ANIMA WG
Internet-Draft
Intended status: Standards Track
Expires: 15 August 2025

S. Fries
T. Werner
Siemens
E. Lear
Cisco Systems
M. Richardson
Sandelman Software Works
11 February 2025

BRSKI with Pledge in Responder Mode (BRSKI-PRM)
draft-ietf-anima-brski-prm-18

Commenté [MB1]: Consider having a reference figure early in the document with the various entities.

Abstract

This document defines enhancements to Bootstrapping a Remote Secure Key Infrastructure (BRSKI, RFC 8995) to enable bootstrapping in domains featuring no or only limited connectivity between a pledge and the domain registrar. It specifically changes the interaction model from a pledge-initiated mode, as used in BRSKI, to a pledge-responding mode, where the pledge is in server role. For this, BRSKI with Pledge in Responder Mode (BRSKI-PRM) introduces new endpoints for the Domain Registrar and pledge, and a new component, the Registrar-Agent, which facilitates the communication between pledge and registrar during the bootstrapping phase. To establish the trust relation between pledge and registrar, BRSKI-PRM relies on object security rather than transport security. Thise approach ~~defined here~~ is agnostic to the enrollment protocol that connects the domain registrar to ~~the~~ a Key Infrastructure (e.g., domain Certification ~~A~~Authority).

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-anima-brski-prm/.

Source for this draft and an issue tracker can be found at https://github.com/anima-wg/anima-brski-prm.

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 15 August 2025.

Copyright Notice

Copyright (c) 2025 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.

1. Introduction

BRSKI as defined in [RFC8995] specifies a solution for secure zero-touch (automated) bootstrapping of devices (pledges) in a customer domain, which may be associated with a specific installation location. This includes the discovery of the BRSKI registrar in the customer domain and the exchange of security information necessary to establish trust between a pledge and the domain.

Security information about the customer domain, specifically the customer domain certificate, ~~are~~ is exchanged and authenticated utilizing signed data objects, the voucher artifacts as defined in [RFC8995]. In response to a voucher-request, the Manufacturer Authorized Signing Authority (MASA) issues the voucher and provides it via the domain registrar to the pledge. [I-D.ietf-anima-rfc8366bis] specifies the format of the voucher artifacts, including the voucher-request artifact.

For the certificate enrollment of devices, BRSKI relies on Enrollment over Secure Transport (EST) (~~Enrollment over Secure Transport,~~ [RFC7030]~~)~~ to request and distribute customer domain specific device certificates. EST in turn relies for the authentication and authorization of the certification request on the credentials used by the underlying TLS between the EST client and ~~the~~ an EST server.

BRSKI addresses scenarios in which ~~the~~ a pledge initiates the bootstrapping acting as client (referred to as initiator mode by this document). BRSKI with Pledge in Responder Mode (BRSKI-PRM) defined in this document allows the pledge to act as server, so that it can be triggered externally and, at a specific time, to generate bootstrapping requests in the customer domain. For this approach, this document:

* defines additional endpoints for the domain registrar and new endpoints for the pledge to enable responder mode.

* introduces the Registrar-Agent as new component to facilitate the communication between the pledge and ~~the~~ a domain registrar. The Registrar-Agent may be implemented as an integrated functionality

Commenté [MB2]: Be consistent how this is expanded Xccc Xccc Xcc (XXX) or XXX (Xccc Xccc Xcc).

Commenté [MB3]: There might be many. No?

of a commissioning tool or be co-located with the domain registrar itself. BRSKI-PRM supports the identification of the Registrar-Agent that was performing the bootstrapping allowing for accountability of the pledges' installation, when the Registrar-Agent is a component used by an installer and not co-located with the domain registrar.

* specifies additional artifacts for the exchanges between a pledge acting as server, the Registrar-Agent acting as client, and the domain registrar acting as server toward the Registrar-Agent.

* allows the application of Registrar-Agent credentials to establish TLS connections to ~~the~~ a domain registrar; these are different from the pledge IDevID credentials.

* also enables the usage of alternative transports, both IP-based and non-IP, between the pledge and the domain registrar via the Registrar-Agent; security is addressed at the application layer through object security with an additional signature wrapping the exchanged artifacts.

Commenté [MB4]: Examples to cite here?

The term endpoint used in the context of this document is equivalent to resource in HTTP [RFC9110] and CoAP [RFC7252]; it is not used to describe a device. Endpoints are accessible via Well-Known URIs [RFC8615].

To utilize EST [RFC7030] for enrollment, the domain registrar performs pre-processing of the wrapping signature before actually using EST as defined in [RFC7030].

There may be pledges that can support both modes, initiator and responder mode. In these cases, BRSKI-PRM can be combined with BRSKI as defined in [RFC8995] or BRSKI-AE [I-D.ietf-anima-brski-ae] to allow for more bootstrapping flexibility.

Commenté [MB5]: Need to expose this capabilities? How this is managed?

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~~makes use of the ~~terminology~~ terms defined in Section 1.2 of [RFC8995]. The following terms are defined in addition:

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

CA: Certification Authority. An entity, which issues certificates and maintains certificate revocation information.

Commissioning tool: Tool to interact with devices to provide configuration data.

CSR: Certificate Signing Request.

Commenté [MB6]: This is not a definition. Mau be point to an RFC.

EE: End entity, as defined in [RFC9483]. Typically, a device or service that owns a public-private key pair for which it manages a public key certificate.

EE certificate: the certificate of the EE signed by its owner (e.g., CA). For domain components, the EE certificate is signed by the domain owner. For the pledge, the EE certificate is either the IDevID certificate signed by the manufacturer or the LDevID certificate signed by the domain owner or an application-specific EE certificate signed by the domain owner.

endpoint: Term equivalent to resource in HTTP [RFC9110] and CoAP [RFC7252]. Endpoints are accessible via Well-Known URIs [RFC8615].

Commenté [MB7]: Note that CoAP has also «Endpoint», which is not identical to resource.

IDevID: An Initial Device Identifier X.509 certificate installed by the vendor on new equipment. This is a term from 802.1AR [IEEE-802.1AR].

LDevID: A Local Device Identifier X.509 certificate installed by the owner of the equipment. This is a term from 802.1AR [IEEE-802.1AR].

mTLS: mutual Transport Layer Security.

Commenté [MB8]: Add reference

PER: Pledge Enroll-Request is a signature-wrapped CSR, signed by the pledge that requests enrollment to a domain via the Registrar-Agent.

POI: Proof-of-Identity, as defined in [RFC5272].

POP: Proof-of-Possession (of a private key), as defined in [RFC5272].

Commenté [MB9]: Unfortunate as this one is widely used for Point of Presence

PVR: Pledge Voucher-Request is a signature-wrapped voucher-request, signed by the pledge that sends it to the domain registrar via the Registrar-Agent.

RA: Registration Authority, an optional system component to which a CA delegates certificate management functions such as authorization checks. In BRSKI-PRM, this is a functionality of the domain registrar, as in BRSKI [RFC8995].

Registrar-Agent~~Registrar_Agent~~: Component facilitating the data exchange between a pledge in responder mode and a domain registrar.

Commenté [MB10]: Consistent with the use in the doc.

RVR: Registrar Voucher-Request is a signature-wrapped voucher-request, signed by the domain registrar that sends it to the MASA. For BRSKI-PRM, it contains a copy of the original PVR received from the pledge.

This document uses the following encoding notations in the given JWS-signed artifact examples:

BASE64(OCTETS): Denotes the base64 encoding of an octet sequence using the character set defined in Section 4 of [RFC4648] and without the inclusion of any line breaks, whitespace, or other additional characters. Note that the base64 encoding of the empty octet sequence is the empty string.

BASE64URL(OCTETS): Denotes the base64url encoding of an octet sequence, per Section 2 of [RFC7515].

UTF8(STRING): Denotes the octet sequence of the UTF-8 [RFC3629] representation of STRING, per Section 1 of [RFC7515].

This document includes many examples that would contain many long sequences of base64-encoded objects with no content directly comprehensible to a human reader. In order to keep those examples short, they use the token base64encodedvalue== as a placeholder for base64 data. The full base64 data is included in the appendices of this document.

3. Scope of Solution

3.1. Supported Environments and Use Case Examples

BRSKI-PRM is applicable to scenarios where pledges may have no direct connection to ~~the~~ a domain registrar, may have no continuous connection, or require coordination of the pledge requests to be provided to a domain registrar.

This can be motivated by pledges deployed in environments not yet connected to the operational customer domain network, e.g., at a building construction site, or environments intentionally disconnected from the Internet, e.g., critical industrial facilities. Another example is the assembly of electrical cabinets, which are prepared in advance before the installation at a customer domain.

3.1.1. Building Automation

In building automation, a typical use case exists where a detached building or the basement is equipped with sensors, actuators, and controllers, but with only limited or no connection to the central building management system. This limited connectivity may exist during installation time or also during operation time.

During the installation, for instance, a service technician collects the device-specific information from the basement network and provides them to the central building management system. This could be done using a laptop, common mobile device, or dedicated commissioning tool to transport the information. The service technician may successively collect device-specific information in different parts of the building before connecting to the domain registrar for bulk bootstrapping.

A domain registrar may be part of the central building management system and already be operational in the installation network. The central building management system can then provide operational parameters for the specific devices in the basement or other detached areas. These operational parameters may comprise values and settings required in the operational phase of the sensors/actuators, among them a certificate issued by the operator to authenticate against other components and services. These operational parameters are then provided to the devices in the basement facilitated by the service technician's laptop. The Registrar-Agent, defined in this document, may be run on the technician's laptop to interact with pledges.

3.1.2. Infrastructure Isolation Policy

This refers to any case in which the 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 a domain registrar may be allowed in carefully controlled short periods of time, for example when a batch of new devices are deployed, but prohibited at other times.

3.1.3. Less Operational Security in the Target-Domain

The registration authority (RA) performing the authorization of a certificate request is a critical PKI component and therefore requires higher operational security than other components utilizing the issued certificates. CAs may also require higher security in the registration procedures. There may be situations in which the customer domain does not offer enough physical security to operate an RA/CA and therefore this service is transferred to a backend that offers a higher level of operational security.

3.2. Potential Limitations

The mechanism described in this document presumes the ability of the pledge and the Registrar-Agent to communicate with one another. This may not be possible in constrained environments where, in particular, power must be conserved. In these situations, it is anticipated that the transceiver will be powered down most of the time. This presents a rendezvous problem: the pledge is unavailable for certain periods of time, and the Registrar-Agent is similarly presumed to be unavailable for certain periods of time. To overcome this situation, the pledges may need to be powered on, either manually or by sending a trigger signal.

4. Requirements Discussion and Mapping to Solution-Elements

Commenté [MB11]: What does that mean?

Based on the intended target environment described in Section 3.1, the following boundary conditions are derived to support bootstrapping of pledges in responder mode (acting as server):

* To facilitate the communication between a pledge in responder mode and ~~the~~ a registrar, additional functionality is needed either on the registrar or as a stand-alone component. This new functionality is defined as Registrar-Agent and acts as an agent of the registrar to trigger the pledge to generate requests for voucher and enrollment. These requests are then provided by the Registrar-Agent to the registrar. This requires the definition of pledge endpoints to allow interaction with the Registrar-Agent.

Commenté [MB12]: Repeatition.

* The security of communication between the Registrar-Agent and the pledge ~~must~~ does not rely on Transport Layer Security (TLS) to enable

application of BRSKI-PRM in environments, in which the communication between the Registrar-Agent and the pledge is done over other technologies like BTLE or NFC, which may not support TLS protected communication. In addition, the pledge does not have a certificate that can easily be verified by [RFC9525] methods.

* The use of authenticated self-contained objects addresses both, the TLS challenges and the technology stack challenge.

* By contrast, the Registrar-Agent can be authenticated by the registrar as a component, acting on behalf of the registrar. In addition, the registrar must be able to verify, which Registrar-Agent was in direct contact with the pledge.

* It would be inaccurate for the voucher-request and voucher-response to use the assertion type proximity in the voucher, as the pledge was not in direct contact with the registrar for bootstrapping. Therefore, a new assertion type is necessary for distinguishing assertions the MASA can state.

At least the following properties are required for the voucher and enrollment processing:

* POI: provides data-origin authentication of an artifact, e.g., a voucher-request or an Enroll-Request, utilizing an existing IDevID. Certificate updates may utilize the certificate that is to be updated.

* POP: proves that an entity possesses and controls the private key corresponding to the public key contained in the certification request, typically by adding a signature computed using the private key to the certification request.

Solution examples based on existing technology are provided with the focus on existing ~~IETF~~ RFCs:

* Voucher-Requests and Vouchers as used in [RFC8995] already provide both, POP and POI, through a digital signature to protect the integrity of the voucher, while the corresponding signing certificate contains the identity of the signer.

* Enroll-Requests are data structures containing the information from a requester for a CA to create a certificate. The certification request format in BRSKI is PKCS#10 [RFC2986]. In PKCS#10, the structure is signed to ensure integrity protection and POP of the private key of the requester that corresponds to the contained public key. In the application examples, this POP alone is not sufficient. A POI is also required for the certification request and therefore the certification request needs to be additionally bound to the existing pledge IDevID credential. This binding supports the authorization decision for the certification request and may be provided directly with the certification request. While BRSKI uses the binding to TLS, BRSKI-PRM aims at an additional signature of the PKCS#10 using existing credentials on the pledge (IDevID). This allows the process to be independent of the selected transport.

5. Solution Architecture

5.1. Overview

For ~~BRSKI with Pledge in Responder Mode (~~BRSKI-PRM~~)~~, the base system architecture defined in BRSKI [RFC8995] is enhanced to facilitate new use cases in which the pledge acts as server. The responder mode allows delegated bootstrapping using a Registrar-Agent instead of a direct connection between the pledge and the domain registrar.

Necessary enhancements to support authenticated self-contained objects for certificate enrollment are kept at a minimum to enable reuse of already defined architecture elements and interactions. The format of the bootstrapping objects produced or consumed by the pledge is usually based on JSON Web Signature (JWS) [RFC7515] and further specified in Section 7 to address the requirements stated in Section 4~~ above~~. In constrained environments, it may be based on COSE [RFC9052].

a mis en forme : Surlignage

An abstract overview of the BRSKI-PRM protocol can be found on slide 8 of [BRSKI-PRM-abstract].

Commenté [MB13]: Thanks for this.

To support mutual trust establishment between the domain registrar and pledges not directly connected to the customer domain, this document specifies the exchange of authenticated self-contained objects with the help of the Registrar-Agent.

This leads to extensions of the logical components in the BRSKI architecture as shown in Figure 1.

Note that the Join Proxy is not shown in the figure. In certain situations the Join Proxy may still be present and could be used by the Registrar-Agent to connect to the Registrar. For example, a Registrar-Agent application on a smartphone often can connect to local Wi-Fi without giving up their cellular network connection [androidnsd], but only can make link-local connections.

The Registrar-Agent interacts with the pledge to transfer the required data objects for bootstrapping, which are then also exchanged between the Registrar-Agent and the domain registrar. The addition of the Registrar-Agent influences the sequences of the data exchange between the pledge and the domain registrar described in [RFC8995]. To enable reuse of BRSKI defined functionality as much as possible, BRSKI-PRM:

* uses existing endpoints where the required functionality is provided.

* enhances existing endpoints with new supported media types, e.g., for JWS voucher.

* defines new endpoints where additional functionality is required, e.g., for wrapped certification request, wrapped CA certificates, and new status information.

```
                             +---------------------------+
          ..... Drop Ship .....| Vendor Services           |
          :                    +---------------+-----------+
```

```
       :                    | M anufacturer |           |
       :                    | A uthorized   | Ownership |
       :                    | S igning      | Tracker   |
       :                    | A uthority    |           |
       :                    +---------------+-----------+
       :                                         ^
       :                                         | BRSKI-
       :                                         | MASA
       :          ...............................|.........
       V          .                              v        .
   +---------+    .  +------------+       +-----------+   .
   |         |    .  |            |       |           |   .
   | Pledge  | BRSKI-| Registrar- | BRSKI-| Domain    |   .
   |         |  PRM  | Agent      |  PRM  | Registrar |   .
   |         |<----->|            |<----->|           |   .
   |         |    .  |   EE cert. |       |  EE cert. |   .
   |         |    .  +------------+       +-----+-----+   .
   | IDevID  |    .                             |         .
   |         |    .           +-----------------+-----+   .
   +---------+    .           | Key Infrastructure    |   .
                  .           | (e.g., PKI CA)        |   .
                  .           +-----------------------+   .
                  .........................................
                              Customer Domain
```

Figure 1: BRSKI-PRM architecture overview using Registrar-Agent

Figure 1 shows the relations between the following main components:

* Pledge: Is expected to respond with the necessary data objects for bootstrapping to ~~the~~ a Registrar-Agent. The protocol used between the pledge and the Registrar-Agent is assumed to be HTTP(S) in the context of this document. Any other protocol can be used as long as it supports the exchange of the necessary artifacts. This includes CoAP or protocols to be used over Bluetooth or NFC connections. A pledge acting as server leads to the following differences compared to BRSKI [RFC8995]:

  - The pledge no longer initiates bootstrapping, but is discovered and triggered by ~~the~~ a Registrar-Agent as defined in Section 6.1.2.

  - The pledge offers additional endpoints as defined in Section 6.2, so that ~~the~~ a Registrar-Agent can request data required for bootstrapping the pledge.

  - The pledge includes additional data in the PVR, which is provided and signed by ~~the~~ a Registrar-Agent as defined in Section 7.1. This allows the registrar to identify with which Registrar-Agent the pledge was in contact (see Section 5.4).

  - The artifacts exchanged between the pledge and the registrar via the Registrar-Agent are authenticated self-contained objects (i.e., signature-wrapped artifacts).

* Registrar-Agent: Is a new component defined in Section 6.1 that provides a store and forward communication path to exchange data objects between the pledge and ~~the~~ a domain registrar. This is for

Commenté [MB14]: Can this be outsourced/external to the customer domain?

Commenté [MB15]: In which cases the non-secure mode is used?

situations in which ~~the~~ a domain registrar is not directly reachable by the pledge, which may be due to a different technology stacks or due to missing connectivity. A Registrar-Agent acting as client leads to the following new aspects:

- The order of exchanges in the BRSKI-PRM call flow is different from that in BRSKI [RFC8995], as the Registrar-Agent can trigger one or more pledges and collects the PVR and PER artifacts simultaneously as defined in Section 7. This enables bulk bootstrapping of several devices.

- There is no trust assumption between the pledge and the Registrar-Agent as only authenticated self-contained objects are used, which are transported via the Registrar-Agent and provided either by the pledge or the domain registrar.

- The trust assumption between the Registrar-Agent and the domain registrar may be based on EE certificates that are both signed by the domain owner.

- The Registrar-Agent may be realized as stand-alone component supporting nomadic activities of a service technician moving between different installation sites.

- Alternatively, the Registrar-Agent may also be realized as co-located functionality for a registrar, to support pledges in responder mode.

* Join Proxy (not shown): Has the same functionality as described in [RFC8995] if needed. Note that a Registrar-Agent may use a join proxy to facilitate the TLS connection to the registrar in the same way that a BRSKI pledge would use a join proxy. This is useful in cases where the Registrar-Agent does not have full IP connectivity via the domain network or cases where it has no other means to locate the registrar on the network.

* Domain registrar: In general, fulfills the same functionality regarding the bootstrapping of the pledge in a customer domain by facilitating the communication of the pledge with the MASA service and the domain key infrastructure (PKI). However, there are ~~also~~ differences compared to BRSKI [RFC8995]:

  - A BRSKI-PRM domain registrar does not interact with a pledge directly, but through the Registrar-Agent as defined in Section 7.

  - A BRSKI-PRM domain registrar offers additional endpoints as defined in Section 6.3 to support the signature-wrapped artifacts used by BRSKI-PRM.

* Vendor services: Encompass MASA and Ownership Tracker and are used as defined in [RFC8995]. A MASA responsible for pledges that implement BRSKI-PRM is expected to support BRSKI-PRM extensions:

  - The default format for voucher artifacts (incl. voucher-request) is JWS-signed JSON as defined in [I-D.ietf-anima-jws-voucher].

