LDevID certificate). This approach typically requires online or on-site availability of the RA for performing the final authorization decision for the certification request.

Using EST for BRSKI has the advantage that the mutually authenticated TLS connection established between the pledge and the registrar can be reused for protecting the message exchange needed for enrolling the LDevID certificate. This strongly simplifies the implementation of the enrollment message exchange.

Yet the use of TLS has the limitation that this cannot provide auditability nor end-to-end security for the certificate enrollment request because the TLS session is transient and terminates at the registrar. This is a problem in particular if the enrollment is done via multiple hops, part of which may not even be network-based.

A further limitation of using EST as the certificate enrollment protocol is that due to using PKCS#10 structures in enrollment requests, the only possible proof-of-possession method is a self-signature, which excludes requesting certificates for key types that do not support signing.

Appendix B. Application examples

This informative annex provides some detail to the application examples listed in Section 1.3.

B.1. Rolling stock

Rolling stock or railroad cars contain a variety of sensors, actuators, and controllers, which communicate within the railroad car but also exchange information between railroad cars building a train, with track-side equipment, and/or possibly with backend systems. These devices are typically unaware of backend system connectivity. Managing certificates may be done during maintenance cycles of the railroad car, but can already be prepared during operation. Preparation will include generating certification requests, which are collected and later forwarded for processing, once the railroad car is connected to the operator backend. The authorization of the certification request is then done based on the operator's asset/inventory information in the backend.

UNISIG has included a CMP profile for enrollment of TLS certificates of on-board and track-side components in the Subset-137 specifying the ETRAM/ETCS on-line key management for train control systems [UNISIG-Subset-137].

B.2. Building automation

In building automation scenarios, a detached building or the basement of a building may be equipped with sensors, actuators, and controllers that are connected with each other in a local network but with only limited or no connectivity to a central building management system. This problem may occur during installation time but also during operation. In such a situation a service technician collects the necessary data and transfers it between the local network and the central building management system, e.g., using a laptop or a mobile phone. This data may comprise parameters and settings required in the operational phase of the sensors/actuators, like a component certificate issued by the operator to authenticate against other components and services.

The collected data may be provided by a domain registrar already existing in the local network. In this case connectivity to the backend PKI may be facilitated by the service technician's laptop. Alternatively, the data can also be collected from the pledges directly and provided to a domain registrar deployed in a different network as preparation for the operational phase. In this case, connectivity to the domain registrar may also be facilitated by the service technician's laptop.

B.3. Substation automation

In electrical substation automation scenarios, a control center typically hosts PKI services to issue certificates for Intelligent Electronic Devices (IEDs) operated in a substation. Communication between the substation and control center is performed through a proxy/gateway/DMZ, which terminates protocol flows. Note that [NERC-CIP-005-5] requires inspection of protocols at the boundary of a security perimeter (the substation in this case). In addition, security management in substation automation assumes central support of several enrollment protocols in order to support the various capabilities of IEDs from different vendors. The IEC standard IEC62351-9 [IEC-62351-9] specifies mandatory support of two enrollment protocols: SCEP [RFC8894] and EST [RFC7030] for the infrastructure side, while the IED must only support one of the two.

B.4. Electric vehicle charging infrastructure

For electric vehicle charging infrastructure, protocols have been defined for the interaction between the electric vehicle and the charging point (e.g., ISO 15118-2 [ISO-IEC-15118-2]) as well as between the charging point and the charging point operator (e.g. OCPP [OCPP]). Depending on the authentication model, unilateral or mutual authentication is required. In both cases the charging point uses an X.509 certificate to authenticate itself in TLS connections between the electric vehicle and the charging point. The management of this certificate depends, among others, on the selected backend connectivity protocol. In the case of OCPP, this protocol is meant to be the only communication protocol between the charging point and the backend, carrying all information to control the charging operations and maintain the charging point itself. This means that the certificate management needs to be handled in-band of OCPP. This requires the ability to encapsulate the certificate management messages in a transport-independent way. Authenticated self-containment will support this by allowing the transport without a separate enrollment protocol, binding the messages to the identity of the communicating endpoints.

B.5. Infrastructure isolation policy

This refers to any case in which network infrastructure is normally isolated from the Internet as a matter of policy, most likely for security reasons. In such a case, limited access to external PKI services will be allowed in carefully controlled short periods of time, for example when a batch of new devices is deployed, and forbidden or prevented at other times.

B.6. Sites with insufficient level of operational security

The registration authority performing (at least part of) the authorization of a certification request is a critical PKI component and therefore requires higher operational security than components utilizing the issued certificates for their security features. CAs may also demand higher security in the registration procedures. Especially the CA/Browser forum currently increases the security requirements in the certificate issuance procedures for publicly trusted certificates. In case the on-site components of the target domain cannot be operated securely enough for the needs of a registration authority, this service should be transferred to an off-site backend component that has a sufficient level of security.

Appendix C. History of changes TBD RFC Editor: please delete

From IETF draft 04 -> IETF draft 05:

- * David von Oheimb became the editor.
- * Streamline wording, consolidate terminology, improve grammar, etc.
- * Shift the emphasis towards supporting alternative enrollment protocols.
- * Update the title accordingly - preliminary change to be approved.
- * Move comments on EST and detailed application examples to informative annex.
- * Move the remaining text of section 3 as two new sub-sections of section 1.

From IETF draft 03 -> IETF draft 04:

- * Moved UC2 related parts defining the pledge in responder mode to a separate document. This required changes and adaptations in several sections. Main changes concerned the removal of the subsection for UC2 as well as the removal of the YANG model related text as it is not applicable in UC1.
- * Updated references to the Lightweight CMP Profile.
- * Added David von Oheimb as co-author.

From IETF draft 02 -> IETF draft 03:

- * Housekeeping, deleted open issue regarding YANG voucher-request in UC2 as voucher-request was enhanced with additional leaf.
- * Included open issues in YANG model in UC2 regarding assertion value agent-proximity and CSR encapsulation using SZTP sub module).

From IETF draft 01 -> IETF draft 02:

- * Defined call flow and objects for interactions in UC2. Object format based on draft for JOSE signed voucher artifacts and aligned the remaining objects with this approach in UC2 .
- * Terminology change: issue #2 pledge-agent -> registrar-agent to better underline agent relation.
- * Terminology change: issue #3 PULL/PUSH -> pledge-initiator-mode and pledge-responder-mode to better address the pledge operation.
- * Communication approach between pledge and registrar-agent changed by removing TLS-PSK (former section TLS establishment) and associated references to other drafts in favor of relying on higher layer exchange of signed data objects. These data objects are included also in the pledge-voucher-request and lead to an extension of the YANG module for the voucher-request (issue #12).
- * Details on trust relationship between registrar-agent and registrar (issue #4, #5, #9) included in UC2.
- * Recommendation regarding short-lived certificates for registrar-agent authentication towards registrar (issue #7) in the security considerations.
- * Introduction of reference to agent signing certificate using SKID in agent signed data (issue #11).
- * Enhanced objects in exchanges between pledge and registrar-agent to allow the registrar to verify agent-proximity to the pledge (issue #1) in UC2.
- * Details on trust relationship between registrar-agent and pledge (issue #5) included in UC2.
- * Split of use case 2 call flow into sub sections in UC2.

From IETF draft 00 -> IETF draft 01:

- * Update of scope in Section 1.2 to include in which the pledge acts as a server. This is one main motivation for use case 2.
- * Rework of use case 2 to consider the transport between the pledge and the pledge-agent. Addressed is the TLS channel establishment between the pledge-agent and the pledge as well as the endpoint definition on the pledge.
- * First description of exchanged object types (needs more work)
- * Clarification in discovery options for enrollment endpoints at the domain registrar based on well-known endpoints in Section 4.3 do not result in additional /.well-known URIs. Update of the illustrative example. Note that the change to /brski for the voucher related endpoints has been taken over in the BRSKI main document.
- * Updated references.
- * Included Thomas Werner as additional author for the document.

From individual version 03 -> IETF draft 00:

- * Inclusion of discovery options of enrollment endpoints at the domain registrar based on well-known endpoints in Section 4.3 as replacement of section 5.1.3 in the individual draft. This is intended to support both use cases in the document. An illustrative example is provided.
- * Missing details provided for the description and call flow in pledge-agent use case UC2, e.g. to accommodate distribution of CA certificates.
- * Updated CMP example in Section 5 to use Lightweight CMP instead of CMP, as the draft already provides the necessary /.well-known endpoints.
- * Requirements discussion moved to separate section in Section 3. Shortened description of proof of identity binding and mapping to existing protocols.
- * Removal of copied call flows for voucher exchange and registrar discovery flow from [RFC8995] in Section 4 to avoid doubling or text or inconsistencies.

- * Reworked abstract and introduction to be more crisp regarding the targeted solution. Several structural changes in the document to have a better distinction between requirements, use case description, and solution description as separate sections. History moved to appendix.

From individual version 02 -> 03:

- * Update of terminology from self-contained to authenticated self-contained object to be consistent in the wording and to underline the protection of the object with an existing credential. Note that the naming of this object may be discussed. An alternative name may be attestation object.
- * Simplification of the architecture approach for the initial use case having an offsite PKI.
- * Introduction of a new use case utilizing authenticated self-contained objects to onboard a pledge using a commissioning tool containing a pledge-agent. This requires additional changes in the BRSKI call flow sequence and led to changes in the introduction, the application example, and also in the related BRSKI-AE call flow.
- * Update of provided examples of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 4.2.5.

From individual version 01 -> 02:

- * Update of introduction text to clearly relate to the usage of IDevID and LDevID.
- * Definition of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 4.2.5. This section also contains a first discussion of an optional discovery mechanism to address situations in which the registrar supports more than one enrollment approach. Discovery should avoid that the pledge performs a trial and error of enrollment protocols.
- * Update of description of architecture elements and changes to BRSKI in Section 4.1.
- * Enhanced consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 3 and in Section 5.

From individual version 00 -> 01:

- * Update of examples, specifically for building automation as well as two new application use cases in Appendix B.
- * Deletion of asynchronous interaction with MASA to not complicate the use case. Note that the voucher exchange can already be handled in an asynchronous manner and is therefore not considered further. This resulted in removal of the alternative path the MASA in Figure 1 and the associated description in Section 4.1.
- * Enhancement of description of architecture elements and changes to BRSKI in Section 4.1.
- * Consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 3.
- * New section starting Section 5 with the mapping to existing enrollment protocols by collecting boundary conditions.

Authors' Addresses

David von Oheimb (editor)
Siemens AG
Otto-Hahn-Ring 6
81739 Munich
Germany
Email: david.von.oheimb@siemens.com
URI: <https://www.siemens.com/>

Steffen Fries
Siemens AG
Otto-Hahn-Ring 6
81739 Munich
Germany
Email: steffen.fries@siemens.com
URI: <https://www.siemens.com/>

Hendrik Brockhaus
Siemens AG
Otto-Hahn-Ring 6
81739 Munich
Germany
Email: hendrik.brockhaus@siemens.com
URI: <https://www.siemens.com/>

Eliot Lear
Cisco Systems
Richtistrasse 7
CH-8304 Wallisellen
Switzerland
Phone: +41 44 878 9200
Email: lear@cisco.com

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2021

M. Richardson
Sandelman Software Works
P. van der Stok
vanderstok consultancy
P. Kampanakis
Cisco Systems
February 04, 2021

Constrained Join Proxy for Bootstrapping Protocols
draft-ietf-anima-constrained-join-proxy-02

Abstract

This document defines a protocol to securely assign a pledge to a domain, represented by a Registrar, using an intermediary node between pledge and Registrar. This intermediary node is known as a "constrained Join Proxy".

This document extends the work of [I-D.ietf-anima-bootstrapping-keyinfra] by replacing the Circuit-proxy by a stateless/stateful constrained (CoAP) Join Proxy. It transports join traffic from the pledge to the Registrar without requiring per-client state.

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 August 8, 2021.

Copyright Notice

Copyright (c) 2021 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 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
2. Terminology	4
3. Requirements Language	4
4. Join Proxy functionality	4
5. Join Proxy specification	5
5.1. Statefull Join Proxy	5
5.2. Stateless Join Proxy	6
5.3. Stateless Message structure	8
6. Comparison of stateless and statefull modes	9
7. Discovery	10
7.1. Pledge discovery of Registrar	11
7.1.1. CoAP discovery	11
7.1.2. Autonomous Network	11
7.1.3. 6tisch discovery	11
7.2. Pledge discovers Join Proxy	11
7.2.1. Autonomous Network	12
7.2.2. CoAP discovery	12
7.3. Join Proxy discovers Registrar join port	12
7.3.1. CoAP discovery	12
8. Security Considerations	13
9. IANA Considerations	13
9.1. Resource Type registry	13
10. Acknowledgements	14
11. Contributors	14
12. Changelog	14
12.1. 01 to 02	14
12.2. 00 to 01	14
12.3. 00 to 00	14
13. References	14
13.1. Normative References	14
13.2. Informative References	16
Appendix A. Stateless Proxy payload examples	17
Authors' Addresses	17

1. Introduction

Enrolment of new nodes into networks with enrolled nodes present is described in [I-D.ietf-anima-bootstrapping-keyinfra] ("BRSKI") and makes use of Enrolment over Secure Transport (EST) [RFC7030] with [RFC8366] vouchers to securely enroll devices. BRSKI connects new devices ("pledges") to "Registrars" via a Join Proxy.

The specified solutions use https and may be too large in terms of code space or bandwidth required for constrained devices. Constrained devices possibly part of constrained networks [RFC7228] typically implement the IPv6 over Low-Power Wireless personal Area Networks (6LoWPAN) [RFC4944] and Constrained Application Protocol (CoAP) [RFC7252].

CoAP can be run with the Datagram Transport Layer Security (DTLS) [RFC6347] as a security protocol for authenticity and confidentiality of the messages. This is known as the "coaps" scheme. A constrained version of EST, using Coap and DTLS, is described in [I-D.ietf-ace-coap-est]. The {I-D.ietf-anima-constrained-voucher} describes the BRSKI extensions to the Registrar.

DTLS is a client-server protocol relying on the underlying IP layer to perform the routing between the DTLS Client and the DTLS Server. However, the new "joining" device will not be IP routable until it is authenticated to the network. A new "joining" device can only initially use a link-local IPv6 address to communicate with a neighbour node using neighbour discovery [RFC6775] until it receives the necessary network configuration parameters. However, before the device can receive these configuration parameters, it needs to authenticate itself to the network to which it connects. IPv6 routing is necessary to establish a connection between joining device and the Registrar.

A DTLS connection is required between Pledge and Registrar.

This document specifies a new form of Join Proxy and protocol to act as intermediary between joining device and Registrar to establish a connection between joining device and Registrar.

This document is very much inspired by text published earlier in [I-D.kumar-dice-dtls-relay]. [I-D.richardson-anima-state-for-joinrouter] outlined the various options for building a join proxy. [I-D.ietf-anima-bootstrapping-keyinfra] adopted only the Circuit Proxy method (1), leaving the other methods as future work. This document standardizes the CoAP/DTLS (method 4).

2. Terminology

The following terms are defined in [RFC8366], and are used identically as in that document: artifact, imprint, domain, Join Registrar/Coordinator (JRC), Manufacturer Authorized Signing Authority (MASA), pledge, Trust of First Use (TOFU), and Voucher.

3. 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.

4. Join Proxy functionality

As depicted in the Figure 1, the joining Device, or pledge (P), in an LLN mesh can be more than one hop away from the Registrar (R) and not yet authenticated into the network.

In this situation, it can only communicate one-hop to its nearest neighbour, the Join Proxy (J) using their link-local IPv6 addresses. However, the Pledge (P) needs to communicate with end-to-end security with a Registrar hosting the Registrar (R) to authenticate and get the relevant system/network parameters. If the Pledge (P) initiates a DTLS connection to the Registrar whose IP address has been pre-configured, then the packets are dropped at the Join Proxy (J) since the Pledge (P) is not yet admitted to the network or there is no IP routability to Pledge (P) for any returned messages.

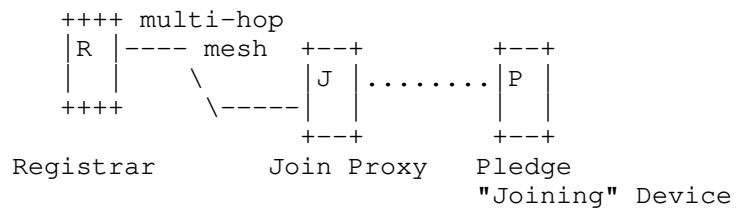


Figure 1: multi-hop enrolment.

Without routing the Pledge (P) cannot establish a secure connection to the Registrar (R) in the network assuming appropriate credentials are exchanged out-of-band, e.g. a hash of the Pledge (P)'s raw public key could be provided to the Registrar (R).

Furthermore, the Pledge (P) may be unaware of the IP address of the Registrar (R) to initiate a DTLS connection and perform authentication.

To overcome the problems with non-routability of DTLS packets and/or discovery of the destination address of the EST Server to contact, the Join Proxy is introduced. This Join Proxy functionality is configured into all authenticated devices in the network which may act as the Join Proxy for newly joining nodes. The Join Proxy allows for routing of the packets from the Pledge using IP routing to the intended Registrar.

5. Join Proxy specification

A Join Proxy can operate in two modes:

- o Statefull mode
- o Stateless mode

5.1. Statefull Join Proxy

In stateful mode, the joining node forwards the DTLS messages to the Registrar.

Assume that the Pledge does not know the IP address of the Registrar it needs to contact. The Join Proxy has been enrolled via the Registrar and consequently knows the IP address and port of the Registrar. The Pledge first discovers and selects the most appropriate Join Proxy. (Discovery can be based upon [I-D.ietf-anima-bootstrapping-keyinfra] section 4.3, or via DNS-SD service discovery [RFC6763]). The Pledge initiates its request as if the Join Proxy is the intended Registrar. The Join Proxy receives the message at a discoverable "Join" port. The Join Proxy changes the IP packet (without modifying the DTLS message) by modifying both the source and destination addresses to forward the message to the intended Registrar. The Join Proxy maintains a 4-tuple array to translate the DTLS messages received from the Registrar and forward it to the EST Client. This is a form of Network Address translation, where the Join Proxy acts as a forward proxy. In Figure 2 the various steps of the message flow are shown, with 5684 being the standard coaps port:

Pledge (P)	Join Proxy (J)	Registrar (R)	Message	
			Src_IP:port	Dst_IP:port
--ClientHello-->			IP_P:p_P	IP_Ja:p_J
	--ClientHello-->		IP_Jb:p_Jb	IP_R:5684
		<--ServerHello--	IP_R:5684	IP_Jb:p_Jb
		:		
<--ServerHello--		:	IP_Ja:p_J	IP_P:p_P
	:	:		
	:	:	:	:
	:	:	:	:
--Finished-->		:	IP_P:p_P	IP_Ja:p_J
	--Finished-->		IP_Jb:p_Jb	IP_R:5684
		<--Finished--	IP_R:5684	IP_Jb:p_Jb
<--Finished--			IP_Ja:p_J	IP_P:p_P
:		:	:	:

IP_P:p_P = Link-local IP address and port of Pledge (DTLS Client)

IP_R:5684 = Global IP address and coaps port of Registrar

IP_Ja:P_J = Link-local IP address and join port of Join Proxy

IP_Jb:p_Rb = Global IP address and client port of Join proxy

Figure 2: constrained statefull joining message flow with Registrar address known to Join Proxy.

5.2. Stateless Join Proxy

The stateless Join Proxy aims to minimize the requirements on the constrained Join Proxy device. Stateless operation requires no memory in the Join Proxy device, but may also reduce the CPU impact as the device does not need to search through a state table.

If an untrusted Pledge that can only use link-local addressing wants to contact a trusted Registrar, and the Registrar is more than one hop away, it sends the DTLS message to the Join Proxy.

When a Pledge attempts a DTLS connection to the Join Proxy, it uses its link-local IP address as its IP source address. This message is transmitted one-hop to a neighbouring (Join Proxy) node. Under normal circumstances, this message would be dropped at the neighbour node since the Pledge is not yet IP routable or is not yet authenticated to send messages through the network. However, if the neighbour device has the Join Proxy functionality enabled, it routes the DTLS message to its Registrar of choice.

The Join Proxy extends this message into a new type of message called Join ProxY (JPY) message and sends it on to the Registrar.

The JPY message payload consists of two parts:

- o Header (H) field: consisting of the source link-local address and port of the Pledge (P), and
- o Contents (C) field: containing the original DTLS message.

On receiving the JPY message, the Registrar retrieves the two parts.

The Registrar transiently stores the Header field information. The Registrar uses the Contents field to execute the Registrar functionality. However, when the Registrar replies, it also extends its DTLS message with the header field in a JPY message and sends it back to the Join Proxy. The Registrar SHOULD NOT assume that it can decode the Header Field, it should simply repeat it when responding. The Header contains the original source link-local address and port of the pledge from the transient state stored earlier and the Contents field contains the DTLS message.

On receiving the JPY message, the Join Proxy retrieves the two parts. It uses the Header field to route the DTLS message retrieved from the Contents field to the Pledge.

In this scenario, both the Registrar and the Join Proxy use discoverable "Join" ports.

The Figure 3 depicts the message flow diagram:

EST	Client (P)	Join Proxy (J)	Registrar (R)	Message	
				Src_IP:port	Dst_IP:port
		--ClientHello-->		IP_P:p_P	IP_Ja:p_Ja
		--JPY[H(IP_P:p_P),-->		IP_Jb:p_Jb	IP_R:p_Ra
		C(ClientHello)]			
		<--JPY[H(IP_P:p_P),--		IP_R:p_Ra	IP_Jb:p_Jb
		C(ServerHello)]			
		<--ServerHello--		IP_Ja:p_Ja	IP_P:p_P
		:		:	:
		:		:	:
		--Finished-->		IP_P:p_P	IP_Ja:p_Ja
		--JPY[H(IP_P:p_P),-->		IP_Jb:p_Jb	IP_R:p_Ra
		C(Finished)]			
		<--JPY[H(IP_P:p_P),--		IP_R:p_Ra	IP_Jb:p_Jb
		C(Finished)]			
		<--Finished--		IP_Ja:p_Ja	IP_P:p_P
		:		:	:

IP_P:p_P = Link-local IP address and port of the Pledge

IP_R:p_Ra = Global IP address and join port of Registrar

IP_Ja:p_Ja = Link-local IP address and join port of Join Proxy

IP_Jb:p_Jb = Global IP address and port of Join Proxy

JPY[H()],C()] = Join Proxy message with header H and content C

Figure 3: constrained stateless joining message flow.

5.3. Stateless Message structure

The JPY message is constructed as a payload with media-type application/cbor

Header and Contents fields together are one cbor array of 5 elements:

1. header field: containing a CBOR array [RFC7049] with the pledge IPv6 Link Local address as a cbor byte string, the pledge's UDP port number as a CBOR integer, the IP address family (IPv4/IPv6) as a cbor integer, and the proxy's ifindex or other identifier for the physical port as cbor integer. The header field is not DTLS encrypted.
2. Content field: containing the DTLS encrypted payload as a CBOR byte string.

The join_proxy cannot decrypt the DTLS encrypted payload and has no knowledge of the transported media type.

```
JPY_message =  
[  
    ip      : bstr,  
    port    : int,  
    family   : int,  
    index    : int  
    payload : bstr  
]
```

Figure 4: CDDL representation of JPY message

The content fields are DTLS encrypted. In CBOR diagnostic notation the payload JPY[H(IP_P:p_P)], will look like:

```
[h'IP_p', p_P, family, ident, h'DTLS-content']
```

Examples are shown in Appendix A.

6. Comparison of stateless and statefull modes

The stateful and stateless mode of operation for the Join Proxy have their advantages and disadvantages. This section should enable to make a choice between the two modes based on the available device resources and network bandwidth.

Properties	Stateful mode	Stateless mode
State Information	The Join Proxy needs additional storage to maintain mapping between the address and port number of the pledge and those of the Registrar.	No information is maintained by the Join Proxy. Registrar needs to store the packet header.
Packet size	The size of the forwarded message is the same as the original message.	Size of the forwarded message is bigger than the original, it includes additional source and destination addresses.
Specification complexity	The Join Proxy needs additional functionality to maintain state information, and modify the source and destination addresses of the DTLS handshake messages	New JPY message to encapsulate DTLS message The Registrar and the Join Proxy have to understand the JPY message in order to process it.
Ports	Join Proxy needs discoverable "Join" port	Join Proxy and Registrar need discoverable "Join" ports

Figure 5: Comparison between stateful and stateless mode

7. Discovery

It is assumed that Join Proxy seamlessly provides a coaps connection between Pledge and coaps Registrar. In particular this section replaces section 4.2 of [I-D.ietf-anima-bootstrapping-keyinfra].

The discovery follows two steps:

1. The pledge is one hop away from the Registrar. The pledge discovers the link-local address of the Registrar as described in {I-D.ietf-ace-coap-est}. From then on, it follows the BRSKI process as described in {I-D.ietf-ace-coap-est}, using link-local addresses.
2. The pledge is more than one hop away from a relevant Registrar, and discovers the link-local address and join port of a Join

Proxy. The pledge then follows the BRSKI procedure using the link-local address of the Join Proxy.

3. The stateless Join Proxy discovers the join port of the Registrar

Once a pledge is enrolled, it may function as Join Proxy. The Join Proxy functions are advertised as described below. In principle, the Join Proxy functions are offered via a "join" port, and not the standard coaps port. Also the Registrar offer a "join" port to which the stateless join proxy sends the JPY message. The Join Proxy and Registrar MUST show the extra join port number when repending to the .well-known/core request addressed to the standard coap/coaps port.

Three discovery cases are discussed: coap discovery, 6tisch discovery and GRASP discovery.

7.1. Pledge discovery of Registrar

The Pledge and Join Proxy are assumed to communicate via Link-Local addresses.

7.1.1. CoAP discovery

The discovery of the coaps Registrar, using coap discovery, by the Join Proxy follows section 6 of [I-D.ietf-ace-coap-est]. The extension to discover the additional port needed by the stateless proxy is described in Section 7.2.2.

7.1.2. Autonomous Network

In the context of autonomous networks, the Join Proxy uses the DULL GRASP M_FLOOD mechanism to announce itself. Section 4.1.1 of [I-D.ietf-anima-bootstrapping-keyinfra] discusses this in more detail. The Registrar announces itself using ACP instance of GRASP using M_FLOOD messages. Autonomous Network Join Proxies MUST support GRASP discovery of Registrar as decribed in section 4.3 of [I-D.ietf-anima-bootstrapping-keyinfra] .

7.1.3. 6tisch discovery

The discovery of Registrar by the pledge uses the enhanced beacons as discussed in [I-D.ietf-6tisch-enrollment-enhanced-beacon].

7.2. Pledge discovers Join Proxy

7.2.1. Autonomous Network

The pledge MUST listen for GRASP M_FLOOD [I-D.ietf-anima-grasp] announcements of the objective: "AN_Proxy". See section Section 4.1.1 [I-D.ietf-anima-bootstrapping-keyinfra] for the details of the objective.

7.2.2. CoAP discovery

In the context of a coap network without Autonomous Network support, discovery follows the standard coap policy. The Pledge can discover a Join Proxy by sending a link-local multicast message to ALL CoAP Nodes with address FF02::FD. Multiple or no nodes may respond. The handling of multiple responses and the absence of responses follow section 4 of [I-D.ietf-anima-bootstrapping-keyinfra].

The join port of the Join Proxy is discovered by sending a GET request to `"/.well-known/core"` including a resource type (rt) parameter with the value `"brski-proxy"` [RFC6690]. Upon success, the return payload will contain the join port.

The example below shows the discovery of the join port of the Join Proxy.

```
REQ: GET coap://[FF02::FD]/.well-known/core?rt=brski-proxy
```

```
RES: 2.05 Content
```

```
<coaps://[IP_address]:join-port>; rt="brski-proxy"
```

Port numbers are assumed to be the default numbers 5683 and 5684 for coap and coaps respectively (sections 12.6 and 12.7 of [RFC7252] when not shown in the response. Discoverable port numbers are usually returned for Join Proxy resources in the `<href>` of the payload (see section 5.1 of [I-D.ietf-ace-coap-est]).

7.3. Join Proxy discovers Registrar join port

7.3.1. CoAP discovery

The stateless Join Proxy can discover the join port of the Registrar by sending a GET request to `"/.well-known/core"` including a resource type (rt) parameter with the value `"join-proxy"` [RFC6690]. Upon success, the return payload will contain the join Port of the Registrar.

```
REQ: GET coap://[IP_address]/.well-known/core?rt=brski-proxy
```

```
RES: 2.05 Content
```

```
<coaps://[IP_address]:join-port>; rt="join-proxy"
```

The discoverable port numbers are usually returned for Join Proxy resources in the <href> of the payload (see section 5.1 of [I-D.ietf-ace-coap-est]).

8. Security Considerations

It should be noted here that the contents of the CBOR map used to convey return address information is not protected. However, the communication is between the Proxy and a known registrar are over the already secured portion of the network, so are not visible to eavesdropping systems.

All of the concerns in [I-D.ietf-anima-bootstrapping-keyinfra] section 4.1 apply. The pledge can be deceived by malicious AN_Proxy announcements. The pledge will only join a network to which it receives a valid [RFC8366] voucher.

If the proxy/Registrar was not over a secure network, then an attacker could change the cbor array, causing the pledge to send traffic to another node. If the such scenario needed to be supported, then it would be reasonable for the Proxy to encrypt the CBOR array using a locally generated symmetric key. The Registrar would not be able to examine the result, but it does not need to do so. This is a topic for future work.

9. IANA Considerations

This document needs to create a registry for key indices in the CBOR map. It should be given a name, and the amending formula should be IETF Specification.

9.1. Resource Type registry

This specification registers a new Resource Type (rt=) Link Target Attributes in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

rt="brski-proxy". This BRSKI resource is used to query and return the supported BRSKI port of the Join Proxy.

rt="join-proxy". This BRSKI resource is used to query and return the supported BRSKI port of the Registrar.

10. Acknowledgements

Many thanks for the comments by Brian Carpenter and Esko Dijk.

11. Contributors

Sandeep Kumar, Sye loong Keoh, and Oscar Garcia-Morchon are the co-authors of the draft-kumar-dice-dtls-relay-02. Their draft has served as a basis for this document. Much text from their draft is copied over to this draft.

12. Changelog

12.1. 01 to 02

- o Discovery of Join Proxy and Registrar ports

12.2. 00 to 01

- o Registrar used throughout instead of EST server
- o Emphasized additional Join Proxy port for Join Proxy and Registrar
- o updated discovery accordingly
- o updated stateless Join Proxy JPY header
- o JPY header described with CDDL
- o Example simplified and corrected

12.3. 00 to 00

- o copied from vanderstok-anima-constrained-join-proxy-05

13. References

13.1. Normative References

[I-D.ietf-6tisch-enrollment-enhanced-beacon]
Dujovne, D. and M. Richardson, "IEEE 802.15.4 Information Element encapsulation of 6TiSCH Join and Enrollment Information", draft-ietf-6tisch-enrollment-enhanced-beacon-14 (work in progress), February 2020.

- [I-D.ietf-ace-coap-est]
Stok, P., Kampanakis, P., Richardson, M., and S. Raza,
"EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-
est-18 (work in progress), January 2020.
- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-45 (work in progress), November 2020.
- [I-D.ietf-anima-constrained-voucher]
Richardson, M., Stok, P., and P. Kampanakis, "Constrained
Voucher Artifacts for Bootstrapping Protocols", draft-
ietf-anima-constrained-voucher-09 (work in progress),
November 2020.
- [I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic
Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-
grasp-15 (work in progress), July 2017.
- [I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart
Content-Format for CoAP", draft-ietf-core-multipart-ct-04
(work in progress), August 2019.
- [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>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,
January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert,
"A Voucher Artifact for Bootstrapping Protocols",
RFC 8366, DOI 10.17487/RFC8366, May 2018,
<<https://www.rfc-editor.org/info/rfc8366>>.

13.2. Informative References

- [I-D.kumar-dice-dtls-relay]
Kumar, S., Keoh, S., and O. Garcia-Morchon, "DTLS Relay for Constrained Environments", draft-kumar-dice-dtls-relay-02 (work in progress), October 2014.
- [I-D.richardson-anima-state-for-joinrouter]
Richardson, M., "Considerations for stateful vs stateless join router in ANIMA bootstrap", draft-richardson-anima-state-for-joinrouter-03 (work in progress), September 2020.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [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>>.

Appendix A. Stateless Proxy payload examples

The examples show the get coaps://[192.168.1.200]:5965/est/crts to a Registrar. The header generated between Client and registrar and from registrar to client are shown in detail. The DTLS encrypted code is not shown.

The request from Join Proxy to Registrar looks like:

```

85                                     # array(5)
  50                                 # bytes(16)
    000000000000000000000000FFFC0A801C8 #
  19 BDA7                             # unsigned(48551)
  0A                                   # unsigned(10)
  00                                   # unsigned(0)
  58 2D                               # bytes(45)
<cacrts DTLS encrypted request>

```

In CBOR Diagnostic:

```

[h'000000000000000000000000FFFC0A801C8', 48551, 10, 0,
 h'<cacrts DTLS encrypted request>']

```

The response is:

```

85                                     # array(5)
  50                                 # bytes(16)
    000000000000000000000000FFFC0A801C8 #
  19 BDA7                             # unsigned(48551)
  0A                                   # unsigned(10)
  00                                   # unsigned(0)
  59 026A                             # bytes(618)
<cacrts DTLS encrypted response>

```

In CBOR diagnostic:

```

[h'000000000000000000000000FFFC0A801C8', 48551, 10, 0,
 h'<cacrts DTLS encrypted response>']

```

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 24, 2021

M. Richardson
Sandelman Software Works
P. van der Stok
vanderstok consultancy
P. Kampanakis
Cisco Systems
E. Dijk
IoTconsultancy.nl
February 20, 2021

Constrained Voucher Artifacts for Bootstrapping Protocols
draft-ietf-anima-constrained-voucher-10

Abstract

This document defines a protocol to securely assign a Pledge to an owner and to enroll it into the owner's network. The protocol uses an artifact that is signed by the Pledge's manufacturer. This artifact is known as a "voucher".

This document builds upon the work in [RFC8366] and [BRSKI], but defines an encoding of the voucher in CBOR rather than JSON, and enables the Pledge to perform its transactions using CoAP rather than HTTPS.

The use of Raw Public Keys instead of X.509 certificates for security operations is also explained.

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 August 24, 2021.

Copyright Notice

Copyright (c) 2021 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 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
2. Terminology	4
3. Requirements Language	4
4. Survey of Voucher Types	5
5. Discovery and URI	6
6. BRSKI-EST Protocol	7
6.1. Discovery, URIs and Content Formats	7
6.2. Discovery, URIs and Content Formats	7
6.3. Extensions to BRSKI	8
6.4. Extensions to EST-coaps	8
6.4.1. Pledge Extensions	8
6.4.2. Registrar Extensions	10
7. BRSKI-MASA Protocol	10
8. Pinning in Voucher Artifacts	11
8.1. Registrar Identity Selection and Encoding	11
8.2. MASA Pinning Policy	12
8.3. Pinning of Raw Public Keys	13
9. Artifacts	15
9.1. Voucher Request artifact	15
9.1.1. Tree Diagram	15
9.1.2. SID values	16
9.1.3. YANG Module	17
9.1.4. Example voucher request artifact	21
9.2. Voucher artifact	21
9.2.1. Tree Diagram	21
9.2.2. SID values	22
9.2.3. YANG Module	22
9.2.4. Example voucher artifacts	25
9.3. Signing voucher and voucher-request artifacts with COSE	26
10. Design Considerations	26
11. Security Considerations	27

11.1.	Clock Sensitivity	27
11.2.	Protect Voucher PKI in HSM	27
11.3.	Test Domain Certificate Validity when Signing	27
12.	IANA Considerations	27
12.1.	Resource Type Registry	27
12.2.	The IETF XML Registry	27
12.3.	The YANG Module Names Registry	28
12.4.	The RFC SID range assignment sub-registry	28
12.5.	Media-Type Registry	28
12.5.1.	application/voucher-cose+cbor	28
12.6.	CoAP Content-Format Registry	29
13.	Acknowledgements	29
14.	Changelog	30
15.	References	30
15.1.	Normative References	30
15.2.	Informative References	32
Appendix A.	EST messages to EST-coaps	33
A.1.	enrollstatus	33
A.2.	voucher_status	34
Appendix B.	COSE examples	35
B.1.	Pledge, Registrar and MASA keys	38
B.1.1.	Pledge private key	38
B.1.2.	Registrar private key	39
B.1.3.	MASA private key	39
B.2.	Pledge, Registrar and MASA certificates	40
B.2.1.	Pledge IDevID certificate	40
B.2.2.	Registrar Certificate	41
B.2.3.	MASA Certificate	43
B.3.	COSE signed voucher request from Pledge to Registrar	45
B.4.	COSE signed voucher request from Registrar to MASA	47
B.5.	COSE signed voucher from MASA to Pledge via Registrar	48
Authors' Addresses	49

1. Introduction

Secure enrollment of new nodes into constrained networks with constrained nodes presents unique challenges. There are network bandwidth and code size issues to contend with. A solution for autonomous enrollment such as [I-D.ietf-anima-bootstrapping-keyinfra] may be too large in terms of code size or bandwidth required.

Therefore, this document defines a constrained version of the voucher artifact [RFC8366], along with a constrained version of BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] that makes use of the constrained CoAP-based version of EST, EST-coaps [I-D.ietf-ace-coap-est] rather than EST over HTTPS [RFC7030].

While the [RFC8366] voucher is by default serialized to JSON with a signature in CMS, this document defines a new voucher serialization to CBOR ([RFC7049]) with a signature in COSE [I-D.ietf-cose-rfc8152bis-struct]. This COSE-signed CBOR-encoded voucher can be transported using secured CoAP or HTTP. The CoAP connection (between Pledge and Registrar) is to be protected by either OSCORE+EDHOC, or DTLS (CoAPS). The HTTP connection (between Registrar and MASA) is to be protected using TLS (HTTPS).

This document has a similar structure to [RFC8366] but adds sections concerning:

1. Voucher-request artifact specification based on Section 3 of [I-D.ietf-anima-bootstrapping-keyinfra],
2. Voucher(-request) transport over CoAP based on Section 3 of [I-D.ietf-anima-bootstrapping-keyinfra] and on [I-D.ietf-ace-coap-est].

The CBOR definitions for the constrained voucher format are defined using the mechanism described in [I-D.ietf-core-yang-cbor] using the SID mechanism explained in [I-D.ietf-core-sid]. As the tooling to convert YANG documents into a list of SID keys is still in its infancy, the table of SID values presented here should be considered normative rather than the output of the pyang tool.

There is additional work when the voucher is integrated into the key-exchange, described in [I-D.selander-ace-ake-authz]. This work is not in scope for this document.

2. Terminology

The following terms are defined in [RFC8366], and are used identically as in that document: artifact, domain, imprint, Join Registrar/Coordinator (JRC), Manufacturer Authorized Signing Authority (MASA), Pledge, Registrar, Trust of First Use (TOFU), and Voucher.

The following terms from [I-D.ietf-anima-bootstrapping-keyinfra] are used identically as in that document: Domain CA, enrollment, IDevID, Join Proxy, LDevID, manufacturer, nonced, nonceless, PKIX.

3. 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.

4. Survey of Voucher Types

[RFC8366] provides for vouchers that assert proximity, that authenticate the Registrar and that can offer varying levels of anti-replay protection.

This document does not make any extensions to the semantic meanings of vouchers, only the encoding has been changed to optimize for constrained devices and networks.

Time-based vouchers are supported in this definition, but given that constrained devices are extremely unlikely to have accurate time, their use is very unlikely. Most Pledges using these constrained vouchers will be online during enrollment and will use live nonces to provide anti-replay protection.

[RFC8366] defined only the voucher artifact, and not the Voucher Request artifact, which was defined in [I-D.ietf-anima-bootstrapping-keyinfra]. This document defines both a constrained voucher and a constrained voucher-request. They are presented in the order "voucher-request", followed by a "voucher" response as this is the order that they occur in the protocol.

The constrained voucher request MUST be signed by the Pledge. It can sign using its IDevID X.509 certificate, or if an IDevID is not available its manufacturer-installed raw public key (RPK). The constrained voucher MUST be signed by the MASA.

For the constrained voucher request this document defines two distinct methods for the Pledge to identify the Registrar: using either the Registrar's X.509 certificate, or using a raw public key (RPK) of the Registrar. For the constrained voucher also these two methods are supported to indicate (pin) a trusted domain identity: using either a pinned domain X.509 certificate, or a pinned raw public key (RPK).

When the Pledge is known by MASA to support RPK but not X.509 certificates, the voucher produced by the MASA pins the raw public key of the Registrar in the "pinned-domain-subject-public-key-info" field of a voucher. This is described in more detail in the YANG definition for the constrained voucher and in section Section 8.

When the Pledge is known by MASA to support PKIX format certificates, the "pinned-domain-cert" field present in a voucher typically pins a domain certificate. That can be either the End-Entity certificate of

the Registrar, or the certificate of a domain CA of the Registrar's domain. However, if the Pledge is known to also support RPK pinning and the MASA intends to pin the Registrar's identity (not a CA), then MASA MAY pin the RPK of the Registrar instead of the Registrar's End-Entity certificate in order to save space in the voucher.

5. Discovery and URI

This section describes the BRSKI extensions to EST-coaps [I-D.ietf-ace-coap-est] to transport the voucher between Registrar, join proxy and Pledge over CoAP. The extensions are targeted to low-resource networks with small packets. Saving header space is important and the EST-coaps URI is shorter than the EST URI.

The presence and location of (path to) the management data are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"ace.est"` [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. It is up to the implementation to choose its root resource; throughout this document the example root resource `/est` is used.

The EST-coaps server URIs differ from the EST URI by replacing the scheme `https` by `coaps` and by specifying shorter resource path names:

```
coaps://www.example.com/est/short-name
```

Figure 5 in section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the BRSKI extension URI path to the EST-coaps URI path.

BRSKI	EST-coaps
<code>/requestvoucher</code>	<code>/rv</code>
<code>/voucher_status</code>	<code>/vs</code>
<code>/enrollstatus</code>	<code>/es</code>

Table 1: BRSKI path to EST-coaps path

`/requestvoucher`, `/voucher_status` and `/enrollstatus` occur between the Pledge and Registrar (the BRSKI-EST protocol) and also between Registrar and MASA, but, as described in Section 7, this document addresses only the BRSKI-EST portion of the protocol.

When discovering the root path for the EST resources, the server MAY return the full resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. For example, the following more complete response is possible.

6. BRSKI-EST Protocol

The constrained BRSKI-EST protocol described in this section is between the Pledge and the Registrar only. (probably via a join proxy, such as described in [I-D.ietf-anima-constrained-join-proxy]) It extends both the BRSKI and EST-coaps protocols.

6.1. Discovery, URIs and Content Formats

The constrained BRSKI-EST protocol described in this section is between the Pledge and the Registrar only. (probably via a join proxy, such as described in [I-D.ietf-anima-constrained-join-proxy]) It extends both the BRSKI and EST-coaps protocols.

6.2. Discovery, URIs and Content Formats

TBD: content overlaps with Section 5, to be fixed - issue #79

The Pledge MAY perform a discovery operation on the `"/.well-known/core?rt=brski"` resource of the Registrar if it wishes to discover possibly shorter URLs for the functions, or if it has the possibility to use a variety of onboarding protocols or certificate enrollment protocols and it wants to discover which of these protocols are available.

For example, if the Registrar supports a short BRSKI URL (`/b`) and supports the voucher format `"application/voucher-cose+cbor"` (TBD3), and status reporting in both CBOR and JSON formats:

```
REQ: GET /.well-known/core?rt=brski*
```

```
RES: 2.05 Content
Content-Format: 40
```

```
Payload:
```

```
</b>;rt=brski,
</b/rv>;rt=brski.rv;ct=TBD3,
</b/vs>;rt=brski.vs;ct="50 60",
</b/es>;rt=brski.es;ct="50 60"
```

The Registrar is under no obligation to provide shorter URLs, and MAY respond to this query with only the `"/.well-known/brski"` end points defined in [I-D.ietf-anima-bootstrapping-keyinfra] section 5.

Registrars that have implemented shorter URLs MUST also respond in equivalent ways to the `"/.well-known/brski"` URLs, and MUST NOT distinguish between them. In particular, a Pledge MAY use the longer and shorter URLs in combination.

The return of multiple content-types in the `"ct"` attribute allows the Pledge to choose the most appropriate one. Note that Content-Format TBD3 is defined in this document.

The Content-Format (`"application/json"`) 50 MAY be supported and 60 MUST be supported by the Registrar for the `/vs` and `/es` resources. Content-Format TBD3 MUST be supported by the Registrar for the `/rv` resource. If the `"ct"` attribute is not indicated for this resource, this implies that at least TBD3 is supported.

The Pledge and MASA need to support one or more formats (at least TBD3) for the voucher and for the voucher request. The MASA needs to support all formats that the Pledge, produced by that manufacturer, supports.

6.3. Extensions to BRSKI

A Pledge that only supports the EST-coaps enrollment method SHOULD NOT use discovery for BRSKI resources, since it is more efficient to just try the supported enrollment method via the well-known BRSKI/EST-coaps resources, and it avoids the Pledge having to do complex CoRE Link Format parsing. A Registrar SHOULD host any discoverable BRSKI resources on the same (UDP) server port that the Pledge's DTLS connection is using. This avoids the Pledge having to reconnect using DTLS, in order to access these resources.

6.4. Extensions to EST-coaps

A Pledge that only supports the EST-coaps enrollment method SHOULD NOT use discovery for EST-coaps resources, for similar reasons as stated in the previous section. A Registrar SHOULD host any discoverable EST-coaps resources on the same (UDP) server port that the Pledge's DTLS connection is using. This avoids the Pledge having to reconnect using DTLS, in order to access these resources.

6.4.1. Pledge Extensions

A constrained Pledge SHOULD NOT perform the optional `"CSR attributes request"` (`/att`) to minimize network traffic and reduce code size (i.e. by not implementing the complex CSR attributes parsing code).