- The Agent Proximity Assertion (~~see~~ Section 5.4) requires additional validation steps as defined in Section 7.3.1.

5.2. Nomadic Connectivity

In one example instance of the PRM architecture as shown in Figure 2, there is no connectivity between the location in which the pledge is installed and the location of the domain registrar. This is often the case in the ~~aforementioned~~ building automation use case mentioned in ~~(~~Section 3.1.1~~)~~.

```
                            +--------------------------+
     ..... Drop Ship .....| Vendor Services          |
     :                      +--------------------------+
     :                                            ^
 .......................................          |
 .   v                                   .        |
 . +--------+           .-.-.-.-.-.-.-. .        |
 . |        | BRSKI-PRM : Registrar-  : .        |
 . | Pledge |<---------->: Agent       : .        |
 . +--------+ L2 or L3  :-.-.-.-.-.-.-: .        | BRSKI-
 .          connectivity   ^            .        | MASA
 .........................!............          |
    Pledge Installation    !                      |
    Location               ! Nomadic              |
                           ! connectivity         |
                           !                      |
                ...........!....................|.........
                .          v                    v        .
                .  .-.-.-.-.-.-.-. BRSKI- +-----------+  .
                .  : Registrar-  :  PRM   | Domain    |  .
                .  : Agent       :<------>| Registrar |  .
                .  :-.-.-.-.-.-.-:        +-----+-----+  .
                .                               |        .
                .            +------------------+-----+  .
                .            | Key Infrastructure     |  .
                .            | (e.g., PKI CA)         |  .
                .            +------------------------+  .
                ..........................................
                             Customer Domain
```

Figure 2: Registrar-Agent nomadic connectivity example

BRSKI-PRM enables support of this case through nomadic connectivity of the Registrar-Agent. To perform enrollment in this setup, multiple round trips of the Registrar-Agent between the pledge installation location and the domain registrar are required.

1. Connectivity to domain registrar: preparation tasks for pledge bootstrapping not part of the BRSKI-PRM protocol definition, like retrieval of list of pledges to enroll.

2. Connectivity to pledge installation location: retrieve information about available pledges (IDevID), collect request

objects (i.e., Pledge Voucher-Requests and Pledge Enroll-Requests using the BRSKI-PRM approach described in Section 7.1 and Section 7.2).

3. Connectivity to domain registrar, submit collected request information of pledges, retrieve response objects (i.e., Voucher and Enroll-Response) using the BRSKI-PRM approach described in Section 7.3 and Section 7.4.

4. Connectivity to pledge installation location, provide retrieved objects to the pledges to enroll pledges and collect status using the BRSKI-PRM approach described in Section 7.6, Section 7.7, and Section 7.8.

5. Connectivity to domain registrar, submit Voucher Status and Enrollment Status using the BRSKI-PRM approach described in Section 7.9 and Section 7.10.

Variations of this setup include cases where the Registrar-Agent uses, for example, Wi-Fi to connect to the pledge installation network, and mobile network connectivity to connect to the domain registrar. Both connections may also be possible in a single location at the same time, based on installation building conditions.

5.3. Co-located Registrar-Agent and Domain Registrar

Compared to ~~[RFC8995] ~~BRSKI, pledges supporting BRSKI-PRM can be completely passive and only need to react when being requested to react by a Registrar-Agent. In [RFC8995], pledges instead need to continuously interact with ~~the ~~a domain registrar during onboarding, through discovery, voucher exchange, and enrollment. This may increase the load on the domain registrar, specifically, if a larger number of pledges onboards simultaneously.

```
                          +---------------------------+
    ..... Drop Ship .....| Vendor Service            |
    :                     +---------------------------+
    :                                         ^
    :                                         | BRSKI-MASA
    :          ...............................|.........
    :          .                              v        .
    v          .          +-------------------------+  .
 +--------+    . BRSKI-   |.............            |  .
 |        |    .  PRM     |. Registrar- . Domain    |  .
 | Pledge |<------------->|. Agent      . Registrar |  .
 +--------+ L2 or L3      |.............            |  .
            connectivity  +--------------------+-----+  .
               .                              |        .
               .           +-------------------+-----+  .
               .           | Key Infrastructure     |  .
               .           +------------------------+  .
               .........................................
                           Customer Domain
```

Figure 3: Registrar-Agent integrated into Domain Registrar example

The benefits of BRSKI-PRM can be achieved even without the operational complexity of stand-alone Registrar-Agents by integrating the necessary functionality of the Registrar-Agent as a module into the domain registrar as shown in Figure 3 so that it can support the BRSKI-PRM communications to the pledge.

Commenté [MB16]: May be move to the ops section.

5.4. Agent Proximity Assertion

"Agent proximity" is a statement in the PVR and the voucher that the registrar communicates via ~~the~~ a Registrar-Agent as defined in Section 7 and not directly to the pledge. It is therefore a different assertion than "network proximity", which is defined in Section 3 of [RFC8995]. Hence, [I-D.ietf-anima-rfc8366bis] defines the additional assertion type agent-proximity. This assertion type can be verified by the registrar and MASA during BRSKI-PRM voucher-request processing.

In BRSKI, the pledge verifies POP of the registrar ~~end-entity (~~EE~~)~~ credentials via the TLS handshake and pins that public key as the proximity-registrar-cert into the voucher request. This allows the MASA to verify the proximity of the pledge and registrar, facilitating a decision to assign the pledge to that domain owner. In BRSKI, the TLS session is considered provisional until the pledge receives the voucher to verify POI.

In contrast, in BRSKI-PRM the pledge has no direct connection to the registrar and MUST accept the supplied registrar EE certificate provisionally until it receives the voucher as described in Section 7.6 to verify both POP and POI. The provisional registrar EE certificate is used for the object security along the authenticated self-contained objects that in BRSKI-PRM replace the direct TLS connection to the registrar available in BRSKI [RFC8995]. See also Section 5 of [RFC8995] on "provisional state".

a mis en forme : Surlignage

For the Agent Proximity Assertion, the Registrar-Agent EE certificate and registrar EE certificate must be signed by the same domain owner, i.e., MUST possess a common domain trust anchor in their certificate chain. Akin to the Network Proximity Assertion in BRSKI [RFC8995], the Agent Proximity Assertion provides pledge proximity evidence to the MASA. But additionally, the Agent Proximity Assertion allows the domain registrar to be sure that the PVR supplied by the Registrar-Agent was in fact collected by the Registrar-Agent to which the registrar is connected by utilizing an agent-signed data object.

6. System Components

6.1. Registrar-Agent

The Registrar-Agent ~~is a new component in BRSKI-PRM that provides a store and forward communication path with secure message passing between pledges in responder mode and the domain registrar. It~~ uses its own ~~end-entity (~~EE~~)~~ certificate and corresponding credentials (i.e., private key) for TLS client authentication and for signing agent-signed data objects.

Commenté [MB17]: Already said several times

The Registrar-Agent EE certificate MUST include a SubjectKeyIdentifier as defined in Section 4.2.1.2 of [RFC5280], which is used as a reference within agent-signed data objects as defined in Section 7.1.1.1. Note that this is an additional requirement for issuing the Registrar-Agent EE certificate. [RFC8995] has a similar requirement for the registrar EE certificate.

The SubjectKeyIdentifier is used in favor of providing the complete Registrar-Agent EE certificate in agent-signed data objects to accommodate also constrained environments and reduce bandwidth needed for communication with the pledge. In addition, it follows the recommendation from BRSKI to use SubjectKeyIdentifier in favor of a certificate fingerprint to avoid additional computations.

The provisioning of the Registrar-Agent EE certificate is out of scope for this document, but may be done using its own BRSKI run or by other means such as configuration. It is RECOMMENDED to use short-lived Registrar-Agent EE certificates in the range of days or weeks. This is to address the assumed nature of stand-alone Registrar-Agents as nomadic devices (see Section 5.2) and to avoid potential misuse as outlined in Section 12.3.

Further, the Registrar-Agent requires the registrar EE certificate to provide it to the pledge. It MAY use the certificate verified during server authentication within an initial TLS session with the registrar; in this case, the Registrar-Agent MUST possess the domain trust anchor (i.e., CA certificate) for the registrar EE certificate to verify the certificate chain. Alternatively, the registrar EE certificate MAY be provided via configuration or a repository. The registrar IP address or hostname is provided either by configuration or by using the discovery mechanism defined in [RFC8995] (see Section 6.1.1).

In addition to the certificates, the Registrar-Agent is provided with the product-serial-number(s) of the pledge(s) to be bootstrapped. This is necessary to allow for the discovery of pledges by the Registrar-Agent using DNS-SD with mDNS (see Section 6.1.2). The list may be provided by prior administrative means or the Registrar-Agent may get the information via an (out-of-band) interaction with the pledge. For instance, [RFC9238] describes scanning of a QR code, where the product-serial-number would be initialized from the 12N B005 Product Serial Number data record.

In summary, the following information MUST be available at the Registrar-Agent before the interaction with a pledge:

* Registrar-Agent EE certificate and corresponding private key: own operational credentials to authenticate and sign agent-signed data

* Registrar EE certificate: certificate of the domain registrar to be provided to the pledge

* Serial number(s): product-serial-number(s) of pledge(s) to be bootstrapped; used for discovery

Further, the Registrar-Agent SHOULD have synchronized time.

Finally, the Registrar-Agent MAY possess the IDevID (root or issuing) CA certificate of the pledge manufacturer/vendor to validate the IDevID certificate on returned PVR or in case of optional TLS usage for pledge communication (see Appendix B). The distribution of IDevID CA certificates to the Registrar-Agent is out of scope of this

Commenté [MB18]: Can this be trusted?

Commenté [MB19]: Should we provide more concrete behavior here?

document and may be done by a manual configuration.

6.1.1. Discovery of the Registrar

While the Registrar-Agent requires ~~the~~ an IP address of the domain registrar to initiate a TLS session, a separate discovery of the registrar is likely not needed and a configuration of the domain registrar IP address or hostname is assumed. Registrar-Agent and registrar are domain components that already have a trust relation, as a Registrar-Agent acts as representative of the domain registrar towards the pledge or may even be collocated with the domain registrar. Further, other communication (not part of this document) between the Registrar-Agent and the registrar is assumed, e.g., to exchange information about product-serial-number(s) of pledges to be discovered as outlined in Section 5.2.

Moreover, the ~~standard~~ discovery described in Section 4 of [RFC8995] and ~~the~~ Appendix A.2 of [RFC8995] does not support identification of registrars with an enhanced feature set (like the support of BRSKI-PRM), and hence ~~this~~ that ~~standard~~ discovery is not applicable.

As a more general solution, the BRSKI discovery mechanism can be extended to provide upfront information on the capabilities of registrars, such as the mode of operation (pledge-responder-mode or registrar-responder-mode). Defining discovery extensions is out of scope of this document. For further discussion, see [I-D.ietf-anima-brski-discovery].

6.1.2. Discovery of the Pledge

The discovery of the pledge by ~~the~~ a Registrar-Agent in the context of this document describes the minimum discovery approach that MUST be supported. A more general discovery mechanism, also supporting GRASP besides DNS-SD with mDNS, is discussed in [I-D.ietf-anima-brski-discovery].

Discovery in BRSKI-PRM uses DNS-based Service Discovery [RFC6763] over Multicast DNS [RFC6762] to discover the pledge. Note that Section 9 of [RFC6762] provides support for conflict resolution in situations when a DNS-SD with mDNS responder receives an mDNS response with inconsistent data. Note that [RFC8990] does not support conflict resolution of mDNS, which may be a limitation for its application.

The pledge constructs a Service Instance Name based on device local information (manufacturer/vendor name and serial number), which results in <product-serial-number>._brski-pledge._tcp.local. The product-serial-number composition is ~~manufacturer~~ manufacturer-dependent and may contain information regarding the manufacturer, the product type, and further information specific to the product instance. To allow distinction of pledges, the product-serial-number therefore needs to be sufficiently unique.

Note that this goes against the naming recommendation of [RFC6763]. The _brski-pledge._tcp service, however, targets machine-to-machine discovery.

Commenté [MB20]: This is more about non-discovery :-)

Consider changing to «Explicit Configuration of Registrars»

Commenté [MB21]: How to reconcile this with the intermittent connectivity, including with a name resolution service?

In the absence of a more general discovery as defined in [I-D.ietf-anima-brski-discovery] the Registrar-Agent MUST use

* <product-serial-number>._brski-pledge._tcp.local, to discover a specific pledge, e.g., when connected to a local network.

* _brski-pledge._tcp.local to get a list of pledges to be bootstrapped.

A manufacturer may allow the pledge to react on DNS-SD with mDNS discovery without its product-serial-number contained. This allows a commissioning tool to discover pledges to be bootstrapped in the domain. The manufacturer supports this functionality as outlined in Section 12.4.

Establishing network connectivity of the pledge is out of scope of this document but necessary to apply DNS-SD with mDNS. For Ethernet, it is provided by simply connecting the network cable. For Wi-Fi networks, connectivity can be provided by using a pre-agreed SSID for bootstrapping, e.g., as proposed in [I-D.richardson-emu-eap-onboarding]. The same approach can be used by 6LoWPAN/mesh using a pre-agreed PAN ID. How to gain network connectivity is out of scope of this document.

6.2. Pledge in Responder Mode

In BRSKI-PRM, the pledge is triggered by ~~the~~ a Registrar-Agent to create the PVR and PER. It is also triggered for processing of the responses and the generation of status information once the Registrar-Agent has received the responses from the registrar later in the process.

To enable interaction as responder with ~~the~~ a Registrar-Agent, pledges in responder mode MUST act as servers and ~~MUST~~ SHOULD provide the endpoints defined in Table 1 within the BRSKI-defined /.well-known/brski/ URI path, except for the OPTIONAL endpoint "qps". The endpoints are defined with short names to also accommodate for resource-constrained devices.

Endpoint	Operation	Exchange and Artifacts
tpvr	Trigger Pledge Voucher-Request	Section 7.1
tper	Trigger Pledge Enroll-Request	Section 7.2
svr	Supply Voucher to Pledge	Section 7.6
scac	Supply CA Certificates to Pledge	Section 7.7
ser	Supply Enroll-Response to Pledge	Section 7.8

Commenté [MB22]: Do we need to have a control parameter here?

Commenté [MB23]: Check expired individual I-Ds. I suggest we remove all those.

Commenté [MB24]: No need to repeat this.

Commenté [MB25]: Please fix.

qps	Query Pledge Status	Section 7.11

Table 1: Well-Known Endpoints on a Pledge in Responder Mode

HTTP(S) uses the Host header field (or :authority in HTTP/2) to allow for name-based virtual hosting as explained in Section 7.2 of [RFC9110]. This header field is mandatory, and so a compliant HTTP(S) client is going to insert it, which may be just an IP address. The pledge MUST respond to all requests regardless of the Host header field provided by the client (i.e., ignore it). Note that there is no requirement for the pledge to operate its BRSKI-PRM service on port numbers 80 or ~~port~~ 443, so there is no reason for name-based virtual hosting.

Commenté [MB26]: I'm afraid this needs some scoping; as there are other legitimate conditions where the pledge doe snot have to reply.

For instance, when the Registrar-Agent reaches out to the "tpvr" endpoint on a pledge in responder mode with the full URI http://pledge.example.com/.well-known/brski/tpvr, it sets the Host header field to pledge.example.com and the absolute path /.well-known/brski/tpbr. In practice, however, the pledge is usually known by a .local hostname or only its IP address as returned by a discovery protocol, which will be included in the Host header field.

As BRSKI-PRM uses authenticated self-contained objects between the pledge and the domain registrar, the binding of the pledge identity to the voucher-requests is provided by the wrapping signature employing the pledge IDevID credential. Hence, pledges MUST have an Initial Device Identifier (IDevID) installed in them at the factory.

6.2.1. Pledge with Combined Functionality

Pledges MAY support both initiator and responder mode.

A pledge in initiator mode should listen for announcement messages as described in Section 4.1 of [RFC8995]. Upon discovery of a potential registrar, it initiates the bootstrapping to that registrar. At the same time (so as to avoid the Slowloris-like attack described in [RFC8995]), it SHOULD also respond to the triggers for responder mode described in this document.

Once a pledge with combined functionality has been bootstrapped, it MAY act as client for enrollment of further certificates needed, e.g., using the enrollment protocol of choice. If it still acts as server, the defined BRSKI-PRM endpoints to trigger a Pledge Enroll-Request (PER) or to provide an Enroll-Response can be used for further certificates.

6.3. Domain Registrar

~~In BRSKI-PRM, the~~ The domain registrar provides the endpoints already specified in [RFC8995] (derived from EST [RFC7030]) where suitable. In addition, it MUST provide the endpoints defined in Table 2 within the BRSKI-defined /.well-known/brski/ Well-Known URI path. These endpoints accommodate for the authenticated self-contained objects used by BRSKI-PRM to provide Pledge Enroll-Request (PER) artifacts and signature-wrapped CA certificates via the Registrar-Agent.

Endpoint	Operation	Exchange and Artifacts
requestenroll	Supply PER to Registrar	Section 7.4
wrappedcacerts	Obtain CA Certificates	Section 7.5

Table 2: Additional Well-Known Endpoints on a BRSKI-PRM Registrar

The registrar possesses its own EE certificate and corresponding private key for authenticating and signing. It MUST use the same certificate/credentials for authentication in the TLS session with a Registrar-Agent and for signing artifacts for that Registrar-Agent and its pledges (see Section 7.3.6). Overall, this may have operational implications when the registrar is part of a scalable framework as described in Section 1.3.1 of [I-D.richardson-anima-registrar-considerations].

Commenté [MB27]: May be grouped in one single OPS section

According to Section 5.3 of [RFC8995], ~~the~~ a domain registrar performs the pledge authorization for bootstrapping within its domain based on the Pledge Voucher-Request. For this, it MUST possess the IDevID trust anchor(s) (i.e., root or issuing CA certificate(s)) of the pledge vendor(s)/manufacturer(s). This behavior is retained in BRSKI-PRM.

In its role as EST server [RFC7030], the domain registrar MUST also possess the domain CA certificates as defined in Section 5.9 of [RFC8995].

Finally, the domain registrar MUST possess the Registrar-Agent EE certificate(s) to validate agent-signed data and to provide it to the MASA. The registrar MAY use the certificate verified during client authentication within the TLS sessions with the Registrar-Agent; in this case, the registrar MUST possess the domain trust anchor (i.e., domain CA certificate) for the Registrar-Agent EE certificate to verify the certificate chain. Alternatively, the Registrar-Agent EE certificate(s) MAY be provided via configuration or a repository.

6.3.1. Domain Registrar with Combined Functionality

A registrar with combined BRSKI and BRSKI-PRM functionality MAY detect if the bootstrapping is performed by the pledge directly (BRSKI case) or by a Registrar-Agent (BRSKI-PRM case) based on the utilized credentials for client authentication during the TLS session establishment and switch the operational mode from BRSKI to BRSKI-PRM.

This may be supported by a specific naming in the SAN (subject alternative name) component of the Registrar-Agent EE certificate, which allows the domain registrar to explicitly detect already in the TLS session establishment that the connecting client is a Registrar-

Agent.

The registrar MAY be restricted by configuration, if it accepts every Registrar-Agent, which can authenticate with a domain issued certificate or only explicitly authorized ones.

Note that using an EE certificate for TLS client authentication of the Registrar-Agent is a deviation from [RFC8995], in which the pledge IDevID certificate is used to perform TLS client authentication.

6.4. MASA

The Manufacturer Authorized Signing Authority (MASA) is a vendor service that generates and signs voucher artifacts for pledges by the same vendor. When these pledges support BRSKI-PRM, the MASA needs to implement the following functionality in addition to BRSKI [RFC8995].

A MASA for pledges in responder mode MUST support the voucher format defined in [I-D.ietf-anima-jws-voucher] to parse and process JWS-signed voucher-request artifacts and generate JWS-signed voucher artifacts.

Further, a MASA for pledges in responder mode MUST support the Agent Proximity Assertion (~~see~~ Section 5.4) through the validation steps defined in Section 7.3.1 based on the Pledge Voucher-Request (PVR) and Registrar Voucher-Request (RVR) artifact fields defined in Section 7.1.2 and Section 7.3.4, respectively.

7. Exchanges and Artifacts

The interaction of the pledge with the Registrar-Agent may be accomplished using different transports (i.e., protocols and/or network technologies). This specification utilizes HTTP(S) as default transport. Other specifications may define alternative transports such as CoAP, Bluetooth Low Energy (BLE), or Near Field Communication (NFC). These transports may differ from and are independent of the ones used between the Registrar-Agent and the registrar.

Transport independence is realized through authenticated self-contained objects that are not bound to a specific transport security and stay the same along the communication path from the pledge via the Registrar-Agent to the registrar. [I-D.ietf-anima-rfc8366bis] defines CMS-signed JSON structures as format for artifacts representing authenticated self-contained objects. This specification utilizes JWS-signed JSON structures as default format for BRSKI-PRM. Other specifications may define alternative formats for representing authenticated self-contained objects such as COSE-signed CBOR structures.

Figure 4 provides an overview of the exchanges detailed in the following subsections.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
```

```
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                  |                  |   Internet |
 |     discover     |                  |                  |            |
 |      pledge      |                  |                  |            |
 |    mDNS query    |                  |                  |            |
 |<-----------------|                  |                  |            |
 |----------------->|                  |                  |            |
 |                  |                  |                  |            |
 ~                  ~                  ~                  ~            ~
(1) Trigger Pledge Voucher-Request
 ~                  ~                  ~                  ~            ~
 |                  |                  |                  |            |
 |<----opt. TLS---->|                  |                  |            |
 |<------tPVR-------|                  |                  |            |
 |--------PVR------>|                  |                  |            |
 |                  |                  |                  |            |
 ~                  ~                  ~                  ~            ~
(2) Trigger Pledge Enroll-Request
 ~                  ~                  ~                  ~            ~
 |                  |                  |                  |            |
 |<----opt. TLS---->|                  |                  |            |
 |<------tPER-------|                  |                  |            |
 |--------PER------>|                  |                  |            |
 |                  |                  |                  |            |
 ~                  ~                  ~                  ~            ~
(3) Supply PVR to Registrar (including MASA interaction)
 ~                  ~                  ~                  ~            ~
 |                  |                  |                  |            |
 |                  |<-----mTLS------>|                  |            |
 |                  |                  |                  |            |
 |                  |         [Registrar-Agent            |            |
 |                  |    authenticated&authorized?]       |            |
 |                  |                  |                  |            |
 |                  |-------PVR------>|                  |            |
 |                  |                  |                  |            |
 |                  |          [accept device?]           |            |
 |                  |                  |                  |            |
 |                  |                  |<------------mTLS------------>|
 |                  |                  |--------------RVR------------>|
 |                  |                  |                  ~            |
 |                  |                  |            [extract DomainID]
 |                  |                  |            [update audit-log]
 |                  |                  |                  ~            |
 |                  |                  |<-----------Voucher-----------|
 |                  |<----Voucher''---|                  |            |
 |                  |                  |                  |            |
 ~                  ~                  ~                  ~            ~
(4) Supply PER to Registrar (including Key Infrastructure interaction)
 ~                  ~                  ~                  ~            ~
 |                  |                  |                  |            |
 |                  |<---((mTLS))---->|                  |            |
 |                  |-------PER------>|                  |            |
 |                  |                  |----[Request]--->|            |
 |                  |                  |<--[Certificate]-|            |
 |                  |<--Enroll-Resp---|                  |            |
 |                  |                  |                  |            |
 ~                  ~                  ~                  ~            ~
(5) Obtain CA Certificates
```

```
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |                  |<----(mTLS)------>|                  |             |
 |                  |<----caCerts------|                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(6) Supply Voucher to Pledge
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |<----opt. TLS---->|                  |                  |             |
 |<-----Voucher''---|                  |                  |             |
 |------vStatus---->|                  |                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(7) Supply CA Certificates to Pledge
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |<----opt. TLS---->|                  |                  |             |
 |<-----caCerts-----|                  |                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(8) Supply Enroll-Response to Pledge
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |<----opt. TLS---->|                  |                  |             |
 |<---Enroll-Resp---|                  |                  |             |
 |-----eStatus----->|                  |                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(9) Voucher Status Telemetry (including backend interaction)
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |                  |<----(mTLS)------>|                  |             |
 |                  |-----vStatus----->|                  |             |
 |                  |                  |<-----------(mTLS)----------->|
 |                  |                  |-----req device audit-log---->|
 |                  |                  |<------device audit-log-------|
 |                  |                  |                  |             |
 |                  |         [verify audit-log]          |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(10) Enroll Status Telemetry
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |                  |<----(mTLS)------>|                  |             |
 |                  |-----eStatus----->|                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
(11) Query Pledge Status
 ~                  ~                  ~                  ~             ~
 |                  |                  |                  |             |
 |<----opt. TLS---->|                  |                  |             |
 |<-----tStatus-----|                  |                  |             |
 |------pStatus---->|                  |                  |             |
 |                  |                  |                  |             |
 ~                  ~                  ~                  ~             ~
```

Figure 4: Overview pledge-responder-mode exchanges

The following subsections split the interactions shown in Figure 4 between the different components into:

1. Section 7.1 describes the acquisition exchange for the Pledge Voucher-Request initiated by the Registrar-Agent to the pledge.

2. Section 7.2 describes the acquisition exchange for the Pledge Enroll-Request initiated by the Registrar-Agent to the pledge.

3. Section 7.3 describes the issuing exchange for the Voucher initiated by the Registrar-Agent to the registrar, including the interaction of the registrar with the MASA using the RVR Section 7.3.4, as well as the artifact processing by these entities.

4. Section 7.4 describes the enroll exchange initiated by the Registrar-Agent to the registrar including the interaction of the registrar with the CA using the PER as well as the artifact processing by these entities.

5. Section 7.5 describes the retrieval exchange for the optional CA certificate provisioning to the pledge initiated by the Registrar-Agent to the CA.

6. Section 7.6 describes the Voucher exchange initiated by the Registrar-Agent to the pledge and the returned status information.

7. Section 7.7 describes the CA certificate exchange initiated by the Registrar-Agent to the pledge.

8. Section 7.8 describes the Enroll-Response exchange initiated by the Registrar-Agent to the pledge (containing a new pledge EE certificate) and the returned status information.

9. Section 7.9 describes the Voucher Status telemetry exchange initiated by the Registrar-Agent to the registrar, including the interaction of the registrar with the MASA.

10. Section 7.10 describes the Enroll Status telemetry exchange initiated by the Registrar-Agent to the registrar.

11. Section 7.11 describes the Pledge Status exchange about the general bootstrapping state initiated by the Registrar-Agent to the pledge.

7.1. Trigger Pledge Voucher-Request

The Registrar-Agent MUST begin the sequence of exchanges by sending the Pledge Voucher-Request Trigger (tPVR). This assumes that the Registrar-Agent has already discovered the pledge, for instance as described in Section 6.1.2 based on DNS-SD or similar.

~~Optionally,~~ TLS MAY be used to provide transport security, e.g., privacy and peer authentication, for the exchange between the Registrar-Agent and the pledge (see Appendix B).

Commenté [MB28]: Redundant with «MAY»

Figure 5 shows the acquisition of the Pledge Voucher-Request (PVR) and the following subsections describe the corresponding artifacts.

```
+---------+    +-------------+    +------------+    +---------+    +-------+
| Pledge |    | Registrar- |    |  Domain    |    |  Key    |    | MASA |
|         |    |   Agent     |    | Registrar |    | Infra. |    |       |
+---------+    +-------------+    +------------+    +---------+    +-------+
 |                  |                  |                 |   Internet |
 ~                  ~                  ~                 ~            ~
(1) Trigger Pledge Voucher-Request
 ~                  ~                  ~                 ~            ~
 |                  |                  |                 |            |
 |<----opt. TLS---->|                  |                 |            |
 |<------tPVR-------|                  |                 |            |
 |--------PVR------>|                  |                 |            |
 |                  |                  |                 |            |
 ~                  ~                  ~                 ~            ~
```

Figure 5: PVR acquisition exchange

The Registrar-Agent SHALL trigger the pledge to create a PVR via HTTP(S) POST to the pledge endpoint at /.well-known/brski/tpvr. The request body MUST contain the JSON-based Pledge Voucher-Request Trigger (tPVR) artifact as defined in Section 7.1.1. In the request header, the Content-Type field MUST be set to application/json and the Accept field SHOULD be set to application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

Upon receiving a valid tPVR, the pledge MUST reply with the PVR artifact as defined in Section 7.1.2 in the body of an HTTP 200 OK response. If the Accept header was not provided in the PVR, the pledge assumes that the accepted response format is application/voucher-jws+json and proceeds processing. In the response header, the Content-Type field MUST be set to application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

Note that the pledge provisionally accepts the registrar EE certificate contained in the tPVR until it receives the voucher (see Section 5.4).

If the pledge is unable to create the PVR, it SHOULD respond with an HTTP error status code to the Registrar-Agent. The following client error status codes SHOULD be used:

* 400 Bad Request: if the pledge detects an error in the format of the request, e.g., missing field, wrong data types, etc. or if the request is not valid JSON even though the Content-Type request header field was set to application/json.

* 406 Not Acceptable: if the Accept request header field indicates a type that is unknown or unsupported, e.g., a type other than application/voucher-jws+json.

* 415 Unsupported Media Type: if the Content-Type request header field indicates a type that is unknown or unsupported, e.g., a type other than application/json.

a mis en forme : Surlignage

Commenté [MB29]: Shouldn't be this be subject to policy? E.g., rate-limit requests?

Commenté [MB30]: Move to be next the SHOULD right above

Commenté [MB31]: Shouldn't that be TTLed as well?

Commenté [MB32]: ---DISCUSS

The use of normative language is IMO not compliant with the guidance in RFC9205

The pledge MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent.

While BRSKI-PRM does not specify which content may be provided in the response body, it is recommended to provided it as JSON encoded information as other BRSKI-PRM exchanges also utilize this encoding.

7.1.1. Request Artifact: Pledge Voucher-Request Trigger (tPVR)

The Pledge Voucher-Request Trigger (tPVR) artifact SHALL be an unsigned data object, providing the necessary parameters for generating the Pledge Voucher-Request (PVR) artifact such that the Agent Proximity Assertion can be verified by registrar and MASA: the registrar EE certificate and an agent-signed data object containing the product-serial-number and a timestamp. The artifact is unsigned because at the time of receiving the tPVR, the pledge could not verify any signature.

For the JSON-based format used by this specification, the tPVR artifact SHALL be a UTF-8 encoded JSON document [RFC8259] that conforms with the CDDL [RFC8610] data model defined in Figure 6:

```
pledgevoucherrequesttrigger = {
  "agent-provided-proximity-registrar-cert": bytes,
  "agent-signed-data": bytes
}
```

Figure 6: CDDL for Pledge Voucher-Request Trigger (pledgevoucherrequesttrigger)

The agent-provided-proximity-registrar-cert member SHALL contain the base64-encoded registrar EE certificate in X.509 v3 (DER) format. The agent-signed-data member SHALL contain the base64-encoded JWS Agent-Signed Data as defined in Section 7.1.1.1. Figure 7 summarizes the serialization the JSON tPVR artifact:

```
{
  "agent-provided-proximity-registrar-cert": "base64encodedvalue==",
  "agent-signed-data": BASE64(UTF8(JWS Agent-Signed Data))
}
```

Figure 7: tPVR Representation in JSON

7.1.1.1. JWS Agent-Signed Data

To enable alternative formats, the YANG module in [I-D.ietf-anima-rfc8366bis] defines the leaf "agent-signed-data" as binary. For the JWS-signed JSON format used by this specification, the agent-signed-data leaf SHALL be a UTF-8 encoded JWS structure in "General JWS JSON Serialization Syntax" as defined in Section 7.2.1 of [RFC7515] signing the JSON Agent-Signed Data defined in Section 7.1.1.1.1. Figure 8 summarizes this JWS structure for the agent-signed-data member of the tPVR artifact:

```
{
  "payload": BASE64URL(UTF8(JSON Agent-Signed Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
```

```
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 8: JWS Agent-Signed Data in General JWS JSON Serialization Syntax

The JSON Agent-Signed Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.1.1.1.1. JSON Agent-Signed Data

The JSON Agent-Signed Data SHALL be a JSON document [RFC8259] that MUST conform with the CDDL [RFC8610] data model defined in Figure 9.~~:~~

```
prmasd = {
  "created-on": tdate,
  "serial-number": text
}
```

Figure 9: CDDL for JSON Agent-Signed Data (prmasd)

The created-on member SHALL contain the current date and time at tPVR creation as standard date/time string as defined in Section 5.6 of [RFC3339].

The serial-number member SHALL contain the product-serial-number of the pledge with which the Registrar-Agent assumes to communicate as string. The format MUST correspond to the X520SerialNumber field of IDevID certificates.

Figure 10 ~~below~~ shows an example for the JSON Agent-Signed Data:

```
{
  "created-on": "2021-04-16T00:00:01.000Z",
  "serial-number": "vendor-pledge4711"
}
```

Figure 10: JSON Agent-Signed Data Example

7.1.1.1.2. JWS Protected Header

The JWS Protected Header of the agent-signed-data member MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515].

* kid: SHALL contain the base64-encoded OCTET STRING value of the SubjectKeyIdentifier of the Registrar-Agent EE certificate as described in Section 6.1.

Figure 11 ~~below~~ shows an example for this JWS Protected Header:

```
{
  "alg": "ES256",
  "kid": "base64encodedvalue=="
}
```

Figure 11: JWS Protected Header Example for

7.1.1.1.3. JWS Signature

The Registrar-Agent MUST sign the agent-signed-data member using its EE credentials. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

7.1.2. Response Artifact: Pledge Voucher-Request (PVR)

The Pledge Voucher-Request (PVR) artifact SHALL be an authenticated self-contained object signed by the pledge, containing an extended Voucher-Request artifact based on Section 5.2 of [RFC8995]. The BRSKI-PRM related enhancements of the "ietf-voucher-request" YANG module are defined in [I-D.ietf-anima-rfc8366bis].

For the JWS-signed JSON format used by this specification, the PVR artifact MUST be a JWS Voucher structure as defined in [I-D.ietf-anima-jws-voucher], which MUST contain the JSON PVR Data defined in Section 7.1.2.1 in the JWS Payload. Figure 12 summarizes the serialization of the JWS-signed JSON PVR artifact~~:~~.

```
{
  "payload": BASE64URL(UTF8(JSON PVR Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 12: PVR Representation in General JWS JSON Serialization Syntax

7.1.2.1. JSON PVR Data

The JSON PVR Data MUST contain the following fields of the "ietf-voucher-request" YANG module as defined in [I-D.ietf-anima-rfc8366bis]; note that this makes optional ~~leaves~~ leaf data nodes in the YANG definition mandatory for the PVR artifact:

Commenté [MB33]: I think this spec should be clustered with 8366bis.

* created-on: SHALL contain the current date and time at PVR creation as standard date/time string as defined in Section 5.6 of [RFC3339]; if the pledge does not have synchronized time, it SHALL use the created-on value from the JSON Agent-Signed Data received with the tPVR artifact and SHOULD advance that value based on its

local clock to reflect the PVR creation time.

* nonce: SHALL contain a cryptographically strong pseudo-random Number.

* serial-number: SHALL contain the product-serial-number in the X520SerialNumber field of the pledge IDevID certificate as string as defined in Section 2.3.1 of [RFC8995].

* assertion: SHALL contain the assertion type agent-proximity to indicate the pledge request (different from BRSKI [RFC8995]).

* agent-provided-proximity-registrar-cert: SHALL contain the base64-encoded registrar EE certificate provided in the tPVR by the Registrar-Agent; enables the registrar and MASA to verify the Agent Proximity Assertion.

* agent-signed-data: SHALL contain the same value as the agent-signed-data member in the tPVR provided by the Registrar-Agent; enables the registrar and MASA to verify the Agent Proximity Assertion; also enables the registrar to log which Registrar-Agent was in contact with the pledge.

Figure 13 ~~below~~ shows an example for the JSON PVR Data:


```
{
  "ietf-voucher-request:voucher": {
     "created-on": "2021-04-16T00:00:02.000Z",
     "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
     "serial-number": "vendor-pledge4711",
     "assertion": "agent-proximity",
     "agent-provided-proximity-registrar-cert": "base64encodedvalue==",
     "agent-signed-data": "base64encodedvalue=="
  }
}
```


Figure 13: JSON PVR Data Example

7.1.2.2. JWS Protected Header

The JWS Protected Header MUST follow the definitions of Section 3.2 of [I-D.ietf-anima-jws-voucher].

7.1.2.3. JWS Signature

The pledge MUST sign the PVR artifact using its IDevID credential following the definitions of Section 3.3 of [I-D.ietf-anima-jws-voucher].

7.2. Trigger Pledge Enroll-Request

Once the Registrar-Agent has received the PVR it can trigger the pledge to generate a Pledge Enroll-Request (PER).

~~Optionally,~~ TLS MAY be used to provide privacy for this exchange between the Registrar-Agent and the pledge (see Appendix B). Figure 14 shows the acquisition of the PER and the following subsections describe the corresponding artifacts.

Commenté [MB34]: Redundant with «MAY»

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(2) Trigger Pledge Enroll-Request
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |<----opt. TLS---->|                 |                 |            |
 |<------tPER-------|                 |                 |            |
 |--------PER------>|                 |                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 14: PER acquisition exchange

The Registrar-Agent SHALL trigger the pledge to create the PER via HTTP(S) POST to the pledge endpoint at /.well-known/brski/tper. The request body MUST contain the JSON-based Pledge Enroll-Request Trigger (tPER) artifact as defined in Section 7.2.1. In the request header, the Content-Type field MUST be set to application/json and the Accept field SHOULD be set to application/jose+json.

Upon receiving a valid tPER, the pledge MUST reply with the PER artifact as defined in Section 7.2.2 in the body of an HTTP 200 OK response. If the Accept header was not provided in the PER, the pledge assumes that the accepted response format is application/voucher-jws+json and proceeds processing. In the response header, the Content-Type field MUST be set to application/jose+json.

If the pledge is unable to create the PER, it SHOULD respond with an HTTP error status code to the Registrar-Agent. The following client error status codes MAY be used:

* 400 Bad Request: if the pledge detects an error in the format of the request.

* 406 Not Acceptable: if the Accept request header field indicates a type that is unknown or unsupported, e.g., a type other than application/jose+json.

* 415 Unsupported Media Type: if the Content-Type request header field indicates a type that is unknown or unsupported, e.g., a type other than application/json.

The pledge MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent. While BRSKI-PRM does not specify which content may be provided in the response body, it is recommended to provided it as JSON encoded information as other BRSKI-PRM exchanges also utilize this encoding.

7.2.1. Request Artifact: Pledge Enroll-Request Trigger (tPER)

The Pledge Enroll-Request Trigger (tPVR) artifact SHALL be an unsigned data object, providing enrollment parameters. This document specifies only the basic parameter for a generic, device-related

Commenté [MB35]: ---DISCUSS

Idem as a similar construct above.

Commenté [MB36]: Idem as above. Need to follow the guidance in RFC9205

LDevID certificate with no CSR attributes provided to the pledge. If specific attributes in the certificate are required, they have to be inserted by the issuing Key Infrastructure.

The Pledge Enroll-Request Trigger (tPER) artifact MAY be used to provide additional enrollment parameters such as CSR attributes. How to provide and use such additional data is out of scope for this specification.

For the JSON-based format used by this specification, the tPER artifact MUST be a UTF-8 encoded JSON document [RFC8259] that conforms with the CDDL [RFC8610] data model defined in Figure 15~~:~~.