When creating the CSR, the Pledge selects itself which attributes to include. One or more Subject Distinguished Name fields MUST be

included. If the Pledge has no specific information on what attributes/fields are desired in the CSR, it MUST use the Subject Distinguished Name fields from its LDevID unmodified. The Pledge may receive such information via the voucher (encoded in a vendor-specific way) or some other, out-of-band means.

A constrained Pledge MAY use the following optimized EST-coaps procedure to minimize both network traffic and code size:

1. if the BRSKI-received voucher, validating the current EST server, contains a pinned domain CA certificate, the Pledge provisionally considers this single certificate as the sole EST trust anchor, in other words, the single result of "CA certificates request" (/crts) to the EST server.
2. Using this trust anchor it proceeds with EST simple enrollment (/sen) to obtain its provisionally trusted LDevID.
3. Then, the Pledge attempts to validate that the trust anchor CA is the signer of the LDevID. If this is the case, the Pledge finally accepts the pinned domain CA certificate as the legitimate trust anchor CA for its domain and it also accepts its LDevID.
4. If this is not the case, the Pledge MUST perform an actual "CA certificates request" (/crts) to the EST server to obtain the EST CA trust anchors since these obviously differ from the (temporary) pinned domain CA.
5. When doing this request, the Pledge MAY use a CoAP Accept Option with value TBD287 ("application/pkix-cert") to limit the number of returned EST CA trust anchors to only one. Such limiting to only one has the advantages that storage requirements for CA certificates are reduced, network traffic can be reduced, and code size can be reduced (by not having to parse the alternative format 281 "application/pkcs7-mime;smime-type=certs-only" and not having to support CoAP block-wise transfer).
6. If the Pledge cannot obtain the single CA certificate or the finally validated CA certificate cannot be chained to the LDevID, then the Pledge MUST abort the enrollment process and report the error using the enrollment status telemetry (/es).

The Content-Format ("application/json") 50 MAY be supported and 60 MUST be supported by the Registrar for the /vs and /es resources. Content-Format TBD3 MUST be supported by the Registrar for the /rv resource. If the "ct" attribute is not indicated for this resource, this implies that at least TBD3 is supported.

When a Registrar receives a "CA certificates request" (/crt) request with a CoAP Accept Option with value TBD287 it SHOULD return only the single CA certificate that is the envisioned or actual authority for the current, authenticated Pledge making the request. The only exception case is when the Registrar is configured to not support a request for a single CA certificate for operational or security reasons, e.g. because every device enrolled into the domain needs to use at least multiple CAs. In such exception case the Registrar returns the CoAP response 4.06 Not Acceptable to indicate that only the default Content-Format of 281 "application/pkcs7-mime;smime-type=certs-only" is available.

6.4.2. Registrar Extensions

When a Registrar receives a "CA certificates request" (/crt) request with a CoAP Accept Option with value TBD287 it SHOULD return only the single CA certificate that is the envisioned or actual authority for the current, authenticated Pledge making the request. The only exception case is when the Registrar is configured to not support a request for a single CA certificate for operational or security reasons, e.g. because every device enrolled into the domain needs to use at least multiple CAs. In such exception case the Registrar returns the CoAP response 4.06 Not Acceptable to indicate that only the default Content-Format of 281 "application/pkcs7-mime;smime-type=certs-only" is available.

7. BRSKI-MASA Protocol

[I-D.ietf-anima-bootstrapping-keyinfra] section 5.4 describes a connection between the Registrar and the MASA as being a normal TLS connection using HTTPS. This document does not change that. The use of CoAP for the BRSKI-MASA connection is NOT supported.

Some consideration was made to specify CoAP support for consistency but:

- o the Registrar is not expected to be so constrained that it cannot support HTTPS client connections.
- o the technology and experience to build Internet-scale HTTPS responders (which the MASA is) is common, while the experience doing the same for CoAP is much less common.
- o in many Enterprise networks, outgoing UDP connections are often treated as suspicious, and there seems to be no advantage to using CoAP in that environment.

- o a Registrar is likely to provide onboarding services to both constrained and non-constrained devices. Such a Registrar would need to speak HTTPS anyway.
- o similarly, a manufacturer is likely to offer both constrained and non-constrained devices, so there may in practice be no situation in which the MASA could be CoAP-only. Additionally, as the MASA is intended to be a function that can easily be outsourced to a third-party service provider, reducing the complexity would also seem to reduce the cost of that function.

8. Pinning in Voucher Artifacts

The voucher is a statement from the MASA to the Pledge indicating who the Pledge's owner is. This section deals with the question of how that owner's identity is determined and how it is encoded within the voucher.

8.1. Registrar Identity Selection and Encoding

Section 5.5 of [I-D.ietf-anima-bootstrapping-keyinfra] describes BRSKI policies for selection of the owner identity. It indicates some of the flexibility that is possible for the Registrar. The recommendation made there is for the Registrar to include only certificates in the (CMS) signing structure which participate in the certificate chain that is to be pinned.

The MASA is expected to evaluate the certificates included by the Registrar in its voucher request, forming them into a chain with the Registrar's (signing) identity on one end. Then, it pins a certificate selected from the chain. For instance, for a domain with a two-level certification authority, where the voucher-request has been signed by "Registrar" its signing structure includes two additional CA certificates:

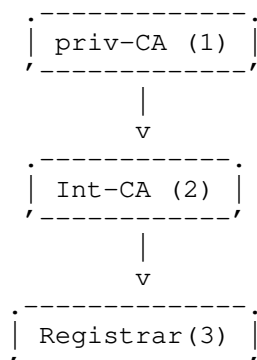


Figure 1: Two Level PKI

When the Registrar is using a COSE-signed constrained format voucher request towards MASA, instead of a regular CMS-signed voucher request, the COSE_Sign1 object contains a protected and an unprotected header, and according to [I-D.ietf-cose-x509], would carry all the certificates of the chain in an "x5bag" attribute placed in the unprotected header.

8.2. MASA Pinning Policy

The MASA, having assembled and verified the chain in the signing structure, will now need to select a certificate to pin in the voucher in case there are multiple available. (For the case that only the Registrar's End-Entity certificate is included, only this certificate can be selected and this section does not apply.) The BRSKI policy for pinning by the MASA as described in Section 5.5.2 of [I-D.ietf-anima-bootstrapping-keyinfra] leaves much flexibility to the manufacturer. The present document adds the following rules to the MASA pinning policy, in order to reduce on average the duration of BRSKI/EST on constrained, low-bandwidth networks:

1. for a voucher containing a nonce, it SHOULD select the most specific (lowest-level) CA certificate in the chain.
2. for a nonceless voucher, it SHOULD select the least-specific (highest-level) CA certificate in the chain that is allowed under the MASA's policy for this specific customer (domain).

The rationale for 1. is that in case of a voucher with nonce, the voucher is valid only in scope of the present DTLS connection between Pledge and Registrar anyway, so it would have no benefit to pin a higher-level CA. By pinning the most specific CA the constrained Pledge can validate its DTLS connection using less crypto operations.

The rationale for pinning a CA instead of the Registrar's End-Entity certificate directly is the following benefit on constrained networks: the pinned certificate in the voucher can in common cases be re-used as a Domain CA trust anchor during the EST enrollment and during the operational phase that follows after EST enrollment, as explained elsewhere in this document. Doing so avoids an additional transmission of this trust anchor over the network during the EST enrollment, saving potentially 100s of bytes and a CoAP transaction.

The rationale for 2. follows from the flexible BRSKI trust model for, and purpose of, nonceless vouchers (Sections 5.5.* and 7.4.1 of [I-D.ietf-anima-bootstrapping-keyinfra]).

Using the previous example of a domain with a two-level certification authority, the most specific CA ("Sub-CA") is the identity that is pinned by MASA in a nonced voucher. A Registrar that wished to have only the Registrar's End-Entity certificate pinned would omit the "priv-CA" and "Sub-CA" certificates from the voucher-request.

In case of a nonceless voucher, the MASA would depending on trust level pin only "Registrar" certificate (low trust in customer), or the "Sub-CA" certificate (in case of medium trust, implying that any Registrar of that sub-domain is acceptable), or even the "priv-CA" certificate (in case of high trust in the customer, and possibly a pre-agreed need of the customer to obtain flexible long-lived vouchers).

8.3. Pinning of Raw Public Keys

Specifically for constrained use cases, the pinning of the raw public key (RPK) of the Registrar is also supported in the constrained voucher, instead of an X.509 certificate. If an RPK is pinned it MUST be the RPK of the Registrar.

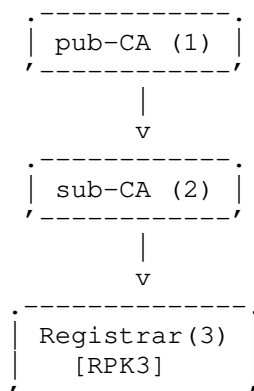


Figure 2: Raw Public Key pinning

When the Pledge is known by MASA to support RPK but not X.509 certificates, the voucher produced by the MASA pins the RPK of the Registrar in the "pinned-domain-subject-public-key-info" field of a voucher. This is described in more detail in the YANG definition for the constrained voucher. A Pledge that does not support X.509 certificates cannot use EST to enroll; it has to use another method for certificate-less enrollment and the Registrar has to support this method also. It is possible that the Pledge will not enroll, but instead only a network join operation will occur, such as described in [I-D.ietf-6tisch-minimal-security]. How the Pledge discovers this method and details of the enrollment method are out of scope of this document.

When the Pledge is known by MASA to support PKIX format certificates, the "pinned-domain-cert" field present in a voucher typically pins a domain certificate. That can be either the End-Entity certificate of the Registrar, or the certificate of a domain CA of the Registrar's domain. However, if the Pledge is known to also support RPK pinning and the MASA intends to pin the Registrar's identity (not a CA), then MASA SHOULD pin the RPK (RPK3 in figure Figure 2) of the Registrar instead of the Registrar's End-Entity certificate in order to save space in the voucher.

To Be Completed further (TBD): Note, the above paragraphs are duplicated from the section Section 4 so we may have to resolve this duplication.

9. Artifacts

This section describes the abstract (tree) definition as explained in [I-D.ietf-netmod-yang-tree-diagrams] first. This provides a high-level view of the contents of each artifact.

Then the assigned SID values are presented. These have been assigned using the rules in [I-D.ietf-core-sid], with an allocation that was made via the <http://comi.space> service.

9.1. Voucher Request artifact

9.1.1. Tree Diagram

The following diagram is largely a duplicate of the contents of [RFC8366], with the addition of proximity-registrar-subject-public-key-info, proximity-registrar-cert, and prior-signed-voucher-request.

prior-signed-voucher-request is only used between the Registrar and the MASA. proximity-registrar-subject-public-key-info replaces proximity-registrar-cert for the extremely constrained cases.

```
module: ietf-constrained-voucher-request
```

```
  grouping voucher-request-constrained-grouping
```

```
    +-- voucher
```

```
      +-- created-on?
```

```
        | yang:date-and-time
```

```
      +-- expires-on?
```

```
        | yang:date-and-time
```

```
      +-- assertion
```

```
        | enumeration
```

```
      +-- serial-number
```

```
        | string
```

```
      +-- idevid-issuer?
```

```
        | binary
```

```
      +-- pinned-domain-cert?
```

```
        | binary
```

```
      +-- domain-cert-revocation-checks?
```

```
        | boolean
```

```
      +-- nonce?
```

```
        | binary
```

```
      +-- last-renewal-date?
```

```
        | yang:date-and-time
```

```
      +-- proximity-registrar-subject-public-key-info?
```

```
        | binary
```

```
      +-- proximity-registrar-sha256-of-subject-public-key-info?
```

```
        | binary
```

```
      +-- proximity-registrar-cert?
```

```
        | binary
```

```
      +-- prior-signed-voucher-request?
```

```
        | binary
```

9.1.2. SID values

SID Assigned to

```
-----
2501 data /ietf-constrained-voucher-request:voucher
2502 data .../assertion
2503 data .../created-on
2504 data .../domain-cert-revocation-checks
2505 data .../expires-on
2506 data .../idevid-issuer
2507 data .../last-renewal-date
2508 data /ietf-constrained-voucher-request:voucher/nonce
2509 data .../pinned-domain-cert
2510 data .../prior-signed-voucher-request
2511 data .../proximity-registrar-cert
2512 data mity-registrar-sha256-of-subject-public-key-info
2513 data .../proximity-registrar-subject-public-key-info
2514 data .../serial-number
```

WARNING, obsolete definitions

9.1.3. YANG Module

In the constrained-voucher-request YANG module, the voucher is "augmented" within the "used" grouping statement such that one continuous set of SID values is generated for the constrained-voucher-request module name, all voucher attributes, and the constrained-voucher-request attribute. Two attributes of the voucher are "refined" to be optional.

```
<CODE BEGINS> file "ietf-constrained-voucher-request@2019-09-01.yang"
module ietf-constrained-voucher-request {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-voucher-request";
  prefix "constrained";

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  import ietf-voucher {
    prefix "v";
  }
}
```

organization

"IETF ANIMA Working Group";

contact

"WG Web: <<http://tools.ietf.org/wg/anima/>>

WG List: <<mailto:anima@ietf.org>>

Author: Michael Richardson
<<mailto:mcr+ietf@sandelman.ca>>

Author: Peter van der Stok
<<mailto:consultancy@vanderstok.org>>

Author: Panos Kampanakis
<<mailto:pkampana@cisco.com>>;

description

"This module defines the format for a voucher request, which is produced by a pledge to request a voucher. The voucher-request is sent to the potential owner's Registrar, which in turn sends the voucher request to the manufacturer or delegate (MASA).

A voucher is then returned to the pledge, binding the pledge to the owner. This is a constrained version of the voucher-request present in draft-ietf-anima-bootstrap-keyinfra.txt.

This version provides a very restricted subset appropriate for very constrained devices.

In particular, it assumes that nonce-ful operation is always required, that expiration dates are rather weak, as no clocks can be assumed, and that the Registrar is identified by a pinned Raw Public Key.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119.";

revision "2019-09-01" {

description

"Initial version";

reference

"RFC XXXX: Voucher Profile for Constrained Devices";

}

rc:yang-data voucher-request-constrained-artifact {

// YANG data template for a voucher.

uses voucher-request-constrained-grouping;

}

```
// Grouping defined for future usage
grouping voucher-request-constrained-grouping {
  description
    "Grouping to allow reuse/extensions in future work.";

  uses v:voucher-artifact-grouping {

    refine voucher/created-on {
      mandatory false;
    }

    refine voucher/pinned-domain-cert {
      mandatory false;
    }

  }

  augment "voucher" {
    description "Base the constrained voucher-request upon the
      regular one";

    leaf proximity-registrar-subject-public-key-info {
      type binary;
      description
        "The proximity-registrar-subject-public-key-info replaces
        the proximit-registrar-cert in constrained uses of
        the voucher-request.
        The proximity-registrar-subject-public-key-info is the
        Raw Public Key of the Registrar. This field is encoded
        as specified in RFC7250, section 3.
        The ECDSA algorithm MUST be supported.
        The EdDSA algorithm as specified in
        draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
        Support for the DSA algorithm is not recommended.
        Support for the RSA algorithm is MAY, but due to
        size is discouraged.";
    }

    leaf proximity-registrar-sha256-of-subject-public-key-info {
      type binary;
      description
        "The proximity-registrar-sha256-of-subject-public-key-info
        is an alternative to
        proximity-registrar-subject-public-key-info.
        and pinned-domain-cert. In many cases the
        public key of the domain has already been transmitted
        during the key agreement protocol, and it is wasteful
        to transmit the public key another two times.
        The use of a hash of public key info, at 32-bytes for
```

```
    sha256 is a significant savings compared to an RSA
    public key, but is only a minor savings compared to
    a 256-bit ECDSA public-key.
    Algorithm agility is provided by extensions to this
    specifications which define new leaf for other hash
    types.";
}

leaf proximity-registrar-cert {
  type binary;
  description
    "An X.509 v3 certificate structure as specified by
    RFC 5280,
    Section 4 encoded using the ASN.1 distinguished encoding
    rules (DER), as specified in ITU-T X.690.

    The first certificate in the Registrar TLS server
    certificate_list sequence (see [RFC5246]) presented by
    the Registrar to the Pledge. This MUST be populated in a
    Pledge's voucher request if the proximity assertion is
    populated.";
}

leaf prior-signed-voucher-request {
  type binary;
  description
    "If it is necessary to change a voucher, or re-sign and
    forward a voucher that was previously provided along a
    protocol path, then the previously signed voucher
    SHOULD be included in this field.

    For example, a pledge might sign a proximity voucher,
    which an intermediate registrar then re-signs to
    make its own proximity assertion. This is a simple
    mechanism for a chain of trusted parties to change a
    voucher, while maintaining the prior signature
    information.

    The pledge MUST ignore all prior voucher information
    when accepting a voucher for imprinting. Other
    parties MAY examine the prior signed voucher
    information for the purposes of policy decisions.
    For example this information could be useful to a
    MASA to determine that both pledge and registrar
    agree on proximity assertions. The MASA SHOULD
    remove all prior-signed-voucher-request information when
    signing a voucher for imprinting so as to minimize the
    final voucher size.";
```

```

    }
  }
}
}
<CODE ENDS>

```

9.1.4. Example voucher request artifact

Below is a CBOR serialization of an example constrained voucher request from a Pledge to a Registrar, shown in CBOR diagnostic notation. The enum value of the assertion field is calculated to be 2 by following the algorithm described in section 9.6.4.2 of [RFC7950]. Four dots ("....") in a CBOR byte string denotes a sequence of bytes that are not shown for brevity.

```

{
  2501: {
    +2 : "2016-10-07T19:31:42Z", / SID= 2503, created-on /
    +4 : "2016-10-21T19:31:42Z", / SID= 2505, expires-on /
    +1 : 2,                      / SID= 2502, assertion /
                                / "proximity" /
    +13: "JADA123456789",        / SID= 2514, serial-number /
    +5 : h'01020D0F',           / SID= 2506, idevid-issuer /
    +10: h'cert.der',           / SID=2511, proximity-registrar-cert/
    +3 : true,                  / SID= 2504, domain-cert
                                -revocation-checks/
    +6 : "2017-10-07T19:31:42Z", / SID= 2507, last-renewal-date /
    +12: h'key_info'            / SID= 2513, proximity-registrar
                                -subject-public-key-info /
  }
}

```

```

<CODE ENDS>

```

9.2. Voucher artifact

The voucher's primary purpose is to securely assign a Pledge to an owner. The voucher informs the Pledge which entity it should consider to be its owner.

9.2.1. Tree Diagram

The following diagram is largely a duplicate of the contents of [RFC8366], with only the addition of pinned-domain-subject-public-key-info.

```

module: ietf-constrained-voucher

  grouping voucher-constrained-grouping
    +-- voucher
      +-- created-on?
      |   yang:date-and-time
      +-- expires-on?
      |   yang:date-and-time
      +-- assertion
      |   enumeration
      +-- serial-number
      |   string
      +-- idevid-issuer?
      |   binary
      +-- pinned-domain-cert?
      |   binary
      +-- domain-cert-revocation-checks?
      |   boolean
      +-- nonce?
      |   binary
      +-- last-renewal-date?
      |   yang:date-and-time
      +-- pinned-domain-subject-public-key-info?
      |   binary
      +-- pinned-sha256-of-subject-public-key-info?
      |   binary
<CODE ENDS>

```

9.2.2. SID values

```

      SID Assigned to
      -----
2451 data /ietf-constrained-voucher:voucher
2452 data /ietf-constrained-voucher:voucher/assertion
2453 data /ietf-constrained-voucher:voucher/created-on
2454 data .../domain-cert-revocation-checks
2455 data /ietf-constrained-voucher:voucher/expires-on
2456 data /ietf-constrained-voucher:voucher/idevid-issuer
2457 data .../last-renewal-date
2458 data /ietf-constrained-voucher:voucher/nonce
2459 data .../pinned-domain-cert
2460 data .../pinned-domain-subject-public-key-info
2461 data .../pinned-sha256-of-subject-public-key-info
2462 data /ietf-constrained-voucher:voucher/serial-number

```

```

WARNING, obsolete definitions
<CODE ENDS>

```

9.2.3. YANG Module

In the constrained-voucher YANG module, the voucher is "augmented" within the "used" grouping statement such that one continuous set of SID values is generated for the constrained-voucher module name, all voucher attributes, and the constrained-voucher attribute. Two attributes of the voucher are "refined" to be optional.


```
<CODE BEGINS> file "ietf-constrained-voucher@2019-09-01.yang"
module iETF-constrained-voucher {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-voucher";
  prefix "constrained";

  import iETF-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  import iETF-voucher {
    prefix "v";
  }

  organization
    "IETF ANIMA Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/anima/>
    WG List:  <mailto:anima@ietf.org>
    Author:   Michael Richardson
              <mailto:mcr+ietf@sandelman.ca>
    Author:   Peter van der Stok
              <mailto:consultancy@vanderstok.org>
    Author:   Panos Kampanakis
              <mailto:pkampana@cisco.com>";

  description
    "This module defines the format for a voucher, which is produced
    by a pledge's manufacturer or delegate (MASA) to securely assign
    one or more pledges to an 'owner', so that the pledges may
    establish a secure connection to the owner's network
    infrastructure.

    This version provides a very restricted subset appropriate
    for very constrained devices.
    In particular, it assumes that nonce-ful operation is
    always required, that expiration dates are rather weak, as no
    clocks can be assumed, and that the Registrar is identified
    by a pinned Raw Public Key.

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
    'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY',
```

and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119.";

```
revision "2019-09-01" {
  description
    "Initial version";
  reference
    "RFC XXXX: Voucher Profile for Constrained Devices";
}

rc:yang-data voucher-constrained-artifact {
  // YANG data template for a voucher.
  uses voucher-constrained-grouping;
}

// Grouping defined for future usage
grouping voucher-constrained-grouping {
  description
    "Grouping to allow reuse/extensions in future work.";

  uses v:voucher-artifact-grouping {

    refine voucher/created-on {
      mandatory false;
    }

    refine voucher/pinned-domain-cert {
      mandatory false;
    }

    augment "voucher" {
      description "Base the constrained voucher
                  upon the regular one";
      leaf pinned-domain-subject-public-key-info {
        type binary;
        description
          "The pinned-domain-subject-public-key-info replaces the
           pinned-domain-cert in constrained uses of
           the voucher. The pinned-domain-subject-public-key-info
           is the Raw Public Key of the Registrar.
           This field is encoded as specified in RFC7250,
           section 3.
           The ECDSA algorithm MUST be supported.
           The EdDSA algorithm as specified in
           draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
           Support for the DSA algorithm is not recommended.
           Support for the RSA algorithm is a MAY.";
      }
    }
  }
}
```

```

leaf pinned-sha256-of-subject-public-key-info {
  type binary;
  description
    "The pinned-hash-subject-public-key-info is a second
    alternative to pinned-domain-cert. In many cases the
    public key of the domain has already been transmitted
    during the key agreement process, and it is wasteful
    to transmit the public key another two times.
    The use of a hash of public key info, at 32-bytes for
    sha256 is a significant savings compared to an RSA
    public key, but is only a minor savings compared to
    a 256-bit ECDSA public-key.
    Algorithm agility is provided by extensions to this
    specifications which define new leaf for other hash types";
}
}
}
}
}
<CODE ENDS>

```

9.2.4. Example voucher artifacts

Below the CBOR serialization of an example constrained voucher is shown in CBOR diagnostic notation. The enum value of the assertion field is calculated to be zero by following the algorithm described in section 9.6.4.2 of [RFC7950]. Four dots ("....") in a CBOR byte string denotes a sequence of bytes that are not shown for brevity.

```

{
  2451: {
    +2 : "2016-10-07T19:31:42Z", / SID = 2453, created-on /
    +4 : "2016-10-21T19:31:42Z", / SID = 2455, expires-on /
    +1 : 0, / SID = 2452, assertion "verified" /
    +11: "JADA123456789", / SID = 2462, serial-number /
    +5 : h'E40393B4....68A3', / SID = 2456, idevid-issuer /
    +8 : h'30820275....C35F', / SID = 2459, pinned-domain-cert/
    +3 : true, / SID = 2454, domain-cert /
    / -revocation-checks /
    +6 : "2017-10-07T19:31:42Z" / SID = 2457, last-renewal-date /
  }
}

```

<CODE ENDS>

9.3. Signing voucher and voucher-request artifacts with COSE

The COSE_Sign1 structure is discussed in section 4.2 of [I-D.ietf-cose-rfc8152bis-struct]. The CBOR object that carries the body, the signature, and the information about the body and signature is called the COSE_Sign1 structure. It is used when only one signature is used on the body. Support for ECDSA with sha256 (secp256k1 and prime256v1 curves) is compulsory.

The supported COSE-sign1 object structure is shown in Figure 3. Support for EdDSA is encouraged. [EDNOTE: Expand and add a reference why.]

```
COSE_Sign1(  
  [  
    h'A101382E',          # { "alg": EC256K1 }  
    {  
      "kid" : h'789' # hash256(public key)  
    },  
    h'123', #voucher-request binary content  
    h'456', #voucher-request binary public signature  
  ]  
)
```

Figure 3: cose-sign1 example

The [COSE-registry] specifies the integers that replace the strings and the mnemonics in Figure 3. The value of the "kid" parameter is an example value. Usually a hash of the public key is used to identify the public key. The public key and its hash are derived from the relevant certificate (Pledge, Registrar or MASA certificate).

In Appendix B a binary cose-sign1 object is shown based on the voucher-request example of Section 9.1.4.

10. Design Considerations

The design considerations for the CBOR encoding of vouchers is much the same as for [RFC8366].

One key difference is that the names of the leaves in the YANG does not have a material effect on the size of the resulting CBOR, as the SID translation process assigns integers to the names.

Any POST request to the Registrar with resource /est/vs or /est/es returns a 2.05 response with empty payload. The client should be aware that the server may use a piggybacked CoAP response (ACK, 2.05)

but may also respond with a separate CoAP response, i.e. first an (ACK, 0.0) that is an acknowledgement of the request reception followed by a (CON, 2.05) response in a separate CoAP message.

11. Security Considerations

11.1. Clock Sensitivity

TBD.

11.2. Protect Voucher PKI in HSM

TBD.

11.3. Test Domain Certificate Validity when Signing

TBD.

12. IANA Considerations

12.1. Resource Type Registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE parameters" registry are specified below. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

ace.rt.rv needs registration with IANA
ace.rt.vs needs registration with IANA
ace.rt.es needs registration with IANA
ace.rt.ra needs registration with IANA

12.2. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-constrained-voucher
Registrant Contact: The ANIMA WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-constrained-voucher-request
Registrant Contact: The ANIMA WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

12.3. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registration is requested:

```

name:      ietf-constrained-voucher
namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-voucher
prefix:    vch
reference:  RFC XXXX

name:      ietf-constrained-voucher-request
namespace: urn:ietf:params:xml:ns:yang:ietf-constrained
              -voucher-request
prefix:    vch
reference:  RFC XXXX

```

12.4. The RFC SID range assignment sub-registry

Entry-point	Size	Module name	RFC Number
2450	50	ietf-constrained-voucher	[ThisRFC]
2500	50	ietf-constrained-voucher -request	[ThisRFC]

Warning: These SID values are defined in [I-D.ietf-core-sid], not as an Early Allocation.

12.5. Media-Type Registry

This section registers the the 'application/voucher-cose+cbor' in the "Media Types" registry. These media types are used to indicate that the content is a CBOR voucher either signed with a COSE_Sign1 structure [I-D.ietf-cose-rfc8152bis-struct].

12.5.1. application/voucher-cose+cbor

Type name: application
 Subtype name: voucher-cose+cbor
 Required parameters: none
 Optional parameters: cose-type
 Encoding considerations: COSE_Sign1 CBOR vouchers are COSE objects signed with one signer.
 Security considerations: See Security Considerations, Section
 Interoperability considerations: The format is designed to be broadly interoperable.
 Published specification: THIS RFC.
 Applications that use this media type: ANIMA, 6tisch, and other zero-touch imprinting systems
 Additional information:
 Magic number(s): None
 File extension(s): .vch
 Macintosh file type code(s): none
 Person & email address to contact for further information: IETF ANIMA WG
 Intended usage: LIMITED
 Restrictions on usage: NONE
 Author: ANIMA WG
 Change controller: IETF
 Provisional registration? (standards tree only): NO

12.6. CoAP Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for two media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

Media type	mime type	Encoding	ID	References
application/voucher-cose+cbor	"COSE-Sign1"	CBOR	TBD3	[This RFC]

13. Acknowledgements

We are very grateful to Jim Schaad for explaining COSE and CMS choices. Also thanks to Jim Schaad for correcting earlier version of the COSE Sign1 objects.

Michel Veillette did extensive work on pyang to extend it to support the SID allocation process, and this document was among the first users.

14. Changelog

- 10 Design considerations extended Examples made consistent
- 08 Examples for cose_sign1 are completed and improved.
- 06 New SID values assigned; regenerated examples
- 04 voucher and request-voucher MUST be signed examples for signed request are added in appendix IANA SID registration is updated SID values in examples are aligned signed cms examples aligned with new SIDs
- 03
Examples are inverted.
- 02
Example of requestvoucher with unsigned application/cbor is added
attributes of voucher "refined" to optional
CBOR serialization of vouchers improved
Discovery port numbers are specified
- 01
application/json is optional, application/cbor is compulsory
Cms and cose mediatypes are introduced

15. References

15.1. Normative References

- [I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson,
"Constrained Join Protocol (CoJP) for 6TiSCH", draft-ietf-6tisch-minimal-security-15 (work in progress), December 2019.
- [I-D.ietf-ace-coap-est]
Stok, P. V. D., Kampanakis, P., Richardson, M. C., and S. Raza,
"EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-18 (work in progress), January 2020.

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M. C., Eckert, T., Behringer, M. H., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-45 (work in progress), November 2020.
- [I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (YANG SID)", draft-ietf-core-sid-15 (work in progress), January 2021.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-15 (work in progress), January 2021.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-15 (work in progress), February 2021.
- [I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", draft-ietf-cose-x509-08 (work in progress), December 2020.
- [I-D.selander-ace-ake-authz]
Selander, G., Mattsson, J. P., Vucinic, M., Richardson, M., and A. Schellenbaum, "Lightweight Authorization for Authenticated Key Exchange.", draft-selander-ace-ake-authz-02 (work in progress), November 2020.
- [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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.

15.2. Informative References

- [COSE-registry] IANA, ., "CBOR Object Signing and Encryption (COSE) registry", 2017, <<https://www.iana.org/assignments/cose/cose.xhtml>>.
- [I-D.ietf-anima-constrained-join-proxy] Richardson, M., Stok, P. V. D., and P. Kampanakis, "Constrained Join Proxy for Bootstrapping Protocols", draft-ietf-anima-constrained-join-proxy-02 (work in progress), February 2021.
- [I-D.ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

Appendix A. EST messages to EST-coaps

This section extends the examples from Appendix A of [I-D.ietf-ace-coap-est]. The CoAP headers are only worked out for the enrollstatus example.

A.1. enrollstatus

A coaps enrollstatus message can be :

```
POST coaps://192.0.2.1:8085/est/es
```

The corresponding coap header fields are shown below.

```
Ver = 1
T = 0 (CON)
Code = 0x02 (0.02 is POST)
Options
  Option (Uri-Path)
    Option Delta = 0xb    (option nr = 11)
    Option Length = 0x3
    Option Value = "est"
  Option (Uri-Path)
    Option Delta = 0x0    (option nr = 11)
    Option Length = 0x2
    Option Value = "es"
Payload = [Empty]
```

The Uri-Host and Uri-Port Options are omitted because they coincide with the transport protocol destination address and port respectively.

A 2.05 Content response with an unsigned voucher status (ct=60) will then be:

```
2.05 Content (Content-Format: application/cbor)
```

With CoAP fields and payload:

```

Ver=1
T=2 (ACK)
Code = 0x45 (2.05 Content)
Options
  Option1 (Content-Format)
  Option Delta = 0xC (option nr 12)
  Option Length = 0x2
  Option Value = 60 (application/cbor)

Payload (CBOR diagnostic) =
{
  "version":"1",
  "Status": 1, / 1 = Success, 0 = Fail /
  "Reason":"Informative human readable message",
  "reason-context": "Additional information"
}

```

The binary payload is:

```

A46776657273696F6E6131665374617475730166526561736F6E7822
496E666F726D61746976652068756D616E207265616461626C65206D
6573736167656E726561736F6E2D636F6E74657874
764164646974696F6E616C20696E666F726D6174696F6E

```

The binary payload disassembles to the above CBOR diagnostic code.

A.2. voucher_status

A coaps voucher_status message can be:

```
POST coaps://[2001:db8::2:1]:61616/est/vs
```

A 2.05 Content response with a non signed CBOR voucher status (ct=60) will then be:

```

2.05 Content (Content-Format: application/cbor)
Payload =
a46776657273696f6e6131665374617475730166526561736f6e7822496e666f7
26d61746976652068756d616e207265616461626c65206d6573736167656e7265
61736f6e2d636f6e74657874764164646974696f6e616c20696e666f726d61746
96f6e<CODE ENDS>

```

The payload above is represented in hexadecimal.

```

{"version": "1", "Status": 1, "Reason": "Informative human
readable message", "reason-context": "Additional information"}<CODE ENDS>

```

Appendix B. COSE examples

These examples are generated on a pie 4 and a PC running BASH. Keys and Certificates have been generated with openssl with the following shell script:

```
#!/bin/bash
#try-cert.sh
export dir=./brski/intermediate
export cadir=./brski
export cnfdir=./conf
export format=pem
export default_crl_days=30
sn=8

DevID=pledge.1.2.3.4
serialNumber="serialNumber=$DevID"
export hwType=1.3.6.1.4.1.6715.10.1
export hwSerialNum=01020304 # Some hex
export subjectAltName="otherName:1.3.6.1.5.5.7.8.4;SEQ:hmodname"
echo $hwType - $hwSerialNum
echo $serialNumber
OPENSSL_BIN="openssl"

# remove all files
rm -r ./brski/*
#
# initialize file structure
# root level
cd $cadir
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
touch serial
echo 11223344556600 >serial
echo 1000 > crlnumber
# intermediate level
mkdir intermediate
cd intermediate
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 11223344556600 >serial
echo 1000 > crlnumber
cd ../../
```

```
# file structure is cleaned start filling

echo "#####"
echo "create registrar keys and certificates "
echo "#####"

echo "create root registrar certificate using ecdsa with sha 256 key"
$OPENSSL_BIN ecparam -name prime256v1 -genkey \
    -noout -out $cadir/private/ca-regis.key

$OPENSSL_BIN req -new -x509 \
    -config $cnfdir/openssl-regis.cnf \
    -key $cadir/private/ca-regis.key \
    -out $cadir/certs/ca-regis.crt \
    -extensions v3_ca \
    -days 365 \
    -subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=consultancy \
    /CN=registrar.stok.nl"

# Combine authority certificate and key
echo "Combine authority certificate and key"
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet \
    -inkey $cadir/private/ca-regis.key \
    -in $cadir/certs/ca-regis.crt -export \
    -out $cadir/certs/ca-regis-comb.pfx

# converteer authority pkcs12 file to pem
echo "converteer authority pkcs12 file to pem"
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet \
    -in $cadir/certs/ca-regis-comb.pfx \
    -out $cadir/certs/ca-regis-comb.crt -nodes

#show certificate in registrar combined certificate
$OPENSSL_BIN x509 -in $cadir/certs/ca-regis-comb.crt -text

#
# Certificate Authority for MASA
#
echo "#####"
echo "create MASA keys and certificates "
echo "#####"

echo "create root MASA certificate using ecdsa with sha 256 key"
$OPENSSL_BIN ecparam -name prime256v1 -genkey -noout \
    -out $cadir/private/ca-masa.key

$OPENSSL_BIN req -new -x509 \
```

```
-config $cnfdir/openssl-masa.cnf \  
-days 1000 -key $cadir/private/ca-masa.key \  
-out $cadir/certs/ca-masa.crt \  
-extensions v3_ca\  
-subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=manufacturer\  
/CN=masa.stok.nl"  
  
# Combine authority certificate and key  
echo "Combine authority certificate and key for masa"  
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet\  
-inkey $cadir/private/ca-masa.key \  
-in $cadir/certs/ca-masa.crt -export \  
-out $cadir/certs/ca-masa-comb.pfx  
  
# converteer authority pkcs12 file to pem for masa  
echo "converteer authority pkcs12 file to pem for masa"  
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet\  
-in $cadir/certs/ca-masa-comb.pfx \  
-out $cadir/certs/ca-masa-comb.crt -nodes  
  
#show certificate in pledge combined certificate  
$OPENSSL_BIN x509 -in $cadir/certs/ca-masa-comb.crt -text  
  
#  
# Certificate for Pledge derived from MASA certificate  
#  
echo "#####"  
echo "create pledge keys and certificates "  
echo "#####"  
  
# Pledge derived Certificate  
  
echo "create pledge derived certificate using ecdsa with sha 256 key"  
$OPENSSL_BIN ecparam -name prime256v1 -genkey -noout \  
-out $dir/private/pledge.key  
  
echo "create pledge certificate request"  
$OPENSSL_BIN req -nodes -new -sha256 \  
-key $dir/private/pledge.key -out $dir/csr/pledge.csr \  
-subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=manufacturing\  
/CN=uuid:$DevID/$serialNumber"  
  
# Sign pledge derived Certificate  
echo "sign pledge derived certificate "  
$OPENSSL_BIN ca -config $cnfdir/openssl-pledge.cnf \  
-extensions 8021ar_idevid\
```

```
-days 365 -in $dir/csr/pledge.csr \  
-out $dir/certs/pledge.crt  
  
# Add pledge key and pledge certificate to pkcs12 file  
echo "Add derived pledge key and derived pledge \  
certificate to pkcs12 file"  
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet\  
-inkey $dir/private/pledge.key \  
-in $dir/certs/pledge.crt -export \  
-out $dir/certs/pledge-comb.pfx  
  
# converteer pledge pkcs12 file to pem  
echo "converteer pledge pkcs12 file to pem"  
$OPENSSL_BIN pkcs12 -passin pass:watnietweet -passout pass:watnietweet\  
-in $dir/certs/pledge-comb.pfx \  
-out $dir/certs/pledge-comb.crt -nodes  
  
#show certificate in pledge-comb.crt  
$OPENSSL_BIN x509 -in $dir/certs/pledge-comb.crt -text  
  
#show private key in pledge-comb.crt  
$OPENSSL_BIN ecparam -name prime256v1\  
-in $dir/certs/pledge-comb.crt -text  
<CODE ENDS>
```

The xxxx-comb certificates have been generated as required by libcoap for the DTLS connection generation.

B.1. Pledge, Registrar and MASA keys

This first section documents the public and private keys used in the subsequent test vectors below. These keys come from test code and are not used in any production system, and should only be used only to validate implementations.

B.1.1. Pledge private key

Private-Key: (256 bit)

priv:

9b:4d:43:b6:a9:e1:7c:04:93:45:c3:13:d9:b5:f0:
41:a9:6a:9c:45:79:73:b8:62:f1:77:03:3a:fc:c2:
9c:9a

pub:

04:d6:b7:6f:74:88:bd:80:ae:5f:28:41:2c:72:02:
ef:5f:98:b4:81:e1:d9:10:4c:f8:1b:66:d4:3e:5c:
ea:da:73:e6:a8:38:a9:f1:35:11:85:b6:cd:e2:04:
10:be:fe:d5:0b:3b:14:69:2e:e1:b0:6a:bc:55:40:
60:eb:95:5c:54

ASN1 OID: prime256v1

NIST CURVE: P-256

<CODE ENDS>

B.1.2. Registrar private key

Private-Key: (256 bit)

priv:

81:df:bb:50:a3:45:58:06:b5:56:3b:46:de:f3:e9:
e9:00:ae:98:13:9e:2f:36:68:81:fc:d9:65:24:fb:
21:7e

pub:

04:50:7a:c8:49:1a:8c:69:c7:b5:c3:1d:03:09:ed:
35:ba:13:f5:88:4c:e6:2b:88:cf:30:18:15:4f:a0:
59:b0:20:ec:6b:eb:b9:4e:02:b8:93:40:21:89:8d:
a7:89:c7:11:ce:a7:13:39:f5:0e:34:8e:df:0d:92:
3e:d0:2d:c7:b7

ASN1 OID: prime256v1

NIST CURVE: P-256

<CODE ENDS>

B.1.3. MASA private key

Private-Key: (256 bit)

priv:

c6:bb:a5:8f:b6:d3:c4:75:28:d8:d3:d9:46:c3:31:
83:6d:00:0a:9a:38:ce:22:5c:e9:d9:ea:3b:98:32:
ec:31

pub:

04:59:80:94:66:14:94:20:30:3c:66:08:85:55:86:
db:e7:d4:d1:d7:7a:d2:a3:1a:0c:73:6b:01:0d:02:
12:15:d6:1f:f3:6e:c8:d4:84:60:43:3b:21:c5:83:
80:1e:fc:e2:37:85:77:97:94:d4:aa:34:b5:b6:c6:
ed:f3:17:5c:f1

ASN1 OID: prime256v1

NIST CURVE: P-256

<CODE ENDS>

B.2. Pledge, Registrar and MASA certificates

Below the certificates that accompany the keys. The certificate description is followed by the hexadecimal DER of the certificate

B.2.1. Pledge IDevID certificate

Certificate:

Data:

```
Version: 3 (0x2)
Serial Number: 4822678189204992 (0x11223344556600)
Signature Algorithm: ecdsa-with-SHA256
Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=manufacturer,
                                             CN=masa.stok.nl
Validity
  Not Before: Dec  9 10:02:36 2020 GMT
  Not After : Dec 31 23:59:59 9999 GMT
Subject: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=manufacturing,
        CN=uuid:pledge.1.2.3.4/serialNumber=pledge.1.2.3.4
Subject Public Key Info:
  Public Key Algorithm: id-ecPublicKey
  Public-Key: (256 bit)
  pub:
    04:d6:b7:6f:74:88:bd:80:ae:5f:28:41:2c:72:02:
    ef:5f:98:b4:81:e1:d9:10:4c:f8:1b:66:d4:3e:5c:
    ea:da:73:e6:a8:38:a9:f1:35:11:85:b6:cd:e2:04:
    10:be:fe:d5:0b:3b:14:69:2e:e1:b0:6a:bc:55:40:
    60:eb:95:5c:54
  ASN1 OID: prime256v1
  NIST CURVE: P-256
X509v3 extensions:
  X509v3 Basic Constraints:
    CA:FALSE
  X509v3 Authority Key Identifier:
    keyid:
E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3
```

```
Signature Algorithm: ecdsa-with-SHA256
30:46:02:21:00:d2:e6:45:3b:b0:c3:00:b3:25:8d:f1:83:fe:
d9:37:c1:a2:49:65:69:7f:6b:b9:ef:2c:05:07:06:31:ac:17:
bd:02:21:00:e2:ce:9e:7b:7f:74:50:33:ad:9e:ff:12:4e:e9:
a6:f3:b8:36:65:ab:7d:80:bb:56:88:bc:03:1d:e5:1e:31:6f
```

<CODE ENDS>

This is the hexadecimal representation in (request-)voucher examples referred to as pledge-cert-hex.