```
pledgeenrollrequesttrigger = {
        "enroll-type": $enroll-type
}

$enroll-type /= "enroll-generic-cert"
```

Figure 15: CDDL for Pledge Enroll-Request Trigger (pledgeenrollrequesttrigger)

The enroll-type member allows for specifying which type of certificate is to be enrolled. As shown in Figure 15, BRSKI-PRM only defines the enumeration value enroll-generic-cert for the enrollment of the generic, device-related LDevID certificate. Other specifications using this artifact may define further enum values, e.g., to bootstrap application-related EE certificates with additional CSR attributes.

7.2.2. Response Artifact: Pledge Enroll-Request (PER)

The Pledge Enroll-Request (PER) artifact SHALL be an authenticated self-contained object signed by the pledge, containing a PKCS#10 Certificate Signing Request (CSR) [RFC2986]. The CSR already assures POP of the private key corresponding to the contained public key. In addition, based on the PER signature using the IDevID of the pledge, POI is provided.

For the JWS-signed JSON format used by this specification, the PER artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON CSR Data defined in Section 7.2.2.1 in the JWS Payload. Figure 16 summarizes the serialization of the JWS-signed JSON PER artifact:

```
{
  "payload": BASE64URL(UTF8(JSON CSR Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 16: PER Representation in General JWS JSON Serialization Syntax

The JSON CSR Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.2.2.1. JSON CSR Data

The JSON CSR Data SHALL be a JSON document [RFC8259] that MUST conform with the data model described by the csr-grouping of the ietf-ztp-types YANG module defined in Section 3.2 of [RFC9646] and MUST be encoded using the rules defined in [RFC7951]. Note that [RFC9646] also allows for inclusion of CSRs in different formats used by CMP and CMC. For PKCS#10 CSRs as used in BRSKI and BRSKI-PRM, the p10-csr case of the csr-grouping MUST be used.

Figure 17 below shows an example for the JSON CSR Data:


```
{
  "ietf-ztp-types": {
     "p10-csr": "base64encodedvalue=="
   }
}
```


Figure 17: JSON CSR Data Example

7.2.2.2. JWS Protected Header

The JWS Protected Header of the PER artifact MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515]

* x5c: SHALL contain the base64-encoded pledge EE certificate used to sign the PER artifact; it SHOULD also contain the certificate chain for this certificate; if the certificate chain is not included in the x5c Header Parameter, it MUST be available at the domain registrar for verification

* crit: SHALL indicate the extension Header Parameter created-on to ensure that it must be understood and validated by the receiver as defined in Section 4.1.11 of [RFC7515]

In addition, the JWS Protected Header of the PER artifact MUST contain the following extension Header Parameter:

* created-on: SHALL contain the current date and time at PER creation as standard date/time string as defined in Section 5.6 of [RFC3339]; if the pledge does not have synchronized time, it SHALL use the created-on value from the JSON Agent-Signed Data received with the tPVR artifact and SHOULD advance that value based on its local clock to reflect the PER creation time

The new protected Header Parameter created-on is introduced to reflect freshness of the PER. It allows the registrar to verify the timely correlation between the PER artifact and previous exchanges, i.e., created-on of PER >= created-on of PVR >= created-on of PVR trigger. The registrar MAY consider ignoring any but the newest PER artifact from the same pledge in case the registrar has at any point in time more than one pending PER from the pledge.

Figure 18 below shows an example for this JWS Protected Header:


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "crit": ["created-on"],
  "created-on": "2022-09-13T00:00:02.000Z"
}
```


Figure 18: JWS Protected Header Example within PER

7.2.2.3. JWS Signature

The pledge MUST sign the PER artifact using its IDevID credential. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

While BRSKI-PRM targets the initial enrollment, re-enrollment can be supported similarly. In this case, the pledge MAY use its current, potentially application-related EE credential instead of its IDevID credential to sign the PER artifact. The issuing CA can associate the re-enrollment request with the pledge based on the previously issued and still valid EE certificate. Note that a pledge that does not have synchronized time needs to advance the last known current date and time based on its local clock over a longer period, which also requires persisting the local clock advancements across reboots.

7.3. Supply PVR to Registrar (including MASA interaction)

Once the Registrar-Agent has acquired one or more PVR and PER object pairs, it starts the interaction with the domain registrar. Collecting multiple pairs allows bulk bootstrapping of several pledges using the same session with the registrar.

The Registrar-Agent MUST establish a TLS session to the registrar with mutual authentication. In contrast to BRSKI [RFC8995], the TLS client authentication uses the Registrar-Agent EE certificate instead of the pledge IDevID certificate. Consequently, the domain registrar can distinguish BRSKI (pledge-initiator-mode) from BRSKI-PRM (pledge-responder-mode).

The registrar SHOULD verify the TLS client authentication of the Registrar-Agent, in particular if the TLS session is used to obtain the Registrar-Agent EE certificate (see Section 6.3). Note that authentication and authorization of the pledge verified during the TLS session based on the signatures inside the PVR artifact.

As already stated in [RFC8995], the use of TLS 1.3 (or newer) is encouraged. TLS 1.2 or newer is REQUIRED on the Registrar-Agent side. TLS 1.3 (or newer) SHOULD be available on the registrar, but TLS 1.2 MAY be used. TLS 1.3 (or newer) SHOULD be available on the MASA, but TLS 1.2 MAY be used.

Commenté [MB37]: --DISCUSS

Should this updated to reflect draft-ietf-uta-require-tls13?

Figure 19 shows the voucher-request processing and the following subsections describe the corresponding artifacts.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(3) Supply PVR to Registrar (including backend interaction)
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |                  |<-----mTLS------>|                 |            |
 |                  |                 |                 |            |
 |                  |         [Registrar-Agent          |            |
 |                  |    authenticated&authorized?]     |            |
 |                  |                 |                 |            |
 |                  |-------PVR------>|                 |            |
 |                  |                 |                 |            |
 |                  |          [accept device?]         |            |
 |                  |                 |                 |            |
 |                  |                 |<------------mTLS------------>|
 |                  |                 |--------------RVR------------>|
 |                  |                 |                 ~            |
 |                  |                 |              [extract DomainID]
 |                  |                 |              [update audit-log]
 |                  |                 |                 ~            |
 |                  |                 |<-----------Voucher-----------|
 |                  |<----Voucher''---|                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 19: Voucher issuing exchange

As a first step of the interaction with the domain registrar, the Registrar-Agent SHALL supply the PVR artifact(s) to the registrar via HTTP-over-TLS POST to the registrar endpoint at /.well-known/brski/requestvoucher. Note that this is the same endpoint as for BRSKI described in Section 5.2 of [RFC8995]. The request body MUST contain one previously acquired PVR artifact as defined in Section 7.1.2. In the request header, the Content-Type field MUST be set to application/voucher-jws+json and the Accept field SHOULD be set to application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

Upon receiving a PVR artifact, the registrar accepts or declines the request to join the domain. For this, it MUST perform pledge authorization as defined in Section 5.3 of [RFC8995]. Due to the Registrar-Agent in the middle, the registrar MUST verify in addition that

* the agent-provided-proximity-registrar-cert field of the PVR contains a registrar EE certificate signed by the same domain owner as the registrar EE certificate used to sign the RVR; note

that this check allows for installations with multiple domain registrars and for registrar EE certificate renewal between exchanges with the Registrar-Agent (see Section 5.2); in many installations with a single registrar the contained certificate is identical to the signing certificate

* the agent-signed-data field of the PVR is signed with the private key corresponding to the Registrar-Agent EE certificate as known by the registrar (see Section 6.3); this is done via the SubjectKeyIdentifier of the certificate in the kid Header Parameter of the JWS Protected Header of the agent-signed-data field.

* the product-serial-number inside the agent-signed-data is equal to the serial-number field of the PVR as well as the X520SerialNumber field of the pledge IDevID certificate, which is contained in the JWS Protected Header of the PVR.

* the Registrar-Agent EE certificate is still valid; this is necessary to avoid that a rogue Registrar-Agent generates agent-signed-data objects to onboard arbitrary pledges at a later point in time, see also Section 12.3.

If the registrar is unable to process the request or validate the PVR, it MUST respond with an HTTP client error status code to the Registrar-Agent. The following client error status codes SHOULD be used:

* 400 Bad Request: if the registrar detects an error in the format of the request

* 403 Forbidden: if the registrar detected that one or more security related fields are not valid or if the pledge-provided information could not be used with automated allowance

* 406 Not Acceptable: if the Accept request header field indicates a type that is unknown or unsupported

* 415 Unsupported Media Type: if the Content-Type request header field indicates a type that is unknown or unsupported

Otherwise, the registrar converts the PVR artifact to a Registrar Voucher-Request (RVR) artifact (see Section 7.3.4) and starts the backend interaction with the MASA.

Optionally, the domain registrar MAY respond with an HTTP 202 Accepted response status code to the Registrar-Agent at this point following Section 5.6 of [RFC8995], while the rules defined for the pledge also apply to the Registrar-Agent; in this case, the registrar still continues with the MASA interaction to provide the Voucher artifact to the retry request.

The registrar MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent.

7.3.1. MASA Interaction

The domain registrar MUST establish a TLS session with mutual authentication to the MASA of the pledge according to Section 5.4 of

Commenté [MB38]: Idem as above.

[RFC8995]. It requests the voucher from the MASA according to Section 5.5 of [RFC8995] via HTTP-over-TLS POST to the MASA endpoint at /.well-known/brski/requestvoucher. The request body MUST contain the RVR artifact as defined in Section 7.3.4. In the request header, the Content-Type field and the Accept field MUST be set to the same media type as the incoming PVR artifact. For the default format used in this specification, this is application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

The assumption is that a pledge typically supports a single artifact format and creates the PVR in the supported format; to ensure that the pledge is able to process the voucher, the registrar requests this format via the HTTP Accept header field when requesting the voucher. Further, the RVR artifact and the PVR artifact inside should also use the same format to limit the number of required format encoders. Note that BRSKI-PRM allows for alternative formats such as CMS-signed JSON as used in BRSKI [RFC8995] or COSE-signed

CBOR for constrained environments, when defined by other specifications. Overall, a MASA responsible for BRSKI-PRM capable pledges MUST support the same formats as supported by those pledges.

Once the MASA receives the RVR artifact, it MUST perform the verification as described in Section 5.5 of [RFC8995]. Depending on policy, the MASA MAY choose the type of assertion to perform. For the Agent Proximity Assertion of BRSKI-PRM (see Section 5.4), the MASA MUST skip the verification described in Section 5.5.5 of [RFC8995] and instead MUST verify for the PVR contained in the prior-signed-voucher-request field of the RVR that

* the agent-provided-proximity-registrar-cert field contains an EE certificate that is signed by the same domain owner as the EE certificate/credentials used to sign the RVR; note that this check allows for installations with multiple domain registrars and for registrar EE certificate renewal while PVRs are collected by the Registrar-Agent

* the registrar EE certificate in the agent-provided-proximity-registrar-cert field and the Registrar-Agent EE certificate in the agent-sign-cert field of the RVR are signed by the same domain owner.

* the agent-signed-data field is signed with the credentials corresponding to the Registrar-Agent EE certificate in the agent-sign-cert field of the RVR; this is done via the SubjectKeyIdentifier of the certificate in the kid Header Parameter of the JWS Protected Header in the agent-signed-data field.

* the product-serial-number inside the agent-signed-data is equal to the serial-number field of PVR and the serial-number field of the RVR as well as the X520SerialNumber field of the pledge IDevID certificate, which is contained in the JWS Protected Header of the PVR.

If the agent-sign-cert field in the RVR is not set, the MASA MAY state a lower level assertion value instead of failing the verification, e.g., "logged" or "verified".

If the verification fails, the MASA SHOULD respond with an HTTP client error status code to the registrar. The client error status codes are kept the same as defined in Section 5.6 of [RFC8995]:

Commenté [MB39]: Idem as above

* 403 Forbidden: if the voucher-request is not signed correctly or is stale or if the pledge has another outstanding voucher that cannot be overridden

* 404 Not Found: if the request is for a device that is not known to the MASA

* 406 Not Acceptable: if a voucher of the desired type or that uses the desired algorithms (as indicated by the "Accept" header fields and algorithms used in the signature) cannot be issued as such because the MASA knows the pledge cannot process that type

* 415 Unsupported Media Type: if the request uses an artifact format or Accept header value that is not supported by the MASA

Otherwise, the MASA creates a Voucher artifact as defined in Section 7.3.5 and updates the audit-log as described in Section 5.5 of [RFC8995]. The Voucher is then supplied to the registrar within the body of an HTTP 200 OK response according to Section 5.6 of [RFC8995]. In the response header, the Content-Type field MUST be set to the media type of the incoming RVR artifact. For the default format used in this specification, this is application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

7.3.2. Supply Voucher to Registrar-Agent

After receiving the Voucher from the MASA, the registrar SHOULD evaluate it for transparency and logging purposes as outlined in Section 5.6 of [RFC8995]. It then countersigns the Voucher for delivery to the pledge via the Registrar-Agent.

The registrar MUST reply to the Registrar-Agent with the Registrar-Countersigned Voucher artifact (Voucher') as defined in Section 7.3.6 in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to the media type of the incoming PVR artifact. For the default format used in this specification, this is application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

If the domain registrar is unable to return the Voucher, it MUST respond with an HTTP server error status code to the Registrar-Agent. The following server error status codes SHOULD be used:

Commenté [MB40]: Idem as above

* 500 Internal Server Error: if both Registrar-Agent request and MASA response are valid, but the registrar still failed to return the Voucher, e.g., due to missing configuration or a program failure

* 502 Bad Gateway: if the registrar received an invalid response from the MASA

* 503 Service Unavailable: if a simple retry of the Registrar-Agent request might lead to a successful response; this error response

SHOULD include the Retry-After response header field with an appropriate value

* 504 Gateway Timeout: if the backend request to the MASA timed out

7.3.3. Request Artifact: Pledge Voucher-Request (PVR)

Identical to the PVR artifact received from the pledge as defined in Section 7.1.2. The Registrar-Agent MUST NOT modify PVRs.

7.3.4. Backend Request Artifact: Registrar Voucher-Request (RVR)

The Registrar Voucher-Request (RVR) artifact SHALL be an extended Voucher-Request artifact based on Section 5.5 of [RFC8995]. The BRSKI-PRM related enhancements of the ietf-voucher-request YANG module are defined in [I-D.ietf-anima-rfc8366bis].

For the JWS-signed JSON format used by this specification, the RVR artifact MUST be a JWS Voucher structure as defined in [I-D.ietf-anima-jws-voucher], which MUST contain the JSON RVR Data defined in Section 7.3.4.1 in the JWS Payload. Figure 20 summarizes the serialization of the JWS-signed JSON RVR artifact:

```
{
  "payload": BASE64URL(UTF8(JSON RVR Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 20: RVR Representation in General JWS JSON Serialization Syntax

7.3.4.1. JSON RVR Data

The JSON RVR Data MUST contain the following fields of the ietf-voucher-request YANG module as defined in [I-D.ietf-anima-rfc8366bis]; note that this makes optional leaves in the YANG definition mandatory for the RVR artifact:

* created-on: SHALL contain the current date and time at RVR creation as standard date/time string as defined in Section 5.6 of [RFC3339]

* nonce: SHALL contain a copy of the nonce field from the JSON PVR Data the registrar provides this information to assure successful verification of Registrar-Agent proximity based on the agent-signed-data

* serial-number: SHALL contain the product-serial-number of the pledge; note the required verification by the registrar defined in Section 7.3

* idevid-issuer: SHALL contain the issuer value from the pledge IDevID certificate obtained from the PVR JWS Protected Header x5c

field

* prior-signed-voucher-request: SHALL contain the PVR artifact as received from the Registrar-Agent, see Section 7.1

As BRSKI-PRM uses the Agent Proximity Assertion (see Section 5.4), the JSON RVR Data MUST also contain the following fields:

* assertion: SHALL contain the value agent-proximity to indicate successful verification of the Agent Proximity Assertion (see Section 5.4) by the registrar

* agent-sign-cert: SHALL be a JSON array that contains the base64-encoded Registrar-Agent EE certificate as possessed by the registrar (see Section 6.3) as the first item; subsequent items MUST contain the corresponding certificate chain for verification at the MASA; the field is used for verification of the agent-signed-data field of the contained PVR

Note that the ietf-voucher-request YANG module defines the leaf agent-sign-cert as binary; this specification refines it as a JSON array structure similar to the x5c Header Parameter defined in Section 4.1.6 of [RFC7515].

Figure 21 ~~below~~ shows an example for the JSON RVR Data:


```
{
  "ietf-voucher-request:voucher": {
     "created-on": "2022-01-04T02:37:39.235Z",
     "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
     "serial-number": "vendor-pledge4711",
     "idevid-issuer": "base64encodedvalue==",
     "prior-signed-voucher-request": "base64encodedvalue==",
     "assertion": "agent-proximity",
     "agent-sign-cert": [
       "base64encodedvalue==",
       "base64encodedvalue==",
       "..."
     ]
  }
}
```


Figure 21: JSON RVR Data Example

7.3.4.2. JWS Protected Header

The JWS Protected Header MUST follow the definitions of Section 3.2 of [I-D.ietf-anima-jws-voucher].

7.3.4.3. JWS Signature

The domain registrar MUST sign the RVR artifact using its EE credentials following the definitions of Section 3.3 of [I-D.ietf-anima-jws-voucher].

7.3.5. Backend Response Artifact: Voucher

Commenté [MB41]: May update the date

The Voucher artifact is defined in Section 5.6 of [RFC8995] (cf. "voucher response"). The only difference for BRSKI-PRM is that the assertion field MAY contain the value agent-proximity as defined in [I-D.ietf-anima-rfc8366bis], when the Agent-Proximity Assertion (see Section 5.4) is performed by the MASA.

For the JWS-signed JSON format used by this specification, the Voucher artifact MUST be a JWS Voucher structure as defined in [I-D.ietf-anima-jws-voucher]. It contains JSON Voucher Data in the JWS Payload, for which an example is given in Figure 22:


```
{
  "ietf-voucher:voucher": {
    "created-on": "2022-01-04T00:00:02.000Z",
    "nonce": "base64encodedvalue==",
    "assertion": "agent-proximity",
    "pinned-domain-cert": "base64encodedvalue==",
    "serial-number": "vendor-pledge4711"
  }
}
```


Commenté [MB42]: Refresh the date?

Figure 22: JSON RVR Data Example

7.3.6. Response Artifact: Registrar-Countersigned Voucher

The Registrar-Countersigned Voucher (Voucher') artifact SHALL be an extended Voucher artifact based on Section 5.6 of [RFC8995] using the format defined in Section 7.3.5.

For BRSKI-PRM, the domain registrar MUST add an JWS Protected Header and JWS Signature to the MASA-provided Voucher. Figure 23 summarizes the serialization of the JWS-signed JSON Voucher' artifact.~~:~~

```
{
  "payload": BASE64URL(JSON Voucher Data),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header (MASA))),
      "signature": BASE64URL(JWS Signature (MASA))
    },
    {
      "protected": BASE64URL(UTF8(JWS Protected Header (Registrar))),
      "signature": BASE64URL(JWS Signature (Registrar))
    }
  ]
}
```

Figure 23: Voucher' Representation in General JWS JSON Serialization Syntax

In BRSKI [RFC8995], the registrar proves possession of its credential through the server authentication within the TLS session with the pledge. While the pledge cannot verify the registrar certificate at the time of TLS session establishment, it can verify the TLS server certificate through the certificate in the pinned-domain-cert field upon receiving the Voucher artifact (see Section 5.6.2 of [RFC8995]).

In BRSKI-PRM with the Registrar-Agent mediating all communication, this second signature provides verification and POP of the private key for the registrar EE certificate provided in the initial tPVR artifact from the Registrar-Agent (see Section 7.1.1).

Depending on the security policy of the operator, this signature can also be interpreted as explicit authorization of the registrar to install the contained trust anchor (i.e., pinned domain certificate).

7.3.6.1. JSON Voucher Data

As provided by the MASA inside the JWS Payload. The domain registrar MUST NOT modify the JWS Payload.

7.3.6.2. JWS Protected Header (Registrar)

The registrar-added JWS Protected Header (Registrar) MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515]

* x5c: SHALL contain the base64-encoded registrar EE certificate used to sign the voucher as well as the certificate chain up to (but not including) the pinned domain certificate (the initial domain trust anchor); the pinned domain certificate is already contained in the JSON Voucher Data

Note that for many installations with a single registrar credential, the registrar EE certificate is pinned.

7.3.6.3. JWS Signature (Registrar)

The signature is created by signing the registrar-added JWS Protected Header (Registrar) and the original JWS Payload produced by the MASA as described in Section 5.1 of [RFC7515]. The registrar MUST use its EE credentials to sign.

Note that the credentials need to be the same as used for server authentication in the TLS session with the Registrar-Agent receiving this artifact (see Section 6.3).