30820226308201cba003020102020711223344556600300a06082a8648ce3d04
0302306f310b3009060355040613024e4c310b300906035504080c024e423110
300e06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e6465
7273746f6b31153013060355040b0c0c6d616e75666163747572657231153013
06035504030c0c6d6173612e73746f6b2e6e6c3020170d323031323039313030
3233365a180f39393939313233313233353935395a308190310b300906035504
0613024e4c310b300906035504080c024e423110300e06035504070c0748656c
6d6f6e6431133011060355040a0c0a76616e64657273746f6b31163014060355
040b0c0d6d616e75666163747572696e67311c301a06035504030c1375756964
3a706c656467652e312e322e332e34311730150603550405130e706c65646765
2e312e322e332e343059301306072a8648ce3d020106082a8648ce3d03010703
420004d6b76f7488bd80ae5f28412c7202ef5f98b481e1d9104cf81b66d43e5c
eada73e6a838a9f1351185b6cde20410befed50b3b14692ee1b06abc554060eb
955c54a32e302c30090603551d1304023000301f0603551d23041830168014e4
0393b4c3d3f42a80a47718f6964903011768a3300a06082a8648ce3d04030203
49003046022100d2e6453bb0c300b3258df183fed937c1a24965697f6bb9ef2c
05070631ac17bd022100e2ce9e7b7f745033ad9eff124ee9a6f3b83665ab7d80
bb5688bc031de51e316f<CODE ENDS>

B.2.2. Registrar Certificate

Certificate:

Data:

```
Version: 3 (0x2)
Serial Number:
  70:56:ea:aa:30:66:d8:82:6a:55:5b:90:88:d4:62:bf:9c:f2:8c:fd
Signature Algorithm: ecdsa-with-SHA256
Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=consultancy,
      CN=registrar.stok.nl
Validity
  Not Before: Dec  9 10:02:36 2020 GMT
  Not After : Dec  9 10:02:36 2021 GMT
Subject: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=consultancy,
      CN=registrar.stok.nl
Subject Public Key Info:
  Public Key Algorithm: id-ecPublicKey
  Public-Key: (256 bit)
  pub:
    04:50:7a:c8:49:1a:8c:69:c7:b5:c3:1d:03:09:ed:
    35:ba:13:f5:88:4c:e6:2b:88:cf:30:18:15:4f:a0:
    59:b0:20:ec:6b:eb:b9:4e:02:b8:93:40:21:89:8d:
    a7:89:c7:11:ce:a7:13:39:f5:0e:34:8e:df:0d:92:
    3e:d0:2d:c7:b7
  ASN1 OID: prime256v1
  NIST CURVE: P-256
X509v3 extensions:
  X509v3 Subject Key Identifier:
08:C2:BF:36:88:7F:79:41:21:85:87:2F:16:A7:AC:A6:EF:B3:D2:B3
  X509v3 Authority Key Identifier:
    keyid:
08:C2:BF:36:88:7F:79:41:21:85:87:2F:16:A7:AC:A6:EF:B3:D2:B3

  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Extended Key Usage:
    CMC Registration Authority, TLS Web Server
    Authentication, TLS Web Client Authentication
  X509v3 Key Usage: critical
    Digital Signature, Non Repudiation, Key Encipherment,
    Data Encipherment, Certificate Sign, CRL Sign
Signature Algorithm: ecdsa-with-SHA256
  30:44:02:20:74:4c:99:00:85:13:b2:f1:bc:fd:f9:02:1a:46:
  fb:17:4c:f8:83:a2:7c:a1:d9:3f:ae:ac:f3:1e:4e:dd:12:c6:
  02:20:11:47:14:db:f5:1a:5e:78:f5:81:b9:42:1c:6e:47:02:
  ab:53:72:70:c5:ba:fb:2d:16:c3:de:9a:a1:82:c3:5f
```

<CODE ENDS>

This the hexadecimal representation, in (request-)voucher examples referred to as regis-cert-hex

```
308202753082021ca00302010202147056eaaa3066d8826a555b9088d462bf9c
f28cfd300a06082a8648ce3d0403023073310b3009060355040613024e4c310b
300906035504080c024e423110300e06035504070c0748656c6d6f6e64311330
11060355040a0c0a76616e64657273746f6b31143012060355040b0c0b636f6e
73756c74616e6379311a301806035504030c117265676973747261722e73746f
6b2e6e6c301e170d3230313230393130303233365a170d323131323039313030
3233365a3073310b3009060355040613024e4c310b300906035504080c024e42
3110300e06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e
64657273746f6b31143012060355040b0c0b636f6e73756c74616e6379311a30
1806035504030c117265676973747261722e73746f6b2e6e6c3059301306072a
8648ce3d020106082a8648ce3d03010703420004507ac8491a8c69c7b5c31d03
09ed35ba13f5884ce62b88cf3018154fa059b020ec6bebb94e02b8934021898d
a789c711cea71339f50e348edf0d923ed02dc7b7a3818d30818a301d0603551d
0e0416041408c2bf36887f79412185872f16a7aca6efb3d2b3301f0603551d23
04183016801408c2bf36887f79412185872f16a7aca6efb3d2b3300f0603551d
130101ff040530030101ff30270603551d250420301e06082b0601050507031c
06082b0601050507030106082b06010505070302300e0603551d0f0101ff0404
030201f6300a06082a8648ce3d04030203470030440220744c99008513b2f1bc
fdf9021a46fb174cf883a27cald93faeacf31e4edd12c60220114714dbf51a5e
78f581b9421c6e4702ab537270c5bafb2d16c3de9aa182c35f<CODE ENDS>
```

B.2.3. MASA Certificate

Certificate:

Data:

```
Version: 3 (0x2)
Serial Number:
    14:26:b8:1c:ce:d8:c3:e8:14:05:cb:87:67:0d:be:ef:d5:81:25:b4
Signature Algorithm: ecdsa-with-SHA256
Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok,
    OU=manufacturer, CN=masa.stok.nl
```

Validity

```
Not Before: Dec 9 10:02:36 2020 GMT
Not After : Sep 5 10:02:36 2023 GMT
Subject: C=NL, ST=NB, L=Helmond, O=vanderstok,
    OU=manufacturer, CN=masa.stok.nl
```

Subject Public Key Info:

```
Public Key Algorithm: id-ecPublicKey
Public-Key: (256 bit)
pub:
```

```
04:59:80:94:66:14:94:20:30:3c:66:08:85:55:86:
db:e7:d4:d1:d7:7a:d2:a3:1a:0c:73:6b:01:0d:02:
12:15:d6:1f:f3:6e:c8:d4:84:60:43:3b:21:c5:83:
80:1e:fc:e2:37:85:77:97:94:d4:aa:34:b5:b6:c6:
```

```
        ed:f3:17:5c:f1
        ASN1 OID: prime256v1
        NIST CURVE: P-256
X509v3 extensions:
  X509v3 Subject Key Identifier:
E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3
  X509v3 Authority Key Identifier:
    keyid:
E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3

  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Extended Key Usage:
    CMC Registration Authority,
    TLS Web Server Authentication,
    TLS Web Client Authentication
  X509v3 Key Usage: critical
    Digital Signature, Non Repudiation, Key Encipherment,
    Data Encipherment, Certificate Sign, CRL Sign
Signature Algorithm: ecdsa-with-SHA256
30:44:02:20:2e:c5:f2:24:72:70:20:ea:6e:74:8b:13:93:67:
8a:e6:fe:fb:8d:56:7f:f5:34:18:a9:ef:a5:0f:c3:99:ca:53:
02:20:3d:dc:91:d0:e9:6a:69:20:01:fb:e4:20:40:de:7c:7d:
98:ed:d8:84:53:61:84:a7:f9:13:06:4c:a9:b2:8f:5c
```

<CODE ENDS>

This is the hexadecimal representation, in (request-)voucher examples referred to as masa-cert-hex.

```
3082026d30820214a00302010202141426b81cced8c3e81405cb87670dbeefd5
8125b4300a06082a8648ce3d040302306f310b3009060355040613024e4c310b
300906035504080c024e423110300e06035504070c0748656c6d6f6e64311330
11060355040a0c0a76616e64657273746f6b31153013060355040b0c0c6d616e
7566616374757265723115301306035504030c0c6d6173612e73746f6b2e6e6c
301e170d3230313230393130303233365a170d3233303930353130303233365a
306f310b3009060355040613024e4c310b300906035504080c024e423110300e
06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e64657273
746f6b31153013060355040b0c0c6d616e756661637475726572311530130603
5504030c0c6d6173612e73746f6b2e6e6c3059301306072a8648ce3d02010608
2a8648ce3d0301070342000459809466149420303c6608855586dbe7d4d1d77a
d2a31a0c736b010d021215d61ff36ec8d48460433b21c583801efce237857797
94d4aa34b5b6c6edf3175cf1a3818d30818a301d0603551d0e04160414e40393
b4c3d3f42a80a47718f6964903011768a3301f0603551d23041830168014e403
93b4c3d3f42a80a47718f6964903011768a3300f0603551d130101ff04053003
0101ff30270603551d250420301e06082b0601050507031c06082b0601050507
030106082b06010505070302300e0603551d0f0101ff0404030201f6300a0608
2a8648ce3d040302034700304402202ec5f224727020ea6e748b1393678ae6fe
fb8d567ff53418a9efa50fc399ca5302203ddc91d0e96a692001fbe42040de7c
7d98edd884536184a7f913064ca9b28f5c<CODE ENDS>
```

B.3. COSE signed voucher request from Pledge to Registrar

In this example the voucher request has been signed by the Pledge, and has been sent to the JRC over CoAPS. The Pledge uses the proximity assertion together with an included proximity-registrar-cert field to inform MASA about its proximity to the specific Registrar.

```
POST coaps://registrar.example.com/est/rv
(Content-Format: application/voucher-cose+cbor)
signed_request_voucher
```

The payload `signed_request_voucher` is shown as hexadecimal dump (with lf added):

```

d28444a101382ea104582097113db094eee8eae48683e7337875c0372164be89d023a5f3d
f52699c0fbfb55902d2a11909c5a60274323032302d31322d32335431323a30353a32325a
0474323032322d31322d32335431323a30353a32325a01020750684ca83e27230aff97630
cf2c1ec409a0d6e706c656467652e312e322e332e340a590279308202753082021ca00302
010202147056eaaa3066d8826a555b9088d462bf9cf28cfd300a06082a8648ce3d0403023
073310b3009060355040613024e4c310b300906035504080c024e423110300e0603550407
0c0748656c6d6f6e6431133011060355040a0c0a76616e64657273746f6b3114301206035
5040b0c0b636f6e73756c74616e6379311a301806035504030c117265676973747261722e
73746f6b2e6e6c301e170d3230313230393130303233365a170d323131323039313030323
3365a3073310b3009060355040613024e4c310b300906035504080c024e423110300e0603
5504070c0748656c6d6f6e6431133011060355040a0c0a76616e64657273746f6b3114301
2060355040b0c0b636f6e73756c74616e6379311a301806035504030c1172656769737472
61722e73746f6b2e6e6c3059301306072a8648ce3d020106082a8648ce3d0301070342000
4507ac8491a8c69c7b5c31d0309ed35ba13f5884ce62b88cf3018154fa059b020ec6bebb9
4e02b8934021898da789c711cea71339f50e348edf0d923ed02dc7b7a3818d30818a301d0
603551d0e0416041408c2bf36887f79412185872f16a7aca6efb3d2b3301f0603551d2304
183016801408c2bf36887f79412185872f16a7aca6efb3d2b3300f0603551d130101ff040
530030101ff30270603551d250420301e06082b0601050507031c06082b06010505070301
06082b06010505070302300e0603551d0f0101ff0404030201f6300a06082a8648ce3d040
30203470030440220744c99008513b2f1bcfd9021a46fb174cf883a27ca1d93faeacf31e
4edd12c60220114714dbf51a5e78f581b9421c6e4702ab537270c5bafb2d16c3de9aa182c
35f58473045022063766c7bbd1b339dbc398e764af3563e93b25a69104befe9aac2b3336b
8f56e1022100cd0419559ad960ccaed4dee3f436eca40b7570b25a52eb60332bc1f299148
4e9

```

<CODE ENDS>

The representation of signed_voucher_request in CBOR diagnostic format is:

```

Diagnose(signed_request_voucher) =
18([
h'A101382E',      # {"alg": -47}
{4: h'97113DB094EEE8EAE48683E7337875C0372164BE89D023A5F3DF52699C0FBFB5'},
h'request_voucher',
h'3045022063766C7BBD1B339DBC398E764AF3563E93B25A69104BEFE9AAC2B3336B8F56E
1022100CD0419559AD960CCAED4DEE3F436ECA40B7570B25A52EB60332BC1F2991484E9'
])

```

```

Diagnose(request_voucher) =
{2501: {2: "2020-12-23T12:05:22Z",
4: "2022-12-23T12:05:22Z",
1: 2,
7: h'684CA83E27230AFF97630CF2C1EC409A',
13: "pledge.1.2.3.4",
10: h'regis-cert-hex'}}

```

<CODE ENDS>

B.4. COSE signed voucher request from Registrar to MASA

In this example the voucher request has been signed by the JRC using the private key from Appendix B.1.2. Contained within this voucher request is the voucher request from the Pledge to JRC.

```
POST coaps://masa.example.com/est/rv
(Content-Format: application/voucher-cose+cbor)
signed_masa_request_voucher
```

The payload `signed_masa_voucher_request` is shown as hexadecimal dump (with `lf` added):

[illegible]

<CODE ENDS>

The representation of signed_masa_voucher_request in CBOR diagnostic format is:

```
Diagnose(signed_registrar_request-voucher)
18([
h'A101382E',      # {"alg": -47}
h'E8735BC4B470C3AA6A7AA9AA8EE584C09C11131B205EFEC5D0313BAD84C5CD0
5'},
h'registrar_request_voucher',
h'3045022100E6B45558C1B806BBA23F4AC626C9BDB6FD354EF4330D8DFB7C529
F29CCA934C802203C1F2CCBBAC89733D17EE7775BC2654C5F1CC96AFBA2741CC3
1532444AA8FED8'
])

Diagnose(registrar_request_voucher)
{2501:
  {2: "2020-12-28T10:03:35Z",
    4: "2022-12-28T10:03:35Z",
    7: h'1551631F6E0416BD162BA53EA00C2A05',
    13: "pledge.1.2.3.4",
    5: h'31322D32385431303A30333A33355A07501551631F6E0416BD162BA53EA00C2
A050D6E706C656467652E312E322E332E3405587131322D32385431303A300000
000000000000000000000416BD162BA53EA00C2A050D6E706C656467652E312E3
22E332E3405587131322D32385431303A',
    9: h'signed_request_voucher' }}
<CODE ENDS>
```

B.5. COSE signed voucher from MASA to Pledge via Registrar

The resulting voucher is created by the MASA and returned via the JRC to the Pledge. It is signed by the MASA's private key Appendix B.1.3 and can be verified by the Pledge using the MASA's public key contained within the MASA certificate.

This is the raw binary signed_voucher, encoded in hexadecimal (with lf added):

```
d28444a101382ea104582039920a34ee92d3148ab3a729f58611193270c9029f7784daf11
2614b19445d5158cfa1190993a70274323032302d31322d32335431353a30333a31325a04
74323032302d31322d32335431353a32333a31325a010007506508e06b2959d5089d7a316
9ea889a490b6e706c656467652e312e322e332e340858753073310b300906035504061302
4e4c310b300906035504080c024e423110300e06035504070c0748656c6d6f6e643113301
1060355040a0c0a76616e64657273746f6b31143012060355040b0c0b636f6e73756c7461
6e6379311a301806035504030c117265676973747261722e73746f6b2e6e6c03f45847304
5022022515d96cd12224ee5d3ac673237163bba24ad84815699285d9618f463ee73fa0221
00a6bfff9d8585c1c9256371ece94da3d26264a5dfec0a354fe7b3aef58344c512f

<CODE ENDS>
```

The representation of signed_voucher in CBOR diagnostic format is:

```
Diagnose(signed_voucher) =  
18([  
  h'A101382E',      # {"alg": -47}  
  {4: h'39920A34EE92D3148AB3A729F58611193270C9029F7784DAF112614B194  
    45D51'},  
  h'voucher',  
  h'3045022022515D96CD12224EE5D3AC673237163BBA24AD84815699285D9618F  
    463EE73FA022100A6BFF9D8585C1C9256371ECE94DA3D26264A5DFEC0A354FE7B  
    3AEF58344C512F'  
])
```

```
Diagnose(voucher) =  
{2451:  
  {2: "2020-12-23T15:03:12Z",  
    4: "2020-12-23T15:23:12Z",  
    1: 0,  
    7: h'6508E06B2959D5089D7A3169EA889A49',  
    11: "pledge.1.2.3.4",  
    8: h'regis-cert-hex',  
    3: false}}  
<CODE ENDS>
```

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Esko Dijk
IoTconsultancy.nl

Email: esko.dijk@iotconsultancy.nl

anima Working Group
Internet-Draft
Updates: 8366, 8995 (if approved)
Intended status: Standards Track
Expires: 9 October 2022

M. Richardson
Sandelman Software Works
P. van der Stok
vanderstok consultancy
P. Kampanakis
Cisco Systems
E. Dijk
IoTconsultancy.nl
7 April 2022

Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)
draft-ietf-anima-constrained-voucher-17

Abstract

This document defines the Constrained Bootstrapping Remote Secure Key Infrastructure (Constrained BRSKI) protocol, which provides a solution for secure zero-touch bootstrapping of resource-constrained (IoT) devices into the network of a domain owner. This protocol is designed for constrained networks, which may have limited data throughput or may experience frequent packet loss. Constrained BRSKI is a variant of the BRSKI protocol, which uses an artifact signed by the device manufacturer called the "voucher" which enables a new device and the owner's network to mutually authenticate. While the BRSKI voucher is typically encoded in JSON, Constrained BRSKI defines a compact CBOR-encoded voucher. The BRSKI voucher is extended with new data types that allow for smaller voucher sizes. The Enrollment over Secure Transport (EST) protocol, used in BRSKI, is replaced with EST-over-CoAPS; and HTTPS used in BRSKI is replaced with CoAPS.

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 9 October 2022.

Copyright Notice

Copyright (c) 2022 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 extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
2. Terminology	5
3. Requirements Language	6
4. Overview of Protocol	6
5. Updates to RFC8366 and RFC8995	7
6. BRSKI-EST Protocol	8
6.1. Registrar and the Server Name Indicator (SNI)	8
6.2. TLS Client Certificates: IDevID authentication	9
6.3. Discovery, URIs and Content Formats	10
6.3.1. RFC8995 Telemetry Returns	12
6.4. Join Proxy options	13
6.5. Extensions to BRSKI	13
6.5.1. Discovery	13
6.5.2. CoAP responses	13
6.6. Extensions to EST-coaps	14
6.6.1. Pledge Extensions	15
6.6.2. EST-client Extensions	16
6.6.3. Registrar Extensions	18
6.7. DTLS handshake fragmentation Considerations	19
7. BRSKI-MASA Protocol	19
7.1. Protocol and Formats	19
7.2. Registrar Voucher Request	20
7.3. MASA and the Server Name Indicator (SNI)	20
7.3.1. Registrar Certificate Requirement	21
8. Pinning in Voucher Artifacts	21
8.1. Registrar Identity Selection and Encoding	21
8.2. MASA Pinning Policy	23
8.3. Pinning of Raw Public Keys	24
8.4. Considerations for use of IDevID-Issuer	25
9. Artifacts	26
9.1. Voucher Request artifact	26
9.1.1. Tree Diagram	26

9.1.2.	SID values	27
9.1.3.	YANG Module	28
9.1.4.	Example voucher request artifact	32
9.2.	Voucher artifact	32
9.2.1.	Tree Diagram	32
9.2.2.	SID values	33
9.2.3.	YANG Module	33
9.2.4.	Example voucher artifacts	36
9.3.	Signing voucher and voucher-request artifacts with COSE	37
10.	Deployment-specific Discovery Considerations	38
10.1.	6TSCH Deployments	39
10.2.	Generic networks using GRASP	39
10.3.	Generic networks using mDNS	39
10.4.	Thread networks using Mesh Link Establishment (MLE)	39
10.5.	Non-mesh networks using CoAP Discovery	40
11.	Design Considerations	40
12.	Raw Public Key Use Considerations	40
12.1.	The Registrar Trust Anchor	40
12.2.	The Pledge Voucher Request	41
12.3.	The Voucher Response	41
13.	Use of constrained vouchers with HTTPS	41
14.	Security Considerations	42
14.1.	Duplicate serial-numbers	42
14.2.	IDevID security in Pledge	43
14.3.	Security of CoAP and UDP protocols	44
14.4.	Registrar Certificate may be self-signed	45
14.5.	Use of RPK alternatives to proximity-registrar-cert	45
14.6.	MASA support of CoAPS	46
15.	IANA Considerations	46
15.1.	Resource Type Registry	46
15.2.	The IETF XML Registry	47
15.3.	The YANG Module Names Registry	47
15.4.	The RFC SID range assignment sub-registry	47
15.5.	Media Types Registry	48
15.5.1.	application/voucher-cose+cbor	48
15.6.	CoAP Content-Format Registry	48
15.7.	Update to BRSKI Parameters Registry	49
16.	Acknowledgements	49
17.	Changelog	50
18.	References	50
18.1.	Normative References	50
18.2.	Informative References	54
Appendix A.	Library Support for BRSKI	56
A.1.	OpenSSL	57
A.2.	mbedtls	58
Appendix B.	Constrained BRSKI-EST Message Examples	59
B.1.	enrollstatus	59

B.2. voucher_status	60
Appendix C. COSE-signed Voucher (Request) Examples	61
C.1. Pledge, Registrar and MASA Keys	61
C.1.1. Pledge IDevID private key	61
C.1.2. Registrar private key	61
C.1.3. MASA private key	62
C.2. Pledge, Registrar and MASA Certificates	62
C.2.1. Pledge IDevID Certificate	62
C.2.2. Registrar Certificate	64
C.2.3. MASA Certificate	66
C.3. COSE-signed Pledge Voucher Request (PVR)	68
C.4. COSE-signed Registrar Voucher Request (RVR)	70
C.5. COSE-signed Voucher from MASA	72
Appendix D. Generating Certificates with OpenSSL	74
Appendix E. Pledge Device Class Profiles	77
E.1. Minimal Pledge	78
E.2. Typical Pledge	78
E.3. Full-featured Pledge	78
E.4. Comparison Chart of Pledge Classes	78
Contributors	79
Authors' Addresses	80

1. Introduction

Secure enrollment of new nodes into constrained networks with constrained nodes presents unique challenges. As explained in [RFC7228], the networks are challenged and the nodes are constrained by energy, memory space, and code size.

The Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol described in [RFC8995] provides a solution for secure zero-touch (automated) bootstrap of new (unconfigured) devices. In it, new devices, such as IoT devices, are called "pledges", and equipped with a factory-installed Initial Device Identifier (IDevID) (see [ieee802-1AR]), are enrolled into a network.

The BRSKI solution described in [RFC8995] was designed to be modular, and this document describes a version scaled to the constraints of IoT deployments.

Therefore, this document defines a constrained version of the voucher artifact (described in [RFC8366]), along with a constrained version of BRSKI. This constrained-BRSKI protocol makes use of the constrained CoAP-based version of EST (EST-coaps from [I-D.ietf-ace-coap-est]) rather than the EST over HTTPS [RFC7030]. Constrained-BRSKI is itself scalable to multiple resource levels through the definition of optional functions. Appendix E illustrates this.

In BRSKI, the [RFC8366] voucher is by default serialized to JSON with a signature in CMS [RFC5652]. This document defines a new voucher serialization to CBOR [RFC8949] with a signature in COSE [I-D.ietf-cose-rfc8152bis-struct].

This COSE-signed CBOR-encoded voucher is transported using both secured CoAP and HTTPS. The CoAP connection (between Pledge and Registrar) is to be protected by either OSCORE+EDHOC [I-D.ietf-lake-edhoc] or DTLS (CoAPS). The HTTP connection (between Registrar and MASA) is to be protected using TLS (HTTPS).

This document specifies a constrained voucher-request artifact based on Section 3 of [RFC8995], and voucher(-request) transport over CoAP based on Section 3 of [RFC8995] and on [I-D.ietf-ace-coap-est].

The CBOR definitions for the constrained voucher format are defined using the mechanism described in [I-D.ietf-core-yang-cbor] using the SID mechanism explained in [I-D.ietf-core-sid]. As the tooling to convert YANG documents into a list of SID keys is still in its infancy, the table of SID values presented here MUST be considered normative rather than the output of the tool specified in [I-D.ietf-core-sid].

2. Terminology

The following terms are defined in [RFC8366], and are used identically as in that document: artifact, domain, imprint, Join Registrar/Coordinator (JRC), Manufacturer Authorized Signing Authority (MASA), Pledge, Registrar, Trust of First Use (TOFU), and Voucher.

The following terms from [RFC8995] are used identically as in that document: Domain CA, enrollment, IDevID, Join Proxy, LDevID, manufacturer, nonced, nonceless, PKIX.

The term Pledge Voucher Request, or acronym PVR, is introduced to refer to the voucher request between the pledge and the Registrar.

The term Registrar Voucher Request, or acronym RVR, is introduced to refer to the voucher request between the Registrar and the MASA.

In code examples, the string "<CODE BEGINS>" denotes the start of a code example and "<CODE ENDS>" the end of the code example. Four dots ("....") in a CBOR diagnostic notation byte string denotes a further sequence of bytes that is not shown for brevity.

3. 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.

4. Overview of Protocol

[RFC8366] provides for vouchers that assert proximity, authenticate the Registrar, and can offer varying levels of anti-replay protection.

The proximity proof provided for in [RFC8366], is an assertion that the Pledge and the Registrar are believed to be close together, from a network topology point of view. Like in [RFC8995], proximity is shown by making TLS connections between the Pledge and Registrar using IPv6 Link-Local addresses.

The TLS connection is used to make a Voucher Request. This request is verified by an agent of the Pledge's manufacturer, which then issues a voucher. The voucher provides an authorization statement from the manufacturer indicating that the Registrar is the intended owner of the device. The voucher refers to the Registrar through pinning of the Registrar's identity.

This document does not make any extensions to the semantic meaning of vouchers, only the encoding has been changed to optimize for constrained devices and networks. The two main parts of the BRSKI protocol are named separately in this document: BRSKI-EST for the protocol between Pledge and Registrar, and BRSKI-MASA for the protocol between the Registrar and the MASA.

Time-based vouchers are supported in this definition, but given that constrained devices are extremely unlikely to have accurate time, their use will be uncommon. Most Pledges using constrained vouchers will be online during enrollment and will use live nonces to provide anti-replay protection rather than expiry times.

[RFC8366] defines the voucher artifact, while the Voucher Request artifact was defined in [RFC8995]. This document defines both a constrained voucher and a constrained voucher-request. They are presented in the order "voucher-request", followed by a "voucher" response as this is the order that they occur in the protocol.

The constrained voucher request MUST be signed by the Pledge. It signs using the private key associated with its IDevID X.509 certificate, or if an IDevID is not available, then the private key associated with its manufacturer-installed raw public key (RPK). Section 12 provides additional details on PKIX-less operations.

The constrained voucher MUST be signed by the MASA.

For the constrained voucher request this document defines two distinct methods for the Pledge to identify the Registrar: using either the Registrar's X.509 certificate, or using a raw public key (RPK) of the Registrar.

For the constrained voucher both methods are supported to indicate (pin) a trusted domain identity: using either a pinned domain X.509 certificate, or a pinned raw public key (RPK).

The BRSKI architectures mandates that the MASA be aware of the capabilities of the pledge. This is not a drawback as the pledges are constructed by a manufacturer which also arranges for the MASA to be aware of the inventory of devices.

The MASA therefore knows if the pledge supports PKIX operations, PKIX format certificates, or if the pledge is limited to Raw Public Keys (RPK). Based upon this, the MASA can select which attributes to use in the voucher for certain operations, like the pinning of the Registrar identity. This is described in more detail in Section 9.2.3, Section 8 and Section 8.3 (for RPK specifically).

5. Updates to RFC8366 and RFC8995

This section details the ways in which this document updates other RFCs. The terminology for Updates is taken from [I-D.kuehlewind-update-tag].

This document Updates [RFC8366]. It Extends [RFC8366] by creating a new serialization format, and creates a mechanism to pin Raw Public Key (RPK).

This document Updates [RFC8995]. It Amends [RFC8995]

- * by clarifying how pinning is done,
- * adopts clearer explanation of the TLS Server Name Indicator (SNI), see Section 6.1 and Section 7.3
- * clarifies when new trust anchors should be retrieved (Section 6.6.1),

- * clarified what kinds of Extended Key Usage attributes are appropriate for each certificate (Section 7.3.1)

It Extends [RFC8995] as follows: * defines the CoAP version of the BRSKI protocol

- * makes some messages optional if the results can be inferred from other validations (Section 6.6),
- * provides the option to return trust anchors in a simpler format (Section 6.6.3)
- * extends the BRSKI-MASA protocol to carry the new voucher-cose+cbor format.

6. BRSKI-EST Protocol

This section describes the constrained BRSKI extensions to EST-coaps [I-D.ietf-ace-coap-est] to transport the voucher between Registrar and Pledge (optionally via a Join Proxy) over CoAP. The extensions are targeting low-resource networks with small packets.

The constrained BRSKI-EST protocol described in this section is between the Pledge and the Registrar only.

6.1. Registrar and the Server Name Indicator (SNI)

A DTLS connection is established between the Pledge and the Registrar, similar to the TLS connection described in Section 5.1 of [RFC8995]. This may occur via a Join Proxy as described in Section 6.4. Regardless of the Join Proxy mechanism, the DTLS connection should operate identically.

The SNI issue described below affects [RFC8995] as well, and is reported in errata: <https://www.rfc-editor.org/errata/eid6648>

As the Registrar is discovered by IP address, and typically connected via a Join Proxy, the name of the Registrar is not known to the Pledge. The Pledge will not know what the hostname for the Registrar is, so it cannot do RFC6125 DNS-ID validation on the Registrar's certificate. Instead, it must do validation using the RFC8366 voucher.

As the Pledge does not know the name of the Registrar, the Pledge cannot put any reasonable value into the [RFC6066] Server Name Indicator (SNI). Therefore the Pledge SHOULD omit the SNI extension as per Section 9.2 of [RFC8446].

In some cases, particularly while testing BRSKI, a Pledge may be given the hostname of a particular Registrar to connect to directly. Such a bypass of the discovery process may result in the Pledge taking a different code branch to establish a DTLS connection, and may result in the SNI being inserted by a library. The Registrar MUST ignore any SNI seen.

A primary motivation for making the SNI ubiquitous in the public web is because it allows for multi-tenant hosting of HTTPS sites on a single (scarce) IPv4 address. This consideration does not apply to the server function in the Registrar because:

- * it uses DTLS and CoAP, not HTTPS
- * it typically uses IPv6, often [RFC4193] Unique Local Address, which are plentiful
- * the server port number is typically discovered, so multiple tenants can be accommodated via unique port numbers.

As per Section 3.6.1 of [RFC7030], the Registrar certificate MUST have the Extended Key Usage (EKU) id-kp-cmcRA. This certificate is also used as a TLS Server Certificate, so it MUST also have the EKU id-kp-serverAuth.

6.2. TLS Client Certificates: IDevID authentication

As described in Section 5.1 of [RFC8995], the Pledge makes a connection to the Registrar using a TLS Client Certificate for authentication.

Subsequently the Pledge will send a Pledge Voucher Request (PVR).

As explained below in Section 8.1, the "x5bag" element may be used in the RVR to communicate identity of the Registrar to MASA. The Pledge SHOULD NOT use the x5bag attribute in this way in the PVR. A Registrar that processes a PVR with an x5bag attribute MUST ignore it, and MUST use only the TLS Client Certificate extension for authentication of the Pledge.

A situation where the Pledge MAY use the x5bag is for communication of certificate chains to the MASA. This would arise in some vendor-specific situations involving outsourcing of MASA functionality, or rekeying of the IDevID certification authority.

6.3. Discovery, URIs and Content Formats

To keep the protocol messages small the EST-coaps and constrained-BRSKI URIs are shorter than the respective EST and BRSKI URIs.

The EST-coaps server URIs differ from the EST URIs by replacing the scheme https by coaps and by specifying shorter resource path names. Below are some examples; the first two using a discovered short path name and the last one using the well-known URI of EST which requires no discovery.

```
coaps://server.example.com/est/<short-name>
coaps://server.example.com/e/<short-name>
coaps://server.example.com/.well-known/est/<short-name>
```

Similarly the constrained BRSKI server URIs differ from the BRSKI URIs by replacing the scheme https by coaps and by specifying shorter resource path names. Below are some examples; the first two using a discovered short path name and the last one using the well-known URI prefix which requires no discovery. This is the same `"/.well-known/brski"` prefix as defined in Section 5 of [RFC8995].

```
coaps://server.example.com/brski/<short-name>
coaps://server.example.com/b/<short-name>
coaps://server.example.com/.well-known/brski/<short-name>
```

Figure 5 in Section 3.2.2 of [RFC7030] enumerates the operations supported by EST, for which Table 1 in Section 5.1 of [I-D.ietf-ace-coap-est] enumerates the corresponding EST-coaps short path names. Similarly, Table 1 below provides the mapping from the supported BRSKI extension URI paths to the constrained-BRSKI URI paths.

=====	=====
BRSKI resource constrained-BRSKI resource	
=====	=====
/requestvoucher /rv	
-----	-----
/voucher_status /vs	
-----	-----
/enrollstatus /es	
-----	-----

Table 1: BRSKI URI paths mapping to
Constrained BRSKI URI paths

Note that /requestvoucher indicated above occurs between the Pledge and Registrar (in scope of the BRSKI-EST protocol), but it also occurs between Registrar and MASA. However, as described in Section 6, this section and above table addresses only the BRSKI-EST protocol.

Pledges that wish to discover the available BRSKI bootstrap options/formats, or reduce the size of the CoAP headers by eliminating the "/.well-known/brski" path, can do a discovery operation using [RFC6690] Section 4 by sending a discovery query to the Registrar.

For example, if the Registrar supports a short BRSKI URL (/b) and supports the voucher format "application/voucher-cose+cbor" (TBD3), and status reporting in both CBOR and JSON formats:

```
REQ: GET /.well-known/core?rt=brski*
```

```
RES: 2.05 Content
```

```
Content-Format: 40
```

```
Payload:
```

```
</b>;rt=brski,  
</b/rv>;rt=brski.rv;ct=TBD3,  
</b/vs>;rt=brski.vs;ct="50 60",  
</b/es>;rt=brski.es;ct="50 60"
```

The Registrar is under no obligation to provide shorter URLs, and MAY respond to this query with only the "/.well-known/brski/<short-name>" resources for the short names as defined in Table 1.

Registrars that have implemented shorter URLs MUST also respond in equivalent ways to the corresponding "/.well-known/brski/<short-name>" URLs, and MUST NOT distinguish between them. In particular, a Pledge MAY use the longer and shorter URLs in any combination.

When responding to a discovery request for BRSKI resources, the server MAY in addition return the full resource paths and the content types which are supported by these resources as shown in above example. This is useful when multiple content types are specified for a particular resource on the server. The server responds with only the root path for the BRSKI resources (rt=brski, resource /b in above example) and no others in case the client queries for only rt=brski type resources. (So, a query for rt=brski, without the wildcard character.)

Without discovery, a longer well-known URL can only be used, such as:

```
REQ: GET /.well-known/brski/rv
```

while with discovery of shorter URLs, a request such as:

```
REQ: GET /b/rv
```

is possible.

The return of multiple content-types in the "ct" attribute allows the Pledge to choose the most appropriate one. Note that Content-Format TBD3 ("application/voucher-cose+cbor") is defined in this document.

Content-Format TBD3 MUST be supported by the Registrar for the /rv resource. If the "ct" attribute is not indicated for the /rv resource in the link format description, this implies that at least TBD3 is supported.

Note that this specification allows for voucher-cose+cbor format requests and vouchers to be transmitted over HTTPS, as well as for voucher-cms+json and other formats yet to be defined over CoAP. The burden for this flexibility is placed upon the Registrar. A Pledge on constrained hardware is expected to support a single format only.

The Pledge and MASA need to support one or more formats (at least TBD3) for the voucher and for the voucher request. The MASA needs to support all formats that the Pledge supports.

Section 10 details how the Pledge discovers the Registrar and Join Proxy in different deployment scenarios.

6.3.1. RFC8995 Telemetry Returns

[RFC8995] defines two telemetry returns from the Pledge which are sent to the Registrar. These are the BRSKI Status Telemetry [RFC8995], Section 5.7 and the Enrollment Status Telemetry [RFC8995], Section 5.9.4. These are two POST operations made the by Pledge at two key steps in the process.

[RFC8995] defines the content of these POST operations in CDDL, which are serialized as JSON. This document extends the list of acceptable formats to CBOR as well as JSON, using the rules from [RFC8610].

The existing JSON format is described as CoAP Content-Format 50 ("application/json"), and it MAY be supported. The new CBOR format described as CoAP Content-Format 60 ("application/cbor"), MUST be supported by the Registrar for both the /vs and /es resources.

6.4. Join Proxy options

[I-D.ietf-anima-constrained-join-proxy] specifies a constrained Join Proxy that is optionally placed between Pledge and Registrar. This includes methods for discovery of the Join Proxy by the Pledge and discovery of the Registrar by the Join Proxy.

6.5. Extensions to BRSKI

6.5.1. Discovery

The Pledge discovers an IP address and port number that connects to the Registrar (possibly via a Join Proxy), and it establishes a DTLS connection.

No further discovery of hosts or port numbers is required, but a pledge that can do more than one kind of enrollment (future work offers protocols other than [I-D.ietf-ace-coap-est]), then a pledge may need to use CoAP Discovery to determine what other protocols are available.

A Pledge that only supports the EST-coaps enrollment method SHOULD NOT use discovery for BRSKI resources. It is more efficient to just try the supported enrollment method via the well-known BRSKI/EST-coaps resources. This also avoids the Pledge doing any CoRE Link Format parsing, which is specified in [I-D.ietf-ace-coap-est], Section 4.1.

The Registrar MUST support all of the EST resources at their default ".well-known" locations (on the specified port) as well as any server-specific shorter form that might also be supported.

However, when discovery is being done by the Pledge, it is possible for the Registrar to return references to resources which are on different port numbers. The Registrar SHOULD NOT use different ports numbers by default, because a Pledge that is connected via a Join Proxy can only access a single UDP port. A Registrar configured to never use Join Proxies MAY be configured to use multiple port numbers. Therefore a Registrar MUST host all discoverable BRSKI resources on the same (UDP) server port that the Pledge's DTLS connection is using. Using the same UDP server port for all resources allows the Pledge to continue via the same DTLS connection which is more efficient.

6.5.2. CoAP responses

[RFC8995], Section 5 defines a number of HTTP response codes that the Registrar is to return when certain conditions occur.

The 401, 403, 404, 406 and 415 response codes map directly to CoAP codes 4.01, 4.03, 4.04, 4.06 and 4.15.

The 202 Retry process which occurs in the voucher request, is to be handled in the same way as Section 5.7 of [I-D.ietf-ace-coap-est] process for Delayed Responses.

6.6. Extensions to EST-coaps

This document extends [I-D.ietf-ace-coap-est], and it inherits the functions described in that document: specifically, the mandatory Simple (Re-)Enrollment (/sen and /sren) and Certification Authority certificates request (/crtcs). Support for CSR Attributes Request (/att) and server-side key generation (/skg, /skc) remains optional for the EST server.

Collecting the resource definitions from both [RFC8995], [RFC7030], and [I-D.ietf-ace-coap-est] results in the following shorter forms of URI paths for the commonly used resources:

=====	=====	=====
BRSKI + EST	Constrained-BRSKI + EST	Well-known URI namespace
=====	=====	=====
/requestvoucher	/rv	brski
/voucher_status	/vs	brski
/csrattrs	/att	est
/simpleenroll	/sen	est
/cacerts	/crtcs	est
/enrollstatus	/es	brski
/simplereenroll	/sren	est
=====	=====	=====

Table 2: BRSKI/EST URI paths mapping to Constrained BRSKI/EST short URI paths

6.6.1. Pledge Extensions

This section defines extensions to the BRSKI Pledge, which are applicable during the BRSKI bootstrap procedure. A Pledge which only supports the EST-coaps enrollment method, SHOULD NOT use discovery for EST-coaps resources, because it is more efficient to enroll (e.g. /sen) via the well-known EST resource on the current DTLS connection. This avoids an additional round-trip of packets and avoids the Pledge having to unnecessarily implement CoRE Link Format parsing.

A constrained Pledge SHOULD NOT perform the optional EST "CSR attributes request" (/att) to minimize network traffic. The Pledge selects which attributes to include in the CSR.

One or more Subject Distinguished Name fields MUST be included. If the Pledge has no specific information on what attributes/fields are desired in the CSR, it MUST use the Subject Distinguished Name fields from its LDevID unmodified. The Pledge can receive such information via the voucher (encoded in a vendor-specific way) or via some other, out-of-band means.

A constrained Pledge MAY use the following optimized EST-coaps procedure to minimize network traffic.

1. if the voucher, that validates the current Registrar, contains a single pinned domain CA certificate, the Pledge provisionally considers this certificate as the EST trust anchor, as if it were the result of "CA certificates request" (/crts) to the Registrar.
2. Using this CA certificate as trust anchor it proceeds with EST simple enrollment (/sen) to obtain its provisionally trusted LDevID certificate.
3. If the Pledge validates that the trust anchor CA was used to sign its LDevID certificate, the Pledge accepts the pinned domain CA certificate as the legitimate trust anchor CA for the Registrar's domain and accepts the associated LDevID certificate.
4. If the trust anchor CA was NOT used to sign its LDevID certificate, the Pledge MUST perform an actual "CA certificates request" (/crts) to the EST server to obtain the EST CA trust anchor(s) since these can differ from the (temporary) pinned domain CA.

5. When doing this /crtts request, the Pledge MAY use a CoAP Accept Option with value TBD287 ("application/pkix-cert") to limit the number of returned EST CA trust anchors to only one. A constrained Pledge MAY support only this format in a /crtts response, per Section 5.3 of [I-D.ietf-ace-coap-est].
6. If the Pledge cannot obtain the single CA certificate or the finally validated CA certificate cannot be chained to the LDevID certificate, then the Pledge MUST abort the enrollment process and report the error using the enrollment status telemetry (/es).

Note that even though the Pledge may avoid performing any /crtts request using the above EST-coaps procedure during bootstrap, it SHOULD support retrieval of the trust anchor CA periodically as detailed in the next section.

6.6.2. EST-client Extensions

This section defines extensions to EST-coaps clients, used after the BRSKI bootstrap procedure is completed. (Note that such client is not called "Pledge" in this section, since it is already enrolled into the domain.) A constrained EST-coaps client MAY support only the Content-Format TBD287 ("application/pkix-cert") in a /crtts response, per Section 5.3 of [I-D.ietf-ace-coap-est]. In this case, it can only store one trust anchor of the domain.

An EST-coaps client that has an idea of the current time (internally, or via NTP) SHOULD consider the validity time of the trust anchor CA, and MAY begin requesting a new trust anchor CA using the /crtts request when the CA has 50% of its validity time (notAfter - notBefore) left. A client without access to the current time cannot decide if the trust anchor CA has expired, and SHOULD poll periodically for a new trust anchor using the /crtts request at an interval of approximately 1 month. An EST-coaps server SHOULD include the CoAP ETag Option in every response to a /crtts request, to enable clients to perform low-overhead validation whether their trust anchor CA is still valid. The EST-coaps client SHOULD store the ETag resulting from a /crtts response in memory and SHOULD use this value in an ETag Option in its next GET /crtts request.

The above-mentioned limitation that an EST-coaps client may support only one trust anchor CA is not an issue in case the domain trust anchor remains stable. However, special consideration is needed for cases where the domain trust anchor can change over time. Such a change may happen due to relocation of the client device to a new domain, or due to key update of the trust anchor as described in [RFC4210], Section 4.4.

From the client's viewpoint, a trust anchor change typically happens during EST re-enrollment: a change of domain CA requires all devices operating under the old domain CA to acquire a new LDevID issued by the new domain CA. A client's re-enrollment may be triggered by various events, such as an instruction to re-enroll sent by a domain entity, or an imminent expiry of its LDevID certificate. How the re-enrollment is explicitly triggered on the client by a domain entity, such as a commissioner or a Registrar, is out of scope of this specification.

The mechanism described in [RFC4210], Section 4.4 for Root CA key update requires four certificates: OldWithOld, OldWithNew, NewWithOld, and NewWithNew. The OldWithOld certificate is already stored in the EST client's trust store. The NewWithNew certificate will be distributed as the single certificate in a /crt response, during EST re-enrollment. Since the EST client can only accept a single certificate in a /crt response it implies that the EST client cannot obtain the certificates OldWithNew and NewWithOld in this way, to perform the complete verification of the new domain CA. Instead, the client only verifies the EST server (Registrar) using its old domain CA certificate in its trust store as detailed below, and based on this trust in the active and valid DTLS connection it automatically trusts the new (NewWithNew) domain CA certificate that the EST server provides in the /crt response.

In this manner, even during rollover of trust anchors, it is possible to have only a single trust anchor provided in a /crt response.

During the period of the certificate renewal, it is not possible to create new communication channels between devices with NewCA certificates devices with OldCA certificates. One option is that devices should avoid restarting existing DTLS or OSCORE connections during this interval that new certificates are being deployed. The recommended period for certificate renewal is 24 hours. For re-enrollment, the constrained EST-coaps client MUST support the following EST-coaps procedure, where optional re-enrollment to a new domain is under control of the Registrar:

1. The client connects with DTLS to the Registrar, and authenticates with its present domain certificate (LDevID certificate) as usual. The Registrar authenticates itself with its domain certificate that is trusted by the client, i.e. it chains to the single trust anchor that the client has stored. This is the "old" trust anchor, the one that will be eventually replaced in case the Registrar decides to re-enroll the client into a new domain.

2. The client performs the simple re-enrollment request (/sren) and upon success it obtains a new LDevID.
3. The client verifies the new LDevID against its (single) existing domain trust anchor. If it chains successfully, this means the trust anchor did not change and the client MAY skip retrieving the current CA certificate using the "CA certificates request" (/crts). If it does not chain successfully, this means the trust anchor was changed/updated and the client then MUST retrieve the new domain trust anchor using the "CA certificates request" (/crts).
4. If the client retrieved a new trust anchor in step 3, then it MUST verify that the new trust anchor chains with the new LDevID certificate it obtained in step 2. If it chains successfully, the client stores both, accepts the new LDevID certificate and stops using its prior LDevID certificate. If it does not chain successfully, the client MUST NOT update its LDevID certificate, it MUST NOT update its (single) domain trust anchor, and the client MUST abort the enrollment process and report the error to the Registrar using enrollment status telemetry (/es).

Note that even though the EST-coaps client may skip the /crts request in step 3, it SHOULD support retrieval of the trust anchor CA periodically as detailed earlier in this section.

6.6.3. Registrar Extensions

A Registrar SHOULD host any discoverable EST-coaps resources on the same (UDP) server port that the Pledge's DTLS initial connection is using. This avoids the overhead of the Pledge reconnecting using DTLS, when it performs EST enrollment after the BRSKI voucher request.

The Content-Format 50 (application/json) MUST be supported and 60 (application/cbor) MUST be supported by the Registrar for the /vs and /es resources.

Content-Format TBD3 MUST be supported by the Registrar for the /rv resource.

When a Registrar receives a "CA certificates request" (/crts) request with a CoAP Accept Option with value TBD287 ("application/pkix-cert") it SHOULD return only the single CA certificate that is the envisioned or actual authority for the current, authenticated Pledge making the request.

If the Pledge included in its request an Accept Option for only the TBD287 ("application/pkix-cert") Content Format, but the domain has been configured to operate with multiple CA trust anchors only, then the Registrar returns a 4.06 Not Acceptable error to signal that the Pledge needs to use the Content Format 281 ("application/pkcs7-mime; smime-type=certs-only") to retrieve all the certificates.

If the current authenticated client is an EST-coaps client that was already enrolled in the domain, and the Registrar is configured to assign this client to a new domain CA trust anchor during the next EST re-enrollment procedure, then the Registrar MUST respond with the new domain CA certificate in case the client performs the "CA Certificates request" (/crt) with an Accept Option for TBD287 only. This signals the client that a new domain is assigned to it. The client follows the procedure as defined in Section 6.6.2.

6.7. DTLS handshake fragmentation Considerations

DTLS includes a mechanism to fragment the handshake messages. This is described in Section 4.4 of [I-D.ietf-tls-dtls13]. The protocol described in this document will often be used with a Join Proxy described in [I-D.ietf-anima-constrained-join-proxy]. The Join Proxy will need some overhead, while the maximum packet sized guaranteed on 802.15.4 networks is 1280 bytes. It is RECOMMENDED that a PMTU of 1024 bytes be assumed for the DTLS handshake. It is unlikely that any Packet Too Big indications [RFC4443] will be relayed by the Join Proxy.

During the operation of the constrained BRSKI-EST protocol, the CoAP Blockwise transfer mechanism will be used when message sizes exceed the PMTU. A Pledge/EST-client on a constrained network MUST use the (D)TLS maximum fragment length extension ("max_fragment_length") defined in Section 4 of [RFC6066] with the maximum fragment length set to a value of either 2^9 or 2^{10} .

7. BRSKI-MASA Protocol

This section describes extensions to and clarifications of the BRSKI-MASA protocol between Registrar and MASA.

7.1. Protocol and Formats

Section 5.4 of [RFC8995] describes a connection between the Registrar and the MASA as being a normal TLS connection using HTTPS. This document does not change that. The Registrar MUST use the format "application/voucher-cose+cbor" in its voucher request to MASA, when the Pledge uses this format in its requests to the Registrar [RFC8995].

The MASA only needs to support formats for which there are Pledges that use that format.

The Registrar MUST use the same format for the RVR as the Pledge used for its PVR.

The Registrar indicates the voucher format it wants to receive from MASA using the HTTP Accept header. This format MUST be the same as the format of the PVR, so that the Pledge can parse it.

At the moment of writing the creation of coaps based MASAs is deemed unrealistic. The use of CoAP for the BRSKI-MASA connection can be the subject of another document. Some consideration was made to specify CoAP support for consistency, but:

- * the Registrar is not expected to be so constrained that it cannot support HTTPS client connections.
- * the technology and experience to build Internet-scale HTTPS responders (which the MASA is) is common, while the experience doing the same for CoAP is much less common.
- * a Registrar is likely to provide onboarding services to both constrained and non-constrained devices. Such a Registrar would need to speak HTTPS anyway.
- * a manufacturer is likely to offer both constrained and non-constrained devices, so there may in practice be no situation in which the MASA could be CoAP-only. Additionally, as the MASA is intended to be a function that can easily be outsourced to a third-party service provider, reducing the complexity would also seem to reduce the cost of that function.
- * security-related considerations: see Section 14.6.

7.2. Registrar Voucher Request

If the PVR contains a proximity assertion, the Registrar MUST propagate this assertion into the RVR by including the "assertion" field with the value "proximity". This conforms to the example in Section 3.3 of [RFC8995] of carrying the assertion forward.

7.3. MASA and the Server Name Indicator (SNI)

A TLS/HTTPS connection is established between the Registrar and MASA.

Section 5.4 of [RFC8995] explains this process, and there are no externally visible changes. A MASA that supports the unconstrained voucher formats should be able to support constrained voucher formats equally well.

There is no requirement that a single MASA be used for both constrained and unconstrained voucher requests: the choice of MASA is determined by the id-mod-MASAURLExtn2016 extension contained in the IDDevID.

The Registrar MUST do [RFC6125] DNS-ID checks on the contents of the certificate provided by the MASA.

In contrast to the Pledge/Registrar situation, the Registrar always knows the name of the MASA, and MUST always include an [RFC6066] Server Name Indicator. The SNI is optional in TLS1.2, but common. The SNI is considered mandatory with TLS1.3.

The presence of the SNI is needed by the MASA, in order for the MASA's server to host multiple tenants (for different customers).

The Registrar SHOULD use a TLS Client Certificate to authenticate to the MASA per Section 5.4.1 of [RFC8995]. If the certificate that the Registrar uses is marked as a id-kp-cmcRA certificate, via Extended Key Usage, then it MUST also have the id-kp-clientAuth EKU attribute set.

7.3.1. Registrar Certificate Requirement

In summary for typical Registrar use, where a single Registrar certificate is used, then the certificate MUST have EKU of: id-kp-cmcRA, id-kp-serverAuth, id-kp-clientAuth.

8. Pinning in Voucher Artifacts

The voucher is a statement by the MASA for use by the Pledge that provides the identity of the Pledge's owner. This section describes how the owner's identity is determined and how it is specified within the voucher.

8.1. Registrar Identity Selection and Encoding

Section 5.5 of [RFC8995] describes BRSKI policies for selection of the owner identity. It indicates some of the flexibility that is possible for the Registrar, and recommends the Registrar to include only certificates in the voucher request (CMS) signing structure that participate in the certificate chain that is to be pinned.

The MASA is expected to evaluate the certificates included by the Registrar in its voucher request, forming them into a chain with the Registrar's (signing) identity on one end. Then, it pins a certificate selected from the chain. For instance, for a domain with a two-level certification authority (see Figure 1), where the voucher-request has been signed by "Registrar", its signing structure includes two additional CA certificates. The arrows in the figure indicate the issuing of a certificate, i.e. author of (1) issued (2) and author of (2) issued (3).

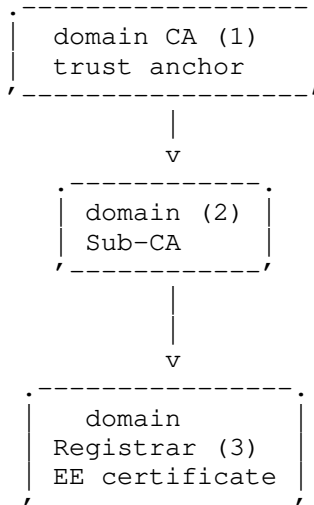


Figure 1: Two-Level CA PKI

When the Registrar is using a COSE-signed constrained voucher request towards MASA, instead of a regular CMS-signed voucher request, the COSE_Sign1 object contains a protected and an unprotected header. The Registrar MUST place all the certificates needed to validate the signature chain from the Registrar on the RVR in an "x5bag" attribute in the unprotected header [I-D.ietf-cose-x509].

The "x5bag" attribute is very important as it provides the required signals from the Registrar to control what identity is pinned in the resulting voucher. This is explained in the next section.

8.2. MASA Pinning Policy

The MASA, having assembled and verified the chain in the signing structure of the voucher request needs to select a certificate to pin. (For the case that only the Registrar's End-Entity certificate is included, only this certificate can be selected and this section does not apply.) The BRSKI policy for pinning by the MASA as described in Section 5.5.2 of [RFC8995] leaves much flexibility to the manufacturer.

The present document adds the following rules to the MASA pinning policy to reduce the network load:

1. for a voucher containing a nonce, it SHOULD select the most specific (lowest-level) CA certificate in the chain.
2. for a nonceless voucher, it SHOULD select the least-specific (highest-level) CA certificate in the chain that is allowed under the MASA's policy for this specific domain.

The rationale for 1. is that in case of a voucher with nonce, the voucher is valid only in scope of the present DTLS connection between Pledge and Registrar anyway, so there is no benefit to pin a higher-level CA. By pinning the most specific CA the constrained Pledge can validate its DTLS connection using less crypto operations. The rationale for pinning a CA instead of the Registrar's End-Entity certificate directly is based on the following benefit on constrained networks: the pinned certificate in the voucher can in common cases be re-used as a Domain CA trust anchor during the EST enrollment and during the operational phase that follows after EST enrollment, as explained in Section 6.6.1.

The rationale for 2. follows from the flexible BRSKI trust model for, and purpose of, nonceless vouchers (Sections 5.5.* and 7.4.1 of [RFC8995]).

Referring to Figure 1 of a domain with a two-level certification authority, the most specific CA ("Sub-CA") is the identity that is pinned by MASA in a nonced voucher. A Registrar that wished to have only the Registrar's End-Entity certificate pinned would omit the "domain CA" and "Sub-CA" certificates from the voucher-request.

In case of a nonceless voucher, depending on the trust level, the MASA pins the "Registrar" certificate (low trust in customer), or the "Sub-CA" certificate (in case of medium trust, implying that any Registrar of that sub-domain is acceptable), or even the "domain CA" certificate (in case of high trust in the customer, and possibly a pre-agreed need of the customer to obtain flexible long-lived vouchers).

8.3. Pinning of Raw Public Keys

Specifically for constrained use cases, the pinning of the raw public key (RPK) of the Registrar is also supported in the constrained voucher, instead of an X.509 certificate. If an RPK is pinned it MUST be the RPK of the Registrar.

When the Pledge is known by MASA to support RPK but not X.509 certificates, the voucher produced by the MASA pins the RPK of the Registrar in either the "pinned-domain-pubk" or "pinned-domain-pubk-sha256" field of a voucher. This is described in more detail in Section 9.2.3. A Pledge that does not support X.509 certificates cannot use EST to enroll; it has to use another method for enrollment without certificates and the Registrar has to support this method also. It is possible that the Pledge will not enroll, but instead only a network join operation will occur (See [RFC9031]). How the Pledge discovers this method and details of the enrollment method are out of scope of this document.

When the Pledge is known by MASA to support PKIX format certificates, the "pinned-domain-cert" field present in a voucher typically pins a domain certificate. That can be either the End-Entity certificate of the Registrar, or the certificate of a domain CA of the Registrar's domain as specified in Section 8.2. However, if the Pledge is known to also support RPK pinning and the MASA intends to identify the Registrar in the voucher (not the CA), then MASA MUST pin the RPK (RPK3 in Figure 2) of the Registrar instead of the Registrar's End-Entity certificate to save space in the voucher.

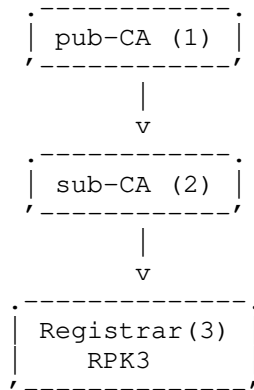


Figure 2: Raw Public Key pinning

8.4. Considerations for use of IDevID-Issuer

[RFC8366] and [RFC8995] defines the idevid-issuer attribute for voucher and voucher-request (respectively), but they summarily explain when to use it.

The use of idevid-issuer is provided so that the serial-number to which the issued voucher pertains can be relative to the entity that issued the devices' IDevID. In most cases there is a one to one relationship between the trust anchor that signs vouchers (and is trusted by the pledge), and the Certification Authority that signs the IDevID. In that case, the serial-number in the voucher must refer to the same device as the serial-number that is in IDevID certificate.

However, there situations where the one to one relationship may be broken. This occurs whenever a manufacturer has a common MASA, but different products (on different assembly lines) are produced with identical serial numbers. A system of serial numbers which is just a simple counter is a good example of this. A system of serial numbers where there is some prefix relating the product type does not fit into this, even if the lower digits are a counter.

It is not possible for the Pledge or the Registrar to know which situation applies. The question to be answered is whether or not to include the idevid-issuer in the PVR and the RVR. A second question arises as to what the format of the idevid-issuer contents are.

Analysis of the situation shows that the pledge never needs to include the idevid-issuer in its PVR, because the pledge's IDevID certificate is available to the Registrar, and the Authority Key Identifier is contained within that. The pledge therefore has no need to repeat this.

For the RVR, the Registrar has to examine the pledge's IDevID certificate to discover the serial number for the Registrar's Voucher Request (RVR). This is clear in Section 5.5 of [RFC8995]. That section also clarifies that the idevid-issuer is to be included in the RVR.

Concerning the Authority Key Identifier, [RFC8366] specifies that the entire object i.e. the extnValue OCTET STRING is to be included: comprising the AuthorityKeyIdentifier, SEQUENCE, Choice as well as the OCTET STRING that is the keyIdentifier.

9. Artifacts

This section describes for both the voucher request and the voucher first the abstract (tree) definition as explained in [RFC8340]. This provides a high-level view of the contents of each artifact.

Then the assigned SID values are presented. These have been assigned using the rules in [I-D.ietf-core-sid].

9.1. Voucher Request artifact

9.1.1. Tree Diagram

The following diagram is largely a duplicate of the contents of [RFC8366], with the addition of the fields proximity-registrar-pubk, proximity-registrar-pubk-sha256, proximity-registrar-cert, and prior-signed-voucher-request.

prior-signed-voucher-request is only used between the Registrar and the MASA. proximity-registrar-pubk or proximity-registrar-pubk-sha256 optionally replaces proximity-registrar-cert for the most constrained cases where RPK is used by the Pledge.

```
module: ietf-voucher-request-constrained
```

```
  grouping voucher-request-constrained-grouping
```

```
    +-- voucher
```

```
      +-- created-on?                yang:date-and-time
      +-- expires-on?               yang:date-and-time
      +-- assertion                  enumeration
      +-- serial-number              string
      +-- idevid-issuer?             binary
      +-- pinned-domain-cert?        binary
      +-- domain-cert-revocation-checks? boolean
      +-- nonce?                    binary
      +-- last-renewal-date?         yang:date-and-time
      +-- proximity-registrar-pubk?  binary
      +-- proximity-registrar-pubk-sha256? binary
      +-- proximity-registrar-cert?  binary
      +-- prior-signed-voucher-request? binary
```

9.1.2. SID values

```
  SID Assigned to
```

```
-----
2501 data /ietf-voucher-request-constrained:voucher
2502 data .../assertion
2503 data .../created-on
2504 data .../domain-cert-revocation-checks
2505 data .../expires-on
2506 data .../idevid-issuer
2507 data .../last-renewal-date
2508 data /ietf-voucher-request-constrained:voucher/nonce
2509 data .../pinned-domain-cert
2510 data .../prior-signed-voucher-request
2511 data .../proximity-registrar-cert
2513 data .../proximity-registrar-pubk
2512 data .../proximity-registrar-pubk-sha256
2514 data .../serial-number
```

WARNING, obsolete definitions

The "assertion" attribute is an enumerated type [RFC8366], and the current PYANG tooling does not document the valid values for this attribute. In the JSON serialization, the literal strings from the enumerated types are used so there is no ambiguity. In the CBOR serialization, a small integer is used. This following values are documented here, but the YANG module should be considered authoritative. No IANA registry is provided or necessary because the YANG module provides for extensions.

Integer	Assertion Type
0	verified
1	logged
2	proximity

Table 3: CBOR integers
for the "assertion"
attribute enum

9.1.3. YANG Module

In the voucher-request-constrained YANG module, the voucher is "augmented" within the "used" grouping statement such that one continuous set of SID values is generated for the voucher-request-constrained module name, all voucher attributes, and the voucher-request-constrained attributes. Two attributes of the voucher are "refined" to be optional.

```
<CODE BEGINS> file "ietf-voucher-request-constrained@2021-04-15.yang"
module ietf-voucher-request-constrained {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-voucher-request-constrained";
  prefix "constrained";

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  import ietf-voucher {
    prefix "v";
  }

  organization
    "IETF ANIMA Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/anima/>
```


WG List: <mailto:anima@ietf.org>
Author: Michael Richardson
<mailto:mcr+ietf@sandelman.ca>
Author: Peter van der Stok
<mailto:consultancy@vanderstok.org>
Author: Panos Kampanakis
<mailto:pkampana@cisco.com>;

description

"This module defines the format for a voucher request, which is produced by a pledge to request a voucher. The voucher-request is sent to the potential owner's Registrar, which in turn sends the voucher request to the manufacturer or its delegate (MASA).

A voucher is then returned to the pledge, binding the pledge to the owner. This is a constrained version of the voucher-request present in
{[I-D.ietf-anima-bootstrap-keyinfra]}

This version provides a very restricted subset appropriate for very constrained devices. In particular, it assumes that nonce-ful operation is always required, that expiration dates are rather weak, as no clocks can be assumed, and that the Registrar is identified by either a pinned Raw Public Key of the Registrar, or by a pinned X.509 certificate of the Registrar or domain CA.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119."

```
revision "2021-04-15" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: Voucher Profile for Constrained Devices";  
}
```

```
rc:yang-data voucher-request-constrained-artifact {  
  // YANG data template for a voucher.  
  uses voucher-request-constrained-grouping;  
}
```

```
// Grouping defined for future usage  
grouping voucher-request-constrained-grouping {  
  description
```

```
"Grouping to allow reuse/extensions in future work.";

uses v:voucher-artifact-grouping {

  refine voucher/created-on {
    mandatory false;
  }

  refine voucher/pinned-domain-cert {
    mandatory false;
  }

  augment "voucher" {
    description "Base the constrained voucher-request upon the
      regular one";

    leaf proximity-registrar-pubk {
      type binary;
      description
        "The proximity-registrar-pubk replaces
        the proximity-registrar-cert in constrained uses of
        the voucher-request.
        The proximity-registrar-pubk is the
        Raw Public Key of the Registrar. This field is encoded
        as specified in RFC7250, section 3.
        The ECDSA algorithm MUST be supported.
        The EdDSA algorithm as specified in
        draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
        Support for the DSA algorithm is not recommended.
        Support for the RSA algorithm is a MAY, but due to
        size is discouraged.";
    }

    leaf proximity-registrar-pubk-sha256 {
      type binary;
      description
        "The proximity-registrar-pubk-sha256
        is an alternative to both
        proximity-registrar-pubk and pinned-domain-cert.
        In many cases the public key of the domain has already
        been transmitted during the key agreement protocol,
        and it is wasteful to transmit the public key another
        two times.
        The use of a hash of public key info, at 32-bytes for
        sha256 is a significant savings compared to an RSA
        public key, but is only a minor savings compared to
        a 256-bit ECDSA public-key."
    }
  }
}
```

```
        Algorithm agility is provided by extensions to this
        specification which may define a new leaf for another
        hash type.";
    }

    leaf proximity-registrar-cert {
        type binary;
        description
            "An X.509 v3 certificate structure as specified by
            RFC 5280,
            Section 4 encoded using the ASN.1 distinguished encoding
            rules (DER), as specified in ITU-T X.690.

            The first certificate in the Registrar TLS server
            certificate_list sequence (see [RFC5246]) presented by
            the Registrar to the Pledge. This field or one of its
            alternatives MUST be populated in a
            Pledge's voucher request if the proximity assertion is
            populated.";
    }

    leaf prior-signed-voucher-request {
        type binary;
        description
            "If it is necessary to change a voucher, or re-sign and
            forward a voucher that was previously provided along a
            protocol path, then the previously signed voucher
            SHOULD be included in this field.

            For example, a pledge might sign a proximity voucher,
            which an intermediate registrar then re-signs to
            make its own proximity assertion. This is a simple
            mechanism for a chain of trusted parties to change a
            voucher, while maintaining the prior signature
            information.

            The pledge MUST ignore all prior voucher information
            when accepting a voucher for imprinting. Other
            parties MAY examine the prior signed voucher
            information for the purposes of policy decisions.
            For example, this information could be useful to a
            MASA to determine that both pledge and registrar
            agree on proximity assertions. The MASA SHOULD
            remove all prior-signed-voucher-request information when
            signing a voucher for imprinting so as to minimize the
            final voucher size.";
    }
}
```

```

    }
  }
}
<CODE ENDS>

```

9.1.4. Example voucher request artifact

Below is a CBOR serialization of an example constrained voucher request from a Pledge to a Registrar, shown in CBOR diagnostic notation. The enum value of the assertion field is calculated to be 2 by following the algorithm described in section 9.6.4.2 of [RFC7950].

```

{
  2501: {
    +2 : "2016-10-07T19:31:42Z", / SID=2503, created-on /
    +4 : "2016-10-21T19:31:42Z", / SID=2505, expires-on /
    +1 : 2, / SID=2502, assertion "proximity" /
    +13: "JADA123456789", / SID=2514, serial-number /
    +5 : h'08C2BF36....B3D2B3', / SID=2506, idevid-issuer /
    +10: h'30820275....82c35f', / SID=2511, proximity-registrar-cert/
    +3 : true, / SID=2504, domain-cert
    +6 : "2017-10-07T19:31:42Z" / SID=2507, last-renewal-date /
  }
}

```

9.2. Voucher artifact

The voucher's primary purpose is to securely assign a Pledge to an owner. The voucher informs the Pledge which entity it should consider to be its owner.

9.2.1. Tree Diagram

The following diagram is largely a duplicate of the contents of [RFC8366], with only the addition of the fields pinned-domain-pubk and pinned-domain-pubk-sha256.

module: ietf-voucher-constrained

grouping voucher-constrained-grouping

```

+-- voucher
  +-- created-on?          yang:date-and-time
  +-- expires-on?         yang:date-and-time
  +-- assertion            enumeration
  +-- serial-number        string
  +-- idevid-issuer?       binary
  +-- pinned-domain-cert?  binary
  +-- domain-cert-revocation-checks? boolean
  +-- nonce?              binary
  +-- last-renewal-date?   yang:date-and-time
  +-- pinned-domain-pubk?  binary
  +-- pinned-domain-pubk-sha256? binary

```

9.2.2. SID values

SID Assigned to

```

-----
2451 data /ietf-voucher-constrained:voucher
2452 data /ietf-voucher-constrained:voucher/assertion
2453 data /ietf-voucher-constrained:voucher/created-on
2454 data .../domain-cert-revocation-checks
2455 data /ietf-voucher-constrained:voucher/expires-on
2456 data /ietf-voucher-constrained:voucher/idevid-issuer
2457 data .../last-renewal-date
2458 data /ietf-voucher-constrained:voucher/nonce
2459 data .../pinned-domain-cert
2460 data .../pinned-domain-pubk
2461 data .../pinned-domain-pubk-sha256
2462 data /ietf-voucher-constrained:voucher/serial-number

```

WARNING, obsolete definitions

The "assertion" enumerated attribute is numbered as per Section 9.1.2.

9.2.3. YANG Module

In the voucher-constrained YANG module, the voucher is "augmented" within the "used" grouping statement such that one continuous set of SID values is generated for the voucher-constrained module name, all voucher attributes, and the voucher-constrained attributes. Two attributes of the voucher are "refined" to be optional.

```
<CODE BEGINS> file "ietf-voucher-constrained@2021-04-15.yang"
module ietf-voucher-constrained {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-voucher-constrained";
  prefix "constrained";

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  import ietf-voucher {
    prefix "v";
  }

  organization
    "IETF ANIMA Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/anima/>
    WG List:    <mailto:anima@ietf.org>
    Author:     Michael Richardson
                <mailto:mcr+ietf@sandelman.ca>
    Author:     Peter van der Stok
                <mailto:consultancy@vanderstok.org>
    Author:     Panos Kampanakis
                <mailto:pkampana@cisco.com>";

  description
    "This module defines the format for a voucher, which
     is produced by a pledge's manufacturer or its delegate
     (MASA) to securely assign one or more pledges to an 'owner',
     so that a pledge may establish a secure connection to the
     owner's network infrastructure.

     This version provides a very restricted subset appropriate
     for very constrained devices.
     In particular, it assumes that nonce-ful operation is
     always required, that expiration dates are rather weak, as no
     clocks can be assumed, and that the Registrar is identified
     by either a pinned Raw Public Key of the Registrar, or by a
     pinned X.509 certificate of the Registrar or domain CA."
```

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119.";

```
revision "2021-04-15" {
  description
    "Initial version";
  reference
    "RFC XXXX: Voucher Profile for Constrained Devices";
}

rc:yang-data voucher-constrained-artifact {
  // YANG data template for a voucher.
  uses voucher-constrained-grouping;
}

// Grouping defined for future usage
grouping voucher-constrained-grouping {
  description
    "Grouping to allow reuse/extensions in future work.";

  uses v:voucher-artifact-grouping {

    refine voucher/created-on {
      mandatory false;
    }

    refine voucher/pinned-domain-cert {
      mandatory false;
    }

    augment "voucher" {
      description "Base the constrained voucher
                  upon the regular one";
      leaf pinned-domain-pubk {
        type binary;
        description
          "The pinned-domain-pubk may replace the
           pinned-domain-cert in constrained uses of
           the voucher. The pinned-domain-pubk
           is the Raw Public Key of the Registrar.
           This field is encoded as a Subject Public Key Info block
           as specified in RFC7250, in section 3.
           The ECDSA algorithm MUST be supported.
           The EdDSA algorithm as specified in
           draft-ietf-tls-rfc4492bis-17 SHOULD be supported.
           Support for the DSA algorithm is not recommended."
```

```

    Support for the RSA algorithm is a MAY.";
}

leaf pinned-domain-pubk-sha256 {
    type binary;
    description
        "The pinned-domain-pubk-sha256 is a second
         alternative to pinned-domain-cert. In many cases the
         public key of the domain has already been transmitted
         during the key agreement process, and it is wasteful
         to transmit the public key another two times.
         The use of a hash of public key info, at 32-bytes for
         sha256 is a significant savings compared to an RSA
         public key, but is only a minor savings compared to
         a 256-bit ECDSA public-key.
         Algorithm agility is provided by extensions to this
         specification which can define a new leaf for another
         hash type.";
}
}
}
}
<CODE ENDS>
```

9.2.4. Example voucher artifacts

Below the CBOR serialization of an example constrained voucher is shown in CBOR diagnostic notation. The enum value of the assertion field is calculated to be zero by following the algorithm described in section 9.6.4.2 of [RFC7950].

```

{
  2451: {
    +2 : "2016-10-07T19:31:42Z", / SID = 2453, created-on /
    +4 : "2016-10-21T19:31:42Z", / SID = 2455, expires-on /
    +1 : 0, / SID = 2452, assertion "verified" /
    +11: "JADA123456789", / SID = 2462, serial-number /
    +5 : h'E40393B4....68A3', / SID = 2456, idevid-issuer /
    +8 : h'30820275....C35F', / SID = 2459, pinned-domain-cert/
    +3 : true, / SID = 2454, domain-cert /
    /
    -revocation-checks /
    +6 : "2017-10-07T19:31:42Z" / SID = 2457, last-renewal-date /
  }
}

```


9.3. Signing voucher and voucher-request artifacts with COSE

The COSE_Sign1 structure is discussed in Section 4.2 of [I-D.ietf-cose-rfc8152bis-struct]. The CBOR object that carries the body, the signature, and the information about the body and signature is called the COSE_Sign1 structure. It is used when only one signature is used on the body.

Support for ECDSA with SHA2-256 using curve secp256r1 (aka prime256k1) is RECOMMENDED. Most current low power hardware has support for acceleration of this algorithm. Future hardware designs could omit this in favour of a newer algorithms. This is the ES256 keytype from Table 1 of [I-D.ietf-cose-rfc8152bis-algs]. Support for curve secp256k1 is OPTIONAL.

Support for EdDSA using Curve 25519 is RECOMMENDED in new designs if hardware support is available. This is keytype EDDSA (-8) from Table 2 of [I-D.ietf-cose-rfc8152bis-algs]. A "crv" parameter is necessary to specify the Curve, which from Table 18. The 'kty' field MUST be present, and it MUST be 'OKP'. (Table 17)

A transition towards EdDSA is occurring in the industry. Some hardware can accelerate only some algorithms with specific curves, other hardware can accelerate any curve, and still other kinds of hardware provide a tool kit for acceleration of any elliptic curve algorithm.

In general, the Pledge is expected to support only a single algorithm, while the Registrar, usually not constrained, is expected to support a wide variety of algorithms: both legacy ones and up-and-coming ones via regular software updates.

An example of the supported COSE_Sign1 object structure is shown in Figure 3.

```
18( / COSE_Sign1 /
  [
    h'A101382E',          / protected header encoding: {1: -47}      /
    {                     /           which means { "alg": ES256K }    /
      4 : h'7890A03F1234' / 4 is the "kid" binary key identifier /
    },
    h'1234....5678', / voucher-request binary content (CBOR)      /
    h'4567....1234' / voucher-request binary public signature     /
  ]
)
```

Figure 3: COSE_Sign1 example in CBOR diagnostic notation

The [COSE-registry] specifies the integers/encoding for the "alg" and "kid" fields in Figure 3. The "alg" field restricts the key usage for verification of this COSE object to a particular cryptographic algorithm.

The "kid" field is optionally present: it is an unprotected field that identifies the public key of the key pair that was used to sign this message. The value of the key identifier "kid" parameter is an example value. Usually a hash of the public key is used to identify the public key, but a device serial number may also be used. The choice of key identifier method is vendor-specific. If "kid" is not present, then a verifying party needs to use other context information to retrieve the right public key to verify the COSE_Sign1 object against. For example, this context information may be a unique serial number encoded in the binary content (CBOR) field.

A Registrar MAY use a "kid" parameter in its RVR to identify its signing key as used to sign the RVR. The method of generating this "kid" is vendor-specific and SHOULD be configurable in the Registrar to support commonly used methods. In order to support future business cases and supply chain integrations, a Registrar MUST be configurable, on a per-manufacturer basis, to be able to configure the "kid" to a particular value. Both binary and string values are to be supported, each being inserted using a CBOR bstr or tstr. By default, a Registrar does not include a "kid" parameter in its RVR since the signing key is already identified by the included signing certificates in the COSE "x5bag" structure.

A Pledge normally SHOULD NOT use a "kid" parameter in its PVR, because its signing key is already identified by the Pledge's unique serial number that is included in the PVR. Still, where needed the Pledge MAY use a "kid" parameter in its PVR to help the MASA identify the right public key to verify against. This can occur for example if a Pledge has multiple IDevID identities. A Registrar normally SHOULD ignore a "kid" parameter used in a received PVR, as this information is intended for the MASA. In other words, there is no need for the Registrar to verify the contents of this field, but it may include it in an audit log.

In Appendix C a binary COSE_Sign1 object is shown based on the voucher-request example of Section 9.1.4.

10. Deployment-specific Discovery Considerations

This section details how discovery is done in specific deployment scenarios.

10.1. 6TSCH Deployments

In 6TISCH networks, the Constrained Join Proxy (CoJP) mechanism is described in [RFC9031]. Such networks are expected to use a [I-D.ietf-lake-edhoc] to do key management. This is the subject of future work. The Enhanced Beacon has been extended in [RFC9032] to allow for discovery of the Join Proxy.

10.2. Generic networks using GRASP

[RFC8995] defines a mechanism for the Pledge to discover a Join Proxy by listening for [RFC8990] GRASP messages. This mechanism can be used on any network which does not have another more specific mechanism. This mechanism supports mesh networks, and can also be used over unencrypted WIFI.

10.3. Generic networks using mDNS

[RFC8995] also defines a non-normative mechanism for the Pledge to discover a Join Proxy by doing mDNS queries. This mechanism can be used on any network which does not have another recommended mechanism. This mechanism does not easily support mesh networks. It can be used over unencrypted WIFI.

10.4. Thread networks using Mesh Link Establishment (MLE)

Thread [Thread] is a wireless mesh network protocol based on 6LoWPAN [RFC6282] and other IETF protocols. In Thread, a new device discovers potential Thread networks and Thread routers to join by using the Mesh Link Establishment (MLE) [I-D.ietf-6lo-mesh-link-establishment] protocol. MLE uses the UDP port number 19788. The new device sends discovery requests on different IEEE 802.15.4 radio channels, to which routers (if any present) respond with a discovery response containing information about their respective network. Once a suitable router is selected the new device initiates a DTLS transport-layer secured connection to the network's commissioning application, over a link-local single radio hop to the selected Thread router. This link is not yet secured at the radio level: link-layer security will be set up once the new device is approved by the commissioning application to join the Thread network, and it gets provisioned with network access credentials.

The Thread router acts here as a Join Proxy. The MLE discovery response message contains UDP port information to signal the new device which port to use for its DTLS connection.

10.5. Non-mesh networks using CoAP Discovery

On unencrypted constrained networks such as 802.15.4, CoAP discover may be done using the mechanism detailed in [I-D.ietf-ace-coap-est] section 5.1.

11. Design Considerations

The design considerations for the CBOR encoding of vouchers are much the same as for JSON vouchers in [RFC8366]. One key difference is that the names of the leaves in the YANG definition do not affect the size of the resulting CBOR, as the SID translation process assigns integers to the names.

Any POST request to the Registrar with resource `/vs` or `/es` returns a 2.04 Changed response with empty payload. The client should be aware that the server may use a piggybacked CoAP response (ACK, 2.04) but may also respond with a separate CoAP response, i.e. first an (ACK, 0.0) that is an acknowledgement of the request reception followed by a (CON, 2.04) response in a separate CoAP message.

12. Raw Public Key Use Considerations

This section explains techniques to reduce the number of bytes that are sent over the wire during the BRSKI bootstrap. The use of a raw public key (RPK) in the pinning process can significantly reduce the number of bytes and round trips, but it comes with a few significant operational limitations.

12.1. The Registrar Trust Anchor

When the Pledge first connects to the Registrar, the connection to the Registrar is provisional, as explained in Section 5.6.2 of [RFC8995]. The Registrar provides its public key in a `TLSServerCertificate`, and the Pledge uses that to validate that integrity of the (D)TLS connection, but it does not validate the identity of the provided certificate.

As the `TLSServerCertificate` object is never verified directly by the pledge, sending it can be considered superfluous. Instead of using a `(TLSServer)Certificate` of type X509 (see section 4.4.2 of [RFC8446]), a `RawPublicKey` object is used.

A Registrar operating in a mixed environment can determine whether to send a Certificate or a Raw Public key: this is determined by the pledge including the `server_certificate_type` of `RawPublicKey`. This is shown in section 5 of [RFC7250].

The Pledge continues to send a `client_certificate_type` of X509, so that the Registrar can properly identify the pledge and distill the MASA URI information from its certificate.

12.2. The Pledge Voucher Request

The Pledge puts the Registrar's public key into the `proximity-registrar-pubk` field of the `voucher-request`. (The `proximity-registrar-pubk-sha256` can also be used if the 32-bytes of a SHA256 hash turns out to be smaller than a typical ECDSA key.)

As the format of the `pubk` field is identical to the TLS Certificate `RawPublicKey`, no manipulation at all is needed to insert this into a `voucher-request`.

12.3. The Voucher Response

A returned voucher will have a `pinned-domain-subk` field with the identical key as was found in the `proximity-registrar-pubk` field above, as well as in the TLS connection.

Validation of this key by the pledge is what takes the DTLS connection out of the provisional state see Section 5.6.2 of [RFC8995].

The voucher needs to be validated first. The Pledge needs to have a public key to validate the signature from the MASA on the voucher.

In certain cases, the MASA's public key counterpart of the (private) signing key is already installed in the Pledge at manufacturing time. In other cases, if the MASA signing key is based upon a PKI (see [I-D.richardson-anima-masa-considerations] Section 2.3), then a certificate chain may need to be included with the voucher in order for the pledge to validate the signature. In CMS signed artifacts, the CMS structure has a place for such certificates.

In the COSE-signed Constrained Vouchers described in this document, the `x5bag` attribute from [I-D.ietf-cose-x509] is to be used for this.

13. Use of constrained vouchers with HTTPS

This specification contains two extensions to [RFC8995]: a constrained voucher format (COSE), and a constrained transfer protocol (CoAP).

On constrained networks with constrained devices, it make senses to use both together. However, this document does not mandate that this be the only way.

A given constrained device design and software may be re-used for multiple device models, such as a model having only an IEEE 802.15.4 radio, or a model having only an IEEE 802.11 (Wi-Fi) radio, or a model having both these radios. A manufacturer of such device models may wish to have code only for the use of the constrained voucher format (COSE), and use it on all supported radios including the IEEE 802.11 radio. For this radio, the software stack to support HTTP/TLS may be already integrated into the radio module hence it is attractive for the manufacturer to reuse this. This type of approach is supported by this document. In the case that HTTPS is used, the normal [RFC8995] resource names are used, together with the media types described in this document.

Other combinations are possible, but they are not enumerated here. New work such as [I-D.ietf-anima-jws-voucher] provides new formats that may be useable over a number of different transports. In general, sending larger payloads over constrained networks makes less sense, while sending smaller payloads over unconstrained networks is perfectly acceptable.

The Pledge will in most cases support a single voucher format, which it uses without negotiation i.e. without discovery of formats supported. The Registrar, being unconstrained, is expected to support all voucher formats. There will be cases where a Registrar does not support a new format that a new Pledge uses, and this is an unfortunate situation that will result in lack of interoperation.

The responsibility for supporting new formats is on the Registrar.

14. Security Considerations

14.1. Duplicate serial-numbers

In the absense of correct use of idevid-issuer by the Registrar as detailed in Section 8.4, it would be possible for a malicious Registrar to use an unauthorized voucher for a device. This would apply only to the case where a Manufacturer Authorized Signing Authority (MASA) is trusted by different products from the same manufacturer, and the manufacturer has duplicated serial numbers as a result of a merge, acquisition or mis-management.

For example, imagine the same manufacturer makes light bulbs as well as gas centrifuges, and said manufacturer does not uniquely allocate product serial numbers. This attack only works for nonceless vouchers. The attacker has obtained a light bulb which happens to have the same serial-number as a gas centrifuge which it wishes to obtain access. The attacker performs a normal BRSKI onboarding for the light bulb, but then uses the resulting voucher to onboard the

gas centrifuge. The attack requires that the gas centrifuge be returned to a state where it is willing to perform a new onboarding operation.

This attack is prevented by the mechanism of having the Registrar include the idevid-issuer in the RVR, and the MASA including it in the resulting voucher. The idevid-issuer is not included by default: a MASA needs to be aware if there are parts of the organization which duplicates serial numbers, and if so, include it.

14.2. IDevID security in Pledge

The security of this protocol depends upon the Pledge identifying itself to the Registrar using its manufacturer installed certificate: the IDevID certificate. Associated with this certificate is the IDevID private key, known only to the Pledge. Disclosure of this private key to an attacker would permit the attacker to impersonate the Pledge towards the Registrar, probably gaining access credentials to that Registrar's network.

If the IDevID private key disclosure is known to the manufacturer, there is little recourse other than recall of the relevant part numbers. The process for communicating this recall would be within the BRSKI-MASA protocol. Neither this specification nor [RFC8995] provides for consultation of a Certification Revocation List (CRL) or Open Certificate Status Protocol (OCSP) by a Registrar when evaluating an IDevID certificate. However, the BRSKI-MASA protocol submits the IDevID from the Registrar to the manufacturer's MASA and a manufacturer would have an opportunity to decline to issue a voucher for a device which they believe has become compromised.

It may be difficult for a manufacturer to determine when an IDevID private key has been disclosed. Two situations present themselves: in the first situation a compromised private key might be reused in a counterfeit device, which is sold to another customer. This would present itself as an onboarding of the same device in two different networks. The manufacturer may become suspicious seeing two voucher requests for the same device from different Registrars. Such activity could be indistinguishable from a device which has been resold from one operator to another, or re-deployed by an operator from one location to another.

In the second situation, an attacker having compromised the IDevID private key of a device might then install malware into the same device and attempt to return it to service. The device, now blank, would go through a second onboarding process with the original Registrar. Such a Registrar could notice that the device has been "factory reset" and alert the operator to this situation. One remedy

against the presence of malware is through the use of Remote Attestation such as described in [I-D.ietf-rats-architecture]. Future work will need to specify a background-check Attestation flow as part of the voucher-request/voucher-response process. Attestation may still require access to a private key (e.g. IDevID private key) in order to sign Evidence, so a primary goal should be to keep any private key safe within the Pledge.

In larger, more expensive, systems there is budget (power, space, and bill of materials) to include more specific defenses for a private key. For instance, this includes putting the IDevID private key in a Trusted Platform Module (TPM), or use of Trusted Execution Environments (TEE) for access to the key. On smaller IoT devices, the cost and power budget for an extra part is often prohibitive.

It is becoming more and more common for CPUs to have an internal set of one-time fuses that can be programmed (often they are "burnt" by a laser) at the factory. This section of memory is only accessible in some privileged CPU state. The use of this kind of CPU is appropriate as it provides significant resistance against key disclosure even when the device can be disassembled by an attacker.

In a number of industry verticals, there is increasing concern about counterfeit parts. These may be look-alike parts created in a different factory, or parts which are created in the same factory during an illegal night-shift, but which are not subject to the appropriate level of quality control. The use of a manufacturer-signed IDevID certificate provides for discovery of the pedigree of each part, and this often justifies the cost of the security measures associated with storing the private key.

14.3. Security of CoAP and UDP protocols

Section 7.1 explains that no CoAPS version of the BRSKI-MASA protocol is proposed. The connection from the Registrar to the MASA continues to be HTTPS as in [RFC8995]. This has been done to simplify the MASA deployment for the manufacturer, because no new protocol needs to be enabled on the server.

The use of UDP protocols across the open Internet is sometimes fraught with security challenges. Denial-of-service attacks against UDP based protocols are trivial as there is no three-way handshake as done for TCP. The three-way handshake of TCP guarantees that the node sending the connection request is reachable using the origin IP address. While DTLS contains an option to do a stateless challenge -- a process actually stronger than that done by TCP -- it is not yet common for this mechanism to be available in hardware at multigigabit speeds. It is for this reason that this document defines using HTTPS for the Registrar to MASA connection.

14.4. Registrar Certificate may be self-signed

The provisional (D)TLS connection formed by the Pledge with the Registrar does not authenticate the Registrar's identity. This Registrar's identity is validated by the [RFC8366] voucher that is issued by the MASA, signed with an anchor that was built-in to the Pledge.

The Registrar may therefore use any certificate, including a self-signed one. The only restrictions on the certificate is that it MUST have EKU bits set as detailed in Section 7.3.1.

14.5. Use of RPK alternatives to proximity-registrar-cert

In Section 9.1 two compact alternative fields for proximity-registrar-cert are defined that include an RPK: proximity-registrar-pubk and proximity-registrar-pubk-sha256. The Pledge can use these fields in its PVR to identify the Registrar based on its public key only. Since the full certificate of the proximate Registrar is not included, use of these fields by a Pledge implies that a Registrar could insert another certificate with the same public key identity into the RVR. For example, an older or a newer version of its certificate. The MASA will not be able to detect such act by the Registrar. But since any 'other' certificate the Registrar could insert in this way still encodes its identity the additional risk of using the RPK alternatives is negligible.

When a Registrar sees a PVR that uses one of proximity-registrar-pubk or proximity-registrar-pubk-sha256 fields, this implies the Registrar must include the certificate identified by these fields into its RVR. Otherwise, the MASA is unable to verify proximity. This requirement is already implied by the "MUST" requirement in Section 8.1.

14.6. MASA support of CoAPS

The use of CoAP for the BRSKI-MASA connection is not in scope of the current document. The following security considerations have led to this choice of scope:

- * the technology and experience to build secure Internet-scale HTTPS responders (which the MASA is) is common, while the experience in doing the same for CoAP is much less common.
- * in many enterprise networks, outgoing UDP connections are often treated as suspicious, which could effectively block CoAP connections for some firewall configurations.
- * reducing the complexity of MASA (i.e. less protocols supported) would also reduce its potential attack surface, which is relevant since the MASA is 24/7 exposed on the Internet and accepting (untrusted) incoming connections.

15. IANA Considerations

15.1. Resource Type Registry

Additions to the sub-registry "Resource Type Link Target Attribute Values", within the "CoRE Parameters" IANA registry are specified below.

Reference: [This RFC]

Attribute	Description
brski	Root path of Bootstrapping Remote Secure Key Infrastructure (BRSKI) resources
brski.rv	BRSKI request voucher resource
brski.vs	BRSKI voucher status telemetry resource
brski.es	BRSKI enrollment status telemetry resource

Table 4: Resource Type (rt) link target attribute values for IANA registration

15.2. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-voucher-constrained
 Registrant Contact: The ANIMA WG of the IETF.
 XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-voucher-request-constrained
 Registrant Contact: The ANIMA WG of the IETF.
 XML: N/A, the requested URI is an XML namespace.

15.3. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registration is requested:

name: ietf-voucher-constrained
 namespace: urn:ietf:params:xml:ns:yang:ietf-voucher-constrained
 prefix: vch
 reference: RFC XXXX

name: ietf-voucher-request-constrained
 namespace: urn:ietf:params:xml:ns:yang:ietf-voucher-request-constrained
 prefix: vch
 reference: RFC XXXX

15.4. The RFC SID range assignment sub-registry

Entry-point	Size	Module name	RFC Number
2450	50	ietf-voucher-constrained	[This RFC]
2500	50	ietf-voucher-request -constrained	[This RFC]

Warning: These SID values are defined in [I-D.ietf-core-sid], not as an Early Allocation.

IANA: please update the names in the Registry to match these revised names, if they have not already been revised.

15.5. Media Types Registry

This section registers the 'application/voucher-cose+cbor' in the IANA "Media Types" registry. This media type is used to indicate that the content is a CBOR voucher or voucher request signed with a COSE_Sign1 structure [I-D.ietf-cose-rfc8152bis-struct].

15.5.1. application/voucher-cose+cbor

Type name: application
Subtype name: voucher-cose+cbor
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: binary (CBOR)
Security considerations: Security Considerations of [This RFC].
Interoperability considerations: The format is designed to be broadly interoperable.
Published specification: [This RFC]
Applications that use this media type: ANIMA, 6tisch, and other zero-touch onboarding systems
Fragment identifier considerations: The syntax and semantics of fragment identifiers specified for application/voucher-cose+cbor are as specified for application/cbor. (At publication of this document, there is no fragment identification syntax defined for application/cbor.)
Additional information:
 Deprecated alias names for this type: N/A
 Magic number(s): N/A
 File extension(s): .vch
 Macintosh file type code(s): N/A
Person & email address to contact for further information: IETF ANIMA Working Group (anima@ietf.org) or IETF Operations and Management Area Working Group (opsawg@ietf.org)
Intended usage: COMMON
Restrictions on usage: N/A
Author: ANIMA WG
Change controller: IETF
Provisional registration? (standards tree only): NO

15.6. CoAP Content-Format Registry

One addition to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry is needed for a new content-format. It can be registered in the Expert Review range (0-255) or the IETF Review range (256-9999).

Media type	Encoding	ID	Reference
application/voucher-cose+cbor	-	TBD3	[This RFC]

15.7. Update to BRSKI Parameters Registry

This section updates the BRSKI Well-Known URIs sub-registry of the IANA Bootstrapping Remote Secure Key Infrastructures (BRSKI) Parameters Registry by adding a new column "Short URI". The contents of this field MUST be specified for any newly registered URI as follows:

Short URI: A short name for the "URI" resource that can be used by a Constrained BRSKI Pledge in a CoAP request to the Registrar. In case the "URI" resource is only used between Registrar and MASA, the value "--" is registered denoting that a short name is not applicable.