7.4. Supply PER to Registrar (including Key Infrastructure interaction; requestenroll)

After receiving the Voucher artifact, the Registrar-Agent sends the PER to the domain registrar within the same TLS session.

In case the TLS session to the registrar is already closed, the Registrar-Agent establishes a new session as described in Section 7.3. The registrar is able to correlate the PVR and PER artifacts based on the signatures and the contained product-serial-number. Note that this also addresses situations in which a nonceless voucher is used and may be pre-provisioned to the pledge.

Figure 24 depicts exchanges for the PER-request handling and the following subsections describe the corresponding artifacts. Note that "Request" and "Certificate" do not denote BRSKI-PRM defined

artifacts, but are data objects depending on the certificate
management protocol used by the domain Key Infrastructure.

```
+--------+     +------------+     +-----------+     +--------+     +------+
| Pledge |     | Registrar- |     |  Domain   |     |  Key   |     | MASA |
|        |     |   Agent    |     | Registrar |     | Infra. |     |      |
+--------+     +------------+     +-----------+     +--------+     +------+
 |                  |                  |                 |   Internet |
 ~                  ~                  ~                 ~            ~
(4) Supply PER to Registrar (including Key Infrastructure interaction)
 ~                  ~                  ~                 ~            ~
 |                  |                  |                 |            |
 |                  |<----(mTLS)------>|                 |            |
 |                  |-------PER------->|                 |            |
 |                  |                  |----[Request]--->|            |
 |                  |                  |<--[Certificate]-|            |
 |                  |<--Enroll-Resp----|                 |            |
 |                  |                  |                 |            |
 ~                  ~                  ~                 ~            ~
```

Figure 24: Enroll exchange

As a second step of the interaction with the domain registrar, the Registrar-Agent SHALL supply the PER artifact(s) to the registrar via HTTP-over-TLS POST to the registrar endpoint at /.well-known/brski/requestenroll. The request body MUST contain one previously acquired PER artifact as defined in Section 7.2.2. In the request header, the Content-Type field MUST be set to application/jose+json and the Accept field SHOULD be set to application/jose+json.

Note that this is different from the EST [RFC7030] endpoint used in BRSKI, as the PER artifact is signature-wrapped. Hence, upon receiving a PER artifact, the registrar MUST verify that

* the PER was signed with the private key corresponding to the pledge EE certificate, which is contained in the JWS Protected Header of the PER.

* the pledge identified by its EE certificate is accepted to join the domain after successful validation of the corresponding PVR.

If the registrar is unable to process the request or validate the PER, it MUST respond with an HTTP client error status code to the Registrar-Agent. The following client error status codes SHOULD be used:

Commenté [MB43]: Idem as above for a similar comment

* 400 Bad Request: if the registrar detects an error in the format of the request

* 401 Unauthorized: if the signature of the PER cannot be verified

* 404 Not Found: if the PER is for a device that is not known to the registrar

* 406 Not Acceptable: if the Accept request header field indicates a type that is unknown or unsupported, e.g., a type other than application/jose+json

* 415 Unsupported Media Type: if the PER uses an artifact format that is not supported by the registrar, e.g., a type other than application/jose+json

Otherwise, the registrar extracts the PKCS#10 Certificate Signing Request (CSR) inside the PER (see Section 7.2.2) and uses the CSR to request a new pledge EE certificate from the domain Key Infrastructure. The exact interaction and exchanged data objects depends on the certificate management protocol used by the Key Infrastructure, and is out of scope for this document.

A successful interaction with the Key Infrastructure will result in a pledge EE certificate signed by the domain owner (e.g., LDevID certificate). The registrar MUST reply to the Registrar-Agent with the Enroll-Response (Enroll-Resp) as defined in Section 7.4.2 in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to application/pkcs7-mime.

If the domain registrar is unable to return the Enroll-Resp, it MUST respond with an HTTP server error status code to the Registrar-Agent. The following server error status codes SHOULD be used:

Commenté [MB44]: Idem as a similar comment above.

* 500 Internal Server Error: if the Key Infrastructure response is valid, but the registrar still failed to return the Enroll-Resp, e.g., due to missing configuration or a program failure

* 502 Bad Gateway: if the registrar received an invalid response from the Key Infrastructure

* 503 Service Unavailable: if a simple retry of the Registrar-Agent request might lead to a successful response; this error response SHOULD include the Retry-After response header field with an appropriate value

* 504 Gateway Timeout: if the backend request to the Key Infrastructure timed out

Note that while BRSKI-PRM targets the initial enrollment, re-enrollment may be supported similarly with the exception that the current, potentially application-related pledge EE certificate is used instead of the IDevID certificate to sign the PER artifact (see also Section 7.2). Hence, there is no verification whether the pledge is accepted to join the domain, as the still valid EE certificate signed by the domain owner identifies the pledge as already accepted component of the domain.

7.4.1. Request Artifact: Pledge Enroll-Request (PER)

Identical to the PER artifact defined in Section 7.2.2. The Registrar-Agent MUST NOT modify PERs received from pledges.

7.4.2. Response Artifact: Registrar Enroll-Response (Enroll-Resp)

The Enroll-Response (Enroll-Resp) artifact SHALL be an authenticated self-contained object signed by the domain owner, containing a pledge EE certificate.

For this specification, the Enroll-Resp artifact MUST be a certs-only

CMC Simple PKI Response (PKCS#7) as defined in Section 4.1 of [RFC5272] (following EST [RFC7030]). Note that it only contains the pledge EE certificate, but not the certificate chain. The chain is provided with the CA certificates.

7.5. Obtain CA Certificates (wrappedcacerts)

The pinned domain certificate in the voucher is only the initial trust anchor for only the domain registrar. To fully trust the domain and ~~also~~ to verify its own EE certificate, the pledge also needs the corresponding domain CA certificate(s). A bag of CA certificates signed by the registrar will allow the pledge to verify the authorization to install the received CA certificate(s) through the pinned domain certificate in the voucher.

Note that this is a deviation from EST [RFC7030] used in BRSKI [RFC8995].

The Registrar-Agent obtains this artifact within the same TLS session. In case the TLS session to the registrar is already closed, the Registrar-Agent establishes a new session as described in Section 7.3. The CA certificates do not need to be correlated to a specific voucher or Enroll-Response; they only need to be fresh.

Figure 25 shows the acquisition of the CA certificate(s) and the following subsections describe the corresponding artifact.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                |                 |                 |   Internet |
 ~                ~                 ~                 ~            ~
(5) Obtain CA Certificates
 ~                ~                 ~                 ~            ~
 |                |                 |                 |            |
 |                |<----(mTLS)----->|                 |            |
 |                |<----caCerts-----|                 |            |
 |                |                 |                 |            |
 ~                ~                 ~                 ~            ~
```

Figure 25: CA certificates retrieval exchange

As a third step of the interaction with the domain registrar, the Registrar-Agent SHALL obtain the CA-Certificates artifact from the registrar via HTTP-over-TLS GET to the registrar endpoint at /.well-known/brski/wrappedcacerts. In the request header, the Accept field SHOULD be set to application/jose+json.

Upon receiving a GET request at /.well-known/brski/wrappedcacerts, the domain registrar MUST reply with the CA-Certificates artifact as defined in Section 7.5.2 in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to application/jose+json.

7.5.1. Request (no artifact)

In this exchange, the request is a result of the HTTP(S) default transport for this specification. There is no artifact provided to the registrar.

7.5.2. Response Artifact: CA-Certificates (caCerts)

The CA-Certificates (caCerts) artifact SHALL be an authenticated self-contained object signed by the registrar, containing the domain trust anchors and the certificate chain for the pledge domain EE certificate, i.e., the root CA certificate(s) and possibly intermediate certificate(s) as described in Section 4.1.3 of [RFC7030].

For the JWS-signed JSON format used by this specification, the caCerts artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON CA Data defined in Section 7.5.2.1 in the JWS Payload.

Figure 26 summarizes the serialization of the JWS-signed JSON caCerts artifact:

```
{
  "payload": BASE64URL(UTF8(JSON CA Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 26: Voucher' Representation in General JWS JSON Serialization Syntax

The JSON CA Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.5.2.1. JSON CA Data

The JSON CA Data SHALL be a JSON document [RFC8259] that MUST conform with the CDDL [RFC8610] data model defined in Figure 27.~~:~~

```
cacerts = {
        "x5bag": bytes / [2* bytes]
}
```

Figure 27: CDDL for JSON CA Data (cacerts)

The x5bag member MUST follow the definition of the x5bag COSE Header Parameter in Section 2 of [RFC9360]. It is either a single X.509 v3 certificate or an array of at least two X.509 v3 certificates in DER format. For JSON syntax, the octet-based certificates MUST be base64-encoded. It SHALL contain one or more domain CA (root or issuing) certificates.

Note that as per [RFC8995], the domain registrar acts as EST server, and hence is expected to possess the CA certificates applicable for the domain and can thus deliver them to the pledge (see Section 6.3).

Figure 28 below shows an example for the JSON CA Data:


```
{
  "x5bag": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 28: JSON CA Data Example

7.5.2.2. JWS Protected Header

The JWS Protected Header of the caCerts artifact MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515]

* x5c: SHALL contain the base64-encoded registrar EE certificate used to sign the caCerts artifact as well as the certificate chain up to (but not including) the pinned domain certificate

Figure 29 ~~below~~ shows an example for this JWS Protected Header:


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 29: JWS Protected Header Example within PER

7.5.2.3. JWS Signature

The registrar MUST sign the caCerts artifact using its EE credentials. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

7.6. Supply Voucher to Pledge (svr)

Once the Registrar-Agent has acquired the following three bootstrapping artifacts, it can supply them to the pledge starting with the Voucher':

* Voucher': voucher countersigned by the registrar (from MASA via Registrar)

* Enroll-Resp: pledge EE certificate signed by the domain owner (from Key Infrastructure via registrar)

* caCerts: domain trust anchors (from Key Infrastructure via Registrar)

Reconnecting to the pledge might require to re-discover the pledge as described in Section 6.1.2. The Registrar-Agent MAY store information from the first connection with the pledge to optimize.

Optionally, TLS MAY be used to provide privacy for this exchange between the Registrar-Agent and the pledge (see Appendix B).

Figure 30 shows the provisioning of the voucher to the pledge and the following subsections describe the corresponding artifacts.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(6) Supply Voucher to Pledge
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |<----opt. TLS---->|                 |                 |            |
 |<-----Voucher''---|                 |                 |            |
 |------vStatus---->|                 |                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 30: Voucher exchange

The Registrar-Agent SHALL supply the voucher to the pledge via HTTP(S) POST to the pledge endpoint at /.well-known/brski/svr. The request body MUST contain the Registrar-Countersigned Voucher (Voucher') artifact previously acquired from the domain registrar as defined in Section 7.3.6. In the request header, the Content-Type field MUST be set to application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher] and the Accept field SHOULD be set to application/jose+json.

Upon receiving the voucher, the pledge SHALL perform the signature verification in the following order:

1. Verify the MASA signature as described in Section 5.6.1 of [RFC8995] against the pre-installed manufacturer trust anchor (e.g., IDevID).

2. Provisionally install the initial domain trust anchor contained in the pinned-domain-cert field of the voucher.

3. Validate the registrar EE certificate received in the agent-provided-proximity-registrar-cert field of the previously received tPVR artifact using the pinned domain certificate; this terminates the "provisional state" for the object security within the authenticated self-contained objects that in BRSKI-PRM replace the direct TLS connection to the registrar in BRSKI [RFC8995] (see Section 5.4).

4. Verify registrar signature of the Voucher' artifact similar as described in Section 5.6.1 of [RFC8995], but using the pinned domain certificate instead of the MASA certificate for the verification.

If all steps above complete successfully, the pledge SHALL terminate the "provisional state" for the initial domain trust anchor (i.e., the pinned domain certificate).

A nonceless voucher MAY be accepted as in [RFC8995] if allowed by the pledge implementation of the manufacturer.

Commenté [MB45]: Should this be controlled/tuned?

After voucher validation and verification, the pledge needs to reply with a status telemetry message as defined in Section 5.7 of [RFC8995]. The pledge MUST generate the Voucher Status (vStatus) artifact as defined in Section 7.6.2 and MUST provide it to the Registrar-Agent in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to application/jose+json.

If the pledge is unable to validate or verify the voucher, it MUST report the reason in the corresponding field of the Voucher Status.

If the pledge did not provide voucher status telemetry information after processing the voucher, the Registrar-Agent MAY query the pledge status explicitly as described in Section 7.11. It MAY resend the voucher depending on the Pledge status following the same procedure.

7.6.1. Request Artifact: Registrar-Countersigned Voucher

Identical to the Registrar-Countersigned Voucher (Voucher') artifact received from the registrar as defined in Section 7.3.6. The Registrar-Agent MUST NOT modify countersigned vouchers.

7.6.2. Response Artifact: Voucher Status (vStatus)

The Voucher Status (vStatus) artifact SHALL be an authenticated self-contained object signed by the pledge, containing status telemetry as defined in Section 5.7 of [RFC8995].

For the JWS-signed JSON format used by this specification, the vStatus artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON Voucher Status Data defined in Section 7.6.2.1 in the JWS Payload. Figure 31 summarizes the serialization of the JWS-signed JSON vStatus artifact:

```
{
  "payload": BASE64URL(UTF8(JSON Voucher Status Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 31: vStatus Representation in General JWS JSON Serialization Syntax

The JSON Status Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.6.2.1. JSON Voucher Status Data

The JSON Status Data SHALL be a JSON document [RFC8259] that MUST conform with the voucherstatus-post CDDL [RFC8610] data model defined in Section 5.7 of [RFC8995]:

* version: contains a version number for the format and semantics of the other fields; this specification assumes version 1 just like BRSKI [RFC8995].

* status: contains the boolean value “true” in case of success and “false” in case of failure.

* reason: contains a human-readable message; SHOULD NOT provide information beneficial to an attacker.

* reason-context: contains a JSON object that provides additional information specific to a failure; in contrast to Section 5.7 of [RFC8995], MUST be provided; SHOULD NOT provide information beneficial to an attacker

BRSKI-PRM implementations utilize the reason-context field to provide a distinguishable token, which enables the registrar to detect status artifacts provided to the wrong endpoint. For vStatus artifacts, the JSON object in the reason-context field MUST contain the member pvs-details.

Figure 32 ~~below~~ shows an example for the JSON Voucher Status Data in case of success and Figure 33 in case of failure:


```
{
  "version": 1,
  "status": true,
  "reason": "Voucher successfully processed.",
  "reason-context": {
    "pvs-details": "Current date 5/23/2024"
  }
}
```


Figure 32: JSON Voucher Status Data Success Example


```
{
  "version": 1,
  "status": false,
  "reason": "Failed to authenticate MASA certificate.",
  "reason-context": {
```

```
      "pvs-details": "Current date 1/1/1970 < valid from 1/1/2023"
    }
  }
```


Figure 33: JSON Voucher Status Data Failure Example

7.6.2.2. JWS Protected Header

The JWS Protected Header of the vStatus artifact MUST contain the following ~~standard~~ Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515]

* x5c: SHALL contain the base64-encoded pledge IDevID certificate used to sign the vStatus artifact; it SHOULD also contain the certificate chain for this certificate; if the certificate chain is not included in the x5c Header Parameter, it MUST be available at the domain registrar for verification

Figure 34 ~~below~~ shows an example for this JWS Protected Header.~~:~~


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 34: JWS Protected Header Example within vStatus

7.6.2.3. JWS Signature

The pledge MUST sign the vStatus artifact using its IDevID credential. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

7.7. Supply CA Certificates to Pledge (scac)

Before supplying the pledge EE certificate, the Registrar-Agent supplies the domain CA certificates to the pledge, so the pledge can verify its EE certificate in the next exchange. As the CA certificate provisioning is crucial from a security perspective, this exchange SHOULD only be done, if supplying the voucher in the previous exchange (Section 7.6) has been successfully processed by the pledge as reflected in the vStatus artifact.

~~Optionally,~~ TLS MAY be used to provide privacy for this exchange between the Registrar-Agent and the pledge (see Appendix B).

Commenté [MB46]: Redundant with «May»

Figure 35 shows the provisioning of the CA certificates to the pledge and the following subsections describe the corresponding artifacts.

```
+--------+     +------------+     +-----------+     +--------+     +------+
| Pledge |     | Registrar- |     |  Domain   |     |  Key   |     | MASA |
|        |     |   Agent    |     | Registrar |     | Infra. |     |      |
+--------+     +------------+     +-----------+     +--------+     +------+
```

```
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(7) Supply CA Certificates to Pledge
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |<----opt. TLS---->|                 |                 |            |
 |<-----caCerts-----|                 |                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 35: Certificate provisioning exchange

The Registrar-Agent SHALL provide the bag of CA certificates requested from and signed by the registrar to the pledge by HTTP(S) POST to the pledge endpoint at /.well-known/brski/scac. The request body MUST contain the caCerts artifact as defined in Section 7.5.2. In the request header, the Content-Type field MUST be set to application/jose+json.

Upon receiving valid caCerts artifact, the pledge MUST first verify the signature of the registrar using the initial trust anchor (pinned domain certificate). In the case of success, the pledge MUST install the contained CA certificates as trust anchors as described in Section 4.1.3 of [RFC7030]. This includes the verification of all intermediate CA certificates (i.e., not self-signed CA certificates).

If the pledge is unable to process the caCerts, it SHOULD respond with an HTTP error status code to the Registrar-Agent. The following client error status codes SHOULD be used:

Commenté [MB47]: Idem as for similar construct above.

* 400 Bad Request: if the pledge detects an error in the format of the request

* 401 Unauthorized: if the signature of the registrar cannot be verified against the installed initial trust anchor (pinned domain certificate)

* 403 Forbidden: if one of the intermediate CA certificates cannot be verified against the available trust anchors (e.g., self-signed CA certificates)

* 415 Unsupported Media Type: if the Content-Type request header field indicates a type that is unknown or unsupported, e.g., a type other than application/jose+json

Otherwise, if processing completes successfully, the pledge SHOULD reply with HTTP 200 OK without a response body. The pledge MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent.

7.7.1. Request Artifact: CA-Certificates (caCerts)

Identical to the CA-Certificates (caCerts) artifact received from the registrar as defined in Section 7.5.2. The Registrar-Agent MUST NOT modify CA-Certificates artifacts.

7.7.2. Response (no artifact)

In this exchange, the response is a result of the HTTP(S) default transport for this specification. There is no artifact provided to the Registrar-Agent.

7.8. Supply Enroll-Response to Pledge (ser)

After supplying the CA certificates, the Registrar-Agent supplies the pledge EE certificate to the pledge.

Optionally, TLS MAY be used to provide privacy for this exchange between the Registrar-Agent and the pledge (see Appendix B).

Figure 36 shows the provisioning of the domain-owner signed EE certificate to the pledge and the following subsections describe the corresponding artifacts.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(8) Supply Enroll-Response to Pledge
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |<----opt. TLS---->|                 |                 |            |
 |<---Enroll-Resp---|                 |                 |            |
 |-----eStatus----->|                 |                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 36: Enroll-Response exchange

The Registrar-Agent SHALL send the domain-owner signed EE certificate to the pledge by HTTP(S) POST to the pledge endpoint at /.well-known/brski/ser. The request body MUST contain the Enroll-Response (Enroll-Resp) artifact previously acquired from the domain registrar as defined in Section 7.4.2. In the request header, the Content-Type field MUST be set to application/pkcs7-mime and the Accept field SHOULD be set to application/jose+json.

Upon reception, the pledge SHALL verify the received EE certificate using the installed trust anchors. After Enroll-Resp validation and verification, the pledge needs to reply with a status telemetry message as defined in Section 5.9.4 of [RFC8995]. The pledge MUST generate the Enroll Status (eStatus) artifact as defined in Section 7.8.2 and MUST provide it to the Registrar-Agent in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to application/jose+json.

If the pledge is unable to validate or verify the Enroll-Response, it MUST report the reason in the corresponding field of the Enroll Status.

7.8.1. Request Artifact: Enroll-Response (Enroll-Resp)

Identical to the Enroll-Response (Enroll-Resp) artifact received from the registrar as defined in Section 7.4.2. The Registrar-Agent MUST NOT modify Enroll-Response artifacts.

7.8.2. Response Artifact: Enroll Status (eStatus)

The Enroll Status (eStatus) artifact SHALL be an authenticated self-contained object signed by the pledge, containing status telemetry as defined in Section 5.9.4 of [RFC8995].

For the JWS-signed JSON format used by this specification, the eStatus artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON Enroll Status Data defined in Section 7.8.2.1 in the JWS Payload. Figure 37 summarizes the serialization of the JWS-signed JSON eStatus artifact:

```
{
  "payload": BASE64URL(UTF8(JSON Enroll Status Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 37: eStatus Representation in General JWS JSON Serialization Syntax

The JSON Enroll Status Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.8.2.1. JSON Enroll Status Data

The JSON Status Data SHALL be a JSON document [RFC8259] that MUST conform with the enrollstatus-post CDDL [RFC8610] data model defined in Section 5.9.4 of [RFC8995]. The members are the same as for the JSON Voucher Status Data and follow the same definitions as given in Section 7.6.2.1 (incl. making reason-context mandatory).

BRSKI-PRM implementations again utilize the reason-context field to provide a distinguishable token. For eStatus artifacts, the JSON object in the reason-context field MUST contain the member pes-details.

Figure 38 below shows an example for the JSON Enroll Status Data in case of success and Figure 39 in case of failure:

```
{
  "version": 1,
```

```
  "status": true,
  "reason": "Enroll-Response successfully processed.",
  "reason-context": {
    "pes-details": "Success"
  }
}
```


Figure 38: JSON Enroll Status Data Success Example


```
{
  "version": 1,
  "status": false,
  "reason": "Enroll-Response could not be verified.",
  "reason-context": {
    "pes-details": "No matching trust anchor"
  }
}
```


Figure 39: JSON Enroll Status Data Failure Example

7.8.2.2. JWS Protected Header

The JWS Protected Header of the eStatus artifact MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515]

* x5c: SHALL contain the base64-encoded pledge EE certificate used to sign the eStatus artifact; it SHOULD also contain the certificate chain for this certificate; if the certificate chain is not included in the x5c Header Parameter, it MUST be available at the domain registrar for verification

Figure 40 below shows an example for this JWS Protected Header:


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 40: JWS Protected Header Example within eStatus

7.8.2.3. JWS Signature

If the pledge verified the received EE certificate successfully, it MUST sign the eStatus artifact using its new EE credentials. In failure case, the pledge MUST sign it using its IDevID credentials. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

7.9. Voucher Status Telemetry (including MASA interaction)

Once the Registrar-Agent has collected both status artifacts from one or more pledges, it SHALL provide the status information to the domain registrar for further processing, beginning with the voucher status telemetry.

In case the TLS session to the registrar is closed, the Registrar-Agent establishes a new session as described in Section 7.3.

Figure 41 shows the provisioning of the voucher status information from the pledge(s) to the registrar and the following subsections describe the corresponding artifact and MASA interaction.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(9) Voucher Status Telemetry (including backend interaction)
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |                  |<---((mTLS))---->|                 |            |
 |                  |-----vStatus---->|                 |            |
 |                  |                 |<----------((mTLS))---------->|
 |                  |                 |-----req device audit-log---->|
 |                  |                 |<------device audit-log-------|
 |                  |                 |                 |            |
 |                  |        [verify audit-log]         |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 41: Voucher Status telemetry exchange

First, the Registrar-Agent SHALL supply the voucher status telemetry to the registrar via HTTP-over-TLS POST to the registrar endpoint at /.well-known/brski/voucher_status. The request body MUST contain one previously acquired vStatus artifact as defined in Section 7.6.2. In the request header, the Content-Type field MUST be set to application/jose+json.

Upon receiving a vStatus artifact, the registrar MUST process it as described in Section 5.7 of [RFC8995]. Due to the Registrar-Agent in the middle, the registrar MUST in addition verify the signature of the vStatus and that it belongs to an accepted device of the domain based on the serial-number field of the IDevID certificate contained in the JWS Protected Header of the vStatus.

According to Section 5.7 of [RFC8995], the registrar SHOULD respond with an HTTP 200 OK without a response body in the success case or fail with an HTTP error status code. The registrar MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent.

The registrar SHOULD proceed with the audit-log request to the MASA as in BRSKI described in Section 5.8 of [RFC8995].

7.9.1. Request Artifact: Voucher Status (vStatus)

Identical to the Voucher Status (vStatus) artifact received from the pledge as defined in Section 7.6.2. The Registrar-Agent MUST NOT modify vStatus artifacts.

7.9.2. Response (no artifact)

In this exchange, the response is a result of the HTTP(S) default transport for this specification. There is no artifact provided to the Registrar-Agent.

7.10. Enroll Status Telemetry

The Registrar-Agent SHALL complete the sequence of exchanges for bootstrapping with providing the enroll status telemetry to the domain registrar. This status indicates whether the pledge could process the Enroll-Response (pledge EE certificate signed by the domain owner) and holds the corresponding private key.

In case the TLS session to the registrar is already closed, the Registrar-Agent establishes a new session as described in Section 7.3.

Figure 42 shows the provisioning of the enroll status information from the pledge(s) to the registrar and the following subsections describe the corresponding artifact.

```
+--------+    +------------+    +-----------+    +--------+    +------+
| Pledge |    | Registrar- |    |  Domain   |    |  Key   |    | MASA |
|        |    |   Agent    |    | Registrar |    | Infra. |    |      |
+--------+    +------------+    +-----------+    +--------+    +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(10) Enroll Status Telemetry
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |                  |<---((mTLS))---->|                 |            |
 |                  |-----eStatus---->|                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Figure 42: Enroll Status telemetry exchange

The Registrar-Agent SHALL supply the enroll status telemetry to the registrar via HTTP-over-TLS POST to the registrar endpoint at /.well-known/brski/enrollstatus. The request body MUST contain one previously acquired eStatus artifact as defined in Section 7.8.2. In the request header, the Content-Type field MUST be set to application/jose+json.

Upon receiving an eStatus artifact, the registrar MUST process it as described in Section 5.9.4 of [RFC8995]. Due to the Registrar-Agent in the middle, instead of the BRSKI TLS session with the pledge, the registrar MUST verify the signature of the eStatus artifact and that it belongs to an accepted device of the domain based on the serial-number field of the EE certificate contained in the JWS Protected Header of the eStatus. Note that if the Enroll Status indicates success, the eStatus artifact is signed with the new pledge EE credentials; if it indicates failure, the pledge was unable to process the supplied EE certificate and therefore signed with its IDevID credentials.

According to Section 5.9.4 of [RFC8995], the registrar SHOULD respond with an HTTP 200 OK in the success case or MAY fail with an HTTP 404 client error status code. The registrar MAY use the response body to signal success/failure details to the service technician operating the Registrar-Agent.

If the eStatus indicates failure, the registrar MAY decide that for security reasons the pledge is not allowed to reside in the domain. In this case, the registrar MUST revoke the pledge EE certificate. An example case for the registrar revoking the issued certificate is when the pledge was not able to verify the received EE certificate and therefore did not accept it for installation.

7.10.1. Request Artifact: Enroll Status (eStatus)

Identical to the Enroll Status (eStatus) artifact received from the pledge as defined in Section 7.8.2. The Registrar-Agent MUST NOT modify eStatus artifacts.

7.10.2. Response (no artifact)

In this exchange, the response is a result of the HTTP(S) default transport for this specification. There is no artifact provided to the Registrar-Agent.

7.11. Query Pledge Status (qps)

The following assumes that a Registrar-Agent MAY need to query the overall status of a pledge. This information can be useful to solve errors, when the pledge was not able to connect to the target domain during bootstrapping. A pledge MAY omit the dedicated endpoint for the Query Pledge Status operation (see Section 6.2).

~~Optionally,~~ TLS MAY be used to provide privacy for this exchange between the Registrar-Agent and the pledge (see Appendix B).

Figure 43 shows the query and response for the overall pledge status and the following subsections describe the corresponding artifacts.

```
+--------+     +------------+     +-----------+     +--------+     +------+
| Pledge |     | Registrar- |     |  Domain   |     |  Key   |     | MASA |
|        |     |   Agent    |     | Registrar |     | Infra. |     |      |
+--------+     +------------+     +-----------+     +--------+     +------+
 |                  |                 |                 |   Internet |
 ~                  ~                 ~                 ~            ~
(11) Query Pledge Status
 ~                  ~                 ~                 ~            ~
 |                  |                 |                 |            |
 |<----opt. TLS---->|                 |                 |            |
 |<-----tStatus-----|                 |                 |            |
 |------pStatus---->|                 |                 |            |
 |                  |                 |                 |            |
 ~                  ~                 ~                 ~            ~
```

Commenté [MB48]: Change to «may»

Commenté [MB49]: Redundant with «MAY»

Figure 43: Pledge Status exchange

The Registrar-Agent SHALL query the pledge via HTTP(S) POST to the pledge endpoint at /.well-known/brski/qps. The request body MUST contain the Status Trigger (tStatus) artifact as defined in Section 7.11.1. In the request header, the Content-Type field MUST be set to application/jose+json and the Accept field SHOULD be set to application/jose+json.

If the pledge implements the Query Pledge Status endpoint, it MUST first verify the signature of the tStatus artifact using its trust anchors. If the pledge does not possess any domain trust anchor yet, it MAY skip the signature verification and choose to reply without it. In the case of success, it MUST reply with the Pledge Status (pStatus) artifact as defined in Section 7.11.2 in the body of an HTTP 200 OK response. In the response header, the Content-Type field MUST be set to application/jose+json.

If the pledge is unable to create the pStatus artifact, the pledge SHOULD respond with an HTTP error status code to the Registrar-Agent. The following client error status codes SHOULD be used:

Commenté [MB50]: Idem as a comment for similar constructs.

* 400 Bad Request: if the pledge detects an error in the format of the request

* 401 Unauthorized: if the signature of the Registrar-Agent cannot be verified using the installed trust anchors

* 406 Not Acceptable: if the Accept request header field indicates a type that is unknown or unsupported, e.g., a type other than application/jose+json

* 415 Unsupported Media Type: if the Content-Type request header field indicates a type that is unknown or unsupported, e.g., a type other than application/jose+json

The pledge MAY use the response body to signal failure details to the service technician operating the Registrar-Agent.

7.11.1. Request Artifact: Status Trigger (tStatus)

The Status Query (tStatus) artifact SHALL be an authenticated self-contained object signed by the pledge, providing status query parameters.

For the JWS-signed JSON format used by this specification, the tStatus artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON Status Trigger Data defined in Section 7.11.1.1 in the JWS Payload. Figure 44 summarizes the serialization of the JWS-signed JSON PER Artifact.~~:~~

```
{
  "payload": BASE64URL(UTF8(JSON Status Trigger Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
```

```
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 44: tStatus Representation in General JWS JSON Serialization Syntax

The JSON Status Trigger Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.11.1.1. JSON Status Trigger Data

The JSON Status Trigger Data SHALL be a JSON document [RFC8259] that MUST conform with the CDDL [RFC8610] data model defined in Figure 45:

```
statustrigger = {
    "version": uint,
    "serial-number": text,
    "created-on": tdate,
    "status-type": $status-type
}

$status-type /= "bootstrap"
$status-type /= "operation"
```

Figure 45: CDDL for JSON Status Trigger Data (statustrigger)

The version member is included to permit significant changes to the pledge status artifacts in the future. The format and semantics in this document follow the status telemetry definitions of [RFC8995]. Hence, the version SHALL be set to 1. A pledge (or Registrar-Agent) that receives a version larger than it knows about SHOULD log the contents and emit an operational notification.

The serial-number member SHALL contain the product-serial-number corresponding to the X520SerialNumber field of the pledge IDevID certificate; it can be correlated with the product-serial-number in the signing certificate contained in the JWS Protected Header of the Pledge Status response artifact.

The created-on member SHALL contain the current date and time at tStatus creation as standard date/time string as defined in Section 5.6 of [RFC3339]; it can be used as reference time for the corresponding Pledge Status response artifact after correlating via the product-serial-number; note that pledges may not have synchronized time to provide the created-on date and time on their own.

The status-type allows for specifying which status information is to be returned. As shown in Figure 45, BRSKI-PRM defines two enumeration values:

* bootstrap to query current status information regarding the bootstrapping status (e.g., voucher processing and enrollment of the pledge into a domain)

* operation to query current status information regarding the operational status (e.g., utilization of the bootstrapped EE credentials in communication with other peers)

Other specifications using this artifact may define further enumeration values, e.g., to query application-related status.

Figure 46 ~~below~~ shows an example for the JSON Status Trigger Data using the status type bootstrap:


```
{
  "version": 1,
  "created-on": "2022-08-12T02:37:39.235Z",
  "serial-number": "vendor-pledge4711",
  "status-type": "bootstrap"
}
```


Figure 46: JSON Status Trigger Data Example

7.11.1.2. JWS Protected Header

The JWS Protected Header of the tStatus artifact MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515].

* x5c: SHALL contain the base64-encoded Registrar-Agent EE certificate used to sign the tStatus artifact as well as the certificate chain.

Figure 47 ~~below~~ shows an example for this JWS Protected Header:


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 47: JWS Protected Header Example within tStatus

7.11.1.3. JWS Signature

The Registrar-Agent MUST sign the tStatus artifact using its EE credentials. The JWS Signature is generated over the JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

7.11.2. Response Artifact: Pledge Status (pStatus)

The Pledge Status (pStatus) artifact SHALL be an authenticated self-contained object signed by the pledge, containing status telemetry information. The exact content depends on the Status Trigger parameter status-type.

For the JWS-signed JSON format used by this specification, the pStatus artifact MUST use the "General JWS JSON Serialization Syntax" defined in Section 7.2.1 of [RFC7515], which MUST contain the JSON Pledge Status Data defined in Section 7.11.2.1 in the JWS Payload. Figure 48 summarizes the serialization of the JWS-signed JSON PER Artifact.~~:~~

```
{
  "payload": BASE64URL(UTF8(JSON Pledge Status Data)),
  "signatures": [
    {
      "protected": BASE64URL(UTF8(JWS Protected Header)),
      "signature": BASE64URL(JWS Signature)
    }
  ]
}
```

Figure 48: pStatus Representation in General JWS JSON Serialization Syntax

The JSON Pledge Status Data MUST be UTF-8 encoded to become the octet-based JWS Payload defined in [RFC7515]. The JWS Payload is further base64url-encoded to become the string value of the payload member as described in Section 3.2 of [RFC7515]. The octets of the UTF-8 representation of the JWS Protected Header are base64url-encoded to become the string value of the protected member. The generated JWS Signature is base64url-encoded to become the string value of the signature member.

7.11.2.1. JSON Pledge Status Data

The JSON Pledge Status Data SHALL be a JSON document [RFC8259] that MUST conform with the CDDL [RFC8610] data model defined in Figure 49, which has the same members as the voucherstatus-post CDDL defined in Section 5.7 of [RFC8995] and the enrollstatus-post CDDL defined in Section 5.9.4 of [RFC8995].

```
pledgestatus = {
  "version": uint,
  "status": bool,
  ?"reason" : text,
  "reason-context": { * $$arbitrary-map }
}
```

Figure 49: CDDL for JSON Pledge Status Data (pledgestatus)

The version member follows the definition in Section 7.11.1.1 (same as in JSON Status Query Data).

The reason and reason-context members follow the definitions in Section 7.6.2.1, i.e., in contrast to [RFC8995], reason-context MUST be provided.

The new pStatus artifact also utilizes the reason-context field to provide a distinguishable token. For pStatus artifacts, the JSON object in the reason-context field MUST contain either the

* pbs-details member for status information corresponding to the status-type bootstrap, or the

* pos-details member for status information corresponding to the status-type operation (see Section 7.11.1.1)

Other documents may add additional reason-context members correlating to other statustrigger status-types or to include further status information.

For the pbs-details member, the following values with the given semantics are defined, while additional information MAY be provided in the top-level reason member:

* factory-default: Pledge has not been bootstrapped. The pledge signs the response message using its IDevID certificate/credentials.

* voucher-success: Pledge processed the voucher exchange successfully. The pledge signs the response message using its IDevID certificate/credentials.

* voucher-error: Pledge voucher processing terminated with error. Additional information may be provided in the reason or reason-context members. The pledge signs the response message using its IDevID certificate/credentials.

* enroll-success: Pledge processed the enrollment exchange successfully. Additional information may be provided in the reason or reason-context members. The pledge signs the response message using its domain-owner signed EE certificate/credentials.

* enroll-error: Pledge enrollment-response processing terminated with error. Additional information may be provided in the reason or reason-context members. The pledge signs the response message using its IDevID certificate/credentials.

The pbs-details values SHALL be cumulative in the sense that enroll-success and enroll-error imply voucher-success. Figure 50 ~~below~~ provides an example for bootstrap status information in the JSON Pledge Status Data.~~:~~

```
{
  "version": 1,
  "status": true,
  "reason": "Pledge processed enrollment exchange successfully.",
  "reason-context": {
    "pbs-details": "Pledge processed enrollment exchange successfully."
  }
}
```

Figure 50: status-bootstrap JSON Pledge Status Data Example

For the pos-details member, the following values with the given semantics are defined, while additional information MAY be provided in the top-level reason member:

* connect-success: Pledge could successfully establish a connection to another peer. The pledge signs the response message using its domain-owner signed EE certificate/credentials.

* connect-error: Pledge connection establishment terminated with error. The pledge signs the response message using its domain-owner signed EE certificate/credentials.

Figure 51 below provides an example for operational status information in the JSON Pledge Status Data:


```
{
  "version": 1,
  "status": "connect-error",
  "reason": "TLS certificate could not be verified.",
  "reason-context": {
    "connect-error" : "Connection establishment terminated with error."
  }
}
```


Figure 51: status-operation JSON Pledge Status Data Example

7.11.2.2. JWS Protected Header

The JWS Protected Header of the pStatus artifact MUST contain the following standard Header Parameters as defined in [RFC7515]:

* alg: SHALL contain the algorithm type used to create the signature, e.g., ES256, as defined in Section 4.1.1 of [RFC7515].

* x5c: SHALL contain the base64-encoded pledge EE certificate used to sign the pStatus artifact; it SHOULD also contain the certificate chain for this certificate; if the certificate chain is not included in the x5c Header Parameter, it MUST be available at the Registrar-Agent for verification.