The initial contents of the sub-registry including the new column are as follows:

URI	Short URI	Description	Reference
requestvoucher	rv	Request voucher: Pledge to Registrar, and Registrar to MASA	[RFC8995], [This RFC]
voucher_status	vs	Voucher status telemetry: Pledge to Registrar	[RFC8995], [This RFC]
requestauditlog	--	Request audit log: Registrar to MASA	[RFC8995]
enrollstatus	es	Enrollment status telemetry: Pledge to Registrar	[RFC8995], [This RFC]

Table 5: Update of the BRSKI Well-Known URI Sub-Registry

16. Acknowledgements

We are very grateful to Jim Schaad for explaining COSE and CMS choices. Also thanks to Jim Schaad for correcting earlier versions of the COSE_Sign1 objects.

Michel Veillette did extensive work on `_pyang_` to extend it to support the SID allocation process, and this document was among its first users.

Daniel Franke and Henk Birkholtz provided review feedback.

The BRSKI design team has met on many Thursdays for document review. It includes: Aurelio Schellenbaum, David von Oheimb Steffen Fries, Thomas Werner, Toerless Eckert,

17. Changelog

-11 to -16 (For change details see GitHub issues <https://github.com/anima-wg/constrained-voucher/issues>)

-10 Design considerations extended Examples made consistent

-08 Examples for cose_sign1 are completed and improved.

-06 New SID values assigned; regenerated examples

-04 voucher and request-voucher MUST be signed examples for signed request are added in appendix IANA SID registration is updated SID values in examples are aligned signed cms examples aligned with new SIDs

-03

Examples are inverted.

-02

Example of requestvoucher with unsigned application/cbor is added attributes of voucher "refined" to optional CBOR serialization of vouchers improved Discovery port numbers are specified

-01

application/json is optional, application/cbor is compulsory Cms and cose mediatypes are introduced

18. References

18.1. Normative References

[I-D.ietf-ace-coap-est]

Stok, P. V. D., Kampanakis, P., Richardson, M. C., and S. Raza, "EST over secure CoAP (EST-coaps)", Work in Progress, Internet-Draft, draft-ietf-ace-coap-est-18, 6 January 2020, <<https://www.ietf.org/archive/id/draft-ietf-ace-coap-est-18.txt>>.

[I-D.ietf-core-sid]

Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-18, 18 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-sid-18.txt>>.

[I-D.ietf-core-yang-cbor]

Veillette, M., Petrov, I., Pelov, A., Bormann, C., and M. Richardson, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-19, 20 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-yang-cbor-19.txt>>.

[I-D.ietf-cose-rfc8152bis-algs]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[I-D.ietf-cose-x509]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-08, 14 December 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-x509-08.txt>>.

- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-43.txt>>.
- [ieee802-1AR]
IEEE Standard, "IEEE 802.1AR Secure Device Identifier", 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <<https://www.rfc-editor.org/info/rfc4193>>.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.

- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [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/info/rfc8610>>.
- [RFC8949] 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/info/rfc8949>>.

- [RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.
- [RFC9031] Vuini, M., Ed., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", RFC 9031, DOI 10.17487/RFC9031, May 2021, <<https://www.rfc-editor.org/info/rfc9031>>.
- [RFC9032] Dujovne, D., Ed. and M. Richardson, "Encapsulation of 6TiSCH Join and Enrollment Information Elements", RFC 9032, DOI 10.17487/RFC9032, May 2021, <<https://www.rfc-editor.org/info/rfc9032>>.

18.2. Informative References

- [COSE-registry] IANA, "CBOR Object Signing and Encryption (COSE) registry", 2017, <<https://www.iana.org/assignments/cose/cose.xhtml>>.
- [I-D.ietf-6lo-mesh-link-establishment] Kelsey, R., "Mesh Link Establishment", Work in Progress, Internet-Draft, draft-ietf-6lo-mesh-link-establishment-00, 1 December 2015, <<https://www.ietf.org/archive/id/draft-ietf-6lo-mesh-link-establishment-00.txt>>.
- [I-D.ietf-anima-constrained-join-proxy] Richardson, M., Stok, P. V. D., and P. Kampanakis, "Constrained Join Proxy for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-join-proxy-09, 25 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-anima-constrained-join-proxy-09.txt>>.
- [I-D.ietf-anima-jws-voucher] Richardson, M. and T. Werner, "JWS signed Voucher Artifacts for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-jws-voucher-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-anima-jws-voucher-03.txt>>.

- [I-D.ietf-lake-edhoc]
Selandier, G., Mattsson, J. P., and F. Palombini,
"Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in
Progress, Internet-Draft, draft-ietf-lake-edhoc-12, 20
October 2021, <<https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-12.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-
15, 8 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-15.txt>>.
- [I-D.kuehlewind-update-tag]
Kuehlewind, M. and S. Krishnan, "Definition of new tags
for relations between RFCs", Work in Progress, Internet-
Draft, draft-kuehlewind-update-tag-04, 12 July 2021,
<<https://www.ietf.org/archive/id/draft-kuehlewind-update-tag-04.txt>>.
- [I-D.richardson-anima-masa-considerations]
Richardson, M. and W. Pan, "Operational Considerations for
Voucher infrastructure for BRSKI MASA", Work in Progress,
Internet-Draft, draft-richardson-anima-masa-
considerations-06, 13 November 2021,
<<https://www.ietf.org/archive/id/draft-richardson-anima-masa-considerations-06.txt>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet
Control Message Protocol (ICMPv6) for the Internet
Protocol Version 6 (IPv6) Specification", STD 89,
RFC 4443, DOI 10.17487/RFC4443, March 2006,
<<https://www.rfc-editor.org/info/rfc4443>>.
- [RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6
Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
DOI 10.17487/RFC6282, September 2011,
<<https://www.rfc-editor.org/info/rfc6282>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
Format", RFC 6690, DOI 10.17487/RFC6690, August 2012,
<<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<https://www.rfc-editor.org/info/rfc7030>>.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <<https://www.rfc-editor.org/info/rfc8990>>.
- [Thread] Thread Group, Inc, "Thread support page, White Papers", November 2021, <<https://www.threadgroup.org/support#Whitepapers>>.

Appendix A. Library Support for BRSKI

For the implementation of BRSKI, the use of a software library to manipulate certificates and use crypto algorithms is often beneficial. Two C-based examples are OpenSSL and mbedtls. Others more targeted to specific platforms or languages exist. It is important to realize that the library interfaces differ significantly between libraries.

Libraries do not support all known crypto algorithms. Before deciding on a library, it is important to look at their supported crypto algorithms and the roadmap for future support. Apart from availability, the library footprint, and the required execution cycles should be investigated beforehand.

The handling of certificates usually includes the checking of a certificate chain. In some libraries, chains are constructed and verified on the basis of a set of certificates, the trust anchor (usually self signed root CA), and the target certificate. In other libraries, the chain must be constructed beforehand and obey order criteria. Verification always includes the checking of the signatures. Less frequent is the checking the validity of the dates or checking the existence of a revoked certificate in the chain against a set of revoked certificates. Checking the chain on the consistency of the certificate extensions which specify the use of the certificate usually needs to be programmed explicitly.

A library can be used to construct a (D)TLS connection. It is useful to realize that differences between (D)TLS implementations will occur due to the differences in the certificate checks supported by the

library. On top of that, checks between client and server certificates enforced by (D)TLS are not always helpful for a BRSKI implementation. For example, the certificates of Pledge and Registrar are usually not related when the BRSKI protocol is started. It must be verified that checks on the relation between client and server certificates do not hamper a successful DTLS connection establishment.

A.1. OpensSSL

From openssl's apps/verify.c :

```
<CODE BEGINS>
X509 *x = NULL;
int i = 0, ret = 0;
X509_STORE_CTX *csc;
STACK_OF(X509) *chain = NULL;
int num_untrusted;

x = load_cert(file, "certificate file");
if (x == NULL)
    goto end;

csc = X509_STORE_CTX_new();
if (csc == NULL) {
    BIO_printf(bio_err, "error %s: X.509 store context"
               "allocation failed\n",
               (file == NULL) ? "stdin" : file);
    goto end;
}

X509_STORE_set_flags(ctx, vflags);
if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
    X509_STORE_CTX_free(csc);
    BIO_printf(bio_err,
               "error %s: X.509 store context"
               "initialization failed\n",
               (file == NULL) ? "stdin" : file);
    goto end;
}
if (tchain != NULL)
    X509_STORE_CTX_set0_trusted_stack(csc, tchain);
if (crls != NULL)
    X509_STORE_CTX_set0_crls(csc, crls);

i = X509_verify_cert(csc);
X509_STORE_CTX_free(csc);

<CODE ENDS>
```

A.2. mbedTLS

```
<CODE BEGINS>
mbedtls_x509_crt cert;
mbedtls_x509_crt caCert;
uint32_t          certVerifyResultFlags;
...
int result = mbedtls_x509_crt_verify(&cert, &caCert, NULL, NULL,
                                     &certVerifyResultFlags, NULL, NULL);
<CODE ENDS>
```

Appendix B. Constrained BRSKI-EST Message Examples

This appendix extends the message examples from Appendix A of [I-D.ietf-ace-coap-est] with constrained BRSKI messages. The CoAP headers are only fully worked out for the first example, enrollstatus.

B.1. enrollstatus

A coaps enrollstatus message from Pledge to Registrar can be as follows:

```
POST coaps://192.0.2.1:8085/b/es
Content-Format: 60
Payload: <binary CBOR enrollstatus document>
```

The corresponding CoAP header fields are shown below.

```
Ver = 1
T = 0 (CON)
TKL = 1
Code = 0x02 (0.02 is POST method)
Message ID = 0xab0f
Token = 0x4d
Options
  Option (Uri-Path)
    Option Delta = 0xb (option nr = 11)
    Option Length = 0x1
    Option Value = "b"
  Option (Uri-Path)
    Option Delta = 0x0 (option nr = 11)
    Option Length = 0x2
    Option Value = "es"
  Option (Content-Format)
    Option Delta = 0x1 (option nr = 12)
    Option Length = 0x1
    Option Value = 60 (application/cbor)
Payload Marker = 0xFF
Payload = A26776657273696F6E0166737461747573F5 (18 bytes binary)
```

The Uri-Host and Uri-Port Options are omitted because they coincide with the transport protocol (UDP) destination address and port respectively.

The above binary CBOR enrollstatus payload looks as follows in CBOR diagnostic notation, for the case of enrollment success:

```
{
  "version": 1,
  "status": true
}
```

Alternatively the payload could look as follows in case of enrollment failure, using the reason field to describe the failure:

```
Payload = A36776657273696F6E0166737461747573F466726561736F6E782A3C
          496E666F726D61746976652068756D616E207265616461626C652065
          72726F72206D6573736167653E
```

```
{
  "version": 1,
  "status": false,
  "reason": "<Informative human readable error message>"
}
```

To indicate successful reception of the enrollmentstatus telemetry report, a response from the Registrar may then be:

2.04 Changed

With CoAP fields:

```
Ver=1
T=2 (ACK)
TKL=1
Code = 0x44 (2.04 Changed)
Message ID = 0xab0f
Token = 0x4d
```

B.2. voucher_status

A coaps voucher_status message from Pledge to Registrar can be as follows:

```
POST coaps://[2001:db8::2:1]/.well-known/brski/vs
Content-Format: 60 (application/cbor)
Payload:
a46776657273696f6e0166737461747573f466726561736f6e7828496e66
6f726d61746976652068756d616e2d7265616461626c65206572726f7220
6d6573736167656e726561736f6e2d636f6e74657874a100764164646974
696f6e616c20696e666f726d6174696f6e
```

The request payload above is binary CBOR but represented here in hexadecimal for readability. Below is the equivalent CBOR diagnostic format.


```
{
  "version": 1,
  "status": false,
  "reason": "Informative human-readable error message",
  "reason-context": { 0: "Additional information" }
}
```

A success response without payload will then be sent by the Registrar back to the Pledge to indicate reception of the telemetry report:

2.04 Changed

Appendix C. COSE-signed Voucher (Request) Examples

This appendix provides examples of COSE-signed voucher requests and vouchers. First, the used test keys and certificates are described, following by examples of a constrained PVR, RVR and voucher.

C.1. Pledge, Registrar and MASA Keys

This section documents the public and private keys used for all examples in this appendix. These keys are not used in any production system, and must only be used for testing purposes.

C.1.1. Pledge IDevID private key

```
<CODE BEGINS>
Private-Key: (256 bit)
priv:
  9b:4d:43:b6:a9:e1:7c:04:93:45:c3:13:d9:b5:f0:
  41:a9:6a:9c:45:79:73:b8:62:f1:77:03:3a:fc:c2:
  9c:9a
pub:
  04:d6:b7:6f:74:88:bd:80:ae:5f:28:41:2c:72:02:
  ef:5f:98:b4:81:e1:d9:10:4c:f8:1b:66:d4:3e:5c:
  ea:da:73:e6:a8:38:a9:f1:35:11:85:b6:cd:e2:04:
  10:be:fe:d5:0b:3b:14:69:2e:e1:b0:6a:bc:55:40:
  60:eb:95:5c:54
ASN1 OID: prime256v1
NIST CURVE: P-256
<CODE ENDS>
```

C.1.2. Registrar private key

```
<CODE BEGINS>
Private-Key: (256 bit)
priv:
  81:df:bb:50:a3:45:58:06:b5:56:3b:46:de:f3:e9:
  e9:00:ae:98:13:9e:2f:36:68:81:fc:d9:65:24:fb:
  21:7e
pub:
  04:50:7a:c8:49:1a:8c:69:c7:b5:c3:1d:03:09:ed:
  35:ba:13:f5:88:4c:e6:2b:88:cf:30:18:15:4f:a0:
  59:b0:20:ec:6b:eb:b9:4e:02:b8:93:40:21:89:8d:
  a7:89:c7:11:ce:a7:13:39:f5:0e:34:8e:df:0d:92:
  3e:d0:2d:c7:b7
ASN1 OID: prime256v1
NIST CURVE: P-256
<CODE ENDS>
```

C.1.3. MASA private key

```
<CODE BEGINS>
Private-Key: (256 bit)
priv:
  c6:bb:a5:8f:b6:d3:c4:75:28:d8:d3:d9:46:c3:31:
  83:6d:00:0a:9a:38:ce:22:5c:e9:d9:ea:3b:98:32:
  ec:31
pub:
  04:59:80:94:66:14:94:20:30:3c:66:08:85:55:86:
  db:e7:d4:d1:d7:7a:d2:a3:1a:0c:73:6b:01:0d:02:
  12:15:d6:1f:f3:6e:c8:d4:84:60:43:3b:21:c5:83:
  80:1e:fc:e2:37:85:77:97:94:d4:aa:34:b5:b6:c6:
  ed:f3:17:5c:f1
ASN1 OID: prime256v1
NIST CURVE: P-256
<CODE ENDS>
```

C.2. Pledge, Registrar and MASA Certificates

All keys and certificates used for the examples have been generated with OpenSSL – see Appendix D for more details on certificate generation. Below the certificates are listed that accompany the keys shown above. Each certificate description is followed by the hexadecimal representation of the X.509 ASN.1 DER encoded certificate. This representation can be for example decoded using an online ASN.1 decoder.

C.2.1. Pledge IDevID Certificate

<CODE BEGINS>

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 4822678189204992 (0x11223344556600)

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=manufacturer,
CN=masa.stok.nl

Validity

Not Before: Dec 9 10:02:36 2020 GMT

Not After : Dec 31 23:59:59 9999 GMT

Subject: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=manufacturing,
CN=uuid:pledge.1.2.3.4/serialNumber=pledge.1.2.3.4

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:d6:b7:6f:74:88:bd:80:ae:5f:28:41:2c:72:02:

ef:5f:98:b4:81:e1:d9:10:4c:f8:1b:66:d4:3e:5c:

ea:da:73:e6:a8:38:a9:f1:35:11:85:b6:cd:e2:04:

10:be:fe:d5:0b:3b:14:69:2e:e1:b0:6a:bc:55:40:

60:eb:95:5c:54

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

X509v3 Authority Key Identifier:

keyid:

E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3

Signature Algorithm: ecdsa-with-SHA256

30:46:02:21:00:d2:e6:45:3b:b0:c3:00:b3:25:8d:f1:83:fe:

d9:37:c1:a2:49:65:69:7f:6b:b9:ef:2c:05:07:06:31:ac:17:

bd:02:21:00:e2:ce:9e:7b:7f:74:50:33:ad:9e:ff:12:4e:e9:

a6:f3:b8:36:65:ab:7d:80:bb:56:88:bc:03:1d:e5:1e:31:6f

<CODE ENDS>

Below is the hexadecimal representation:

<CODE BEGINS>

```
30820226308201cba003020102020711223344556600300a06082a8648ce3d04
0302306f310b3009060355040613024e4c310b300906035504080c024e423110
300e06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e6465
7273746f6b31153013060355040b0c0c6d616e756666163747572657231153013
06035504030c0c6d6173612e73746f6b2e6e6c3020170d323031323039313030
3233365a180f393939393132333313233353935395a308190310b300906035504
0613024e4c310b300906035504080c024e423110300e06035504070c0748656c
6d6f6e6431133011060355040a0c0a76616e64657273746f6b31163014060355
040b0c0d6d616e756666163747572696e67311c301a06035504030c1375756964
3a706c656467652e312e322e332e34311730150603550405130e706c65646765
2e312e322e332e343059301306072a8648ce3d020106082a8648ce3d03010703
420004d6b76f7488bd80ae5f28412c7202ef5f98b481e1d9104cf81b66d43e5c
eada73e6a838a9f1351185b6cde20410befed50b3b14692ee1b06abc554060eb
955c54a32e302c30090603551d1304023000301f0603551d23041830168014e4
0393b4c3d3f42a80a47718f6964903011768a3300a06082a8648ce3d04030203
49003046022100d2e6453bb0c300b3258df183fed937c1a24965697f6bb9ef2c
05070631ac17bd022100e2ce9e7b7f745033ad9eff124ee9a6f3b83665ab7d80
bb5688bc031de51e316f
```

<CODE ENDS>

C.2.2. Registrar Certificate

<CODE BEGINS>

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

70:56:ea:aa:30:66:d8:82:6a:55:5b:90:88:d4:62:bf:9c:f2:8c:fd

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=consultancy,
CN=registrar.stok.nl

Validity

Not Before: Dec 9 10:02:36 2020 GMT

Not After : Dec 9 10:02:36 2021 GMT

Subject: C=NL, ST=NB, L=Helmond, O=vanderstok, OU=consultancy,
CN=registrar.stok.nl

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:50:7a:c8:49:1a:8c:69:c7:b5:c3:1d:03:09:ed:

35:ba:13:f5:88:4c:e6:2b:88:cf:30:18:15:4f:a0:

59:b0:20:ec:6b:eb:b9:4e:02:b8:93:40:21:89:8d:

a7:89:c7:11:ce:a7:13:39:f5:0e:34:8e:df:0d:92:

3e:d0:2d:c7:b7

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Subject Key Identifier:

08:C2:BF:36:88:7F:79:41:21:85:87:2F:16:A7:AC:A6:EF:B3:D2:B3

X509v3 Authority Key Identifier:

keyid:

08:C2:BF:36:88:7F:79:41:21:85:87:2F:16:A7:AC:A6:EF:B3:D2:B3

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Extended Key Usage:

CMC Registration Authority, TLS Web Server

Authentication, TLS Web Client Authentication

X509v3 Key Usage: critical

Digital Signature, Non Repudiation, Key Encipherment,

Data Encipherment, Certificate Sign, CRL Sign

Signature Algorithm: ecdsa-with-SHA256

30:44:02:20:74:4c:99:00:85:13:b2:f1:bc:fd:f9:02:1a:46:

fb:17:4c:f8:83:a2:7c:a1:d9:3f:ae:ac:f3:1e:4e:dd:12:c6:

02:20:11:47:14:db:f5:1a:5e:78:f5:81:b9:42:1c:6e:47:02:

ab:53:72:70:c5:ba:fb:2d:16:c3:de:9a:a1:82:c3:5f

<CODE ENDS>

Below is the hexadecimal representation which is in (request-)voucher examples referred to as regis-cert-hex:

```
<CODE BEGINS>
308202753082021ca00302010202147056eaaa3066d8826a555b9088d462bf9c
f28cfd300a06082a8648ce3d0403023073310b3009060355040613024e4c310b
300906035504080c024e423110300e06035504070c0748656c6d6f6e64311330
11060355040a0c0a76616e64657273746f6b31143012060355040b0c0b636f6e
73756c74616e6379311a301806035504030c117265676973747261722e73746f
6b2e6e6c301e170d3230313230393130303233365a170d323131323039313030
3233365a3073310b3009060355040613024e4c310b300906035504080c024e42
3110300e06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e
64657273746f6b31143012060355040b0c0b636f6e73756c74616e6379311a30
1806035504030c117265676973747261722e73746f6b2e6e6c3059301306072a
8648ce3d020106082a8648ce3d03010703420004507ac8491a8c69c7b5c31d03
09ed35ba13f5884ce62b88cf3018154fa059b020ec6bebb94e02b8934021898d
a789c711cea71339f50e348edf0d923ed02dc7b7a3818d30818a301d0603551d
0e0416041408c2bf36887f79412185872f16a7aca6efb3d2b3301f0603551d23
04183016801408c2bf36887f79412185872f16a7aca6efb3d2b3300f0603551d
130101fff040530030101fff30270603551d250420301e06082b0601050507031c
06082b0601050507030106082b06010505070302300e0603551d0f0101fff0404
030201f6300a06082a8648ce3d04030203470030440220744c99008513b2f1bc
fdf9021a46fb174cf883a27cald93faeacf31e4edd12c60220114714dbf51a5e
78f581b9421c6e4702ab537270c5bafb2dl6c3de9aa182c35f
<CODE ENDS>
```

C.2.3. MASA Certificate

```
<CODE BEGINS>
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      14:26:b8:1c:ce:d8:c3:e8:14:05:cb:87:67:0d:be:ef:d5:81:25:b4
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C=NL, ST=NB, L=Helmond, O=vanderstok,
      OU=manufacturer, CN=masa.stok.nl

    Validity
      Not Before: Dec  9 10:02:36 2020 GMT
      Not After : Sep  5 10:02:36 2023 GMT
    Subject: C=NL, ST=NB, L=Helmond, O=vanderstok,
      OU=manufacturer, CN=masa.stok.nl
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
      Public-Key: (256 bit)
      pub:
        04:59:80:94:66:14:94:20:30:3c:66:08:85:55:86:
```

```
db:e7:d4:d1:d7:7a:d2:a3:1a:0c:73:6b:01:0d:02:
12:15:d6:1f:f3:6e:c8:d4:84:60:43:3b:21:c5:83:
80:1e:fc:e2:37:85:77:97:94:d4:aa:34:b5:b6:c6:
ed:f3:17:5c:f1
ASN1 OID: prime256v1
NIST CURVE: P-256
X509v3 extensions:
  X509v3 Subject Key Identifier:
E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3
  X509v3 Authority Key Identifier:
    keyid:
E4:03:93:B4:C3:D3:F4:2A:80:A4:77:18:F6:96:49:03:01:17:68:A3

  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Extended Key Usage:
    CMC Registration Authority,
    TLS Web Server Authentication,
    TLS Web Client Authentication
  X509v3 Key Usage: critical
    Digital Signature, Non Repudiation, Key Encipherment,
    Data Encipherment, Certificate Sign, CRL Sign
Signature Algorithm: ecdsa-with-SHA256
30:44:02:20:2e:c5:f2:24:72:70:20:ea:6e:74:8b:13:93:67:
8a:e6:fe:fb:8d:56:7f:f5:34:18:a9:ef:a5:0f:c3:99:ca:53:
02:20:3d:dc:91:d0:e9:6a:69:20:01:fb:e4:20:40:de:7c:7d:
98:ed:d8:84:53:61:84:a7:f9:13:06:4c:a9:b2:8f:5c
<CODE ENDS>
```

Below is the hexadecimal representation:

```

<CODE BEGINS>
3082026d30820214a00302010202141426b81cced8c3e81405cb87670dbeefd5
8125b4300a06082a8648ce3d040302306f310b3009060355040613024e4c310b
300906035504080c024e423110300e06035504070c0748656c6d6f6e64311330
11060355040a0c0a76616e64657273746f6b31153013060355040b0c0c6d616e
7566616374757265723115301306035504030c0c6d6173612e73746f6b2e6e6c
301e170d3230313230393130303233365a170d3233303930353130303233365a
306f310b3009060355040613024e4c310b300906035504080c024e423110300e
06035504070c0748656c6d6f6e6431133011060355040a0c0a76616e64657273
746f6b31153013060355040b0c0c6d616e756661637475726572311530130603
5504030c0c6d6173612e73746f6b2e6e6c3059301306072a8648ce3d02010608
2a8648ce3d0301070342000459809466149420303c6608855586dbe7d4d1d77a
d2a31a0c736b010d021215d61ff36ec8d48460433b21c583801efce237857797
94d4aa34b5b6c6edf3175cf1a3818d30818a301d0603551d0e04160414e40393
b4c3d3f42a80a47718f6964903011768a3301f0603551d23041830168014e403
93b4c3d3f42a80a47718f6964903011768a3300f0603551d130101ff04053003
0101ff30270603551d250420301e06082b0601050507031c06082b0601050507
030106082b06010505070302300e0603551d0f0101ff0404030201f6300a0608
2a8648ce3d040302034700304402202ec5f224727020ea6e748b1393678ae6fe
fb8d567ff53418a9efa50fc399ca5302203ddc91d0e96a692001fbe42040de7c
7d98edd884536184a7f913064ca9b28f5c
<CODE ENDS>

```

C.3. COSE-signed Pledge Voucher Request (PVR)

In this COSE example the voucher request has been signed by the Pledge using the private key of Appendix C.1.1, and has been sent to the link-local JRC (Registrar) over CoAPS.

```

POST coaps://[JRC-link-local-address]/b/rv
Content-Format: TBD3
Payload: signed_request_voucher

```

The payload `signed_request_voucher` is shown as hexadecimal dump (with lf added):

<CODE BEGINS>

```
d28444a101382ea104582097113db094eee8eae48683e7337875c0372164
be89d023a5f3df52699c0fbfb55902d2a11909c5a60274323032302d3132
2d32335431323a30353a32325a0474323032322d31322d32335431323a30
353a32325a01020750684ca83e27230aff97630cf2c1ec409a0d6e706c65
6467652e312e322e332e340a590279308202753082021ca0030201020214
7056eaaa3066d8826a555b9088d462bf9cf28cfd300a06082a8648ce3d04
03023073310b3009060355040613024e4c310b300906035504080c024e42
3110300e06035504070c0748656c6d6f6e6431133011060355040a0c0a76
616e64657273746f6b31143012060355040b0c0b636f6e73756c74616e63
79311a301806035504030c117265676973747261722e73746f6b2e6e6c30
1e170d3230313230393130303233365a170d323131323039313030323336
5a3073310b3009060355040613024e4c310b300906035504080c024e4231
10300e06035504070c0748656c6d6f6e6431133011060355040a0c0a7661
6e64657273746f6b31143012060355040b0c0b636f6e73756c74616e6379
311a301806035504030c117265676973747261722e73746f6b2e6e6c3059
301306072a8648ce3d020106082a8648ce3d03010703420004507ac8491a
8c69c7b5c31d0309ed35ba13f5884ce62b88cf3018154fa059b020ec6beb
b94e02b8934021898da789c711cea71339f50e348edf0d923ed02dc7b7a3
818d30818a301d0603551d0e0416041408c2bf36887f79412185872f16a7
aca6efb3d2b3301f0603551d2304183016801408c2bf36887f7941218587
2f16a7aca6efb3d2b3300f0603551d130101ff040530030101ff30270603
551d250420301e06082b0601050507031c06082b0601050507030106082b
06010505070302300e0603551d0f0101ff0404030201f6300a06082a8648
ce3d04030203470030440220744c99008513b2f1bcfd9021a46fb174cf8
83a27ca1d93faeacf31e4edd12c60220114714dbf51a5e78f581b9421c6e
4702ab537270c5bafb2d16c3de9aa182c35f58473045022063766c7bbd1b
339dbc398e764af3563e93b25a69104befe9aac2b3336b8f56e1022100cd
0419559ad960ccaed4dee3f436eca40b7570b25a52eb60332bc1f2991484
e9
```

<CODE ENDS>

The Pledge uses the "proximity" (SID 2502, enum 2) assertion together with an included proximity-registrar-cert field (SID 2511) to inform MASA about its proximity to the specific Registrar. The representation of signed_voucher_request in CBOR diagnostic format is:

```

<CODE BEGINS>
Diagnose(signed_request_voucher) =
18([
h'A101382E',      / {"alg": -47} /
{4: h' 97113DB094EEE8EAE48683E7337875C0372164B
    E89D023A5F3DF52699C0FBFB5'},
h'<request_voucher>', / byte string as detailed below /
h' 3045022063766C7BBD1B339DBC398E764AF3563E93B
25A69104BEFE9AAC2B3336B8F56E1022100CD0419559A
D960CCAED4DEE3F436ECA40B7570B25A52EB60332BC1F
2991484E9'
])

Diagnose(request_voucher) =
{2501: {2: "2020-12-23T12:05:22Z",
        4: "2022-12-23T12:05:22Z",
        1: 2,
        7: h' 684CA83E27230AFF97630CF2C1EC409A',
        13: "pledge.1.2.3.4",
        10: h'<regis-cert-hex>' / byte string as defined in C.2.2 /
      }}
<CODE ENDS>

```

C.4. COSE-signed Registrar Voucher Request (RVR)

In this example the Registrar's voucher request has been signed by the JRC (Registrar) using the private key from Appendix C.1.2. Contained within this voucher request is the voucher request PVR that was made by the Pledge to JRC. Note that the RVR uses the HTTPS protocol (not CoAP) and corresponding long URI path names as defined in [RFC8995]. The Content-Type and Accept headers indicate the constrained voucher format that is defined in the present document. Because the Pledge used this format in the PVR, the JRC must also use this format in the RVR.

```

POST https://masa.example.com/.well-known/brski/requestvoucher
Content-Type: application/voucher-cose+cbor
Accept: application/voucher-cose+cbor
Body: signed_masa_request_voucher

```

The payload `signed_masa_voucher_request` is shown as hexadecimal dump (with lf added):

<CODE BEGINS>

```
d28444a101382ea1045820e8735bc4b470c3aa6a7aa9aa8ee584c09c1113
1b205efec5d0313bad84c5cd05590414a11909c5a60274323032302d3132
2d32385431303a30333a33355a0474323032322d31322d32385431303a30
333a33355a07501551631f6e0416bd162ba53ea00c2a050d6e706c656467
652e312e322e332e3405587131322d32385431303a30333a33355a075015
51631f6e0416bd162ba53ea00c2a050d6e706c656467652e312e322e332e
3405587131322d32385431303a3000000000000000000000000000000000416bd16
2ba53ea00c2a050d6e706c656467652e312e322e332e3405587131322d32
385431303a09590349d28444a101382ea104582097113db094eee8eae486
83e7337875c0372164be89d023a5f3df52699c0fbfb55902d2a11909c5a6
0274323032302d31322d32385431303a30333a33355a0474323032322d31
322d32385431303a30333a33355a010207501551631f6e0416bd162ba53e
a00c2a050d6e706c656467652e312e322e332e340a590279308202753082
021ca00302010202147056eaaa3066d8826a555b9088d462bf9cf28cfd30
0a06082a8648ce3d0403023073310b3009060355040613024e4c310b3009
06035504080c024e423110300e06035504070c0748656c6d6f6e64311330
11060355040a0c0a76616e64657273746f6b31143012060355040b0c0b63
6f6e73756c74616e6379311a301806035504030c11726567697374726172
2e73746f6b2e6e6c301e170d3230313230393130303233365a170d323131
3230393130303233365a3073310b3009060355040613024e4c310b300906
035504080c024e423110300e06035504070c0748656c6d6f6e6431133011
060355040a0c0a76616e64657273746f6b31143012060355040b0c0b636f
6e73756c74616e6379311a301806035504030c117265676973747261722e
73746f6b2e6e6c3059301306072a8648ce3d020106082a8648ce3d030107
03420004507ac8491a8c69c7b5c31d0309ed35ba13f5884ce62b88cf3018
154fa059b020ec6bebb94e02b8934021898da789c711cea71339f50e348e
df0d923ed02dc7b7a3818d30818a301d0603551d0e0416041408c2bf3688
7f79412185872f16a7aca6efb3d2b3301f0603551d2304183016801408c2
bf36887f79412185872f16a7aca6efb3d2b3300f0603551d130101ff0405
30030101ff30270603551d250420301e06082b0601050507031c06082b06
01050507030106082b06010505070302300e0603551d0f0101ff04040302
01f6300a06082a8648ce3d04030203470030440220744c99008513b2f1bc
fdf9021a46fb174cf883a27cald93faeacf31e4edd12c60220114714dbf5
1a5e78f581b9421c6e4702ab537270c5bafb2d16c3de9aa182c35f584730
45022063766c7bbdlb339dbc398e764af3563e93b25a69104befe9aac2b3
336b8f56e1022100cd0419559ad960ccaed4dee3f436eca40b7570b25a52
eb60332bc1f2991484e958473045022100e6b45558c1b806bba23f4ac626
c9bdb6fd354ef4330d8dfb7c529f29cca934c802203c1f2ccbbac89733d1
7ee7775bc2654c5f1cc96afba2741cc31532444aa8fed8
```

<CODE ENDS>

The representation of signed_masa_voucher_request in CBOR diagnostic format is:

```

<CODE BEGINS>
Diagnose(signed_registrar_request-voucher)
18([
h'A101382E',      / {"alg": -47} /
h'E8735BC4B470C3AA6A7AA9AA8EE584C09C11131B205EFEC5D0313BAD84
C5CD05'},
h'<registrar_request_voucher>', / byte string as detailed below /
h'3045022100E6B45558C1B806BBA23F4AC626C9BDB6FD354EF4330D8DFB
7C529F29CCA934C802203C1F2CCBBAC89733D17EE7775BC2654C5F1CC96A
FBA2741CC31532444AA8FED8'
])

Diagnose(registrar_request_voucher)
{2501:
  {2: "2020-12-28T10:03:35Z",
   4: "2022-12-28T10:03:35Z",
   7: h'1551631F6E0416BD162BA53EA00C2A05',
  13: "pledge.1.2.3.4",
   5: h'31322D32385431303A30333A333355A07501551631F6E0416BD
      162BA53EA00C2A050D6E706C656467652E312E322E332E3405
      587131322D32385431303A30000000000000000000000000000004
      16BD162BA53EA00C2A050D6E706C656467652E312E322E332E
      3405587131322D32385431303A', / idevid-issuer /
   9: h'<prior-pvr>' / prior-signed-voucher-request = PVR /
  }
}
<CODE ENDS>

```

C.5. COSE-signed Voucher from MASA

The resulting voucher is created by the MASA and returned via the JRC to the Pledge. It is signed by the MASA's private key (see Appendix C.1.3) and can be verified by the Pledge using the MASA's public key that it stores.

Below is the binary signed_voucher, encoded in hexadecimal (with lf added):

```

<CODE BEGINS>
d28444a101382ea104582039920a34ee92d3148ab3a729f58611193270c9
029f7784daf112614b19445d5158cfa1190993a70274323032302d31322d
32335431353a30333a31325a0474323032302d31322d32335431353a3233
3a31325a010007506508e06b2959d5089d7a3169ea889a490b6e706c6564
67652e312e322e332e340858753073310b3009060355040613024e4c310b
300906035504080c024e423110300e06035504070c0748656c6d6f6e6431
133011060355040a0c0a76616e64657273746f6b31143012060355040b0c
0b636f6e73756c74616e6379311a301806035504030c1172656769737472
61722e73746f6b2e6e6c03f458473045022022515d96cd12224ee5d3ac67
3237163bba24ad84815699285d9618f463ee73fa022100a6bff9d8585c1c
9256371ece94da3d26264a5dfec0a354fe7b3aef58344c512f
<CODE ENDS>

```

The representation of signed_voucher in CBOR diagnostic format is:

```

<CODE BEGINS>
Diagnose(signed_voucher) =
18([
h'A101382E',      / {"alg": -47} /
{4: h'39920A34EE92D3148AB3A729F58611193270C9029F7784DAF112614B194
45D51'},
h'<voucher>',      / byte string as detailed below /
h'3045022022515D96CD12224EE5D3AC673237163BBA24AD84815699285D9618F
463EE73FA022100A6BFF9D8585C1C9256371ECE94DA3D26264A5DFEC0A354FE7B
3AEF58344C512F'
])

Diagnose(voucher) =
{2451:
  {2: "2020-12-23T15:03:12Z",
   4: "2020-12-23T15:23:12Z",
   1: 0,
   7: h'6508E06B2959D5089D7A3169EA889A49',
  11: "pledge.1.2.3.4",
   8: h'<regis-cert-hex>', / as detailed in C.2.2 /
   3: false}
}
<CODE ENDS>

```

In above, regis-cert-hex represents the hexadecimal encoding of the Registrar certificate of Appendix C.2.2.

Appendix D. Generating Certificates with OpenSSL

This informative appendix shows an example of a Bash shell script to generate test certificates for the Pledge IDevID, the Registrar and the MASA. This shell script cannot be run stand-alone because it depends on particular input files which are not included in this appendix. Nevertheless, this example script may provide guidance on how OpenSSL can be configured for generating Constrained BRSKI certificates.

Note: the *-comb.crt certificate files combine the certificate with the private key. These are generated to be used by libcoap for DTLS connection establishment.

```
<CODE BEGINS>
#!/bin/bash
#try-cert.sh
export dir=./brski/intermediate
export cadir=./brski
export cnfdir=./conf
export format=pem
export default_crl_days=30
sn=8

DevID=pledge.1.2.3.4
serialNumber="serialNumber=$DevID"
export hwType=1.3.6.1.4.1.6715.10.1
export hwSerialNum=01020304 # Some hex
export subjectAltName="otherName:1.3.6.1.5.5.7.8.4;SEQ:hmodname"
echo $hwType - $hwSerialNum
echo $serialNumber
OPENSSL_BIN="openssl"

# remove all files
rm -r ./brski/*
#
# initialize file structure
# root level
cd $cadir
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
touch serial
echo 11223344556600 >serial
echo 1000 >crlnumber
# intermediate level
mkdir intermediate
cd intermediate
```

```
mkdir certs crl csr newcerts private
chmod 700 private
touch index.txt
echo 11223344556600 >serial
echo 1000 > crlnumber
cd ../..

# file structure is cleaned start filling

echo "#####"
echo "create registrar keys and certificates "
echo "#####"

echo "create root registrar certificate using ecDSA with sha 256 key"
$OPENSSL_BIN ecparam -name prime256v1 -genkey \
    -noout -out $cadir/private/ca-regis.key

$OPENSSL_BIN req -new -x509 \
    -config $cnfdir/openssl-regis.cnf \
    -key $cadir/private/ca-regis.key \
    -out $cadir/certs/ca-regis.crt \
    -extensions v3_ca \
    -days 365 \
    -subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=consultancy \
/CN=registrar.stok.nl"

# Combine authority certificate and key
echo "Combine authority certificate and key"
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT \
    -inkey $cadir/private/ca-regis.key \
    -in $cadir/certs/ca-regis.crt -export \
    -out $cadir/certs/ca-regis-comb.pfx

# converteer authority pkcs12 file to pem
echo "converteer authority pkcs12 file to pem"
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT \
    -in $cadir/certs/ca-regis-comb.pfx \
    -out $cadir/certs/ca-regis-comb.crt -nodes

#show certificate in registrar combined certificate
$OPENSSL_BIN x509 -in $cadir/certs/ca-regis-comb.crt -text

#
# Certificate Authority for MASA
#
```

```
echo "#####"
echo "create MASA keys and certificates "
echo "#####"

echo "create root MASA certificate using ecdsa with sha 256 key"
$OPENSSL_BIN ecparam -name prime256v1 -genkey -noout \
    -out $cadir/private/ca-masa.key

$OPENSSL_BIN req -new -x509 \
    -config $cnfdir/openssl-masa.cnf \
    -days 1000 -key $cadir/private/ca-masa.key \
    -out $cadir/certs/ca-masa.crt \
    -extensions v3_ca \
    -subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=manufacturer\
/CN=masa.stok.nl"

# Combine authority certificate and key
echo "Combine authority certificate and key for masa"
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT\
    -inkey $cadir/private/ca-masa.key \
    -in $cadir/certs/ca-masa.crt -export \
    -out $cadir/certs/ca-masa-comb.pfx

# converteer authority pkcs12 file to pem for masa
echo "converteer authority pkcs12 file to pem for masa"
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT\
    -in $cadir/certs/ca-masa-comb.pfx \
    -out $cadir/certs/ca-masa-comb.crt -nodes

#show certificate in pledge combined certificate
$OPENSSL_BIN x509 -in $cadir/certs/ca-masa-comb.crt -text

#
# Certificate for Pledge derived from MASA certificate
#
echo "#####"
echo "create pledge keys and certificates "
echo "#####"

# Pledge derived Certificate

echo "create pledge derived certificate using ecdsa with sha 256 key"
$OPENSSL_BIN ecparam -name prime256v1 -genkey -noout \
    -out $dir/private/pledge.key

echo "create pledge certificate request"
```



```
$OPENSSL_BIN req -nodes -new -sha256 \  
-key $dir/private/pledge.key -out $dir/csr/pledge.csr \  
-subj "/C=NL/ST=NB/L=Helmond/O=vanderstok/OU=manufacturing\  
/CN=uuid:$DevID/$serialNumber"  
  
# Sign pledge derived Certificate  
echo "sign pledge derived certificate "  
$OPENSSL_BIN ca -config $cnfdir/openssl-pledge.cnf \  
-extensions 8021ar_idevid\  
-days 365 -in $dir/csr/pledge.csr \  
-out $dir/certs/pledge.crt  
  
# Add pledge key and pledge certificate to pkcs12 file  
echo "Add derived pledge key and derived pledge \  
certificate to pkcs12 file"  
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT\  
-inkey $dir/private/pledge.key \  
-in $dir/certs/pledge.crt -export \  
-out $dir/certs/pledge-comb.pfx  
  
# convertteer pledge pkcs12 file to pem  
echo "convertteer pledge pkcs12 file to pem"  
$OPENSSL_BIN pkcs12 -passin pass:watnietWT -passout pass:watnietWT\  
-in $dir/certs/pledge-comb.pfx \  
-out $dir/certs/pledge-comb.crt -nodes  
  
#show certificate in pledge-comb.crt  
$OPENSSL_BIN x509 -in $dir/certs/pledge-comb.crt -text  
  
#show private key in pledge-comb.crt  
$OPENSSL_BIN ecparam -name prime256v1\  
-in $dir/certs/pledge-comb.crt -text  
  
<CODE ENDS>
```

Appendix E. Pledge Device Class Profiles

This specification allows implementers to select between various functional options for the Pledge, yielding different code size footprints and different requirements on Pledge hardware. Thus for each product an optimal trade-off between functionality, development/maintenance cost and hardware cost can be made.

This appendix illustrates different selection outcomes by means of defining different example "profiles" of constrained Pledges. In the following subsections, these profiles are defined and a comparison is provided.

E.1. Minimal Pledge

The Minimal Pledge profile (Min) aims to reduce code size and hardware cost to a minimum. This comes with some severe functional restrictions, in particular:

- * No support for EST re-enrollment: whenever this would be needed, a factory reset followed by a new bootstrap process is required.
- * No support for change of Registrar: for this case, a factory reset followed by a new bootstrap process is required.

This profile would be appropriate for single-use devices which must be replaced rather than re-deployed. That might include medical devices, but also sensors used during construction, such as concrete temperature sensors.

E.2. Typical Pledge

The Typical Pledge profile (Typ) aims to support a typical Constrained BRSKI feature set including EST re-enrollment support and Registrar changes.

E.3. Full-featured Pledge

The Full-featured Pledge profile (Full) illustrates a Pledge category that supports multiple bootstrap methods, hardware real-time clock, BRSKI/EST resource discovery, and CSR Attributes request/response. It also supports most of the optional features defined in this specification.

E.4. Comparison Chart of Pledge Classes

The below table specifies the functions implemented in the three example Pledge classes Min, Typ and Full.

Function	Min	Typ	Full
General	===	===	====
Support Constrained BRSKI bootstrap	Y	Y	Y
Support other bootstrap method(s)	-	-	Y
Real-time clock and cert time checks	-	-	Y
Constrained BRSKI	===	===	====

Discovery for rt=brski*	-	-	Y
Support pinned Registrar public key (RPK)	Y	-	Y
Support pinned Registrar certificate	-	Y	Y
Support pinned Domain CA	-	Y	Y
Constrained EST	===	===	=====
Discovery for rt=ace.est*	-	-	Y
GET /att and response parsing	-	-	Y
GET /crtts format 281 (multiple CA certs)	-	-	Y
GET /crtts only format TBD287 (one CA cert only)	Y	Y	-
ETag handling support for GET /crtts	-	Y	Y
Re-enrollment supported	- (1)	Y	Y
6.6.1 optimized procedure	Y	Y	-
Pro-active cert re-enrollment at own initiative	N/A	-	Y
Periodic trust anchor retrieval GET /crtts	- (1)	Y	Y
Supports change of Registrar identity	- (1)	Y	Y

Table 6

Notes: (1) is possible only by doing a factory-reset followed by a new bootstrap procedure.

Contributors

Russ Housley
Email: housley@vigilsec.com

Authors' Addresses

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy
Email: stokcons@bbhmail.nl

Panos Kampanakis
Cisco Systems
Email: pkampana@cisco.com

Esko Dijk
IoTconsultancy.nl
Email: esko.dijk@iotconsultancy.nl

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2020

B. Liu (Ed.)
Huawei Technologies
X,. Xiao (Ed.)
A,. Hecker
MRC, Huawei Technologies
S. Jiang
Huawei Technologies
Z,. Despotovic
MRC, Huawei Technologies
February 25, 2020

Information Distribution over GRASP
draft-ietf-anima-grasp-distribution-00

Abstract

This document proposes a solution for information distribution in autonomic networks. Information distribution is categorized into two different modes: 1) instantaneous distribution; 2) publication for retrieval. In the former case, the information is sent, propagates and is disposed of after reception. In the latter case, information needs to be stored in the network.