Figure 52 below shows an example for this JWS Protected Header:


```
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```


Figure 52: JWS Protected Header Example within pStatus

7.11.2.3. JWS Signature

The pledge MUST sign the tStatus artifact using its IDevID or domain-owner signed EE credentials according to its bootstrapping status as defined in Section 7.11.2.1. The JWS Signature is generated over the

JWS Protected Header and the JWS Payload as described in Section 5.1 of [RFC7515].

8. Logging Considerations

The registrar SHOULD log certain events to provide an audit trail for the onboarding of pledges into its domain. This audit trail may support the root cause analysis in case of device or system failures. Recommend key events for logging comprise:

* Communication attempts between the pledge, Registrar-Agent, and registrar.

* Protocol handshakes and onboarding steps.

* Voucher requests and responses.

* Authentication successes or failures.

The logging SHOULD include the identity of the pledge, the identity of the Registrar-Agent that was interacting with the pledge, and relevant artifact fields, in particular telemetry information:

* PVR received from Registrar-Agent

* Acceptance of a pledge into the domain

* Voucher provided to Registrar-Agent

* PER received from Registrar-Agent

* Pledge EE certificate requested

* Pledge EE certificate received from Domain CA

* Pledge EE certificate provided to Registrar-Agent

* CA Certificates provided to Registrar-Agent

* Voucher Status received from Registrar-Agent

* Enroll Status received from Registrar-Agent

* Pledge Status received from Registrar-Agent

* Pledge EE certificate revoked

Furthermore, it is recommended to:

* support adjustable logging levels (severity) to cater to different operational needs or failure situations.

* include meta information to distinguish logs that relate to different BRSKI approaches (e.g., BRSKI, BRSKI-AE, BRSKI-PRM, constraint BRSKI) that are likely supported in the same domain in parallel.

Commenté [MB51]: Logging the same event should be rate-limited.

* include detailed error codes and diagnostics information as defined throughout the document or stemming from other used components or libraries also in the logging information.

* support synchronized time (e.g., via NTP) to include timestamps in logging to enable sequencing and correlation of events.

* utilize standard logging formats (e.g., syslog) to allow for easy integration into log analysis tools and SIEM systems.

* utilize secure transmission of logs to centralized log servers, particularly in cloud or distributed environments (e.g., in case of syslog, [RFC9662] updates the utilized cipher suites for TLS and DTLS).

* allow for definition of key operational thresholds (e.g., high latency, failed onboarding attempts) to trigger alerts for proactive issue resolution.

* avoid inclusion of sensitive information (see also Section 11)

For log analysis the following may be considered:

* The registrar knows which Registrar-Agent collected which PVR from the included agent-signed-data object.

* The registrar always knows the connecting Registrar-Agent from the TLS client authentication using the Registrar-Agent EE certificate and can log it accordingly.

* The telemetry information from the pledge can be correlated to the voucher through the product-serial-number in the EE certificate contained in the JWS Protected Header of the status artifacts and the product-serial-number contained in the voucher. By this it can also be related to the PER.

With this, it can for instance be analyzed if multiple Registrar-Agents are involved in bootstrapping devices. In addition, within the domain it can be analyzed, if the onboarding involved different Registrar-Agents or if different registrars have been used.

9. Operational Considerations

As outlined in Section 5, BRSKI-PRM introduces an additional component with the Registrar-Agent in the BRSKI architecture in addition to new modes of interaction to facilitate the communication between the pledge and the registrar. This has an influence on the configuration and operation not only of the Registrar-Agent, but also for the registrar and the pledge.

As outlined in Section 6, there are additional configuration items dues to the introduction of the Registrar-Agent. This may increase operational complexity and potential misconfigurations in deploying and managing this entity~~.~~:

* A Registrar-Agent needs to be provided with a Registrar-Agent EE certificate, the domain registrar EE certificate and the list of

pledges. BRSKI-PRM is open regarding the selected provisioning method, which may be automated or by configuration.

* Pledges may support either BRSKI-PRM only or combined with other modes of operation.

* Registrars may support either BRSKI-PRM only or combined with other BRSKI modes of operation. The distinction of BRSKI and BRSKI-PRM is done based on the provided endpoints of the registrar. An operator deploying pledges with a mixed set of operation need to ensure that the domain registrar supports all necessary options to ensure bootstrapping of pledges depending of the supported operational mode.

* In addition, registrars may support a co-located Registrar-Agent, if nomadic operation of the Registrar-Agent is not required. This facilitates situations in which an operator wants to deploy BRSKI pledges acting as clients and BSKI pledges acting as servers.

With the Registrar-Agent enhancement a new component is introduced in the communication path between the pledge and the registrar. This likely increases the latency of the communication between the pledge and the registrar. The increase in latency due to this additional component may be neglected given that the Registrar-Agent operates with nomadic connectivity as outlined in Section 5.2.

BRSKI-PRM requires pledges to possess an IDevID to enable onboarding in new domains. IDevID (and corresponding trust anchors) are expected to have a rather long lifetime. This may allow for a longer period between device acquisition and initial onboarding. Contrary, if devices that have been provided with an LDevID (and corresponding trust anchors) and temporarily taken out of service, immediate connectivity when bringing them back to operation may not be given, as the LDevIDs typically have a much shorter validity period compared to IDevIDs. It is therefore recommended to onboard them as new devices to ensure they possess valid LDevIDs.

Besides the above, also consider the existing documents on operational modes for

* BRSKI registrars in [I-D.richardson-anima-registrar-considerations]

* BRSKI MASA in [I-D.richardson-anima-masa-considerations]

10. IANA Considerations

This document requires the following IANA actions.

10.1. BRSKI Well-Known URIs

IANA is requested to enhance the Registry entitled: "BRSKI Well-Known URIs" with the following endpoints:

Path Segment	Description	Reference
requestenroll	Supply PER to registrar	[THISRFC]

wrappedcacerts	Obtain wrapped CA certificates	[THISRFC]
tpvr	Trigger Pledge Voucher-Request	[THISRFC]
tper	Trigger Pledge Enroll-Request	[THISRFC]
svr	Supply voucher to pledge	[THISRFC]
scac	Supply CA certificates to pledge	[THISRFC]
ser	Supply Enroll-Response to pledge	[THISRFC]
qps	Query pledge status	[THISRFC]

Table 3: BRSKI Well-Known URIs Additions

10.2. Service Name and Transport Protocol Port Number Registry

IANA has registered the following service names:

Commenté [MB52]: Add an action for IANA to update that entry

Service Name: brski-pledge
Transport Protocol(s): tcp
Assignee: IESG iesg@ietf.org (mailto:iesg@ietf.org)
Contact: IETF Chair chair@ietf.org (mailto:chair@ietf.org)
Description: The Bootstrapping Remote Secure Key Infrastructure Pledge
Reference: [THISRFC]

11. Privacy Considerations

In general, the privacy considerations of [RFC8995] apply for BRSKI-PRM also. Further privacy aspects need to be considered for:

* the introduction of the additional component Registrar-Agent

* potentially no transport layer security between Registrar-Agent and pledge

Section 7.1 describes to optionally apply TLS to protect the communication between the Registrar-Agent and the pledge. The following is therefore applicable to the communication without the TLS protection.

The credentials used by the Registrar-Agent to sign the data for the pledge SHOULD NOT contain any personal information. Therefore, it is recommended to use an EE certificate associated with the commissioning device instead of an EE certificate associated with the service technician operating the device. This avoids revealing potentially included personal information to Registrar and MASA.

As logging is recommended to better handle failure situations, it is necessary to avoid capturing sensitive or personal data. Privacy-preserving measures in logs SHOULD be applied, such as: * Avoid logging personally identifiable information unless unavoidable. * Anonymize or pseudonymize data where possible.

The communication between the pledge and the Registrar-Agent is performed over plain HTTP. Therefore, it is subject to disclosure by a Dolev-Yao attacker (an "oppressive observer")[onpath]. Depending on the requests and responses, the following information is disclosed.

* the Pledge product-serial-number is contained in the trigger message for the PVR and in all responses from the pledge. This information reveals the identity of the devices being bootstrapped and allows deduction of which products an operator is using in their environment. As the communication between the pledge and the Registrar-Agent may be realized over wireless link, this information could easily be eavesdropped, if the wireless network is not encrypted. Even if the wireless network is encrypted, if it uses a network-wide key, then layer-2 attacks (ARP/ND spoofing) could insert an on-path observer into the path.

* the Timestamp data could reveal the activation time of the device.

* the Status data of the device could reveal information about the current state of the device in the domain network.

12. Security Considerations

In general, the security considerations of [RFC8995] apply for BRSKI-PRM also. Further security aspects are considered here related to:

* the introduction of the additional component Registrar-Agent

* the reversal of the pledge communication direction (push mode, compared to BRSKI)

* no transport layer security between Registrar-Agent and pledge

12.1. Denial of Service (DoS) Attack on Pledge

Disrupting the pledge behavior by a DoS attack may prevent the bootstrapping of the pledge to a new domain. Because in BRSKI-PRM the pledge responds to requests from real or illicit Registrar-Agents, pledges are more subject to DoS-attacks from Registrar-Agents in BRSKI-PRM than they are from illicit registrars in [RFC8995], where pledges do initiate the connections.

A DoS attack with a faked Registrar-Agent may block the bootstrapping of the pledge due changing state on the pledge (the pledge may produce a voucher-request, and refuse to produce another one). One mitigation may be that the pledge does not limit the number of voucher-requests it creates until at least one has finished. An alternative may be that the onboarding state may expire after a certain time, if no further interaction has happened.

In addition, the pledge may assume that repeated triggering for PVR are the result of a communication error with the Registrar-Agent. In that case the pledge MAY simply resend the PVR previously sent. Note that in case of re-sending, a contained nonce and also the contained agent-signed-data in the PVR would consequently be reused.

12.2. Misuse of acquired PVR and PER by Registrar-Agent

A Registrar-Agent that uses previously requested PVR and PER for domain-A, may attempt to onboard the device into domain-B. This can be detected by the domain registrar while PVR processing. The domain registrar needs to verify that the proximity-registrar-cert field in the PVR matches its own registrar EE certificate. In addition, the domain registrar needs to verify the association of the pledge to its domain based on the product-serial-number contained in the PVR and in the pledge IDevID certificate. (This is just part of the supply chain integration). Moreover, the domain registrar verifies if the Registrar-Agent is authorized to interact with the pledge for voucher-requests and enroll-requests, based on the Registrar-Agent EE certificate data contained in the PVR.

Mis-binding of a pledge by a faked domain registrar is countered as described in BRSKI security considerations Section 11.4 of [RFC8995].

12.3. Misuse of Registrar-Agent

Concerns of misuse of a Registrar-Agent with a valid Registrar-Agent EE certificate may be addressed by utilizing short-lived certificates (e.g., valid for a day) to authenticate the Registrar-Agent against the domain registrar. The Registrar-Agent EE certificate may have been acquired by a prior BRSKI run for the Registrar-Agent, if an IDevID is available on Registrar-Agent. Alternatively, the Registrar-Agent EE certificate may be acquired by a service technician from the domain PKI system in an authenticated way.

In addition, it is required that the Registrar-Agent EE certificate is valid for the complete bootstrapping phase. This avoids that a Registrar-Agent could be misused to create arbitrary "agent-signed-data" objects to perform an authorized bootstrapping of a rogue pledge at a later point in time. In this misuse "agent-signed-data" could be dated after the validity time of the Registrar-Agent EE certificate, due to missing trusted timestamp in the Registrar-Agents signature. To address this, the registrar SHOULD verify the certificate used to create the signature on "agent-signed-data".

Furthermore, the registrar also verifies the Registrar-Agent EE certificate used in the TLS handshake with the Registrar-Agent. If both certificates are verified successfully, the Registrar-Agent's signature can be considered as valid. If the registrar detects a mismatch in the utilized certificates, it may conclude the usage of either an outdated "agent-signed-data" component in the PVR or a man-in-the-middle attack by a potentially unauthorized Registrar-Agent.

12.4. Misuse of DNS-SD with mDNS to obtain list of pledges

To discover a specific pledge a Registrar-Agent may query the Service Type in combination with the product-serial-number of a specific pledge, e.g., in the Service Instance Name or Service Subtype. The pledge reacts on this if its product-serial-number is part of the query message.

If the Registrar-Agent performs DNS-based Service Discovery without a specific product-serial-number, all pledges in the domain react if the functionality is supported. This functionality enumerates and reveals the information of devices available in the domain. The information about this is provided here as a feature to support the commissioning of devices. A manufacturer may decide to support this feature only for devices not possessing an LDevID or to not support this feature at all, to avoid an enumeration in an operative domain.

12.5. YANG Module Security Considerations

The enhanced voucher-request described in [I-D.ietf-anima-rfc8366bis] is based on [RFC8995], but uses a different encoding based on [I-D.ietf-anima-jws-voucher]. The security considerations as described in Section 11.7 of [RFC8995] (Security Considerations) apply.

The YANG module specified in [I-D.ietf-anima-rfc8366bis] defines the schema for data that is subsequently encapsulated by a JOSE signed-data Content-type as described in [I-D.ietf-anima-jws-voucher]. As such, all of the YANG-modeled data is protected against modification.

The use of YANG to define data structures via the [RFC8971] "structure" statement, is relatively new and distinct from the common use of YANG to define an API accessed by network management protocols such as NETCONF [RFC6241] and RESTCONF [RFC8040]. For this reason, these guidelines do not follow the template described by Section 3.7 of [RFC8407] (Security Considerations).

Commenté [MB53]: You don't need to have this per 8407bis:

Documents that define exclusively modules following the extension in [RFC8791] are not required to include the security template in Section 3.7.1. Likewise, following the template is not required for modules that define YANG extensions such as [RFC7952].

13. Acknowledgments

We would like to thank the various reviewers, in particular Brian E. Carpenter, Charlie Kaufman (Early SECDIR review), Martin Björklund (Early YANGDOCTORS review), Marco Tiloca (Early IOTDIR review), Oskar Camenzind, Hendrik Brockhaus, and Ingo Wenda for their input and discussion on use cases and call flows. Further review input was provided by Jesser Bouzid, Dominik Tacke, Christian Spindler, and Julian Krieger. Special thanks to Esko Dijk for the in deep review and the improving proposals. Another special thanks for the detailed Shepherad review and connected discussions to Matthias Kovatsch. Support in PoC implementations and comments resulting from the implementation was provided by Hong Rui Li and He Peng Jia. Review comments in the context of a formal analysis of BRSKI-PRM have been provided by Marco Calipari.

14. References

14.1. Normative References

[I-D.ietf-anima-jws-voucher]
Werner, T. and M. Richardson, "JWS signed Voucher

Artifacts for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-jws-voucher-16, 15 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-jws-voucher-16>.

[I-D.ietf-anima-rfc8366bis] Watsen, K., Richardson, M., Pritikin, M., Eckert, T. T., and Q. Ma, "A Voucher Artifact for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-rfc8366bis-12, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-rfc8366bis-12>.

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

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/rfc/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/rfc/rfc5280>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/rfc/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/rfc/rfc6763>.

[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/rfc/rfc7030>.

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

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/rfc/rfc7951>.

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data

Commenté [MB54]: ---DISCUSS

Are we confident that no changes will be induced by ietf-anima-rfc8366bis?

Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

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

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

[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/rfc/rfc8995>.

[RFC9360] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/rfc/rfc9360>.

[RFC9646] Watsen, K., Housley, R., and S. Turner, "Conveying a Certificate Signing Request (CSR) in a Secure Zero-Touch Provisioning (SZTP) Bootstrapping Request", RFC 9646, DOI 10.17487/RFC9646, October 2024, <https://www.rfc-editor.org/rfc/rfc9646>.

14.2. Informative References

[androidnsd]
"Android Developer: Connect devices wirelessly", archived at https://web.archive.org/web/20230000000000*/https://developer.android.com/training/connect-devices-wirelessly, n.d., <https://developer.android.com/training/connect-devices-wirelessly>.

[androidtrustfail]
"Security with Network Protocols", archived at https://web.archive.org/web/20230326153937/https://developer.android.com/training/articles/security-ssl, n.d., <https://developer.android.com/training/articles/security-ssl>.

[BRSKI-PRM-abstract]
"Abstract BRSKI-PRM Protocol Overview", March 2022, <https://datatracker.ietf.org/meeting/113/materials/slides-113-anima-update-on-brski-with-pledge-in-responder-mode-brski-prm-00>.

[I-D.ietf-anima-brski-ae]
von Oheimb, D., Fries, S., and H. Brockhaus, "BRSKI-AE: Alternative Enrollment Protocols in BRSKI", Work in Progress, Internet-Draft, draft-ietf-anima-brski-ae-13, 17 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-brski-ae-13>.

[I-D.ietf-anima-brski-discovery]
Eckert, T. T. and E. Dijk, "BRSKI discovery and variations", Work in Progress, Internet-Draft, draft-ietf-anima-brski-discovery-05, 21 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-brski-discovery-05>.

[I-D.irtf-t2trg-taxonomy-manufacturer-anchors]
Richardson, M., "A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors", Work in Progress, Internet-Draft, draft-irtf-t2trg-taxonomy-manufacturer-anchors-05, 2 January 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-taxonomy-manufacturer-anchors-05>.

[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-09, 22 January 2025, <https://datatracker.ietf.org/doc/html/draft-richardson-anima-masa-considerations-09>.

[I-D.richardson-anima-registrar-considerations]
Richardson, M. and W. Pan, "Operational Considerations for BRSKI Registrar", Work in Progress, Internet-Draft, draft-richardson-anima-registrar-considerations-09, 22 January 2025, <https://datatracker.ietf.org/doc/html/draft-richardson-anima-registrar-considerations-09>.

[I-D.richardson-emu-eap-onboarding]
DeKok, A. and M. Richardson, "EAP defaults for devices that need to onboard", Work in Progress, Internet-Draft, draft-richardson-emu-eap-onboarding-03, 2 April 2023, <https://datatracker.ietf.org/doc/html/draft-richardson-emu-eap-onboarding-03>.

[IEEE-802.1AR]
Institute of Electrical and Electronics Engineers, "IEEE 802.1AR Secure Device Identifier", IEEE 802.1AR, June 2018.

[onpath] "can an on-path attacker drop traffic?", n.d., <https://mailarchive.ietf.org/arch/msg/saag/m1r9uo4xYznOcf85Eyk0Rhut598/>.

[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/rfc/rfc2986>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November

2003, <https://www.rfc-editor.org/rfc/rfc3629>.

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

[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/rfc/rfc6241>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/rfc/rfc8040>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/rfc/rfc8407>.

[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/rfc/rfc8792>.

[RFC8971] Pallagatti, S., Ed., Mirsky, G., Ed., Paragiri, S., Govindan, V., and M. Mudigonda, "Bidirectional Forwarding Detection (BFD) for Virtual eXtensible Local Area Network (VXLAN)", RFC 8971, DOI 10.17487/RFC8971, December 2020, <https://www.rfc-editor.org/rfc/rfc8971>.

[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/rfc/rfc8990>.

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

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9238] Richardson, M., Latour, J., and H. Habibi Gharakheili, "Loading Manufacturer Usage Description (MUD) URLs from QR Codes", RFC 9238, DOI 10.17487/RFC9238, May 2022, <https://www.rfc-editor.org/rfc/rfc9238>.

[RFC9483] Brockhaus, H., von Oheimb, D., and S. Fries, "Lightweight Certificate Management Protocol (CMP) Profile", RFC 9483,

DOI 10.17487/RFC9483, November 2023,
<https://www.rfc-editor.org/rfc/rfc9483>.

[RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <https://www.rfc-editor.org/rfc/rfc9525>.

[RFC9662] Lonvick, C., Turner, S., and J. Salowey, "Updates to the Cipher Suites in Secure Syslog", RFC 9662, DOI 10.17487/RFC9662, October 2024, <https://www.rfc-editor.org/rfc/rfc9662>.

Appendix A. Examples

These examples are folded according to [RFC8792] Single Backslash rule.

A.1. Example Pledge Voucher-Request (PVR) - from Pledge to Registrar-Agent

The following is an example request sent from a Pledge to the Registrar-Agent, in "General JWS JSON Serialization". The message size of this PVR is: 2973 bytes


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

{
  "payload": "eyJpZXRmLXZvdWNoZXItcmVxdWVzdC1wcm06dm91Y2hlciI6eyJhc3\
NlcnRpb24iOiJhZ2VudC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Nj\
c4OSIsIm5vbmNlIjoia2hOeUtwTXRoY2NpYTFyWHc0NC92UT09IiwiY3JlYXRlZC1vbi\
I6IjIwMjQtMDYtMjRUMDk6MDE6MjQuNTU2WiIsImFnZW50LXByb3ZpZGVkLXByb3hpbW\
l0eS1yZWdpc3RyYXItY2VydCI6Ik1JSUI0akNDQVlpZ0F3SUJBZ0lHQVhhZNzJiYlpNQW\
9HQ0NxR1NNNDlCQU1DTURVeEV6QVJCZ05WQkFvTUNrMTVRblZ6YVc1bGMzTXhEVEFMQm\
dOVkJBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUmxjM1JEUVRBZUZ3MHlNREV5TURjd0\
5qRTRNVEphRncwek1ERXlNRGN3TmpFNE1USmFNRDR4RXpBUkJnTlZCQW9NQ2sxNVFuVn\
phVzVsYzNNeERUQUxCZ05WQkFjTUJGTnBkR1V4R0RBV0JnTlZCQU1NRDBSdmJXRnBibE\
psWjJsemRISmhjakJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUFCQmsxNk\
svaTc5b1JrSzVZYmVQZzhVU1I4L3VzMWRQVWlaSE10b2tTZHFLVzVmbldzQmQrcVJMN1\
dSZmZlV2t5Z2Vib0pmSWxsdXJjaTI1d25oaU9WQ0dqZXpCNU1CMEdBMVVkSlFRV01CUU\
dDQ3NHQVFVRkJ3TUJCZ2dyQmdFRkJRY0RIREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdTQV\
lEVlIwUkJFRXdQNElkY21WbmFYTjBjbUZ5TFhSbGMzUXVjMmxsYldWdWN5MWlkQzV1Wl\
hTQ0huSmxaMmx6ZEhKaGNpMTBaWE4wTmk1emFXVnRaVzV6TFdKMExtNWxkREFLQmdncW\
hrak9QUVFEQWdOSUFEQkZBaUJ4bGRCaFpxMEV2NUpMMlByV0N0eVM2aERZVzF5Q08vUm\
F1YnBDN01hSURnSWhBTFNKYmdMbmdoYmJBZzBkY1dGVVZvL2dHTjAvand6SlowU2wyaD\
R4SVhrMSIsImFnZW50LXNpZ25lZC1kYXRhIjoiZXlKd1lYbHNiMkZrSWpvaVpYbEtjRn\
BZVW0xTVdGcDJaRmRPYjFwWVNYUmpiVlo0WkZkV2VtUkRNWGRqYlRBMldWZGtiR0p1VV\
hSak1teHVZbTFXYTB4WFVtaGtSMFZwVDI1emFWa3pTbXhaV0ZKc1drTXhkbUpwU1RaSm\
FrbDNUV3BKZEUxRWEzUk5ha3BWVFVSVk5rNUVUVFpPVkVGMVRWUkpNVmRwU1hOSmJrNX\
NZMjFzYUdKRE1YVmtWekZwV2xoeVNmFVOXBTWGROVkVsNlRrUlZNazU2WnpWSmJqRTVJaX\
dpYzJsbmJtJtRjBkWEpsY3lJJNlczc2ljSEp2ZEdWamRHVmtJam9pWlhsS2NtRlhVV2xQYV\
VwVlZFZE5NMWRZYUV4V2JGWldaVzVLTTFKVVRsSlhWRlpEV2xaa2IyTXlNVVZOTW1NNV\
NXbDNNhVmxYZUc1SmFtOXBVbFpOZVU1VVdXbG1VU0lzSW5OcFoyNWhkSFZ5WlNJNklrd3\
lZVEJsY3pWZkxXZHNZVjkwTjFVME1VbFJXRmxJU1RSQlMxVldVRkZmTTFSbGQxUTFiMF\
ZWWVVOdFVIQktaMmRyU0c1d09WTk1aVFZ1YWkxbldGbFRiMk5sT1RoeFFXSnROa0YwZF\
MxRlIxUkxZMDVSSW4xZGZRMEsifX0",
  "signatures": [
    {
      "protected": "eyJ4NWMiOlsiTUlJQitUQ0NBYUNnQXdJQkFnSUdBWG5WanNV\
```

```
NU1Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RB\
eEthVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJN\
Q0FYRFRJeE1EWXdOREExTkRZeE5Gb1lEems1T1RreE1qTXhNak0xT1RVNVdqQlNNUXN3\
Q1FZRFZRUUdFd0pCVVRFVk1CTUdBMVVFQ2d3TVNtbHVaMHBwYm1kRGIzSndNUk13RVFZ\
RFZRUUZFd293TVRJek5EVTJOemc1TVJjd0ZRWURWUVFEREE1S2FXNW5TbWx1WjBSbGRt\
bGpaVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUFCQzc5bGlhUmNCalpj\
RUVYdzdyVWVhdnRHSkF1SDRwazRJNDJ2YUJNc1UxMWlMRENDTGtWaHRVVjIxbXZhS0N2\
TXgyWStTTWdROGZmd0wyM3ozVElWQldqZFRCek1Dc0dDQ3NHQVFVRkJ3RWdCQjhXSFcx\
aGMyRXRkR1Z6ZEM1emFXVnRaVzV6TFdKMExtNWxkRG81TkRRek1COEdBMVVkSXdRWU1C\
YUFGRlFMak56UC9TL2tvdWpRd2pnNUU1ZnZ3Y1liTUJNR0ExVWRKUVFNTUFvR0NDc0dB\
UVVGQndNQ01BNEdBMVVkRHdFQi93UUVBd0lIZ0RBS0JnZ3Foa2pPUFFRREFnTkhBREJF\
QWlCdTN3UkJMc0pNUDVzTTA3MEgrVUZyeU5VNmdLekxPUmNGeVJST2xxcUhpZ0lnWENt\
SkxUekVsdkQycG9LNmR4NmwxL3V5bVRuYlFERGZKbGF0dVgyUm9PRT0iXSwidHlwIjoi\
dm91Y2hlci1qd3MranNvbiIsImFsZyI6IkVTMjU2In0",
      "signature": "ntAgC7GT7xIDYcHBXoYej8uIUI6WR2Iv-7T1CaR-J6-xS60D\
iWS1-vfc5Uu5INZS1dyWZ4vVH6uaoPceRxNc8g"
    }
  ]
}
```