The capabilities to distribute information are basic and fundamental needs for an autonomous network ([RFC7575]). This document describes typical use cases of information distribution in ANI and requirements to ANI, such that rich information distribution can be natively supported. The document proposes extensions to the autonomic nodes and suggests an implementation based on GRASP ([I-D.ietf-anima-grasp]) extensions as a protocol on the wire.

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 August 28, 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 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
2. Terminology	4
3. Requirements for Information Distribution in ANI	4
4. Node Behaviors	6
4.1. Instant Information Distribution (IID) Sub-module	7
4.1.1. Instant P2P Communication	7
4.1.2. Instant Flooding Communication	7
4.2. Asynchronous Information Distribution (AID) Sub-module	8
4.2.1. Information Storage	8
4.2.2. Event Queue	10
4.3. Summary	12
5. Extending GRASP for Information Distribution	12
5.1. Realizing Instant P2P Transmission	12
5.2. Realizing Instant Selective Flooding	13
5.3. Realizing Subscription as An Event	13
5.4. Un_Subscription Objective Option	14
5.5. Publishing Objective Option	14
6. Security Considerations	15
7. IANA Considerations	15
8. Acknowledgements	15
9. References	15
9.1. Normative References	15
9.2. Informative References	16
Appendix A. Open Issues [RFC Editor: To Be removed before becoming RFC]	16
Appendix B. Closed Issues [RFC Editor: To Be removed before becoming RFC]	17
Appendix C. Change log [RFC Editor: To Be removed before becoming RFC]	17
Appendix D. Real-world Use Cases of Information Distribution	17
D.1. Service-Based Architecture (SBA) in 3GPP 5G	18

D.2. Vehicle-to-Everything (V2X)	19
D.3. Summary	19
Appendix E. Information Distribution Module in ANI	20
Appendix F. Asynchronous ID Integrated with GRASP APIs	20
Authors' Addresses	21

1. Introduction

In an autonomic network, autonomic functions (AFs) running on autonomic nodes constantly exchange information, e.g. AF control/management signaling or AF data exchange. This document discusses the information distribution capability of such exchanges between AFs.

Depending on the number of participants, the information can be distributed in in the following scenarios:

- 1) Point-to-point (P2P) Communication: information is exchanged between parties, i.e. two nodes.
- 2) One-to-Many Communication: information exchanges involve an information source and multiple receivers.

The approaches to information distribution can be chiefly categorized into two basic modes:

- 1) An instantaneous mode (push): a source sends the actual content (e.g. control/management signaling, synchronization data and so on) to all interested receiver(s) immediately. Generally, some preconfiguration is required, as nodes interested in this information must be already known to all nodes in the sense that any receiving node must be able to decide, to which nodes this data is to be sent.
- 2) An asynchronous mode (delayed pull): here, a source publishes the content in some form in the network, which may later be looked for, found and retrieved by some endpoints in the AN. Here, depending on the size of the content, either the whole content or only its metadata might be published into the AN. In the latter case the metadata (e.g. a content descriptor, e.g. a key, and a location in the ANI) may be used for the actual retrieval. Importantly, the source, i.e. here publisher, needs to be able to determine the node, where the information (or its metadata) can be stored.

To avoid repetitive implementations by each AF developer, this document opts for a common support for information distribution

implemented as a basic ANI capability, therefore available to all AFs. In fact, GRASP already provides part of the capabilities.

Regardless, an AF may still define and implement its own information distribution capability. Such a capability may then be advertised using the common information distribution capability defined in this document. Overall, ANI nodes and AFs may decide, which of the information distribution mechanisms they want to use for which type of information, according to their own preferences (e.g. semantic routing table, etc.)

This document first analyzes requirements for information distribution in autonomic networks (Section 3) and then discuss the relevant node behavior (Section 4). After that, the required GRASP extensions are formally introduced (Section 5).

and relevan

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Requirements for Information Distribution in ANI

The question of information distribution in an autonomic network can be discussed through particular use cases or more generally. Depending on the situation it can be quite simple or might require more complex provisions.

Indeed, in the simplest case, the information can be sent:

- 1) at once (in one packet, in one flow),
- 2) straightaway (send-and-forget),
- 3) to all nodes.

Presuming 1), 2) and 3) hold, information distribution in smaller or scarce topologies can be implemented using broadcast, i.e. unconstrained flooding. For reasons well-understood, this approach has its limits in larger and denser networks. In this case, a graph can be constructed such that it contains every node exactly once (e.g. a spanning tree), still allowing to distribute any information to all nodes straightaway. Multicast tree construction protocols could be used in this case. There are reasonable use cases for such scenarios, as presented in Appendix D.

A more complex scenario arises, if only 1) and 2) hold, but the information only concerns a subset of nodes. Then, some kind of selection becomes required, to which nodes the given information should be distributed. Here, a further distinction is necessary; notably, if the selection of the target nodes is with respect to the nature or position of the node, or whether it is with respect to the information content. If the first, some knowledge about the node types, its topological position, etc (e.g. the routing information within ANI) can be used to distinguish nodes accordingly. For instance, edge nodes and forwarding nodes can be distinguished in this way. If the distribution scope is primarily to be defined by the information elements, then a registration / join / subscription or label distribution mechanism is unavoidable. This would be the case, for instance, if the AFs can be dynamically deployed on nodes, and the information is majorily destined to the AFs. Then, depending on the current AF deployment, the distribution scope must be adjusted as well.

If only 1) holds, but the information content might be required again and again, or might not yet be fully available, then more complex mechanisms might be required to store the information within the network for later, for further redistribution, and for notification of interested nodes. Examples for this include distribution of reconfiguration information for different AF instances, which might not require an immediate action, but only an eventual update of the parameters. Also, in some situations, there could be a significant delay between the occurrence of a new event and the full content availability (e.g. if the processing requires a lot of time).

Finally, none of the three might hold. Then, along with the subscription and notification, the actual content might be different from its metadata, i.e. some description of the content and, possibly, its location. The fetching can then be implemented in different, appropriate ways, if necessary as a complex transport session.

In essence, as flooding is usually not an option, and the interest of nodes for particular information elements can change over time, ANI should support autonomies also for the information distribution.

This calls for autonomic mechanisms in the ANI, allowing participating nodes to 1) advertise or publish 2) look for or subscribe to 3) store 4) fetch/retrieve 5) instantaneously push information elements.

In the following cases, situations depicting diverse information distribution needs are discussed.

- 1) Long Communication Intervals. The actual sending of the information is not necessarily instantaneous with some event. Advanced AFs may involve into longer jobs/tasks (e.g. database lookup, authentication etc.) when processing requests, and might not be able to reply immediately. Instead of actively waiting for the reply, a better way for an interested AF might be to get notified, when the reply is finally available.
- 2) Common Interest Distribution. AFs may share interest in common information. For example, the network intent will be distributed to network nodes enrolled, which is usually one-to-many scenario. Intent distribution can also be performed by an instant flooding (e.g. via GRASP) to every network node. However, because of network dynamics, not every node can be just ready at the moment when the network intent is broadcast. Also, a flooding often does not cover all network nodes as there is usually a limitation on the hop number. In fact, nodes may join in the network sequentially. In this situation, an asynchronous communication model could be a better choice where every (newly joining) node can subscribe the intent information and will get notified if it is ready (or updated).
- 3) Distributed Coordination. With computing and storage resources on autonomic nodes, alive AFs not only consume but also generate data information. An example is AFs coordinating with each other as distributed schedulers, responding to service requests and distributing tasks. It is critical for those AFs to make correct decisions based on local information, which might be asymmetric as well. AFs may also need synthetic/aggregated data information (e.g. statistic info, like average values of several AFs, etc.) to make decisions. In these situations, AFs will need an efficient way to form a global view of the network (e.g. about resource consumption, bandwidth and statistics). Obviously, purely relying on instant communication model is inefficient, while a scalable, common, yet distributed data layer, on which AFs can store and share information in an asynchronous way, should be a better choice.

Therefore, for ANI, in order to support various communication scenarios, an information distribution module is required, and both instantaneous and asynchronous communication models should be supported. Some real-world use cases are introduced in Appendix D.

4. Node Behaviors

In this section, how a node should behave in order to support the two identified modes of information distribution is discussed. An ANI is

a distributed system, so the information distribution module must be implemented in a distributed way as well.

4.1. Instant Information Distribution (IID) Sub-module

In this case, An information sender directly specifies the information receiver(s). The instant information distribution sub-module will be the main element.

4.1.1. Instant P2P Communication

IID sub-module performs instant information transmission for ASAs running in an ANI. In specific, IID sub-module will have to retrieve the address of the information receiver specified by an ASA, then deliver the information to the receiver. Such a delivery can be done either in a connectionless or a connection-oriented way.

Current GRASP provides the capability to support instant P2P synchronization for ASAs. A P2P synchronization is a use case of P2P information transmission. However, as mention in Section 3, there are some scenarios where one node needs to transmit some information to another node(s). This is different to synchronization because after transmitting the information, the local status of the information does not have to be the same as the information sent to the receiver. This is not directly support by existing GRASP.

4.1.2. Instant Flooding Communication

IID sub-module finishes instant flooding for ASAs in an ANI. Instant flooding is for all ASAs in an ANI. An information sender has to specify a special destination address of the information and broadcast to all interfaces to its neighbors. When another IID sub-module receives such a broadcast, after checking its TTL, it further broadcast the message to the neighbors. In order to avoid flooding storms in an ANI, usually a TTL number is specified, so that after a pre-defined limit, the flooding message will not be further broadcast again.

In order to avoid unnecessary flooding, a selective flooding can be done where an information sender wants to send information to multiple receivers at once. When doing this, sending information needs to contain criteria to judge on which interfaces the distributed information should and should not be sent. Specifically, the criteria contain:

- o Matching Condition: a set of matching rules such as addresses of recipients, node features and so on.

- o Action: what the node needs to do when the Matching Condition is fulfilled. For example, the action could be forwarding or discarding the distributed message.

Sent information must be included in the message distributed from the sender. The receiving node reacts by first checking the carried Matching Condition in the message to decide who should consume the message, which could be either the node itself, some neighbors or both. If the node itself is a recipient, Action field is followed; if a neighbor is a recipient, the message is sent accordingly.

An exemplary extension to support selective flooding on GRASP is described in Section 5.

4.2. Asynchronous Information Distribution (AID) Sub-module

In asynchronous information distribution, sender(s) and receiver(s) are not immediately specified while they may appear in an asynchronous way. Firstly, AID sub-module enables that the information can be stored in the network; secondly, AID sub-module provides an information publication and subscription (Pub/Sub) mechanism for ASAs.

As sketched in the previous section, in general each node requires two modules: 1) Information Storage (IS) module and 2) Event Queue (EQ) module in the information distribution module. Details of the two modules are described in the following sections.

4.2.1. Information Storage

IS module handles how to save and retrieve information for ASAs across the network. The IS module uses a syntax to index information, generating the hash index value (e.g. a hash value) of the information and mapping the hash index to a certain node in ANI. Note that, this mechanism can use existing solutions. Specifically, storing information in an ANIMA network will be realized in the following steps.

- 1) ASA-to-IS Negotiation. An ASA calls the API provided by information distribution module (directly supported by IS sub-module) to request to store the information somewhere in the network. The IS module performs various checks of the request (e.g. permitted information size).
- 2) Storing Peer Mapping. The information block will be handled by the IS module in order to calculate/map to a peer node in the network. Since ANIMA network is a peer-to-peer network, a typical way is to use distributed hash table (DHT) to map information to a

unique index identifier. For example, if the size of the information is reasonable, the information block itself can be hashed, otherwise, some meta-data of the information block can be used to generate the mapping.

- 3) Storing Peer Negotiation Request. Negotiation request of storing the information will be sent from the IS module to the IS module on the destination node. The negotiation request contains parameters about the information block from the source IS module. According to the parameters as well as the local available resource, the requested storing peer will send feedback the source IS module.
- 4) Storing Peer Negotiation Response. Negotiation response from the storing peer is sent back to the source IS module. If the source IS module gets confirmation that the information can be stored, source IS module will prepare to transfer the information block; otherwise, a new storing peer must be discovered (i.e. going to step 7).
- 5) Information Block Transfer. Before sending the information block to the storing peer that already accepts the request, the IS module of the source node will check if the information block can be afforded by one GRASP message. If so, the information block will be directly sent by calling a GRASP API ([I-D.ietf-anima-grasp-api]). Otherwise, a bulk data transmission is needed. For that, there are multiple ways to do it. The first option is to utilize one of existing protocols that is independent of the GRASP stack. For example, a session connectivity can be established to the storing peer, and over the connection the bulky data can be transmitted part by part. In this case, the IS module should support basic TCP-based session protocols such as HTTP(s) or native TCP. The second option is to directly use GRASP itself for bulky data transferring[I-D.carpenter-anima-grasp-bulk].
- 6) Information Writing. Once the information block (or a smaller block) is received, the IS module of the storing peer will store the data block in the local storage is accessible.
- 7) (Optional) New Storing Peer Discovery. If the previously selected storing peer is not available to store the information block, the source IS module will have to identify a new destination node to start a new negotiation. In this case, the discovery can be done by using discovery GRASP API to identify a new candidate, or more complex mechanisms can be introduced.

Similarly, Getting information from an ANI will be realized in the following steps.

- 1) ASA-to-IS Request. An ASA accesses the IS module via the APIs exposed by the information distribution module. The key/index of the interested information will be sent to the IS module. An assumption here is that the key/index should be known to an ASA before an ASA can ask for the information. This relates to the publishing/subscribing of the information, which are handled by other modules (e.g. Event Queue with Pub/Sub supported by GRASP).
- 2) Storing Peer Mapping. IS module maps the key/index of the requested information to a peer that stores the information, and prepares the information request. The mapping here follows the same mechanism when the information is stored.
- 3) Retrieval Negotiation Request. The source IS module sends a request to the storing peer and asks if such an information object is available.
- 4) Retrieval Negotiation Response. The storing peer checks the key/index of the information in the request, and replies to the source IS module. If the information is found and the information block can be afforded within one GRASP message, the information will be sent together with the response to the source IS module.
- 5) (Optional) New Destination Request. If the information is not found after the source IS module gets the response from the originally identified storing peer, the source IS module will have to discover the location of the requested information.

IS module can reuse distributed databases and key value stores like NoSQL, Cassandra, DHT technologies. storage and retrieval of information are all event-driven responsible by the EQ module.

4.2.2. Event Queue

The Event Queue (EQ) module is to help ASAs to publish information to the network and subscribe to interested information in asynchronous scenarios. In an ANI, information generated on network nodes is an event labeled with an event ID, which is semantically related to the topic of the information. Key features of EQ module are summarized as follows.

- 1) Event Group: An EQ module provides isolated queues for different event groups. If two groups of AFs could have completely different purposes, the EQ module allows to create multiple queues where only AFs interested in the same topic will be aware of the corresponding event queue.

- 2) Event Prioritization: Events can have different priorities in ANI. This corresponds to how much important or urgent the event implies. Some of them are more urgent than regular ones. Prioritization allows AFs to differentiate events (i.e. information) they publish or subscribe to.
- 3) Event Matching: an information consumer has to be identified from the queue in order to deliver the information from the provider. Event matching keeps looking for the subscriptions in the queue to see if there is an exact published event there. Whenever a match is found, it will notify the upper layer to inform the corresponding ASAs who are the information provider and subscriber(s) respectively.

The EQ module on every network node operates as follows.

- 1) Event ID Generation: If information of an ASA is ready, an event ID is generated according to the content of the information. This is also related to how the information is stored/saved by the IS module introduced before. Meanwhile, the type of the event is also specified where it can be of control purpose or user plane data.
- 2) Priority Specification: According to the type of the event, the ASA may specify its priority to say how this event is to be processed. By considering both aspects, the priority of the event will be determined.
- 3) Event Enqueue: Given the event ID, event group and its priority, a queue is identified locally if all criteria can be satisfied. If there is such a queue, the event will be simply added into the queue, otherwise a new queue will be created to accommodate such an event.
- 4) Event Propagation: The published event will be propagated to the other network nodes in the ANIMA domain. A propagation algorithm can be employed to optimize the propagation efficiency of the updated event queue states.
- 5) Event Match and Notification: While propagating updated event states, EQ module in parallel keeps matching published events and its interested consumers. Once a match is found, the provider and subscriber(s) will be notified for final information retrieval.

The category of event priority is defined as the following. In general, there are two event types:

- 1) Network Control Event: This type of events are defined by the ANI for operational purposes on network control. A pre-defined priority levels for required system messages is suggested. For highest level to lowest level, the priority value ranges from NC_PRIOR_HIGH to NC_PRIOR_LOW as integer values. The NC_PRIOR_* values will be defined later according to the total number system events required by the ANI.
- 2) Custom ASA Event: This type of events are defined by the ASAs of users. This specifies the priority of the message within a group of ASAs, therefore it is only effective among ASAs that join the same message group. Within the message group, a group header/leader has to define a list of priority levels ranging from CUST_PRIOR_HIGH to CUST_PRIOR_LOW. Such a definition completely depends on the individual purposes of the message group. When a system message is delivered, its event type and event priority value have to be both specified.

Event contains the address where the information is stored, after a subscriber is notified, it directly retrieves the information from the given location.

4.3. Summary

In summary, the general requirements for the information distribution module on each autonomic node are realized by two sub-modules handling instant communications and asynchronous communications, respectively. For instantaneous mode, node requirements are simple, calling for support for additional signaling. With minimum efforts, reusing the existing GRASP is possible.

For asynchronous mode, information distribution module uses new primitives on the wire, and implements an event queue and an information storage mechanism. An architectural consideration on ANI with the information distribution module is briefly discussed in Appendix E.

5. Extending GRASP for Information Distribution

5.1. Realizing Instant P2P Transmission

This could be a new message in GRASP. In fragmentary CDDL, an Unsolicited Synchronization message follows the pattern:

```
unsolicited_synch-message = [M_UNSOLIDSYNCH, session-id,  
objective]
```


A node MAY actively send a unicast Un-solicited Synchronization message with the Synchronization data, to another node. This MAY be sent to port GRASP_LISTEN_PORT at the destination address, which might be obtained by GRASP Discovery or other possible ways. The synchronization data are in the form of GRASP Option(s) for specific synchronization objective(s).

5.2. Realizing Instant Selective Flooding

Since normal flooding is already supported by GRASP, this section only defines the selective flooding extension.

In fragmentary CDDL, the selective flooding follows the pattern:

```
selective-flood-option = [O_SELECTIVE_FLOOD, +O_MATCH-CONDITION,
match-object, action]

O_MATCH-CONDITION = [O_MATCH-CONDITION, Obj1, match-rule, Obj2]
Obj1 = text

match-rule = GREATER / LESS / WITHIN / CONTAIN

Obj2 = text

match-object = NEIGHBOR / SELF

action = FORWARD / DROP
```

The option field encapsulates a match-condition option which represents the conditions regarding to continue or discontinue flood the current message. For the match-condition option, the Obj1 and Obj2 are to objects that need to be compared. For example, the Obj1 could be the role of the device and Obj2 could be "RSG". The match rules between the two objects could be greater, less than, within, or contain. The match-object represents of which Obj1 belongs to, it could be the device itself or the neighbor(s) intended to be flooded. The action means, when the match rule applies, the current device just continues flood or discontinues.

5.3. Realizing Subscription as An Event

In fragmentary CDDL, a Subscription Objective Option follows the pattern:

```
subscription-objection-option = [SUBSCRIPTION, 2, 2, subobj]
objective-name = SUBSCRIPTION

objective-flags = 2
```

```
loop-count = 2
```

```
subobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a subscription to a specific object.

5.4. Un_Subscription Objective Option

In fragmentary CDDL, a Un_Subscribe Objective Option follows the pattern:

```
Unsubscribe-objection-option = [UNSUBSCRIB, 2, 2, unsubobj]
```

```
objective-name = SUBSCRIPTION
```

```
objective-flags = 2
```

```
loop-count = 2
```

```
unsubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a un-subscription to a specific object.

5.5. Publishing Objective Option

In fragmentary CDDL, a Publish Objective Option follows the pattern:

```
publish-objection-option = [PUBLISH, 2, 2, pubobj]
```

```
objective-name = PUBLISH
```

```
objective-flags = 2
```

```
loop-count = 2
```

```
pubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a publish of a specific object data.

6. Security Considerations

The distribution source authentication could be done at multiple layers:

- o Outer layer authentication: the GRASP communication is within ACP ([I-D.ietf-anima-autonomic-control-plane]). This is the default GRASP behavior.
- o Inner layer authentication: the GRASP communication might not be within a protected channel, then there should be embedded protection in distribution information itself. Public key infrastructure might be involved in this case.

7. IANA Considerations

TBD.

8. Acknowledgements

Valuable comments were received from Brian Carpenter, Michael Richardson, Roland Bless, Mohamed Boucadair, Diego Lopez, Toerless Eckert and other participants in the ANIMA working group.

This document was produced using the xml2rfc tool [RFC2629].

9. References

9.1. Normative References

- [I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-15 (work in progress), July 2017.
- [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>>.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<https://www.rfc-editor.org/info/rfc2629>>.

9.2. Informative References

- [I-D.carpenter-anima-grasp-bulk]
Carpenter, B., Jiang, S., and B. Liu, "Transferring Bulk Data over the GeneRIC Autonomic Signaling Protocol (GRASP)", draft-carpenter-anima-grasp-bulk-05 (work in progress), January 2020.
- [I-D.du-anima-an-intent]
Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", draft-du-anima-an-intent-05 (work in progress), February 2017.
- [I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-22 (work in progress), February 2020.
- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-35 (work in progress), February 2020.
- [I-D.ietf-anima-grasp-api]
Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-ietf-anima-grasp-api-04 (work in progress), October 2019.
- [I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-10 (work in progress), November 2018.
- [RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <<https://www.rfc-editor.org/info/rfc7575>>.

Appendix A. Open Issues [RFC Editor: To Be removed before becoming RFC]

1. More reference to the use cases in the introduction.
2. Better explanation of the required context of the Connected-Car case: Not applicable unless the ACP will be extended to the car,

which may not be desirable with the current ACP design, but maybe refocussing on an "autonomous fleet" use-case (e.g.: all cars operated by some taxi like service).

3. Consider use-case/example of firmware update. By abstracting the location of the firmware from the name of the firmware, while providing a way to notify about it, this significantly supports distribution of firmware updates. References to SUIT would be appropriate.
4. Issues discussed in https://mailarchive.ietf.org/arch/msg/anima/_0fYQPBcLPt8xzQee7P4dILsn3A
5. Rethink/refine terminology, e.g.: "module" seems to be too prescriptive. Refine proposed extensions.
6. Provide more protocol behavior description instead of only implementation / software module architecture description. Reduce the latter or provide better justification for their presence due to explained interoperability requirements.
7. Better motivation in sections 1..4 of the proposed extensions
8. Consider moving examples from appendices into core-text. Ideally craft a single use-case showing/applying all extensions (most simple use case that uses them all).
9. Refine terminology to better match/reuse-the established terminology from the pre-existing ANIMA documents.

Appendix B. Closed Issues [RFC Editor: To Be removed before becoming RFC]

Appendix C. Change log [RFC Editor: To Be removed before becoming RFC]

draft-ietf-anima-grasp-distribution-00, 2020-02-25:

File name changed following WG adoption.

__Added appendix A&B for open/closed issues. The open issues were comments received during the adoption call.

Appendix D. Real-world Use Cases of Information Distribution

The requirement analysis in Section 3 shows that generally information distribution should be better off as an infrastructure layer module, which provides to upper layer utilizations. In this section, we review some use cases from the real-world where an

information distribution module with powerful functions do plays a critical role there.

D.1. Service-Based Architecture (SBA) in 3GPP 5G

In addition to Internet, the telecommunication network (i.e. carrier mobile wireless networks) is another world-wide networking system. The architecture of the 5G mobile networks from 3GPP has been defined to follow a service-based architecture (SBA) where any network function (NF) can be dynamically associated with any other NF(s) when needed to compose a network service. Note that one NF can simultaneously associate with multiple other NFs, instead of being physically wired as in the previous generations of mobile networks. NFs communicate with each other over service-based interface (SBI), which is also standardized by 3GPP [3GPP.23.501].

In order to realize an SBA network system, detailed requirements are further defined to specify how NFs should interact with each other with information exchange over the SBI. We now list three requirements that are related to information distribution here.

- 1) NF Pub/Sub: Any NF should be able to expose its service status to the network and any NF should be able to subscribe the service status of an NF and get notified if the status is available. A concrete example is that a session management function (SMF) can subscribe to the REGISTER notification from an access management function (AMF) if there is a new user equipment trying to access the mobile network [3GPP.23.502].
- 2) Network Exposure Function (NEF): A particular network function that is required to manage the event exposure and distributions. Specifically, SBA requires such a functionality to register network events from the other NFs (e.g. AMF, SMF and so on), classify the events and properly handle event distributions accordingly in terms of different criteria (e.g. priorities) [3GPP.23.502].
- 3) Network Repository Function (NRF): A particular network function where all service status information is stored for the whole network. An SBA network system requires all NFs to be stateless so as to improve the resilience as well as agility of providing network services. Therefore, the information of the available NFs and the service status generated by those NFs will be globally stored in NRF as a repository of the system. This clearly implies storage capability that keeps the information in the network and provides those information when needed. A concrete example is that whenever a new NF comes up, it first of all registers itself at NRF with its profile. When a network service requires a

certain NF, it first inquires NRF to retrieve the availability information and decides whether or not there is an available NF or a new NF must be instantiated [3GPP.23.502].

(Note: 3GPP CT adopted HTTP2.0/JSON to be the protocol communicating between NFs, but autonomic networks can also load HTTP2.0 with in ACP.)

D.2. Vehicle-to-Everything (V2X)

Connected car is one of scenarios interested in automotive manufacturers, carriers and vendors. 5G Automotive Alliance - an industry collaboration organization defines many promising use cases where services from car industry should be supported by the 5G mobile network. Here we list two examples as follows [5GAA.use.cases].

- 1) Software/Firmware Update: Car manufacturers expect that the software/firmware of their car products can be remotely updated/upgraded via 5G network, instead of onsite visiting their 4S stores/dealers offline as nowadays. This requires the network to provide a mechanism for vehicles to receive the latest software updates during a certain period of time. In order to run such a service for a car manufacturer, the network shall not be just like a network pipe anymore. Instead, information data have to be stored in the network, and delivered in a publishing/subscribing fashion. For example, the latest release of a software will be first distributed and stored at the access edges of the mobile network, after that, the updates can be pushed by the car manufacturer or pulled by the car owner as needed.
- 2) Real-time HD Maps: Autonomous driving clearly requires much finer details of road maps. Finer details not only include the details of just static road and streets, but also real-time information on the road as well as the driving area for both local urgent situations and intelligent driving scheduling. This asks for situational awareness at critical road segments in cases of changing road conditions. Clearly, a huge amount of traffic data that are real-time collected will have to be stored and shared across the network. This clearly requires the storage capability, data synchronization and event notifications in urgent cases from the network, which are still missing at the infrastructure layer.

D.3. Summary

Through the general analysis and the concrete examples from the real-world, we realize that the ways information are exchanged in the coming new scenarios are not just short and instant anymore. More advanced as well as diverse information distribution capabilities are

required and should be generically supported from the infrastructure layer. Upper layer applications (e.g. ASAs in ANIMA) access and utilize such a unified mechanism for their own services.

Appendix E. Information Distribution Module in ANI

This appendix describes how the information distribution module fits into the ANI and what extensions of GRASP are required.

(preamble)

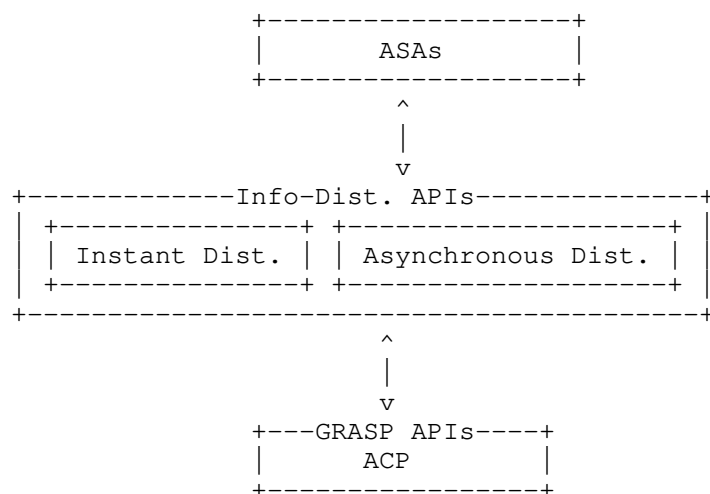


Figure E.1 Information Distribution Module and GRASP Extension.

As the Fig 1 shows, the information distribution module two sub-modules for instant and asynchronous information distributions, respectively, and provides APIs to ASAs. Specific Behaviors of modules are described in Section 5.

Appendix F. Asynchronous ID Integrated with GRASP APIs

Actions triggered to the information distribution module will eventually invoke underlying GRASP APIs. Moreover, EQ and IS modules are usually correlated. When an AF(ASA) publishes information, not only such an event is translated and sent to EQ module, but also the information is indexed and stored simultaneously. Similarly, when an AF(ASA) subscribes information, not only subscribing event is triggered and sent to EQ module, but also the information will be retrieved by IS module at the same time.

- o Storing and publishing information: This action involves both IS and EQ modules where a node that can store the information will be discovered first and related event will be published to the network. For this, GRASP APIs `discover()`, `synchronize()` and `flood()` are combined to compose such a procedure. In specific, `discover()` call will specify its objective being to "store_data" and the return parameters could be either an `ASA_locator` who will accept to store the data, or an error code indicating that no one could afford such data; after that, `synchronize()` call will send the data to the specified `ASA_locator` and the data will be stored at that node, with return of processing results like `store_data_ack`; meanwhile, such a successful event (i.e. data is stored successfully) will be flooded via a `flood()` call to interesting parties (such a multicast group existed).
- o Subscribing and getting information: This action involves both IS and EQ modules as well where a node that is interested in a topic will subscribe the topic by triggering EQ module and if the topic is ready IS module will retrieve the content of the topic (i.e. the data). GRASP APIs such as `register_objective()`, `flood()`, `synchronize()` are combined to compose the procedure. In specific, any subscription action received by EQ module will be translated to `register_objective()` call where the interested topic will be the parameter inside of the call; the registration will be (selectively) flooded to the network by an API call of `flood()` with the option we extended in this draft; once a matched topic is found (because of the previous procedure), the node finding such a match will call API `synchronize()` to send the stored data to the subscriber.

Authors' Addresses

Bing Liu
Huawei Technologies
Q5, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com

Xun Xiao
MRC, Huawei Technologies
German Research Center
Huawei Technologies
Riesstr. 25
Muenchen 80992
Germany

Email: xun.xiao@huawei.com

Artur Hecker
MRC, Huawei Technologies
German Research Center
Huawei Technologies
Riesstr. 25
Muenchen 80992
Germany

Email: artur.hecker@huawei.com

Sheng Jiang
Huawei Technologies
Q27, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: jiangsheng@huawei.com

Zoran Despotovic
MRC, Huawei Technologies
German Research Center
Huawei Technologies
Riesstr. 25
Muenchen 80992
Germany

Email: zoran.despotovic@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 5, 2021

B. Liu (Ed.)
Huawei Technologies
X. Xiao (Ed.)
A. Hecker
MRC, Huawei Technologies
S. Jiang
Huawei Technologies
Z. Despotovic
MRC, Huawei Technologies
B. Carpenter
Univ. of Auckland
September 1, 2020

Information Distribution over GRASP
draft-ietf-anima-grasp-distribution-01

Abstract

This document proposes a solution for information distribution in the autonomic network infrastructure (ANI). Information distribution is categorized into two different modes: 1) instantaneous distribution and 2) publication for retrieval. In the former case, the information is sent, propagated and disposed of after reception. In the latter case, information needs to be stored in the network.

The capability to distribute information is a basic and fundamental need for an autonomous network ([RFC7575]). This document describes typical use cases of information distribution in ANI and requirements to ANI, such that rich information distribution can be natively supported. The document proposes extensions to the autonomic nodes and suggests an implementation based on GRASP ([I-D.ietf-anima-grasp]) extensions as a protocol on the wire.

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 March 5, 2021.

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 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
2. Terminology	4
3. Requirements for Information Distribution in ANI	4
4. Node Behaviors	7
4.1. Instant Information Distribution (IID) Sub-module	7
4.1.1. Instant P2P Communication	7
4.1.2. Instant Flooding Communication	7
4.2. Asynchronous Information Distribution (AID) Sub-module	8
4.2.1. Information Storage	9
4.2.2. Event Queue	11
4.3. Bulk Information Transfer	12
4.4. Summary	14
5. Extending GRASP for Information Distribution	15
5.1. New M_UNSOLIDSYNCH message for Instant P2P Transmission	15
5.2. New O_SELECTIVE_FLOOD option for Selective Flooding	15
5.3. New O_SUBSCRIPTION Objective Option	16
5.4. New O_UNSUBSCRIBE Objective Option	16
5.5. New O_PUBLISH Objective Option	16
6. Security Considerations	17
7. IANA Considerations	17
8. Acknowledgements	17
9. References	17
9.1. Normative References	17
9.2. Informative References	18
Appendix A. Open Issues [RFC Editor: To Be removed before becoming RFC]	19
Appendix B. Closed Issues [RFC Editor: To Be removed before becoming RFC]	20
Appendix C. Change log [RFC Editor: To Be removed before becoming RFC]	

becoming RFC]	20
Appendix D. Implementation Examples and Considerations	20
D.1. GRASP Bulk Transport	20
D.2. Asynchronous ID Integrated with GRASP APIs	23
Appendix E. Real-world Use Cases of Information Distribution	23
E.1. Pub/Sub in 3GPP 5G Networks	24
E.2. Event Queue/Storage in Vehicle-to-Everything (V2X)	25
E.3. Selective Flooding	25
E.4. Summary	27
Appendix F. Information Distribution Module in ANI	27
Authors' Addresses	28

1. Introduction

In an autonomic network, autonomic functions (AFs) running on autonomic nodes constantly exchange information, e.g. AF control/management signaling or AF data exchange. This document discusses the information distribution capability of such exchanges between AFs.

Depending on the number of participants, the information can be distributed in in the following scenarios:

- 1) Point-to-point (P2P) Communication: information is exchanged between parties, i.e. two nodes.
- 2) One-to-Many Communication: information exchanges involve an information source and multiple receivers.

The approaches to information distribution can be mainly categorized into two basic modes:

- 1) An instantaneous mode (push): a source sends the actual content (e.g. control/management signaling, synchronization data and so on) to all interested receiver(s) immediately. Generally, some preconfiguration is required, as nodes interested in this information must be already known to all nodes in the sense that any receiving node must be able to decide, to which nodes this data is to be sent.
- 2) An asynchronous mode (delayed pull): here, a source publishes the content in some form in the network, which may later be looked for, found and retrieved by some endpoints in the AN. Here, depending on the size of the content, either the whole content or only its metadata might be published into the AN. In the latter case the metadata (e.g. a content descriptor, e.g. a key, and a location in the ANI) may be used for the actual retrieval. Importantly, the source, i.e. here publisher, needs to be able to

determine the node, where the information (or its metadata) can be stored.

Note that in both cases, the total size of transferred information can be larger than the payload size of a single GRASP message fitted in one Synchronization and Flood message. In this situation, this document also considers a case of bulk data transfer. To avoid repetitive implementations by each AF developer, this document opts for a common support for information distribution implemented as a basic ANI capability, therefore available to all AFs. In fact, GRASP already provides part of the capabilities.

Regardless, an AF may still define and implement its own information distribution capability. Such a capability may then be advertised using the common information distribution capability defined in this document. Overall, ANI nodes and AFs may decide, which of the information distribution mechanisms they want to use for which type of information, according to their own preferences (e.g. semantic routing table, etc.)

This document first analyzes requirements for information distribution in autonomic networks (Section 3) and then discuss the relevant node behavior (Section 4). After that, the required GRASP extensions are formally introduced (Section 5).

and relevant

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Requirements for Information Distribution in ANI

The question of information distribution in an autonomic network can be discussed through particular use cases or more generally. Depending on the situation it can be quite simple or might require more complex provisions.

Indeed, in the most general case, the information can be sent:

- 1) at once (in one or multiple packets, in one flow),
- 2) straightaway (send-and-forget),
- 3) to all nodes.

For the first scenario, presuming 1), 2) and 3) hold, information distribution in smaller or scarce topologies can be implemented using broadcast, i.e. unconstrained flooding. For reasons well-understood, this approach has its limits in larger and denser networks. In this case, a graph can be constructed such that it contains every node exactly once (e.g. a spanning tree), still allowing to distribute any information to all nodes straightaway. Multicast tree construction protocols could be used in this case. There are reasonable use cases for such scenarios, as presented in Appendix E.

Secondly, a more complex scenario arises, if only 1) and 2) hold, but the information only concerns a subset of nodes. Then, some kinds of selection become required, to which nodes the given information should be distributed. Here, a further distinction is necessary; notably, if the selection of the target nodes is with respect to the nature or position of the node, or whether it is with respect to the information content. If the first, some knowledge about the node types, its topological position, etc (e.g. the routing information within ANI) can be used to distinguish nodes accordingly. For instance, edge nodes and forwarding nodes can be distinguished in this way. If the distribution scope is primarily to be defined by the information elements, then a registration / join / subscription or label distribution mechanism is unavoidable. This would be the case, for instance, if the AFs can be dynamically deployed on nodes, and the information is majorily destined to the AFs. Then, depending on the current AF deployment, the distribution scope must be adjusted as well.

Thirdly, if only 1) holds, but the information content might be required again and again, or might not yet be fully available, then more complex mechanisms might be required to store the information within the network for later, for further redistribution, and for notification of interested nodes. Examples for this include distribution of reconfiguration information for different AF instances, which might not require an immediate action, but only an eventual update of the parameters. Also, in some situations, there could be a significant delay between the occurrence of a new event and the full content availability (e.g. if the processing requires a lot of time).

Finally, none of the three might hold. Then, along with the subscription and notification, the actual content might be different from its metadata, i.e. some descriptions of the content and, possibly, its location. The fetching can then be implemented in different, appropriate ways, if necessary as a complex transport session.

In essence, as flooding is usually not an option, and the interest of nodes for particular information elements can change over time, ANI should support autonomies also for the information distribution.

This calls for autonomic mechanisms in the ANI, allowing participating nodes to 1) advertise/publish, 2) look for/subscribe to 3) store, 4) fetch/retrieve and 5) instantaneously push data information.

In the following cases, situations depicting complicated ways of information distribution are discussed.

- 1) Long Communication Intervals. The actual sending of the information is not necessarily instantaneous with some events. Sophisticated AFs may involve into longer jobs/tasks (e.g. database lookup, validations, etc.) when processing requests, and might not be able to reply immediately. Instead of actively waiting for the reply, a better way for an interested AF might be to get notified, when the reply is finally available.
- 2) Common Interest Distribution. AFs may share information that is a common interest. For example, the network intent will be distributed to network nodes enrolled, which is usually one-to-many scenario. Intent distribution can also be performed by an instant flooding (e.g. via GRASP) to every network node. However, because of network changes, not every node can be just ready at the moment when the network intent is broadcast. Also, a flooding often does not cover all network nodes as there is usually a limitation on the hop number. In fact, nodes may join in the network sequentially. In this situation, an asynchronous communication model could be a better choice where every (newly joining) node can subscribe the intent information and will get notified if it is ready (or updated).
- 3) Distributed Coordination. With computing and storage resources on autonomic nodes, alive AFs not only consume but also generate data information. An example is AFs coordinating with each other as distributed schedulers, responding to service requests and distributing tasks. It is critical for those AFs to make correct decisions based on local information, which might be asymmetric as well. AFs may also need synthetic/aggregated data information (e.g. statistic info, like average values of several AFs, etc.) to make decisions. In these situations, AFs will need an efficient way to form a global view of the network (e.g. about resource consumption, bandwidth and statistics). Obviously, purely relying on instant communication model is inefficient, while a scalable, common, yet distributed data layer, on which AFs

can store and share information in an asynchronous way, should be a better choice.

Therefore, for ANI, in order to support various communication scenarios, an information distribution module is required, and both instantaneous and asynchronous communication models should be supported. Some real-world use cases are introduced in Appendix E.

4. Node Behaviors

In this section, how a node should behave in order to support the two identified modes of information distribution is discussed. An ANI is a distributed system, so the information distribution module must be implemented in a distributed way as well.

4.1. Instant Information Distribution (IID) Sub-module

In this case, an information sender directly specifies the information receiver(s). The instant information distribution sub-module will be the main element.

4.1.1. Instant P2P Communication

IID sub-module performs instant information transmission for ASAs running in an ANI. In specific, IID sub-module will have to retrieve the address of the information receiver specified by an ASA, then deliver the information to the receiver. Such a delivery can be done either in a connectionless or a connection-oriented way.

Current GRASP provides the capability to support instant P2P synchronization for ASAs. A P2P synchronization is a use case of P2P information transmission. However, as mentioned in Section 3, there are some scenarios where one node needs to transmit some information to another node(s). This is different to synchronization because after transmitting the information, the local status of the information does not have to be the same as the information sent to the receiver. This is not directly support by existing GRASP.

4.1.2. Instant Flooding Communication

IID sub-module finishes instant flooding for ASAs in an ANI. Instant flooding is for all ASAs in an ANI. An information sender has to specify a special destination address of the information and broadcast to all interfaces to its neighbors. When another IID sub-module receives such a broadcast, after checking its TTL, it further broadcast the message to the neighbors. In order to avoid flooding storms in an ANI, usually a TTL number is specified, so that after a

pre-defined limit, the flooding message will not be further broadcast again.

In order to avoid unnecessary flooding, a selective flooding can be done where an information sender wants to send information to multiple receivers at once. When doing this, sending information needs to contain criteria to judge on which interfaces the distributed information should and should not be sent. Specifically, the criteria contain:

- o Matching Condition: a set of matching rules such as addresses of recipients, node features and so on.
- o Matching object: the object that the match condition would be applied to. For example, the matching object could be node itself or its neighbors.
- o Action: what the node needs to do when the Matching Condition is fulfilled. For example, the action could be forwarding or discarding the distributed message.

Sent information must be included in the message distributed from the sender. The receiving node reacts by first checking the carried Matching Condition in the message to decide who should consume the message, which could be either the node itself, some neighbors or both. If the node itself is a recipient, Action field is followed; if a neighbor is a recipient, the message is sent accordingly.

An exemplary extension to support selective flooding on GRASP is described in Section 5.

4.2. Asynchronous Information Distribution (AID) Sub-module

In asynchronous information distribution, sender(s) and receiver(s) are not immediately specified while they may appear in an asynchronous way. Firstly, AID sub-module enables that the information can be stored in the network; secondly, AID sub-module provides an information publication and subscription (Pub/Sub) mechanism for ASAs.

As sketched in the previous section, in general each node requires two modules: 1) Information Storage (IS) module and 2) Event Queue (EQ) module in the information distribution module. Details of the two modules are described in the following sections.

4.2.1. Information Storage

IS module handles how to save and retrieve information for ASAs across the network. The IS module uses a syntax to index information, generating the hash index value (e.g. a hash value) of the information and mapping the hash index to a certain node in ANI. Note that, this mechanism can use existing solutions. Specifically, storing information in an ANIMA network will be realized in the following steps.

- 1) ASA-to-IS Negotiation. An ASA calls the API provided by information distribution module (directly supported by IS sub-module) to request to store the information somewhere in the network. The IS module performs various checks of the request (e.g. permitted information size).
- 2) Storing Peer Mapping. The information block will be handled by the IS module in order to calculate/map to a peer node in the network. Since ANIMA network is a peer-to-peer network, a typical way is to use distributed hash table (DHT) to map information to a unique index identifier. For example, if the size of the information is reasonable, the information block itself can be hashed, otherwise, some meta-data of the information block can be used to generate the mapping.
- 3) Storing Peer Negotiation Request. Negotiation request of storing the information will be sent from the IS module to the IS module on the destination node. The negotiation request contains parameters about the information block from the source IS module. According to the parameters as well as the local available resource, the requested storing peer will send feedback the source IS module.
- 4) Storing Peer Negotiation Response. Negotiation response from the storing peer is sent back to the source IS module. If the source IS module gets confirmation that the information can be stored, source IS module will prepare to transfer the information block; otherwise, a new storing peer must be discovered (i.e. going to step 7).
- 5) Information Block Transfer. Before sending the information block to the storing peer that already accepts the request, the IS module of the source node will check if the information block can be afforded by one GRASP message. If so, the information block will be directly sent by calling a GRASP API ([I-D.ietf-anima-grasp-api]). Otherwise, a bulk data transmission is needed. For that, there are multiple ways to do it. The first option is to utilize one of existing protocols that is independent

of the GRASP stack. For example, a session connectivity can be established to the storing peer, and over the connection the bulky data can be transmitted part by part. In this case, the IS module should support basic TCP-based session protocols such as HTTP(s) or native TCP. The second option is to directly use GRASP itself for bulky data transferring [I-D.carpenter-anima-grasp-bulk].

- 6) Information Writing. Once the information block (or a smaller block) is received, the IS module of the storing peer will store the data block in the local storage is accessible.
- 7) (Optional) New Storing Peer Discovery. If the previously selected storing peer is not available to store the information block, the source IS module will have to identify a new destination node to start a new negotiation. In this case, the discovery can be done by using discovery GRASP API to identify a new candidate, or more complex mechanisms can be introduced.

Similarly, Getting information from an ANI will be realized in the following steps.

- 1) ASA-to-IS Request. An ASA accesses the IS module via the APIs exposed by the information distribution module. The key/index of the interested information will be sent to the IS module. An assumption here is that the key/index should be known to an ASA before an ASA can ask for the information. This relates to the publishing/subscribing of the information, which are handled by other modules (e.g. Event Queue with Pub/Sub supported by GRASP).
- 2) Storing Peer Mapping. IS module maps the key/index of the requested information to a peer that stores the information, and prepares the information request. The mapping here follows the same mechanism when the information is stored.
- 3) Retrieval Negotiation Request. The source IS module sends a request to the storing peer and asks if such an information object is available.
- 4) Retrieval Negotiation Response. The storing peer checks the key/index of the information in the request, and replies to the source IS module. If the information is found and the information block can be afforded within one GRASP message, the information will be sent together with the response to the source IS module.
- 5) (Optional) New Destination Request. If the information is not found after the source IS module gets the response from the originally identified storing peer, the source IS module will have to discover the location of the requested information.

IS module can reuse distributed databases and key value stores like NoSQL, Cassandra, DHT technologies. storage and retrieval of information are all event-driven responsible by the EQ module.

4.2.2. Event Queue

The Event Queue (EQ) module is to help ASAs to publish information to the network and subscribe to interested information in asynchronous scenarios. In an ANI, information generated on network nodes is an event labeled with an event ID, which is semantically related to the topic of the information. Key features of EQ module are summarized as follows.

- 1) Event Group: An EQ module provides isolated queues for different event groups. If two groups of AFs could have completely different purposes, the EQ module allows to create multiple queues where only AFs interested in the same topic will be aware of the corresponding event queue.
- 2) Event Prioritization: Events can have different priorities in ANI. This corresponds to how much important or urgent the event implies. Some of them are more urgent than regular ones. Prioritization allows AFs to differentiate events (i.e. information) they publish or subscribe to.
- 3) Event Matching: an information consumer has to be identified from the queue in order to deliver the information from the provider. Event matching keeps looking for the subscriptions in the queue to see if there is an exact published event there. Whenever a match is found, it will notify the upper layer to inform the corresponding ASAs who are the information provider and subscriber(s) respectively.

The EQ module on every network node operates as follows.

- 1) Event ID Generation: If information of an ASA is ready, an event ID is generated according to the content of the information. This is also related to how the information is stored/saved by the IS module introduced before. Meanwhile, the type of the event is also specified where it can be of control purpose or user plane data.
- 2) Priority Specification: According to the type of the event, the ASA may specify its priority to say how this event is to be processed. By considering both aspects, the priority of the event will be determined.

- 3) Event Enqueue: Given the event ID, event group and its priority, a queue is identified locally if all criteria can be satisfied. If there is such a queue, the event will be simply added into the queue, otherwise a new queue will be created to accommodate such an event.
- 4) Event Propagation: The published event will be propagated to the other network nodes in the ANIMA domain. A propagation algorithm can be employed to optimize the propagation efficiency of the updated event queue states.
- 5) Event Match and Notification: While propagating updated event states, EQ module in parallel keeps matching published events and its interested consumers. Once a match is found, the provider and subscriber(s) will be notified for final information retrieval.

The category of event priority is defined as the following. In general, there are two event types:

- 1) Network Control Event: This type of events are defined by the ANI for operational purposes on network control. A pre-defined priority levels for required system messages is suggested. For highest level to lowest level, the priority value ranges from NC_PRIOR_HIGH to NC_PRIOR_LOW as integer values. The NC_PRIOR_* values will be defined later according to the total number system events required by the ANI.
- 2) Custom ASA Event: This type of events are defined by the ASAs of users. This specifies the priority of the message within a group of ASAs, therefore it is only effective among ASAs that join the same message group. Within the message group, a group header/leader has to define a list of priority levels ranging from CUST_PRIOR_HIGH to CUST_PRIOR_LOW. Such a definition completely depends on the individual purposes of the message group. When a system message is delivered, its event type and event priority value have to be both specified.

Event contains the address where the information is stored, after a subscriber is notified, it directly retrieves the information from the given location.

4.3. Bulk Information Transfer

In both cases discussed previously, they are limited to distributing GRASP Objective Options contained in messages that cannot exceed the GRASP maximum message size of 2048 bytes. This places a limit on the size of data that can be transferred directly in a GRASP message such

as a Synchronization or Flood operation for instantaneous information distribution.

There are scenarios in autonomic networks where this restriction is a problem. One example is the distribution of network policy in lengthy formats such as YANG or JSON. Another case might be an Autonomic Service Agent (ASA) uploading a log file to the Network Operations Center (NOC). A third case might be a supervisory system downloading a software upgrade to an autonomic node. A related case might be installing the code of a new or updated ASA to a target node.

Naturally, an existing solution such as a secure file transfer protocol or secure HTTP might be used for this. Other management protocols such as syslog [RFC5424] or NETCONF [RFC6241] might also be used for related purposes, or might be mapped directly over GRASP. The present document, however, applies to any scenario where it is preferable to re-use the autonomic networking infrastructure itself to transfer a significant amount of data, rather than install and configure an additional mechanism.

The node behavior is to use the GRASP Negotiation process to transfer and acknowledge multiple blocks of data in successive negotiation steps, thereby overcoming the GRASP message size limitation. The emphasis is placed on simplicity rather than efficiency, high throughput, or advanced functionality. For example, if a transfer gets out of step or data packets are lost, the strategy is to abort the transfer and try again. In an enterprise network with low bit error rates, and with GRASP running over TCP, this is not considered a serious issue. Clearly, a more sophisticated approach could be designed but if the application requires that, existing protocols could be used, as indicated in the preceding paragraph.

As for any GRASP operation, the two participants are considered to be Autonomic Service Agents (ASAs) and they communicate using a specific GRASP Objective Option, containing its own name, some flag bits, a loop count, and a value. In bulk transfer, we can model the ASA acting as the source of the transfer as a download server, and the destination as a download client. No changes or extensions are required to GRASP itself, but compared to a normal GRASP negotiation, the communication pattern is slightly asymmetric:

- 1) The client first discovers the server by the GRASP discovery mechanism (M_DISCOVERY and M_RESPONSE messages).
- 2) The client then sends a GRASP negotiation request (M_REQ_NEG message). The value of the objective expresses the requested item (e.g., a file name - see the next section for a detailed example).

- 3) The server replies with a negotiation step (M_NEGOTIATE message). The value of the objective is the first section of the requested item (e.g., the first block of the requested file as a raw byte string).
- 4) The client replies with a negotiation step (M_NEGOTIATE message). The value of the objective is a simple acknowledgement (e.g., the text string 'ACK').

The last two steps repeat until the transfer is complete. The server signals the end by transferring an empty byte string as the final value. In this case the client responds with a normal end to the negotiation (M_END message with an O_ACCEPT option).

Errors of any kind are handled with the normal GRASP mechanisms, in particular by an M_END message with an O_DECLINE option in either direction. In this case the GRASP session terminates. It is then the client's choice whether to retry the operation from the start, as a new GRASP session, or to abandon the transfer. The block size must be chosen such that each step does not exceed the GRASP message size limit of 2048 bits.

GRASP bulk transport function doesn't require new GRASP messages/options (as specified in Section 5) to be defined. An implementation example is given in Appendix D.1 .

4.4. Summary

In summary, the general requirements for the information distribution module on each autonomic node are realized by two sub-modules handling instant communications and asynchronous communications, respectively. For instantaneous mode, node requirements are simple, calling for support for additional signaling. With minimum efforts, reusing the existing GRASP is possible.

For asynchronous mode, information distribution module uses new primitives on the wire, and implements an event queue and an information storage mechanism. An architectural consideration on ANI with the information distribution module is briefly discussed in Appendix F.

In both cases, a scenario of bulk information transfer is considered where the retrieved information cannot be fitted in one GRASP message. Based on GRASP Negotiation operation, multiple transmissions can be repeatedly done in order to transfer bulk information piece by piece.

5. Extending GRASP for Information Distribution

5.1. New M_UNSOLIDSYNCH message for Instant P2P Transmission

This could be a new message in GRASP. In fragmentary CDDL, an Un-solicited Synchronization message follows the pattern:

```
unsolicited_synch-message = [M_UNSOLIDSYNCH, session-id,  
objective]
```

A node MAY actively send a unicast Un-solicited Synchronization message with the Synchronization data, to another node. This MAY be sent to port GRASP_LISTEN_PORT at the destination address, which might be obtained by GRASP Discovery or other possible ways. The synchronization data are in the form of GRASP Option(s) for specific synchronization objective(s).

5.2. New O_SELECTIVE_FLOOD option for Selective Flooding

Since normal flooding is already supported by GRASP, this section only defines the selective flooding extension.

In fragmentary CDDL, the selective flooding follows the pattern:

```
selective-flood-option = [O_SELECTIVE_FLOOD, +O_MATCH-CONDITION,  
match-object, action]  
  
O_MATCH-CONDITION = [O_MATCH-CONDITION, Obj1, match-rule, Obj2]  
Obj1 = text  
  
match-rule = GREATER / LESS / WITHIN / CONTAIN  
  
Obj2 = text  
  
match-object = NEIGHBOR / SELF  
  
action = FORWARD / DROP
```

The option field encapsulates a match-condition option which represents the conditions regarding to continue or discontinue flood the current message. For the match-condition option, the Obj1 and Obj2 are to objects that need to be compared. For example, the Obj1 could be the role of the device and Obj2 could be "RSG". The match rules between the two objects could be greater, less than, within, or contain. The match-object represents of which Obj1 belongs to, it could be the device itself or the neighbor(s) intended to be flooded. The action means, when the match rule applies, the current device just continues flood or discontinues.

Some examples of specific O_SELECTIVE_FLOOD option definitions according to some use cases, are described in Appendix E.3 .

5.3. New O_SUBSCRIPTION Objective Option

In fragmentary CDDL, a Subscription Objective Option follows the pattern:

```
subscription-objection-option = [SUBSCRIPTION, 2, 2, subobj]
objective-name = SUBSCRIPTION

objective-flags = 2

loop-count = 2

subobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a subscription to a specific object.

5.4. New O_UNSUBSCRIBE Objective Option

In fragmentary CDDL, a Un_Subscribe Objective Option follows the pattern:

```
Unsubscribe-objection-option = [UNSUBSCRIB, 2, 2, unsubobj]
objective-name = SUBSCRIPTION

objective-flags = 2

loop-count = 2

unsubobj = text
```

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a un-subscription to a specific object.

5.5. New O_PUBLISH Objective Option

In fragmentary CDDL, a Publish Objective Option follows the pattern:

```
publish-objection-option = [PUBLISH, 2, 2, pubobj]
objective-name = PUBLISH
```

objective-flags = 2

loop-count = 2

pubobj = text

This option MAY be included in GRASP M_Synchronization, when included, it means this message is for a publish of a specific object data.

6. Security Considerations

The distribution source authentication could be done at multiple layers:

- o Outer layer authentication: the GRASP communication is within ACP ([I-D.ietf-anima-autonomic-control-plane]). This is the default GRASP behavior.
- o Inner layer authentication: the GRASP communication might not be within a protected channel, then there should be embedded protection in distribution information itself. Public key infrastructure might be involved in this case.

7. IANA Considerations

TBD.

8. Acknowledgements

Valuable comments were received from Michael Richardson, Roland Bless, Mohamed Boucadair, Diego Lopez, Toerless Eckert, Joel Halpern and other participants in the ANIMA working group.

This document was produced using the xml2rfc tool [RFC2629].

9. References

9.1. Normative References

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-15 (work in progress), July 2017.

- [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>>.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<https://www.rfc-editor.org/info/rfc2629>>.
- [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/info/rfc8610>>.

9.2. Informative References

- [I-D.carpenter-anima-grasp-bulk] Carpenter, B., Jiang, S., and B. Liu, "Transferring Bulk Data over the GeneRic Autonomic Signaling Protocol (GRASP)", draft-carpenter-anima-grasp-bulk-05 (work in progress), January 2020.
- [I-D.du-anima-an-intent] Du, Z., Jiang, S., Nobre, J., Ciavaglia, L., and M. Behringer, "ANIMA Intent Policy and Format", draft-du-anima-an-intent-05 (work in progress), February 2017.
- [I-D.ietf-anima-autonomic-control-plane] Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-28 (work in progress), July 2020.
- [I-D.ietf-anima-bootstrapping-keyinfra] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-43 (work in progress), August 2020.
- [I-D.ietf-anima-grasp-api] Carpenter, B., Liu, B., Wang, W., and X. Gong, "Generic Autonomic Signaling Protocol Application Program Interface (GRASP API)", draft-ietf-anima-grasp-api-06 (work in progress), June 2020.

[I-D.ietf-anima-reference-model]

Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-10 (work in progress), November 2018.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <<https://www.rfc-editor.org/info/rfc5424>>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <<https://www.rfc-editor.org/info/rfc7575>>.

Appendix A. Open Issues [RFC Editor: To Be removed before becoming RFC]

1. More reference to the use cases in the introduction.
2. Better explanation of the required context of the Connected-Car case: Not applicable unless the ACP will be extended to the car, which may not be desirable with the current ACP design, but maybe refocussing on an "autonomous fleet" use-case (e.g.: all cars operated by some taxi like service).
3. Consider use-case/example of firmware update. By abstracting the location of the firmware from the name of the firmware, while providing a way to notify about it, this significantly supports distribution of firmware updates. References to SUIT would be appropriate.
4. Issues discussed in https://mailarchive.ietf.org/arch/msg/anima/_0fYQPBcLPt8xzQee7P4dILsn3A
5. Rethink/refine terminology, e.g.: "module" seems to be too prescriptive. Refine proposed extensions.
6. Provide more protocol behavior description instead of only implementation / software module architecture description. Reduce the latter or provide better justification for their presence due to explained interoperability requirements.

7. Better motivation in sections 1..4 of the proposed extensions
8. Consider moving examples from appendices into core-text . Ideally craft a single use-case showing/applying all extensions (most simple use case that uses them all).
9. Refine terminology to better match/reuse-the established terminology from the pre-existing ANIMA documents.

Appendix B. Closed Issues [RFC Editor: To Be removed before becoming RFC]

Appendix C. Change log [RFC Editor: To Be removed before becoming RFC]

draft-ietf-anima-grasp-distribution-01, 2020-09-01:

Merged some essential content of draft-carpenter-anima-grasp-bulk-05.

—

Adjusted appendix structure and content.

draft-ietf-anima-grasp-distribution-00, 2020-02-25:

File name changed following WG adoption.

__Added appendix A&B for open/closed issues. The open issues were comments received during the adoption call.

Appendix D. Implementation Examples and Considerations

D.1. GRASP Bulk Transport

Example for file transfer: this example describes a client ASA requesting a file download from a server ASA.

Firstly we define a GRASP objective informally: ["411:mvFile", 3, 6, value]

The formal CDDL definition [RFC8610] is:

- o mvfile-objective = ["411:mvFile", objective-flags, loop-count, value]
- o objective-flags = ; as in the GRASP specification loop-count = ; as in the GRASP specification value = any

The objective-flags field is set to indicate negotiation. Dry run mode must not be used. The loop-count is set to a suitable value to limit the scope of discovery. A suggested default value is 6.

The value takes the following forms:

- o In the initial request from the client, a UTF-8 string containing the requested file name (with file path if appropriate).
- o In negotiation steps from the server, a byte string containing at most 1024 bytes. However:
 - * If the file does not exist, the first negotiation step will return an M_END, O_DECLINE response.
 - * After sending the last block, the next and final negotiation step will send an empty byte string as the value.
- o In negotiation steps from the client, the value is the UTF-8 string 'ACK'.

Note that the block size of 1024 is chosen to guarantee not only that each GRASP message is below the size limit, but also that only one TCP data packet will be needed, even on an IPv6 network with a minimum link MTU.

We now present outline pseudocode for the client and the server ASA. The API documented in [I-D.ietf-anima-grasp-api] is used in a simplified way, and error handling is not shown in detail.

Pseudo code for client ASA (request and receive a file):

```
requested_obj = objective('411:mvFile')
locator = discover(requested_obj)
requested_obj.value = 'etc/test.json'
received_obj = request_negotiate(requested_obj, locator)
if error_code == declined:
    #no such file
    exit

file = open(requested_obj.value)
file.write(received_obj.value) #write to file
eof = False
while not eof:
    received_obj.value = 'ACK'
    received_obj.loop_count = received_obj.loop_count + 1
    received_obj = negotiate_step(received_obj)
    if received_obj.value == null:
        end_negotiate(True)
        file.close()
        eof = True
    else:
        file.write(received_obj.value) #write to file

#file received
exit
```

Pseudo code for server ASA (await request and send a file):

```
supported_obj = objective('411:mvFile')
requested_obj = listen_negotiate(supported_obj)
file = open(requested_obj.value) #open the source file
if no such file:
    end_negotiate(False) #decline negotiation
    exit

eof = False
while not eof:
    chunk = file.read(1024) #next block of file
    requested_obj.value = chunk
    requested_obj.loop_count = requested_obj.loop_count + 1
    requested_obj = negotiate_step(requested_obj)
    if chunk == null:
        file.close()
        eof = True
        end_negotiate(True)
        exit
    if requested_obj.value != 'ACK':
        #unexpected reply...
```


D.2. Asynchronous ID Integrated with GRASP APIs

Actions triggered to the information distribution module will eventually invoke underlying GRASP APIs. Moreover, EQ and IS modules are usually correlated. When an AF(ASA) publishes information, not only such an event is translated and sent to EQ module, but also the information is indexed and stored simultaneously. Similarly, when an AF(ASA) subscribes information, not only subscribing event is triggered and sent to EQ module, but also the information will be retrieved by IS module at the same time.

- o Storing and publishing information: This action involves both IS and EQ modules where a node that can store the information will be discovered first and related event will be published to the network. For this, GRASP APIs `discover()`, `synchronize()` and `flood()` are combined to compose such a procedure. In specific, `discover()` call will specify its objective being to "store_data" and the return parameters could be either an `ASA_locator` who will accept to store the data, or an error code indicating that no one could afford such data; after that, `synchronize()` call will send the data to the specified `ASA_locator` and the data will be stored at that node, with return of processing results like `store_data_ack`; meanwhile, such a successful event (i.e. data is stored successfully) will be flooded via a `flood()` call to interesting parties (such a multicast group existed).
- o Subscribing and getting information: This action involves both IS and EQ modules as well where a node that is interested in a topic will subscribe the topic by triggering EQ module and if the topic is ready IS module will retrieve the content of the topic (i.e. the data). GRASP APIs such as `register_objective()`, `flood()`, `synchronize()` are combined to compose the procedure. In specific, any subscription action received by EQ module will be translated to `register_objective()` call where the interested topic will be the parameter inside of the call; the registration will be (selectively) flooded to the network by an API call of `flood()` with the option we extended in this draft; once a matched topic is found (because of the previous procedure), the node finding such a match will call API `synchronize()` to send the stored data to the subscriber.

Appendix E. Real-world Use Cases of Information Distribution

The requirement analysis in Section 3 shows that generally information distribution should be better off as an infrastructure layer module, which provides to upper layer utilizations. In this section, we review some use cases from the real-world where an

information distribution module with powerful functions do plays a critical role there.

E.1. Pub/Sub in 3GPP 5G Networks

In addition to Internet, the telecommunication network (i.e. carrier mobile wireless networks) is another world-wide networking system. The architecture of the 5G mobile networks from 3GPP has been defined to follow a service-based architecture (SBA) where any network function (NF) can be dynamically associated with any other NF(s) when needed to compose a network service. Note that one NF can simultaneously associate with multiple other NFs, instead of being physically wired as in the previous generations of mobile networks. NFs communicate with each other over service-based interface (SBI), which is also standardized by 3GPP [3GPP.23.501].

In order to realize an SBA network system, detailed requirements are further defined to specify how NFs should interact with each other with information exchange over the SBI. We now list three requirements that are related to information distribution here.

- 1) NF Pub/Sub: Any NF should be able to expose its service status to the network and any NF should be able to subscribe the service status of an NF and get notified if the status is available. A concrete example is that a session management function (SMF) can subscribe to the REGISTER notification from an access management function (AMF) if there is a new user equipment trying to access the mobile network [3GPP.23.502].
- 2) Network Exposure Function (NEF): A particular network function that is required to manage the event exposure and distributions. Specifically, SBA requires such a functionality to register network events from the other NFs (e.g. AMF, SMF and so on), classify the events and properly handle event distributions accordingly in terms of different criteria (e.g. priorities) [3GPP.23.502].
- 3) Network Repository Function (NRF): A particular network function where all service status information is stored for the whole network. An SBA network system requires all NFs to be stateless so as to improve the resilience as well as agility of providing network services. Therefore, the information of the available NFs and the service status generated by those NFs will be globally stored in NRF as a repository of the system. This clearly implies storage capability that keeps the information in the network and provides those information when needed. A concrete example is that whenever a new NF comes up, it first of all registers itself at NRF with its profile. When a network service requires a

certain NF, it first inquires NRF to retrieve the availability information and decides whether or not there is an available NF or a new NF must be instantiated [3GPP.23.502].

(Note: 3GPP CT adopted HTTP2.0/JSON to be the protocol communicating between NFs, but autonomic networks can also load HTTP2.0 within ACP.)

E.2. Event Queue/Storage in Vehicle-to-Everything (V2X)

Connected car is one of scenarios interested in automotive manufacturers, carriers and vendors. 5G Automotive Alliance - an industry collaboration organization defines many promising use cases where services from car industry should be supported by the 5G mobile network. Here we list two examples as follows [5GAA.use.cases].

- 1) Software/Firmware Update: Car manufacturers expect that the software/firmware of their car products can be remotely updated/upgraded via 5G network, instead of onsite visiting their 4S stores/dealers offline as nowadays. This requires the network to provide a mechanism for vehicles to receive the latest software updates during a certain period of time. In order to run such a service for a car manufacturer, the network shall not be just like a network pipe anymore. Instead, information data have to be stored in the network, and delivered in a publishing/subscribing fashion. For example, the latest release of a software will be first distributed and stored at the access edges of the mobile network, after that, the updates can be pushed by the car manufacturer or pulled by the car owner as needed.
- 2) Real-time HD Maps: Autonomous driving clearly requires much finer details of road maps. Finer details not only include the details of just static road and streets, but also real-time information on the road as well as the driving area for both local urgent situations and intelligent driving scheduling. This asks for situational awareness at critical road segments in cases of changing road conditions. Clearly, a huge amount of traffic data that are real-time collected will have to be stored and shared across the network. This clearly requires the storage capability, data synchronization and event notifications in urgent cases from the network, which are still missing at the infrastructure layer.

E.3. Selective Flooding

Example 1: Selected flooding in hierarchical network:

- o E.g. IPRAN network, which is normally highly hierarchical: large amount of access gateways (CSG) at the low layer, but limited

aggregation gateways (ASG) and core network gateways (RSG) at the upper layer.

- o Some information is not necessary to flood to the CSGs. (E.g. a network policy of VPN mechanisms selection)

In this case, the Selective Flooding Criteria could be defined as:

- o Matching condition: Role=RSG or ASG
- o Matching object: Neighbor devices
- o Action:
 - * If the one neighbor device's "Role" matches the Matching Condition, which is "RSG or ASG", then the node would forward the message to that neighbor.
 - * If not, then the node would discard the message for that neighbor.

Example 2: Selected flooding within a deterministic path:

- o E.g. flood within a MPLS LSP
- o The LSP has been set up
- o One node distributes the information to all the LSRs of the LSP. (e.g. adjust the reserved bandwidth)

In this case, the Selective Flooding Criteria could be defined as:

- o Matching condition: vpn-instance=WCDMA-VPN
- o Matching object: interfaces
- o Action:
 - * If the interface's "vpn-instance" matches the Matching Condition, which is "WCDMA-VPN", then the node would forward the message to that interface.
 - * If not, then the node would discard the message for that interface.

Example 3: Selected flooding for ACP set up:

- o ACP topology should align with the physical topology as much as possible
- o An Anima-Enabled switch should not forwarding the ACP discovery to the nodes attached to it

In this case, the Selective Flooding Criteria could be defined as:

- o Matching condition: Role=switch
- o Matching object: self
- o Action:
 - * If the "Role" of the node itself matches the Matching Condition, which is "switch", then the node would discard the message.
 - * If not, then the node would continue the flood.

E.4. Summary

Through the general analysis and the concrete examples from the real-world, we realize that the ways information are exchanged in the coming new scenarios are not just short and instant anymore. More advanced as well as diverse information distribution capabilities are required and should be generically supported from the infrastructure layer. Upper layer applications (e.g. ASAs in ANIMA) access and utilize such a unified mechanism for their own services.

Appendix F. Information Distribution Module in ANI

This appendix describes how the information distribution module fits into the ANI and what extensions of GRASP are required.

(preamble)

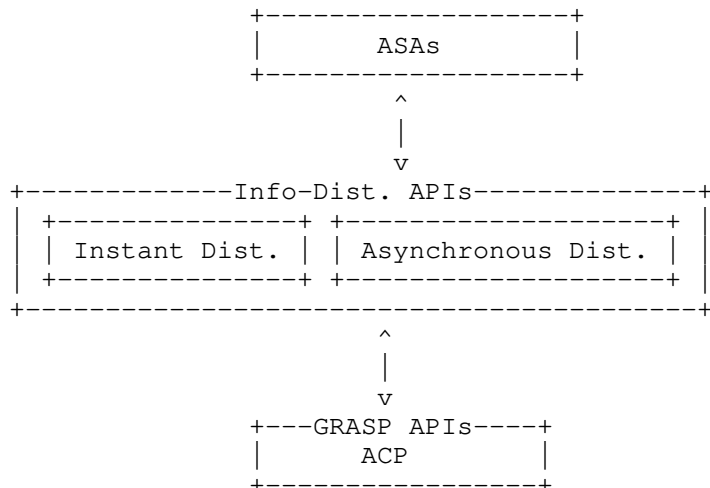


Figure E.1 Information Distribution Module and GRASP Extension.

As the Fig 1 shows, the information distribution module two sub-modules for instant and asynchronous information distributions, respectively, and provides APIs to ASAs. Specific Behaviors of modules are described in Section 5.

Authors' Addresses

Bing Liu
 Huawei Technologies
 Q5, Huawei Campus
 No.156 Beiqing Road
 Hai-Dian District, Beijing 100095
 P.R. China

Email: leo.liubing@huawei.com

Xun Xiao
 MRC, Huawei Technologies
 German Research Center
 Huawei Technologies
 Riesstr. 25
 Muenchen 80992
 Germany

Email: xun.xiao@huawei.com

Artur Hecker
MRC, Huawei Technologies
German Research Center
Huawei Technologies
Riesstr. 25
Muenchen 80992
Germany

Email: artur.hecker@huawei.com

Sheng Jiang
Huawei Technologies
Q27, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: jiangsheng@huawei.com

Zoran Despotovic
MRC, Huawei Technologies
German Research Center
Huawei Technologies
Riesstr. 25
Muenchen 80992
Germany

Email: zoran.despotovic@huawei.com

Brian E. Carpenter
University of Auckland
School of Computer Science
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 June 2021

M. Richardson
Sandelman Software Works
T. Werner
Siemens
22 December 2020

JOSE signed Voucher Artifacts for Bootstrapping Protocols
draft-richardson-anima-jose-voucher-00

Abstract

This document describes a serialiation of the RFC8366 voucher format to a JSON format is then signed using the JSON Object Signing and Encryption mechanism described in RFC7515.

In addition to explaining how the format is created, MIME types are registered and examples are provided.

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 25 June 2021.

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 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	2
2. Terminology	3
3. JSON Web Signatures	3
3.1. Unprotected Header	4
3.2. Protected Header	4
4. Privacy Considerations	4
5. Security Considerations	4
6. IANA Considerations	4
6.1. Media-Type Registry	4
6.1.1. application/voucher-jose+json	4
7. Acknowledgements	5
8. Changelog	5
9. References	5
9.1. Normative References	5
9.2. Informative References	6
Appendix A. Examples	7
A.1. Example Voucher Request (from Pledge to Registrar) . . .	7
A.2. Example Parboiled Voucher Request (from Registrar to MASA)	9
A.3. Example Voucher Result (from MASA to Pledge, via Registrar)	12
Authors' Addresses	15

1. Introduction

[RFC8366] describes a voucher artifact used in [BRSKI] and [RFC8572] to transfer ownership of a device to from a manufacturer to an owner. That document defines the base YANG module, and also the initial serialization to JSON [RFC8259], with a signature provided by [RFC5652].

Other work, [I-D.ietf-anima-constrained-voucher] provides a mapping of the YANG to CBOR [RFC8949] with a signature format of COSE [RFC8812].

This document provides an equivalent mapping of JSON format with the signature format in JOSE format [RFC7515].

This document does not extend the YANG definition of [RFC8366] at all, but accepts that other efforts such as [I-D.richardson-anima-voucher-delegation], [I-D.friel-anima-brski-cloud], and [I-D.ietf-anima-brski-async-enroll] do. This document supports signing any of the extended schemas defined in those documents and any new documents that may appear after this one.

2. Terminology

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.

3. JSON Web Signatures

[RFC7515] defines two serializations: the JWS Compact Serialization and the JWS JSON Serialization. This document restricts itself to the JWS Compact Serialization (JWT) format.

The [RFC8366] JSON structure consists of a nested map, the outer part of which is:

```
{ "ietf-voucher:voucher" : { ..some items }}
```

this is considered the JSON payload as described in [RFC7515] section 3.

The JSON Compact Serialization is exemplained in section 3.1 or section 7.1, and works out to:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||  
BASE64URL(JWS Payload) || '.' ||  
BASE64URL(JWS Signature)
```

Note that this results in a long base64 content (with two interspersed dots). The content is transmitted within the HTTPS session in this base64 format, even though HTTP can accomodate binary content. This is done to be most convenient for available JWT libraries, and for humans who are debugging.

There are a number of attributes. They are:

3.1. Unprotected Header

There is no unprotected header in the Compact Serialization format.

3.2. Protected Header

The standard "typ" and "alg" values described in [RFC7515] are expected in the protected headers.

It is unclear what values, if any, should go into the "typ" header, as in the [BRSKI] use cases, there are additional HTTP MIME type headers to indicate content types.

The "alg" should contain the algorithm type such as "ES256".

If PKIX [RFC5280] format certificates are used then the [RFC7515] section 4.1.6 "x5c" certificate chain SHOULD be used to contain the certificate and chain. Vouchers will often need all certificates in the chain, including what would be considered the trust anchor certificate because intermediate devices (such as the Registrar) may need to audit the artifact, or end systems may need to pin a trust anchor for future operations.

4. Privacy Considerations

The Voucher Request reveals the IDevID of the system that is onboarding.

This request occurs over HTTPS, however the Pledge to Registrar transaction is over a provisional TLS session, and it is subject to disclosure via by a Dolev-Yao attacker (a "malicious messenger") [onpath]. This is explained in [BRSKI] section 10.2.

5. Security Considerations

The issues of how [RFC8366] vouchers are used in a [BRSKI] system is addressed in

6. IANA Considerations

6.1. Media-Type Registry

This section registers the the 'application/voucher-jwt+json' in the "Media Types" registry.

6.1.1. application/voucher-jose+json

Type name: application
Subtype name: voucher-jwt+json
Required parameters: none
Optional parameters: none
Encoding considerations: JWT+JSON vouchers are JOSE objects
signed with one signer.
Security considerations: See Security Considerations, Section
Interoperability considerations: The format is designed to be
broadly interoperable.
Published specification: THIS RFC.
Applications that use this media type: ANIMA, 6tisch, and other
zero-touch imprinting systems
Additional information:
Magic number(s): None
File extension(s): .vjj
Macintosh file type code(s): none
Person & email address to contact for further information: IETF
ANIMA WG
Intended usage: LIMITED
Restrictions on usage: NONE
Author: ANIMA WG
Change controller: IETF
Provisional registration? (standards tree only): NO

7. Acknowledgements

Your name here.

8. Changelog

9. References

9.1. Normative References

- [BRSKI] Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", Work in Progress, Internet-
Draft, draft-ietf-anima-bootstrapping-keyinfra-45, 11
November 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-anima-bootstrapping-keyinfra-45.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>>.

- [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/info/rfc7515>>.
- [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>>.
- [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/info/rfc8259>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8812] Jones, M., "CBOR Object Signing and Encryption (COSE) and JSON Object Signing and Encryption (JOSE) Registrations for Web Authentication (WebAuthn) Algorithms", RFC 8812, DOI 10.17487/RFC8812, August 2020, <<https://www.rfc-editor.org/info/rfc8812>>.

9.2. Informative References

- [I-D.friel-anima-brski-cloud]
Friel, O., Shekh-Yusef, R., and M. Richardson, "BRSKI Cloud Registrar", Work in Progress, Internet-Draft, draft-friel-anima-brski-cloud-03, 24 September 2020, <<http://www.ietf.org/internet-drafts/draft-friel-anima-brski-cloud-03.txt>>.
- [I-D.ietf-anima-brski-async-enroll]
Fries, S., Brockhaus, H., and E. Lear, "Support of asynchronous Enrollment in BRSKI (BRSKI-AE)", Work in Progress, Internet-Draft, draft-ietf-anima-brski-async-enroll-00, 10 July 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-anima-brski-async-enroll-00.txt>>.
- [I-D.ietf-anima-constrained-voucher]
Richardson, M., Stok, P., and P. Kampanakis, "Constrained Voucher Artifacts for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-09, 2 November 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-anima-constrained-voucher-09.txt>>.

- [I-D.richardson-anima-voucher-delegation]
Richardson, M. and J. Yang, "Delegated Authority for Bootstrap Voucher Artifacts", Work in Progress, Internet-Draft, draft-richardson-anima-voucher-delegation-02, 18 September 2020, <<http://www.ietf.org/internet-drafts/draft-richardson-anima-voucher-delegation-02.txt>>.
- [onpath] "can an on-path attacker drop traffic?", n.d., <<https://mailarchive.ietf.org/arch/msg/saag/mlr9uo4xYznOcf85EyK0Rhut598/>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.
- [RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <<https://www.rfc-editor.org/info/rfc8792>>.
- [RFC8949] 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/info/rfc8949>>.

Appendix A. Examples

These examples are folded according to [RFC8792] Single Backslash rule.

A.1. Example Voucher Request (from Pledge to Registrar)

The following is an example request sent from a Pledge to the Registrar. This example is from the Siemens reference Registrar system.

```
<CODE BEGINS> file "voucher_request_01.b64"
eyJhbGciOiAiRVMyNTYiLCAieDVjIjogWyJNSU1CMmpDQ0FzQ2dBd01CQWd\
R0FXZWdkY1NMTUfVR0NDcUdTTTQ5QkFNQ01EMHhDekFKQmdOVkJBWVRBa0Z\
TVJVd0V3WURWUWFLREF4S2FXNW5TbWx1WjBodmNuQXhGekFWQmdOVkJBUTU1\
a3BwYm1kS2FXNW5WR1Z6ZEVOQk1DQVhEVEU0TVRJeE1qQXpNamcxTVZvWUR\
azVPVGt4TWpNeE1qTTFPVFU1V2pCU01Rc3dDUVlEVlFRR0V3SkJVVVEVWTUJ\
R0ExVUVDZ3dNU21sdVowcHBibWREYjNkd01STXdFUVlEVlFRRkV3b3dNVE1\
TkRVMk56ZzVNUmN3RlFZRFZRUUREQTU1YVc1b1NtbHVAMFJsZG1sa1pUQ1p\
Qk1HQNlxR1NNND1BZ0VHQ0NXR1NNND1Bd0VIQTBJQUJNVkdHOFo1cGpmNWp\
bnlyVXJYeVoxalBncUJlM05YdTfkVEFEZStyL3Y2SnPJSgWzNTVJZ2NIQzN\
eHBpYnFKTS9iV1JhRXlqcWNDSmo0akprb3dDdWpWVEJUTUN3R0NTc0dBUVF\
Z3U1U0FnUWZEQjF0WVhOaExYUmxjM1F1YzJsbGJXVnVjeTFpZEM1dVpYUTZ\
VFEWtXpBEJnTlZiU1VFRERBS0JnZ3JCZ0VGQ1FjREFqQU9CZ05WSFE4QkF\
OEVQQU1DQjRBd0NnWU1Lb1pJemowRUF3SURTQUF3U1FJZ1d0UHpJSVhZMm1\
UlhkDEV4S0VoaFpkYTRYK0VwbFpVbUVJMNpBMGRzam9DSVFDm0pwUW1SWE1\
bi9wNEJlOW16aWk5MmVjbFR4NC9PNHJsbTdNeUxxa2hkQT09I119.eyJpZX\
mLXZvdWNoZXItcmVxdWVzdDp2b3VjaGVyIjoGeYJjcmVhdGVkLW9uIjoGJj\
wMjAtMTAtMjJUMDI6Mzc6MzkuMDAwWiIsICJub25jZSI6ICJlRHMmrKy9GdU\
IR1VuUnhOM0UxNENRPT0iLCAic2VyaWFsLW51bWJlciI6ICIwMTIzNDU2Nz\
5In19.Vj9pyo43KDEq0e5tokwHpNhVM0uUkLCatwNQxfCKH8GRQ2iTT2fq\
39k40M-7S-vheDHHuBHFSWb502EPwkda
<CODE ENDS>
```

It contains the following three parts:

Header:

```
<CODE BEGINS> file "voucher_request_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIB2jCCAYCgAwIBAgIGAWEgdcSLMAoGCCqGSM49BAMCMD0xCzAJBg\
VBAYTAKFRMRUwEwYDVQQKDAxKaW5nSmluZ0NvcnAxZzAVBgNVBAMMDkppbm\
KaW5nVGVzdENBMCAXDTE4MTIxMjAzMjg1MVoYDZk5OTkxMjMxMjM1OTU5Wj\
SMQswCQYDVQQGEwJBUTEVMBMGA1UECgwMSmluZ0ppbmddDb3JwMRMwEQYDVQ\
FEwOWMTIzNDU2Nzg5MRCwFQYDVQQDDA5KaW5nSmluZ0Rldm1jZTBZMBMGBY\
GSM49AgEGCCqGSM49AwEHA0IABMVGg8Z5pjf5jXnyrUrXyZ1kPgqBe3NXu1\
TADe+r/v6JzIH1355IgcHC3axpiBqJM/bWRaEyjqcCJj4jJkowCuJVBTMC\
GCSsGAQQBggu5SAgQfDB1tYXNhLXRlc3Quc2l1bWVucy1idC5uZXQ6OTQ0Mz\
TBgNVHSUEDDAKBggrBgEFBQcDAjAOBgNVHQ8BAf8EBAMCB4AwCgYIKoZIzj\
EAwIDSAAwRQIgWtPzIIXY2ixRXJtExKEhhZda4X+Ep1ZomEI2zA0dsjoCIQ\
3JpQmRXMGn/p4Bu9izii92eclTx4/O4rlm7MyLqkhdA=="
  ]
}
<CODE ENDS>
```

Payload:

```
<CODE BEGINS> file "voucher_request_01-payload.b64"
{
  "ietf-voucher-request:voucher": {
    "created-on": "2020-10-22T02:37:39.000Z",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "serial-number": "0123456789"
  }
}
<CODE ENDS>
```

Signature:

```
<CODE BEGINS> file "voucher_request_01-signature.b64"
Vj9pyo43KDEq0e5tokwHpNhVM0uUkLCatwNQxfCKH8GRQ2iTT2fqD39k40\
-7S-vheDHHuBHFSWb502EPwkda
<CODE ENDS>
```

A.2. Example Parboiled Voucher Request (from Registrar to MASA)

The following is an example request sent from the Registrar to the MASA. This example is from the Siemens reference Registrar system. Note that the previous voucher request can be seen in the payload as "prior-signed-voucher-request".

```
<CODE BEGINS> file "parboiled_voucher_request_01.b64"
eyJhbGciOiJFUzI1NiIsIng1YyI6WyJNSU1Cb3pDQ0FVcWdBd0lCQWdJR0F\
MGVMdU1GTUfVR0NDcUdTTTQ5QkFNQ01EVXhFekFSQmdOVkJBb01DazE1UW5\
emFXNWxjM014RFRBTEJnTlZCQWNNQkZOCGRHVXhEekFOQmdOVkJBtU1CbFJ\
YzNSRFFUQWVGdZB4T1RBNU1URXdNak0zTXpKYUZ3MHlPVEE1TVRFd01qTTN\
ekphTUZReEV6QVJCZ05WQkFvTUNrMTVRblZ6YVc1bGMzTXhEVEFMQmdOVk\
Y01CRk5wZEdVeExqXNCZ05WQkFNTUUpWSmxaMmx6ZEhKaGNpQldiM1ZqYUd\
eU1GSmxjWFZsYzNRZ1UybG5ibWx1Wn1CTFpYa3dXVEFUMQmdjcWhrak9QUU1\
QmdncWhrak9QUU1CQndOQ0FBVDZ4VnZBdnFuejFvVW11TldoWHBRc2thUHK\
QUhIUUx3WG1KMGlFTHQ2dU5QYW5BTjBRblDNWU8vMENERWpJa0JRb2J3OF1\
cWp0eEpIV1NHVG05S09veWN3SlRBVEJnTlZlU1VFRERBS0JnZ3JCZ0VGQ1F\
REhEQU9CZ05WSFE4QkFmOEVCQU1DQjRBd0NnWU1Lb1pJemowRUF3SURSd0F\
UkFJZ1lyMkxmcW9hQ0tERjRSQWNNbUppK05DwnFkU211VnVnSVNBN09oSlJ\
M1lDSUR4b1BNTW5wWEFNVHJQSnnVQV31jZUVSMTFQeEhPhiswQ3BTSGkycWd\
V1giLCJNSU1CcERDQ0FVbWdBd0lCQWdJR0FXMGVMdUgrTUfVR0NDcUdTTTQ\
QkFNQ01EVXhFekFSQmdOVkJBb01DazE1UW5WemFXNWxjM014RFRBTEJnTlZ\
QWNNQkZOCGRHVXhEekFOQmdOVkJBtU1CbFJsYzNSRFFUQWVGdZB4T1RBNU1\
RXNdNak0zTXpKYUZ3MHlPVEE1TVRFd01qTTNekphTURVeEV6QVJCZ05WQkF\
TUNrMTVRblZ6YVc1bGMzTXhEVEFMQmdOVkJBtU1CRk5wZEdVeER6QU5CZ05\
QkFNTUJsUmxiM1JEUVRWk1CTUdCeXHU000OUFnRUdDQ3FHU000OUF3RUh\
MElBQk9rdmtUSHU4UWxUM0ZISjFVYUk3K1dzSE9iMFVTM1NBTHRHNXd1S1F\
amlleDA2L1NjWTVQSmliZmdIVEIrRi9RVGpnZWxIR3kxWUtd2NOTWNzU3l\
alJUQkRNQk1HQTFVZEV3RUlvd1FJTUFZQkFmOENBUUV3RGdZRFZSMFBBUUg\
QkFRREFnSUVNqjBHQTfVZERnUVdCQlRvWk1NelFkc0Qvai8rZ1gvN2NCSnV\
```


SC9YbWpBS0JnZ3Foa2pPUFFRREFnTkpbREJHQWlFQXR4UTMrSUxHQ1BJdFN\
NGI5V1hoWE51aHFTUDZIK2IvTEMvZ1ZZRGpRNM9DSVFERzJ1UkNIbFZxM31\
QjU4VFhNVWJ6SDgrT2xoV1V2T2xSRDNWRXFEZGNRdz09I119.eyJpZXRMbX\
vdWN0ZXItcmVxdWVzdDp2b3VjaGVyIjp7InNlcm1hbC1udWliZXIiOiIwMT\
zNDU2Nzg5Iiwibm9uY2UiOiJlRHMtKy9GdURIR1VuUnhOM0UxNENRPT0iLC\
wcm1vcilzaWduZWQtZm91Y2hlcilYzXF1ZXN0IjoizXlKaGJHY2lPaUFpU1\
NeU5UWW1MQ0FpZURWak1qb2dXeUpOU1VsQ01tcERRMEZaUTJkQmQwbENRV2\
KUjBGWFpXZGtZMU5NVFVGd1IwTkRjVWRUVFRRNVFrRk5RMDFFTUhoRGVrRk\
RbWRPvmtKQldWUkJhMFpTVFZKVmQwVjNXVjVXVZGTFJFRjRTMkZYTlc1VG\
XeDFXakJPZG1OdVFYaEdla0ZXUW1kT1ZrSkJUVTFFYTNCd1ltMwTmKZYT1\
1V1IwVjZaRVZPUWsxRFFWaEVWRVUwVFZSSmVFMXFRWHBOYW1jeFRWWnZXVV\
2YXpWUFZHdRUV3BOZUUxcVRUR1BWR1UxVjJwQ1UwMVJjM2REVZsRVZsR1\
SMFYzU2tKV1ZFVldUVUpOUjBFeFZVVkRaM2ROVTIxc2RWb3djSEJpYldSRV\
qTktkMDFTVFhkR1VWbEVWbEZSUmTWM2IzZE5WRWw2VGtSVk1rNTZaelZOvW\
OM1JsRlpSRlpSVVVSrvFUVkxZVmMxYmxOdGJIVmFNRkpzWkcxc2FscFVRbH\
OUWsxSFFubHhSMU5OTkRsQlowVkhRME54UjFOTk5EbeJkMFZJUVRCS1FVSk\
Wa2RIT0ZvMWNHcG1OV3BZYm5seVZYS111Vm94YTFcbmNVSmxNMDVZZFRGa1\
FRkVaU3R5TDNZM1NucEpTR3d6T1RWSloyTk1Rek5oZUhCcFluRktUUzlpVj\
KaFJYbHFjV05EU21vMGFrcHJiM2REZFdWV1ZFSlVUVU4zUjBOVGMwZEJVvk\
DWjNVMVUwRm5VV1pFUWpGMFdWaE9hRXhZVW14ak0xRjFZeKpzYkdKWFZuVm\
lVEZwWkVNMWRWcFlVVFpQVkJZFd1RYcEJWRUpuVGxaSVUxVkJZSRVJCUzBKbl\
zSkNaMFZHUWxGalJFRnFRVTlDWjA1V1NGRTRRa0ZtT0VWQ1FVMURRa1JCZD\
ObldVbExiMXBKZW1vd1JVRjNTVjVUUUVGM1VsRkpaMWQwVUhwS1NwaFpNbW\
0VWxoS2RFVjRTMFZvYUZwallUU11LMFZ3YkZwdmJVVkpNbnBCTUdSemFtOU\
TVkZEttBwd1VXMVNXRTFIYmk5d05FSjFPV2w2YVdrNU1tVmpIR1I0TkM5UE\
ISnNiVGROZVV4eGEyaGtRVDA5SWwxOS5leUpwW1hSbUxYWnZkV05vW1hJdG\
tVnhkV1Z6ZERwMmIzVmphR1Z5SWpvZ2V5SmpjbVZoZEdWa0xXOXVJam9nSW\
Jd01qQXRNVEF0TWpKVU1ESTZNemM2TXprdu1EQXdXaU1zSUNKdWIyNWpaU0\
2SUNKbFJITXJLeTlHZFVSSVixVnVbmbhPTTBVeE5FTlJQVDBpTENBaWMyVn\
hV0ZzTFc1MWJXSmxjaUk2SUNJd01USXpORFUyTnpnNUluMTkuVmo5cHlvND\
LREVxMGUldG9rd0hwTmhWTTB1VWtMQ2F0d05ReGZzQ0tIOEdSUTJpVFQyZn\
EMzlrNDBNLtdTLXZoZURISHVCSEZTV2I1MDJFUHdRZEEiLCJjcmVhdGVkLW\
uIjoimjAyMC0xMC0yM1QwMjJoZnZozOS4yMzVaIn19.S3BRYIKHbsqwQEZsB\
J1COIVAx02NPEc5oo_BnXK_JkQfStTieHFCALdv5MzYdTU9myJO1muaSFEI\
_NFMSFja
<CODE ENDS>

It contains the following three parts:

Header:

```
<CODE BEGINS> file "parboiled_voucher_request_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIBozCCAUqgAwIBAgIGAW0eLuIFMAoGCCqGSM49BAMCMDUxEzARBg\
VBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3\
DQTAeFw0xOTA5MTEwMjMzMzJaFw0yOTA5MTEwMjMzMzJaMFQxEzARBgNVBA\
MCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxLjAsBgNVBAMMJVJlZ21zdH\
hciBWB3VjaGVyIFJlcXVlc3QgU2lnbmhluZyBLZXkwWTATBgqcqhkJOPQIBBg\
qhkJOPQMBBwNCAAT6xVvAvqTz1ZUiuNWhXpQskaPy7AHHQLwXiJ0iELt6uN\
anAN0QnWMyO/0CDEjIkBQobw8YKqjtxJHVSGTj9KOoycwJTATBgNVHSUEDD\
KBggrBgEFBQcDHDAAOBgNVHQ8BAf8EBAMCB4AwCgYIKoZlZj0EAwIDRwAwRA\
gYr2LfqoaCKDF4RACMmJi+NCZqdSiuVugISA7OhKRq3YCIDxnPMMnpXAMTr\
JuPWyceER11PxHOn+0CpSHi2qgpWX",
    "MIIBpDCCAUmGAWIBAgIGAW0eLuH+MAoGCCqGSM49BAMCMDUxEzARBg\
VBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3\
DQTAeFw0xOTA5MTEwMjMzMzJaFw0yOTA5MTEwMjMzMzJaMDUxEzARBgNVBA\
MCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3RDQT\
ZMBMGBByqGSM49AgEGCCqGSM49AwEHA0IABOkvkTHu8Q1T3FHJ1UaI7+WsHO\
0US3SALtG5wuKQDjiex06/ScY5PJibvgHTB+F/QTjge1HGy1YKpwcNMcsSy\
jRTBDMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYDVROPAQH/BAQDAgIEMB0GA1\
dDgQWBBToZIMzQdsD/j/+gX/7cBJucH/XmjAKBgqgqhkJOPQQDAgNJADBGAi\
AtxQ3+ILGBPItSh4b9WXhXNuhqSP6H+b/LC/fVYDjQ6oCIQDG2uRCH1Vq3y\
B58TXMUbzH8+0lhWUv0lRD3VEqDdcQw=="
  ]
}
<CODE ENDS>
```

Payload:

```
<CODE BEGINS> file "parboiled_voucher_request_01-payload.b64"
{
  "ietf-voucher-request:voucher": {
    "serial-number": "0123456789",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "prior-signed-voucher-request": "eyJhbGciOiAiA1RVMyNTYiLC\
ieDVjIjogWyJNSU1CMmpDQ0FZQ2dDd01CQWdJR0FXZlZkdY1NMTUFvR0NDcU\
TTQ5QkFkQ01EMHhDekFKQmdOVkJBWVRBa0ZSTVJlZD0V3WURWUWFLREF4S2\
XNW5TbWx1WjB0dmNuQXhGekFWQmdOVk1JBTU1Ea3BwYmlkS2FXNW5WR1Z6ZE\
OQk1DQVhEVEU0TVRJeE1qQXpNamcxTVZvWUR6azVPVGt4TWpNeE1qTTFPVF\
1V2pCU01Rc3dDUVlEVlFRR0V3SkJVVVEVWTUJNR0ExVUVDZ3dNU21sdVowcH\
ibWREYjNkD01STXdfUVlEVlFRRkV3b3dNVE16TkRVMk56ZzVNUmN3RlFZRF\
RUUREQTvLYVc1b1NtbHVhVAMFJSZG1salpUQ1pNQk1HQnlxR1NNND1BZ0VHQ0\
xR1NNND1Bd0VIQTBJQUJNVkdHOFOlcGpmNWpYbnlyVXJYeVoxa1BncUJlM0\
YdTFkVEFEZStyL3Y2SnPJSgwzNTVJZ2NIQzNheHBpYnFKTS9iV1JhRXlqcW\
DSmo0akprb3dDdWpWVEJUTUN3R0NTc0dBUVFCZ3U1U0FnUWZEQjF0WVhOaE\
YUmxjM1F1YzJsbGJXVnVjeTFpZEM1dVpYUTZPVFEwTxBVEJnTlZlU1VFRE\
BS0JnZ3JCZ0VGQ1FjREFqQU9CZ05WSFE4QkFmOEVCQ1U1DQjRBd0NnWU1Lb1\
JemowRUF3SURTQUF3U1FJZ1d0UHpJSVhZMml4UlhKdEV4S0VoaFpkYTRYK0\
wbFpYvU1JmnpBMGRzam9DSVFDm0pwUW1SWE1Hbi9wNEJlOW16aWk5MmVjbF\
4NC9PNHJsbTdNeUxxa2hkQT09I119.eyJpZXRmLXZvdWNOZXItcmVxdWVzd\
p2b3VjaGVyIjogeyJjcmVhdGVkLW9uIjogIjIwMjAtMTAtMjU1MjU1MjU1\
kuMDAwWiIsICJub25jZSI6ICJlRHRmKy9GdURIR1VuUnhOM0UxNENRPT0iL\
Aic2VyaWFsLW51bWJlciI6ICJlRHRmKy9GdURIR1VuUnhOM0UxNENRPT0iL\
kwHpNhVM0uUkLCatwNQxfSCKH8GRQ2iTT2fQd39k40M-7S-vheDHHuBHF\
502EPwkdA",
    "created-on": "2020-10-22T02:37:39.235Z"
  }
}
<CODE ENDS>
```

Signature:

```
<CODE BEGINS> file "parboiled_voucher_request_01-signature.b64"
S3BRYIKHbsqwQEzSbgJlCOIVAxO2NPEc5oo_BnXK_JkQfStTieHFCALdv5M\
YdTU9myJO1muaSFEIu_NFMSFjA
<CODE ENDS>
```

A.3. Example Voucher Result (from MASA to Pledge, via Registrar)

The following is an example voucher sent from the Registrar to the MASA. This example is from the Siemens reference MASA system.

```
<CODE BEGINS> file "voucher_01.b64"
eyJhbGciOiJFUzI1NiIsIng1YyI6WyJNSU1Ca3pDQ0FUaWdBd01CQWdJR0F\
RkJqQ2tZTUfVr0NDcUdTTTQ5QkFNQ01EMHhDekFKQmdOVkJBWVRBa0ZSTVJ\
d0V3WURWUWFLREF4S2FXNW5TbWx1WjBodmNuQXhGekFWQmdOVkJBWVRBa0ZSTVJ\
Ym1kS2FXNW5WR1Z6ZEVOQk1CNFhEVEU0TURFeU9URXdOVEkwTUZvWERUSTR\
REV5T1RFd05USTBNRm93VHpfTE1Ba0dBMVVFQmhNQ1FWRXhGVEFVQmdOVkK\
b01ERXBWYm1kS2FXNW5RMj15Y0RfcE1DY0dBMVVFQXd3Z1NtbHVAMHBWYm1\
RGIzSndJRlp2ZFd0b1pYSWdVMmxuYm1sdVp5QkxaWGt3V1RBVEJnY3Foa2p\
UFFJQk1JnZ3Foa2pPUFFNQk1J3TkNBQVNDNmJ1TEFtZXExVnc2aVFyUnM4UjB\
Vys0YjFHV3lkVdzMkdBTUZXZD2JpdGyYybk1YSDNpUUhLVnU4czJSdm1CR05\
dk9LR0JISHRCZG1GRVpadmI3b3hJd0VEQU9CZ05WSFE4QkFmOEVCQU1DQjR\
d0NnWU1Lb1pJemowRUF3SURTUUF3UmdJaEFJNFBZYNh0c3NIUDJWSHgvHh\
b1EvU3N5ZEwzMERRSU5FdGNOOW1DVfHQW1FQXZJYjNvK0ZPM0JUbmNMRn\
SlpSQWtkN3pPdXNuLy9aS09hRUtic1ZEaVU9I119.eyJpZXRmLXZvdWNoZX\
6dm91Y2hlc1I6eyJhc3N1cnRpb24iOiJsb2dnZWQ1LCJzZXJpYWwtbnVtYm\
yIjoimDEyMzQ1Njc4OSIsIm5vbmN1IjoizURzKysvRnVESEdVb1J4TjNFMT\
DUT09Iiw1Y3JlYXR1ZC1vbiI6IjIwMjAtMTAtMjU1MDI6MzYzOTI1XW1\
sInBpbm5lZC1kb21haW4tY2VydCI6Ikl1JSUJwRENDQVtZ0F3SUJBZ01HQV\
wUx1SctNQW9HQ0NxR1NNND1CQU1DTURVeEV6QVJCZ05WQkFvTUNrMTVRbl\
6YVc1bGMzTXhEVEFMQmdOVk1JBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUm\
jM1JEUVRBZUZ3MHhPVEE1TVRFd01qTTNNekphRncweU9UQTVNVEV3TWpNM0\
6SmFNRFV4RXpBUk1JN1ZCQW9NQ2sxNVFvVnphVzVsYzNNeERUQUxCZ05WQk\
jTUJGTnBkR1V4RHpBTk1JN1ZCQW9NQ2sxNVFvVnphVzVsYzNNeERUQUxCZ05WQk\
5QWdFR0NDcUdTTTQ5QXdfSEwSUfCT2t2a1RidThRbFQzRkhKMVhStcrV3\
IT2IwVVMzU0FMdEc1d3VLUURqaWV4MDYvU2NZNVBKawJ2Z0hUQitGL1FUam\
lbEhHeTFZS3B3Y05NY3NTEwWFqU1RCRE1CSUdBMVVKRXdfQ193UU1NQV1CQW\
4Q0FRRXdeZ11EV1IwUEFRSC9CQVFQWdJR1CMEdBmVVKRGdRV0JCVG9aSU\
6UWRzRC9qLytnWC83Y0JKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQk\
BaUVBdHhRMytJTEdCUE10U2g0Yj1XWGHYtNvocVNQNkgrYi9MQy9mV11Eal\
2b0NJUURHMnVSQ0hsVnEzeWhCNThUWE1VYnpIOctPbGhXVXZPbFJEM1ZFcu\
kY1F3PT0ifX0.uliO_VB6xIhE8QuhKDGgCkxzsR20IoL0p6qYKpYBDtgkR\
2ykDO_QFjk7W8P5ATW-CQnW1J3ILSeiwMf9nIOg
<CODE ENDS>
```

It contains the following three parts:

Header:

```
<CODE BEGINS> file "voucher_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIBkzCCATigAwIBAgIGAWFBjCkYMAoGCCqGSM49BAMCMD0xCzAJBg\
VBAYTAKFRMRUwEwYDVQQKDAxKaW5nSmluZ0NvcnAxFTATBgNVBAoMDEppbm\
KaW5nVGZvdENBMB4XDTE4MDEyOTEwNTI0MfoXDTI4MDEyOTEwNTI0Mfo\
LMAkGA1UEBhMCQVEFTATBgNVBAoMDEppbmKaW5nQ29ycDEpMCcGA1UEAw\
gSmluZ0ppbmDb3JwIFZvdWNoZXIqU2lnbmduZyBLZXkwWTATBgqcqhkJOPQ\
BBggqhkJOPQMBBwNCAASC6beLameq1Vw6iQrRs8R0ZW+4b1GWydmWs2GAMF\
wbitf2nIXH3OqHKVu8s2RviBGNivOKGBHtBdiFEZZvb7oxIwEDAOBgNVHQ\
BAf8EBAMCB4AwCgYIKoZIzj0EAwIDSQAwwRgIhAI4PYbxtssHP2VHx/tzUoQ\
SsydL30DQINetCn9mCTXPAiEAvIb3o+F03BTncLFsaJZRAkd7zOusn//ZKO\
EKbsVDiU="
  ]
}
<CODE ENDS>
```

Payload:

```
<CODE BEGINS> file "voucher_01-payload.b64"
{
  "ietf-voucher:voucher": {
    "assertion": "logged",
    "serial-number": "0123456789",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "created-on": "2020-10-22T02:37:39.921Z",
    "pinned-domain-cert": "MIIBpDCCAUmGAWIBAgIGAW0eLuH+MAoG\
CqGSM49BAMCMDUxEzARBgNVBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNp\
GUxDzANBgNVBAMMB1Rlc3RDQTAeFw0xOTA5MTEwMjMzZjFwOTA5MTEw\
jM3MzJAMDUxEzARBgNVBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUx\
zANBgNVBAMMB1Rlc3RDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABOkv\
THu8Q1T3FHJ1UaI7+WsHob0US3SALtG5wuKQDjiex06/ScY5PJibvgHTB+F\
QTjgelHGylYKpwcNMcsSyaJRTBDMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYD\
R0PAQH/BAQDAgIEMB0GA1UdDgQWBbToZIMZQdsD/j/+gX/7cBJucH/XmjAK\
ggqhkJOPQDAGNJADBGAiEAtxQ3+ILGBPItSh4b9WXhXNuhqSP6H+b/LC/f\
YDjQ6oCIQDG2uRCH1Vq3yhB58TXMUbzH8+OlhWUvOlRD3VEqDdcQw=="
  }
}
<CODE ENDS>
```

Signature:

```
<CODE BEGINS> file "voucher_01-signature.b64"
u1iO_VB6xIhE8QuhKDGgCkxksnR20IoL0p6qYKpYBDtgkRT2ykDO_QFjk7W\
P5ATW-CQnWlJ3ILSeiwMf9nI0g
<CODE ENDS>
```

Authors' Addresses

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Thomas Werner
Siemens
Email: thomas-werner@siemens.com

anima Working Group
Internet-Draft
Updates: RFC8366 (if approved)
Intended status: Standards Track
Expires: 26 January 2022

M. Richardson
Sandelman Software Works
T. Werner
Siemens
25 July 2021

JWS signed Voucher Artifacts for Bootstrapping Protocols
draft-richardson-anima-jose-voucher-02

Abstract

RFC8366 defines a digital artifact called voucher as a YANG-defined JSON document that has been signed using a Cryptographic Message Syntax (CMS) structure. This memo introduces a variant of the voucher structure in which CMS is replaced by the JSON Object Signing and Encryption (JOSE) mechanism described in RFC7515 to better support use-cases in which JOSE is preferred over CMS.

In addition to explaining how the format is created, MIME types are registered and examples are provided.

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 26 January 2022.

Copyright Notice

Copyright (c) 2021 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 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	2
2. Terminology	3
3. JSON Web Signatures	3
3.1. Unprotected Header	4
3.2. Protected Header	4
4. Privacy Considerations	4
5. Security Considerations	5
6. IANA Considerations	5
6.1. Media-Type Registry	5
6.1.1. application/voucher-jws+json	5
7. Changelog	5
8. References	6
8.1. Normative References	6
8.2. Informative References	6
Appendix A. Examples	8
A.1. Example Voucher Request (from Pledge to Registrar)	8
A.2. Example Parboiled Voucher Request (from Registrar to MASA)	9
A.3. Example Voucher Result (from MASA to Pledge, via Registrar)	13
Authors' Addresses	16

1. Introduction

"A Voucher Artifact for Bootstrapping Protocols", [RFC8366] describes a voucher artifact used in "Bootstrapping Remote Secure Key Infrastructure" [BRSKI] and "Secure Zero Touch Provisioning" [SZTP] to transfer ownership of a device to from a manufacturer to an owner. That document defines the base YANG module, and also the initial serialization to JSON [RFC8259], with a signature provided by [RFC5652].

Other work, [I-D.ietf-anima-constrained-voucher] provides a mapping of the YANG to CBOR [RFC8949] with a signature format of COSE [RFC8812].

This document provides an equivalent mapping of JSON format with the signature format in JOSE format [RFC7515]. The encoding specified in this document is required for [I-D.ietf-anima-brski-async-enroll] and may be required and/or preferred in other use-cases, for example when JOSE is already used in other parts of the use-case, but CMS is not.

This document does not extend the YANG definition of [RFC8366] at all, but accepts that other efforts such as [I-D.richardson-anima-voucher-delegation], [I-D.friel-anima-brski-cloud], and [I-D.ietf-anima-brski-async-enroll] do. This document supports signing any of the extended schemas defined in those documents and any new documents that may appear after this one.

With the availability of different encoded vouchers, it is up to an industry specific application statement to indicate/decide which voucher format is to be used. There is no provision across the different voucher formats that a receiver could safely recognize which format it uses unless additional context is provided. For example, [BRSKI] provides this context via the MIME-Type for the voucher payload.

This document should be considered an Update to [RFC8366] in the category of "See Also" as per [I-D.kuehlewind-update-tag].

2. Terminology

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.

3. JSON Web Signatures

[RFC7515] defines two serializations: the JWS Compact Serialization and the JWS JSON Serialization. The two serializations are mostly equivalent, and the JWS Compact Serialization (JWT) format has better library support in web frameworks, so this document restricts itself to that choice.

The [RFC8366] JSON structure consists of a nested map, the outer part of which is:

```
{ "ietf-voucher:voucher" : { some inner items }}
```

this is considered the JSON payload as described in [RFC7515] section 3.

The JSON Compact Serialization is explained in section 3.1 or section 7.1, and works out to:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||  
BASE64URL(JWS Payload) || '.' ||  
BASE64URL(JWS Signature)
```

Note that this results in a long base64 content (with two interspersed dots). When using HTTPS, the voucher is transmitted in base64 format, even though HTTP can accommodate binary content. This is done to be most convenient for available JWT libraries, and for humans who are debugging.

There are a number of attributes. They are:

3.1. Unprotected Header

There is no unprotected header in the Compact Serialization format.

3.2. Protected Header

The standard "typ" and "alg" values described in [RFC7515] are expected in the protected headers.

It remains to be determined (XXX), what values, if any, should go into the "typ" header, as in the [BRSKI] use cases, there are additional HTTP MIME type headers to indicate content types.

The "alg" should contain the algorithm type such as "ES256".

If PKIX [RFC5280] format certificates are used then the [RFC7515] section 4.1.6 "x5c" certificate chain SHOULD be used to contain the certificate and chain. Vouchers will often need all certificates in the chain, including what would be considered the trust anchor certificate because intermediate devices (such as the Registrar) may need to audit the artifact, or end systems may need to pin a trust anchor for future operations. This is consistent with [BRSKI] section 5.5.2.

4. Privacy Considerations

The Voucher Request reveals the IDevID of the system that is onboarding.

This request occurs over HTTPS, however the Pledge to Registrar transaction is over a provisional TLS session, and it is subject to disclosure via by a Dolev-Yao attacker (a "malicious messenger") [onpath]. This is explained in [BRSKI] section 10.2.

The use of a JWS header brings no new privacy considerations.

5. Security Considerations

The issues of how [RFC8366] vouchers are used in a [BRSKI] system is addressed in section 11 of that document. This document does not change any of those issues, it just changes the signature technology used for vouchers and voucher requests.

[SZTP] section 9 deals with voucher use in Secure Zero Touch Provisioning, and this document also makes no changes to security.

6. IANA Considerations

6.1. Media-Type Registry

This section registers the 'application/voucher-jws+json' in the "Media Types" registry.

6.1.1. application/voucher-jws+json

Type name: application
Subtype name: voucher-jwt+json
Required parameters: none
Optional parameters: none
Encoding considerations: JWS+JSON vouchers are JOSE objects signed with one signer.
Security considerations: See Security Considerations, Section
Interoperability considerations: The format is designed to be broadly interoperable.
Published specification: THIS RFC.
Applications that use this media type: ANIMA, 6tisch, and other zero-touch imprinting systems
Additional information:
 Magic number(s): None
 File extension(s): .vjj
 Macintosh file type code(s): none
Person & email address to contact for further information: IETF ANIMA WG
Intended usage: LIMITED
Restrictions on usage: NONE
Author: ANIMA WG
Change controller: IETF
Provisional registration? (standards tree only): NO

7. Changelog

- * Added adoption call comments from Toerless. Changed from [RFCxxxx] to [THING] style for some key references.

8. References

8.1. Normative References

- [BRSKI] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <<https://www.rfc-editor.org/info/rfc8995>>.
- [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>>.
- [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/info/rfc7515>>.
- [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>>.
- [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/info/rfc8259>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [SZTP] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <<https://www.rfc-editor.org/info/rfc8572>>.

8.2. Informative References

- [I-D.friel-anima-brski-cloud] Friel, O., Shekh-Yusef, R., and M. Richardson, "BRSKI Cloud Registrar", Work in Progress, Internet-Draft, draft-friel-anima-brski-cloud-04, 6 April 2021, <<https://www.ietf.org/archive/id/draft-friel-anima-brski-cloud-04.txt>>.

- [I-D.ietf-anima-brski-async-enroll]
Fries, S., Brockhaus, H., Lear, E., and T. Werner,
"Support of asynchronous Enrollment in BRSKI (BRSKI-AE)",
Work in Progress, Internet-Draft, draft-ietf-anima-brski-
async-enroll-03, 24 June 2021,
<[https://www.ietf.org/archive/id/draft-ietf-anima-brski-
async-enroll-03.txt](https://www.ietf.org/archive/id/draft-ietf-anima-brski-async-enroll-03.txt)>.
- [I-D.ietf-anima-constrained-voucher]
Richardson, M., Stok, P. V. D., Kampanakis, P., and E.
Dijk, "Constrained Voucher Artifacts for Bootstrapping
Protocols", Work in Progress, Internet-Draft, draft-ietf-
anima-constrained-voucher-12, 11 July 2021,
<[https://www.ietf.org/archive/id/draft-ietf-anima-
constrained-voucher-12.txt](https://www.ietf.org/archive/id/draft-ietf-anima-constrained-voucher-12.txt)>.
- [I-D.kuehlewind-update-tag]
Kuehlewind, M. and S. Krishnan, "Definition of new tags
for relations between RFCs", Work in Progress, Internet-
Draft, draft-kuehlewind-update-tag-04, 12 July 2021,
<[https://www.ietf.org/archive/id/draft-kuehlewind-update-
tag-04.txt](https://www.ietf.org/archive/id/draft-kuehlewind-update-tag-04.txt)>.
- [I-D.richardson-anima-voucher-delegation]
Richardson, M. and W. Pan, "Delegated Authority for
Bootstrap Voucher Artifacts", Work in Progress, Internet-
Draft, draft-richardson-anima-voucher-delegation-03, 22
March 2021, <[https://www.ietf.org/archive/id/draft-
richardson-anima-voucher-delegation-03.txt](https://www.ietf.org/archive/id/draft-richardson-anima-voucher-delegation-03.txt)>.
- [onpath] "can an on-path attacker drop traffic?", n.d.,
<[https://mailarchive.ietf.org/arch/msg/saag/
mlr9uo4xYznOcf85EyK0Rhut598/](https://mailarchive.ietf.org/arch/msg/saag/mlr9uo4xYznOcf85EyK0Rhut598/)>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
Housley, R., and W. Polk, "Internet X.509 Public Key
Infrastructure Certificate and Certificate Revocation List
(CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
<<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
RFC 5652, DOI 10.17487/RFC5652, September 2009,
<<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu,
"Handling Long Lines in Content of Internet-Drafts and
RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020,
<<https://www.rfc-editor.org/info/rfc8792>>.

- [RFC8812] Jones, M., "CBOR Object Signing and Encryption (COSE) and JSON Object Signing and Encryption (JOSE) Registrations for Web Authentication (WebAuthn) Algorithms", RFC 8812, DOI 10.17487/RFC8812, August 2020, <<https://www.rfc-editor.org/info/rfc8812>>.
- [RFC8949] 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/info/rfc8949>>.

Appendix A. Examples

These examples are folded according to [RFC8792] Single Backslash rule.

A.1. Example Voucher Request (from Pledge to Registrar)

The following is an example request sent from a Pledge to the Registrar. This example is from the Siemens reference Registrar system.

```
<CODE BEGINS> file "voucher_request_01.b64"
eyJhbGciOiAiRVMyNTYiLCAieDVjIjogWyJNSU1CMmpDQ0FZQ2dBd0lCQWd\
R0FXZWdkY1NMTUfvr0NDcUdTtTQ5QkFNQ01EMHhDekFKQmdOVkJBWVRBa0Z\
TVJVd0V3WURWUWFLREF4S2FXNW5TbWx1WjB0dmNuQXhGekFWQmdOVkJB\
a3BwYm1kS2FXNW5WR1Z6ZEVOQk1DQVhEVEU0TVRJeE1qQXpNamcxTVZvWUR\
azVPVGt4TWpNeE1qTTFPVFU1V2pCU01Rc3dDUVlEVlFRR0V3SkJVVVEVWTUJ\
R0ExVUVDZ3dNU21sdVowcHBibWREYjNkd01STXdfUVVlEVlFRRkV3b3dNVE1\
TkRVMk56ZzVNUmN3RlFzRFRZRUUREQTvLYVclblNtbHVhVAMFJsZGlsalpuQ1p\
Qk1HQNlxR1NNND1BZ0VHQ0Nxr1NNND1Bd0VIQTBJQUJNVkdHOfo1cGpmNWp\
bnlyVXJYeVoxa1BncUJlM05YdTFkVEFEZStyL3Y2SnPJSgwzNTVJZ2NIQzN\
eHBpYnFKTS9iV1JhRXlqcWNSmo0akprb3dDdWpWVEJUTUN3R0NTc0dBUVF\
Z3U1U0FnUWZEQjF0WVhOaExYUmxjM1F1YzJsbGJXVnVjeTFpZEM1dVpYUTZ\
VFEwTXpBVEJnTlZlU1VFRERBS0JnZ3JCZ0VGQ1FjREFqQU9CZ05WSFE4QkF\
OEVCCU1DQjRBd0NnWU1Lb1pJemowRUF3SURTQUF3U1FJZ1d0UHpsJSVhZMm1\
UlhKdEV4S0VoaFpkYTRYK0VwbFpvbUVJMNpBMGRzam9DSVFDM0pwUW1SWE1\
bi9wNEJlOW16aWk5MmVjbFR4NC9PNHJsbTdNeUxxa2hkQT09Il19.eyJpZX\
mLXZvdWNoZXItcmVxdWVzdDp2b3VjaGVyIjogeyJjcmVhdGVkLW9uIjogIj\
wMjAtMTAtMjJUMDI6Mzc6MzkuMDAwWiIsICJub25jZSI6ICJlRHMmKy9GdU\
IR1VuUnhOM0UxNENRPT0iLCAic2VyaWFsLW51bWJlciI6IClwMTIzNDU2Nz\
5In19.Vj9pyo43KDEq0e5tokwHpNhVM0uUkLCatwNQxfSCKH8GRQ2iTT2fq\
39k40M-7S-vheDHHuBHFSWb502EPwkda
<CODE ENDS>
```

It contains the following three parts:

Header:

```

<CODE BEGINS> file "voucher_request_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIB2jCCAYCgAwIBAgIGAWeGdcSLMAoGCCqGSM49BAMCMD0xCzAJBg\
VBAYTAKFRMRUwEwYDVQQKDAxKaW5nSmluZ0NvcnAxZzAVBgNVBAMMDkppbm\
KaW5nVGZvdENBMCAXDTE4MTIxMjAzMjg1MVoYDzK5OTkxMjMxMjM1OTU5Wj\
SMQswCQYDVQQGEwJBUTEVMBMGA1UECgwMSmluZ0ppbmdb3JwMRMwEQYDVQ\
FEowMTIzNDU2Nzg5MRcwFQYDVQQDDA5KaW5nSmluZ0Rldm1jZTBZMBMGBY\
GSM49AgEGCCqGSM49AwEHA0IABMVGG8Z5pjf5jXnyrUrXyZ1kPgqBe3NXu1\
TADe+r/v6JzIH1355IgcHC3axpibqJM/bWRaEyjqcCJj4jJkowCu jVTBTMC\
GCSsGAQQBgu5SAgQfDBltYXNhLXRlc3Quc2l1bWVucylidC5uZXQ6OTQ0Mz\
TBgNVHSUEDDAKBggrBgEFBQcDAjAOBgNVHQ8BAf8EBAMCB4AwCgYIKoZIzj\
EAwIDSAAwRQIgWtPzIIXY2ixRXJtExKEhhZda4X+Ep1ZomEI2zA0dsj0CIQ\
3JpQmRXMGn/p4Bu9izii92ec1Tx4/04rlm7MyLqkhdA=="
  ]
}
<CODE ENDS>

```

Payload:

```

<CODE BEGINS> file "voucher_request_01-payload.b64"
{
  "ietf-voucher-request:voucher": {
    "created-on": "2020-10-22T02:37:39.000Z",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "serial-number": "0123456789"
  }
}
<CODE ENDS>

```

Signature:

```

<CODE BEGINS> file "voucher_request_01-signature.b64"
Vj9pyo43KDEq0e5tokwHpNhVM0uUkLCatwNQxfSCKH8GRQ2iTT2fqD39k40\
-7S-vheDHHuBHFSWb502EPwkDA
<CODE ENDS>

```

A.2. Example Parboiled Voucher Request (from Registrar to MASA)

The term parboiled refers to food which is partially cooked. In [BRSKI], the term refers to a voucher-request which has been received by the Registrar, and then has been processed by the Registrar ("cooked"), and is now being forwarded to the MASA.

DWjNVMVUwRm5VV1pFUWpGMFdWae9hRXhzVW14ak0xRjFZekpzYkdKWFZuVm\
lVEZwWkVNMWRWcFlVVFpQVkJZFd1RYcEJWRUpuVGxaSVUxVkJZSRVJCUzBKb1\
zSkNaMFZHUWxGalJFRnFRVTlDWjA1V1NGRTRRa0ZtT0VWQ1FVMURRa1JCZD\
ObldVbExiMXBKZW1vd1JVRjNTVVJUUVVGM1VsRkpaMWQwVUhwS1NWaFpNbW\
0VWxoS2RFVjRTMFZvYUZwal1UU11LMFZ3YkZwdmJVVkpNbnBCTUdSemFtOU\
TVkZETTBwd1VXMVNXRTFIYmk5d05FSjFPV2w2YVdrNU1tVmpiRlI0TkM5UE\
ISnNiVGROZVV4eGEyaGtRVDA5SWwxOS5leUpwWlhSbUxYWnZkV05vWlhJdG\
tVnhkV1Z6ZERwMmIzVmphR1Z5SWpvZ2V5SmpjbVZoZEdWa0xXOXVJam9nSW\
Jd01qQXRNVEF0TWpKVU1ESTZNemM2TXprdU1EQXdXaU1zSUNKdWIyNWpaU0\
2SUNKbFJITXJLeTlHZFVSSVixVnVbhmhPTTBVeE5FTlJQVDBpTENBaWMyVn\
hV0ZzTFc1MWJXSmxjaUk2SUNJd01USXpORFUyTnpnNUluMTkuVmo5cH1vND\
LREVxMGUldG9rd0hwTmhWTTB1VWtMQ2F0d05ReGZzQ0tIOEdSUTJpVFQyZn\
EMzlrNDBNLtdTLXZoZURISHVCSEZTV2I1MDJFUHdRZEEiLCJjcmVhdGVkLW\
uIjoimjAyMC0xMC0yM1QwMjozNzozOS4yMzVaIn19.S3BRYIKHbsqwQEZsB\
J1COIVAx02NPEc5oo_BnXK_JkQfStTieHFCALdv5MzYdTU9myJO1muaSFEI\
_NFMSFjA
<CODE ENDS>

It contains the following three parts:

Header:

```

<CODE BEGINS> file "parboiled_voucher_request_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIBozCCAUqgAwIBAgIGAW0eLuIFMAoGCCqGSM49BAMCMDUxEzARBg\
VBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3\
DQTAeFw0xOTA5MTEwMjMzMzJaFw0yOTA5MTEwMjMzMzJaMFQxEzARBgNVBA\
MCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxLjAsBgNVBAMMJVJlZ2lzdH\
hciBWB3VjaGVyIFJlcXVlc3QgU2lnbm1uZyBLZXkwWTATBgqcqhkJOPQIBBg\
qhkJOPQMBBwNCAAT6xVvAvqTz1ZUiuNWhXpQskaPy7AHHQLwXiJ0iELt6uN\
anAN0QnWMyO/0CDEjIkBQobw8YKqjtxJHVSGTj9KOoycwJTATBgNVHSUEDD\
KBggrBgEFBQcDHDAAOBgNVHQ8BAf8EBAMCB4AwCgYIKoZlZj0EAwIDRwAwRA\
gYr2LfqoaCKDF4RACMmJi+NCZqdSiuVugISA7OhKRq3YCIDxnPMMnpXAMTr\
JuPWyceER11PxHOn+0CpSHi2qgpWX",
    "MIIBpDCCAUmGAwIBAgIGAW0eLuH+MAoGCCqGSM49BAMCMDUxEzARBg\
VBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3\
DQTAeFw0xOTA5MTEwMjMzMzJaFw0yOTA5MTEwMjMzMzJaMDUxEzARBgNVBA\
MCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUxDzANBgNVBAMMB1Rlc3RDQT\
ZMBMGBByqGSM49AgEGCCqGSM49AwEHA0IABOkvkTHu8Q1T3FHJ1UaI7+WsHO\
0US3SALtG5wuKQDjiex06/ScY5PJibvgHTB+F/QTjge1HGy1YKpwcNMcsSy\
jRTBDMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYDVROPAQH/BAQDAgIEMB0GA1\
dDgQWBBToZIMzQdsD/j/+gX/7cBJucH/XmjAKBgqgqhkJOPQQDAgNJADBGAi\
AtxQ3+ILGBPItSh4b9WXhXNuhqSP6H+b/LC/fVYDjQ6oCIQDG2uRCH1Vq3y\
B58TXMUbzH8+0lhWUv0lRD3VEqDdcQw=="
  ]
}
<CODE ENDS>

```

Payload:

```
<CODE BEGINS> file "parboiled_voucher_request_01-payload.b64"
{
  "ietf-voucher-request:voucher": {
    "serial-number": "0123456789",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "prior-signed-voucher-request": "eyJhbGciOiAiAirmYmNTYiLC\
ieDVjiJogWyJNSU1CMmpDQ0FZQ2dDd01CQWdJR0FXZWNkY1NMTUFvR0NDcU\
TTQ5QkFkQ01EMHhDekFKQmdOVkJBWVRBa0ZSTVJVd0V3WURWUVFLREF4S2\
XNW5TbWx1WjB0dmNuQXhGekFWQmdOVk1JBTU1Ea3BwYm1kS2FXNW5WR1Z6ZE\
OQk1DQVhEVEU0TVRJeE1qQXpNamcxTVZvWUR6azVPVGt4TWpNeE1qTTFPVF\
1V2pCU01Rc3dDUV1EV1FRR0V3SkJVVEVWTUJNR0ExVUVDZ3dNU21sdVowcH\
ibWREYjNkd01STXdfUV1EV1FRRkV3b3dNVE16TkRVMk56ZzVNUmN3RlFZRf\
RUUREQTvLYVc1b1NtbHVhVAMFJSZG1salpUQlpNQk1HQnlxR1NNND1BZ0VHQ0\
xR1NNND1Bd0VIQTBJQUJNVkdHOFolcGpmNWpYbnlyVXJYeVoxa1BncUJlM0\
YdTFkVEFEZStyL3Y2SnPJSgwzNTVJZ2NIQzNheHBpYnFKTS9iV1JhRXlqcW\
DSmo0akprb3dDdWpWVEJUTUN3R0NTc0dBUVFCZ3U1U0FnUWZEQjF0WVhOaE\
YUmxjM1F1YzJsbGJXVnVjeTFpZEM1dVpYUTZPVFEwTXpBVEJnTlZlU1VFRE\
BS0JnZ3JCZ0VGQ1FjREFqQU9CZ05WSFE4QkFmOEVCQU1DQjRDb0NnWU1Lb1\
JemowRUF3SURTQUF3U1FJZ1d0UHpJSVhZMml4U1hKdEV4S0VoaFpkYTRYK0\
wbFpYvUUVJMNpBMGRzam9DSVFDMD0pwUW1SWE1Hbi9wNEJlOW16aWk5MmVjbF\
4NC9PNHJsbTdNeUxxa2hkQT09I119.eyJpZXRmLXZvdWNoZXItcmVxdWVzd\
p2b3VjaGVyIjogeyJjcmVhdGVkLW9uIjogIjIwMjAtMTAtMjU1JUMDI2Mzc2M\
kuMDAwWiIsICJub25jZSI6ICJlRHMkYy9GdURIRlVuUnhOM0UxNENRPT0iLC\
Aic2VyaWFsLW51bWJlciI6ICJlRHMkYy9GdURIRlVuUnhOM0UxNENRPT0iLC\
kwHpNhVM0uUkLCatwNQxfSCKH8GRQ2iTT2fQd39k40M-7S-vheDHHuBHFsw\
502EPwkdA",
    "created-on": "2020-10-22T02:37:39.235Z"
  }
}
<CODE ENDS>
```

Signature:

```
<CODE BEGINS> file "parboiled_voucher_request_01-signature.b64"
S3BRYIKHbsqwQEzSbgJlCOIVAxO2NPEc5oo_BnXK_JkQfStTieHFCALdv5M\
YdTU9myJO1muaSFEIu_NFMSFjA
<CODE ENDS>
```

A.3. Example Voucher Result (from MASA to Pledge, via Registrar)

The following is an example voucher sent from the Registrar to the MASA. This example is from the Siemens reference MASA system.

```
<CODE BEGINS> file "voucher_01.b64"
eyJhbGciOiJFUzI1NiIsIng1YyI6WyJNSU1Ca3pDQ0FUaWdBd01CQWdJR0F\
RkJqQ2tZTUfVr0NDcUdTTTQ5QkFNQ01EMHhDekFKQmdOVkJBWVRBa0ZSTVJ\
d0V3WURWUWFLREF4S2FXNW5TbWx1WjBodmNuQXhGekFWQmdOVkJBWVRBa0ZSTVJ\
Ym1kS2FXNW5WR1Z6ZEVOQk1CNFhEVEU0TURFeU9URXdOVEkwTUZvWERUSTR\
REV5T1RFd05USTBNRm93VHpFTTE1Ba0dBMVVFQmhNQ1FWRXhGVEFUMmdOVk\
b01ERXBWYm1kS2FXNW5RMj15Y0RfcE1DY0dBMVVFQXZ3Z1NtbHVAMHBWYm1\
RGIzSndJRlp2ZFd0b1pYSWdVMmxuYm1sdVp5QkxaWGt3V1RBVEJnY3Foa2p\
UFFJQk1JnZ3Foa2pPUFFNQk1J3TkNBQVNDNmJ1TEFtZXExVnc2aVFyUnM4UjB\
Vys0YjFHV3lkVdzMkdBTUZXZ2JpdG9yYk1YSDNpUUhLVnU4czJSdm1CR05\
dk9LR0JISHRCZG1GRVpadmI3b3hJd0VEQU9CZ05WSFE4QkFmOEVCQU1DQjR\
d0NnWU1Lb1pJemowRUF3SURTUUF3UmdJaEFJNFBZYNh0c3NIUDJWSHgvH\
b1EvU3N5ZEwzMERRSU5FdGNOOW1DVfHQW1FQXZJYjNvK0ZPM0JUbmNMRn\
SlpSQWtkN3pPdXNuLy9aS09hRUtic1ZEaVU9I119.eyJpZXRmLXZvdWNoZX\
6dm91Y2hlc1I6eyJhc3N1cnRpb24iOiJsb2dnZWQ1LCJzZXJpYWwtbnVtYm\
yIjoimDEyMzQ1Njc4OSIsIm5vbmN1IjoizURzKysvRnVESEdVb1J4TjNFMT\
DUT09Iiw1Y3JlYXR1ZC1vbiI6IjIwMjAtMTAtMjU1MDI6MzYzOTIwOTIw\
sInBpbm5lZC1kb21haW4tY2VydCI6Ikl1JSUJwRENDQVVTZ0F3SUJBZ01HQV\
wZUx1SctNQW9HQ0NxR1NNND1CQU1DTURVeEV6QVJCZ05WQkFvTUNrMTVRbl\
6YVc1bGMzTXhEVEFMQmdOVk1JBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUm\
jM1JEUVRBZUZ3MHhPVEE1TVRFd01qTTNNeKphRncweU9UQTVNVEV3TWpNM0\
6SmFNRFV4RXpBUk1JN1ZCQW9NQ2sxnVfUvNphVzVsYzNNeERUQUxCZ05WQk\
jTUJGTnBkR1V4RHpBTk1JN1ZCQW9NQ2sxnVfUvNphVzVsYzNNeERUQUxCZ05WQk\
5QWdFR0NDcUdTTTQ5QXZdFSEwSUFCT2t2a1RidThRbFQzRkhKMVhStcrV3\
IT2IwVVMzU0FMdEc1d3VLUURqaWV4MDYvU2NZNVBKaWJ2Z0hUQitGL1FUam\
lbEhHeTFZS3B3Y05NY3NTEwWFqU1RCRE1CSUdBMVVKRXdfQi93UU1NQV1CQW\
4Q0FRRXdeZ11EV1IwUEFRSC9CQVFEQWdJR1CMEdBmVVKRGdRV0JCVG9aSU\
6UWRzRC9qLytnWC83Y0JKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQk\
BaUVBdHhRMytJTEdCUE10U2g0Yj1XWGHYtNvocVNQNkgrYi9MQy9mV11Ea1\
2b0NJUURHMnVSQ0hsVnEzeWhCNThUWE1VYnpIOctPbGhXVXZPbFJEM1ZFcu\
kY1F3PT0ifX0.uliO_VB6xIhE8QuhKDGgCkxzsR20IoL0p6qYKpYBDtgkR\
2ykDO_QFjk7W8P5ATW-CQnW1J3ILSeiwMf9nI0g
<CODE ENDS>
```

It contains the following three parts:

Header:

```
<CODE BEGINS> file "voucher_01-header.b64"
{
  "alg": "ES256",
  "x5c": [
    "MIIBkzCCATigAwIBAgIGAWFBjCkYMAoGCCqGSM49BAMCMD0xCzAJBg\
VBAYTAKFRMRUwEwYDVQQKDAxKaW5nSmluZ0NvcnAxZzAVBgNVBAMMDkppbm\
KaW5nVGZvdENBMB4XDTE4MDEyOTEwNTI0MfoXDTI4MDEyOTEwNTI0MfoWTz\
LMAkGA1UEBhMCQVEFTATBgNVBAoMDEppbmdKaW5nQ29ycDEpMCcGA1UEAw\
gSmluZ0ppbmdDb3JwIFZvdWNoZXIqU2lnbmhluZyBLZXkwWTATBgqcqhkJOPQ\
BBggqhkJOPQMBBwNCAASC6beLameq1Vw6iQrRs8R0ZW+4b1GWydMws2GAMF\
wbitf2nIXH3OqHKVu8s2RviBGNivOKGBHtBdiFEZZvb7oxIwEDAOBgNVHQ\
BAf8EBAMCB4AwCgYIKoZIzj0EAwIDSQAwwRgIhAI4PYbxtssHP2VHx/tzUoQ\
SsydL30DQINetCn9mCTXPAiEAvIb3o+F03BTncLFsaJZRAkd7zOusn//ZKO\
EKbsVDiU="
  ]
}
<CODE ENDS>
```

Payload:

```
<CODE BEGINS> file "voucher_01-payload.b64"
{
  "ietf-voucher:voucher": {
    "assertion": "logged",
    "serial-number": "0123456789",
    "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
    "created-on": "2020-10-22T02:37:39.921Z",
    "pinned-domain-cert": "MIIBpDCCAUmGAWIBAgIGAW0eLuH+MAoG\
CqGSM49BAMCMDUxEzARBgNVBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNp\
GUxDzANBgNVBAMMB1Rlc3RDQTAeFw0xOTA5MTEwMjMzJaFw0yOTA5MTEw\
jM3MzJaMDUxEzARBgNVBAoMCK15QnVzaW5lc3MxDTALBgNVBACMBFNpdGUx\
zANBgNVBAMMB1Rlc3RDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABOkv\
THu8Q1T3FHJ1UaI7+WsHob0US3SALtG5wuKQDjiex06/ScY5PJibvgHTB+F\
QTjgelHGylYKpwcNMcsSyaJRTBDMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYD\
R0PAQH/BAQDAgIEMB0GA1UdDgQWBbToZIMZQdsD/j/+gX/7cBJucH/XmjAK\
ggqhkJOPQDAGNJADBGAiEAtxQ3+ILGBPItSh4b9WXhXNuhqSP6H+b/LC/f\
YDjQ6oCIQDG2uRCH1Vq3yhB58TXMubzH8+OlhWUvOlRD3VEqDdcQw=="
  }
}
<CODE ENDS>
```

Signature:

```
<CODE BEGINS> file "voucher_01-signature.b64"
u1iO_VB6xIhE8QuhKDGgCcxksnR20IoL0p6qYKpYBDtgkRT2ykDO_QFjk7W\
P5ATW-CQnWlJ3ILSeiwMf9nI0g
<CODE ENDS>
```

Authors' Addresses

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Thomas Werner
Siemens
Email: thomas-werner@siemens.com