Figure 53: Example Pledge-Voucher-Request - PVR

A.2. Example Parboiled Registrar Voucher-Request (RVR) - from Registrar to MASA

The term parboiled refers to food which is partially cooked. In [RFC8995], the term refers to a pledge-voucher-request (PVR) which has been received by the Registrar, and then has been processed by the Registrar ("cooked"), and is now being forwarded to the MASA.

The following is an example registrar-voucher-request (RVR) sent from the Registrar to the MASA, in "General JWS JSON Serialization". Note that the previous PVR can be seen in the payload as "prior-signed-voucher-request". The message size of this RVR is: 7533 bytes


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

{
  "payload": "eyJpZXRmLXZvdWNoZXItcmVxdWVzdC1wcm06dm91Y2hlciI6eyJhc3\
NlcnRpb24iOiJhZ2VudC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Nj\
c4OSIsImlkZXZpZC1pc3N1ZXIiOiJCQmd3Rm9BVVZBdU0zTS85TCtTaTZORENPRGtUbC\
svQnhocz0iLCJub25jZSI6ImtoTnlLcE10aGNjaWExclh3NDQvdlE9PSIsInByaW9yLX\
NpZ25lZC12b3VjaGVyLXJlcXVlc3QiOiJleUp3WVhsc2IyRmtJam9pWlhsS2NGcFlVbT\
FNV0ZwMlpGZE9iMXBZU1hSamJWWjRaRmRXZW1SRE1YZGpiVEEyWkcwNU1Wa3lhR3hqYV\
VrMlpYbEthR016VG14amJsSndZakkwYVU5cFNtaGFNbFoxWkVNeGQyTnRPVFJoVnpGd1\
pFaHJhVXhEU25wYVdFcHdXVmQzZEdKdVZuUlpiVlo1U1dwdmFVMUVSWGxOZWxFeFRtcG\
pORTlUU1hOSmJUVjJZbTFPYkVscWIybGhNbWhQWlZWMGQxUllVbTlaTWs1d1dWUkdlVm\
RJWXpCT1F6a3lWVlF3T1VscGQybFpNMHBzV1ZoU2JGcERNWFppYVVrMlNXcEpkMDFxVV\
hSTlJGbDBUV3BTVlUxRWF6Wk5SRVUyVFdwUmRVNVVWVEpYYVVselNXMUdibHBYTlRCTV\
dFSjVZak5hY0ZwSFZtdE1XRUo1WWpOb2NHSlhhREJsVXpGNVdsZGtjR016VW5sWldFbD\
BXVEpXZVdSRFNUWkphekZLVTFWSk1HRnJJUa1JSVm14d1dqQkdNMU5WU2tKYU1HeElVVl\
pvV2s1NlNtbFpiSEJPVVZZjNVNGRXdUbmhTTVU1T1RrUnNRMUZWTVVSVVZWSldaVVZXTm\
xGV1NrTmFNRFZYVVd0R2RsUlZUbkpOVkZaU1lteGFObGxXWXpGaVIwMTZWRmhvUlZaRl\
JrMVJiV1JQVm10S1Fsa3dNVU5TYXpWM1drVmtWbVZGVWpaUlZUVkRXakExVjFGclJrNV\
VWVXB6VlcxNGFrMHhTa1ZWVmxKQ1dsVmFNMDFJYkU1U1JWWTFWRlZTTYW1Rd05YRlNWRk\
pPVmtWd2FGSnVZM2RsYXpGRlVsaHNUbEpIVGpOVWJYQkdUa1V4VlZOdFJrNVNSRkkwVW\
xod1FsVnJTbTVVYkZwRFVWYzVUbEV5YzNoT1ZrWjFWbTV3YUZaNlZuTlplazVPWlVWU1\
ZWRlZlRU5hTURWWFVXdEdhbFJWU2tkVWJrSnJVakZXTkZJd1VrSldNRXB1Vkd4YVExRl\
```

ZNVTVTUkVKVFpHMUtXRkp1UW1saVJYQnpWMnBLYzJWdFVrbFRiV2hxWVd0S1lWUlZTaz\
VTTUVvMVkxVmtWRlJVVVRWUlYyUkdVakJPUkdOVlpGUlVWRkUxVVZoa1JsTkZSWGRUVl\
VaRFVXMXplRTVyYzNaaFZHTTFZakZLY2xONlZscFpiVlpSV25wb1ZsVXhTVFJNTTFaNl\
RWZFNVVlpYYkdGVFJURXdZakowVkZwSVJreFdlbFp0WW14a2VsRnRVWEpqVmtwTlRqRm\
tVMXB0V214V01uUTFXakpXYVdJd2NHMVRWM2h6WkZoS2FtRlVTVEZrTWpWdllWVTVWMU\
V3WkhGYVdIQkRUbFV4UTAxRlpFSk5WbFpyVTJ4R1VsWXdNVU5WVldSVVVUk9TRkZXUm\
xaU2Ewb3pWRlZZLUTFveVpIbFJiV1JHVW10S1Vsa3dVa2xTUlVaUVVXMWtUMVpyYUZKUF\
JVcENNXbXBvUmxGclJrNVJNRWt3VVZoa1ZGRldiRVZXYkVsM1ZXdEtSbEpZWkZGT1JXeH\
JXVEl4VjJKKdFJsbFVha0pxWWxWYU5WUkdhRk5pUjAxNlZWaFdhazF0ZUhOWmJHUlhaRm\
RPTlUxWGJHdFJlbFl4VjJ4b1ZGRXdhSFZUYlhoaFRXMTRObHBHYUV0aFIwNXdUVlJDWV\
ZkRk5IZFViV3N4WlcxR1dGWnVVbUZXZWxZMlZFWmtTMDFGZUhST1YzaHJVa1ZHVEZGdF\
pHNWpWMmh5WVdzNVVWVldVldSa1ZSVjJSUFUxVkdSVkZyV2tKaFZVbzBZa2RTUTJGR2NIaE\
5SVll5VGxWd1RVMXNRbmxXTUU0d1pWWk5NbUZGVWxwV2VrWTFVVEZVVEE0ZGxWdFJqRlpia0\
pFVGpBeGFGTlZZVmJUVUVjJoQ1ZFWk9TTWx0WkUxaWJXUUnZXVzFLUWxwWlFtdFpwNV1JIVm\
xaYWRrd3laRWhVYWtGMllXNWtObE5zYzYjNkVk1uZDVZVVJTTkZOV2FISk5VMGx6U1cxR2\
JscFhOVEJNV0V0aGVEJNV0U1d1dqSTFiRTVWd0xdGSm9TV3B2YVZaWWFJFdGtNV3haZWtoT2FFVMX\
JXbkpwUVVjNCMllWWndXV0NGQ2RmGZUdwU2JrSmFhRjpWdFZTVGFCYUNSZEpjYmU1pQZHVmGpSRlBxVW1YWVRGMGRzU2JHFVlNRXpKRlFOa2RldThSbXJ1V2hNlNoc3J1RtbFpQWhWQnU1pVWwNE5UVkVFVm1aWnhkWXJYUEJaXdidTlVXJqazE5aQ==

```
   dsWVUzZHBaRWhzZDBscWIybGtiVGt4V1RKb2JHTnBNWEZrTTAxeVlXNU9kbUpwU1hOSm\
   JVWnpXbmxKTmtsclZsUk5hbFV5U1c0d0lpd2ljMmxuYm1GMGRYSmxJam9pYm5SQlowTT\
   NSMVEzZUVsRVdXTklRbGh2V1dWcU9IVkpWVWsyVjFJeVNYWXROMVF4UTJGU0xVbzJMWG\
   hUTmpCRWFWZFRNUzEyWm1NMVZYVTFTVTVhVXpGa2VWZGFOSFpXU0RaMVlXOVFZMlZTZU\
   U1ak9HY2lmVjE5IiwiY3JlYXRlZC1vbiI6IjIwMjQtMDYtMjRUMDk6MDI6MTUuNTczWi\
   IsImFnZW50LXNpZ24tY2VydCI6WyJNSUlCOWpDQ0FaMmdBd0lCQWdJRVl4WHM3VEFLQm\
   dncWhrak9QUVFEQWpBK01STXdFUVlEVlFRS0RBcE5lVUoxYzJsdVpYTnpNUTB3Q3dZRF\
   ZRUUhEQVJUYVhSbE1SZ3dGZ1lEVlFRRERBOVVaWE4wVUhWemFFMXZaR1ZzUTBFd0hoY0\
   5Nakl3T1RBMU1USXpORFV6V2hjTk1qVXdPVEExTVRJek5EVXpXakJnTVFzd0NRWURWUV\
   FHRXdKQlVURVNNQkFHQTFVRUNnd0pUWGxEYjIxd1lXNTVNUlV3RXdZRFZRUUxEQXhOZV\
   ZOMVluTnBaR2xoY25reEpqQWtCZ05WQkFNTUhVMTVVMmwwWlZCMWMyaE5iMlJsYkZKbF\
   oybHpkSEpoY2tGblpXNTBNRmt3RXdZSEtvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFnQU\
   V4aHZuYWtDSmVpZ3pqWkFVYU5adVAwMWUrUWxVY1E5UjJMSWs2UkI2dmtjdFdMS3BaWC\
   85TGthNEdxckFWWmhhM3ZKcmhGc0l4OEdUQkhqWnZLMVd1Nk5uTUdVd0RnWURWUjBQQQV\
   FIL0JBUURBZ09JTUI4R0ExVWRJd1FZTUJhQUZHK2hQVzUxN1ovb3NSQ0ZUc2NlUDY4bj\
   kzc2pNQjBHQTFVZERnUVdCQlJNdHp0akVwVlJUT3ZBVGRCamtGNWFHeVlQZURBVEJnTl\
   ZIU1VFRERBS0JnZ3JCZ0VGQlFjREFqQUtCZ2dxaGtqT1BRUURBZ05IQURCRUFpQmJoRG\
   pwbDJ2cWNONnBSVjRuZVU0dFFsWWFOTit4ZjNnSnUrMHBKblNBL1FJZ0ljcXpsZmhYaU\
   Qxc0g3VTVQdUtwVVpzSWpkRjRSenhzQTZxSnRFTEQyUHM9Il19fQ",
     "signatures": [
       {
         "protected": "eyJ4NWMiOlsiTUlJQm96Q0NBVXFnQXdJQkFnSUdBVzBlTHVJ\
   Rk1Bb0dDQ3FHU000OUJBTUNNRFV4RXpBUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERU\
   QUxCZ05WQkFjTUJGTnBkR1V4RHpBTkJnTlZCQU1NQmxSbGMzUkRRVEFlRncweE9UQTVN\
   VEV3TWpNM016SmFGdzB5T1RBNU1URXdNak0zTXpKYU1GUXhFekFSQmdOVkJBb01DazE1\
   UW5WemFXNWxjM014RFRBTEJnTlZCQWNNQkZOcGRHVXhMakFzQmdOVkJBTU1KVkpsWjJs\
   emRISmhjaUJXYjNWamFHVnlJRkpsY1hWbGMzUWdVMmxuYm1sdVp5QkxaWGt3V1RBVEJn\
   Y3Foa2pPUFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVQ2eFZ2QXZxVHoxWlVpdU5XaFhwUXNr\
   YVB5N0FISFFMd1hpSjBpRUx0NnVOUGFuQU4wUW5XTVlPLzBDREVqSWtCUW9idzhZS3Fq\
   dHhKSFZTR1RqOUtPb3ljd0pUQVRCZ05WSFNVRUREQUtCZ2dyQmdFRkJRY0RIREFPQmdO\
   VkhROEJBZjhFQkFNQ0I0QXdDZ1lJS29aSXpqMEVBd0lEUndBd1JBSWdZcjJMZnFvYUNL\
   REY0UkFjTW1KaStOQ1pxZFNpdVZ1Z0lTQTdPaEtScTNZQ0lEeG5QTU1ucFhBTVRyUEp1\
   UFd5Y2VFUjExUHhIT24rMENwU0hpMnFncFdYIl0sInR5cCI6InZvdWNoZXItandzK2pz\
   b24iLCJhbGciOiJFUzI1NiJ9",
         "signature": "_mcsO5vo0g2rFmBvTb-UsOWkEmhYNfQ5XmbuKHKH0ZLjea-7\
   911BilAMdFORmT4vCzWKBSH6HSqtpIRcSSxx7Q"
       }
     ]
   }
```


Figure 54: Example Registrar-Voucher-Request - RVR

A.3. Example Voucher - from MASA to Pledge, via Registrar and Registrar-Agent

The following is an example voucher-response from MASA to Pledge via Registrar and Registrar-Agent, in "General JWS JSON Serialization". The message size of this Voucher is: 1916 bytes

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


```
   {
     "payload":"eyJpZXRmLXZvdWNoZXI6dm91Y2hlciI6eyJhc3NlcnRpb24iOiJhZ2V\
   udC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Njc4OSIsIm5vbmNlIjo\
   iTDNJSjZocHRIQ0lRb054YWFiOUhXQT09IiwiY3JlYXRlZC1vbiI6IjIwMjItMDQtMjZ\
   UMDU6MTY6MjguNzI2WiIsInBpbm5lZC1kb21haW4tY2VydCI6Ik1JSUJwRENDQVVtZ0F\
   3SUJBZ0lHQVcwZUx1SCtNQW9HQ0NxR1NNNDlCQU1DTURVeEV6QVJCZ05WQkFvTUNrMTV\
   RblZ6YVc1bGMzTXhEVEFMQmdOVkJBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUmxjM1J\
```

```
EUVRBZUZ3MHhPVEE1TVRFd01qTTNNekphRncweU9UQTVNVEV3TWpNM016SmFNRFV4RXp\
BUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERUQUxCZ05WQkFjTUJGTnBkR1V4RHpBTkJ\
nTlZCQU1NQmxSbGMzUkRRVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUF\
CT2t2a1RIdThRbFQzRkhKMVVhSTcrV3NIT2IwVVMzU0FMdEc1d3VLUURqaWV4MDYvU2N\
ZNVBKaWJ2Z0hUQitGL1FUamdlbEhHeTFZS3B3Y05NY3NTeWFqUlRCRE1CSUdBMVVkRXd\
FQi93UUlNQVlCQWY4Q0FRRXdEZ1lEVlIwUEFRSC9CQVFEQWdJRU1CMEdBMVVkRGdRV0J\
CVG9aSU16UWRzRC9qLytnWC83Y0JKKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQkd\
BaUVBdHhRMytJTEdCUEl0U2g0YjlXWGhYTnVocVNQNkgrYi9MQy9mVllEalE2b0NJUUR\
HMnVSQ0hsVnEzeWhCNThUWE1VYnpIOCtPbGhXVXZPbFJEM1ZFcURkY1F3PT0ifX0",
  "signatures":[{
    "protected":"eyJ4NWMiOlsiTUlJQmt6Q0NBVGlnQXdJQkFnSUdBV0ZCakNrWU1\
Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RBeEt\
hVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJNQjR\
YRFRFNE1ERXlPVEV3TlRJME1Gb1hEVEk0TURFeU9URXdOVEkwTUZvd1R6RUxNQWtHQTF\
VRUJoTUNRVkV4RlRBVEJnTlZCQW9NREVwcGJtZEthVzVuUTI5eWNERXBNQ2NHQTFVRUF\
3d2dTbWx1WjBwcGJtZERiM0p3SUZadmRXTm9aWElnVTJsbmJtbHVaeUJMWlhrd1dUQVR\
CZ2NxaGtqT1BRSUJCZ2dxaGtqT1BRTUJCd05DQUFTQzZiZUxBbWVxMVZ3NmlRclJzOFI\
wWlcrNGIxR1d5ZG1XczJHQU1GV3diaXRmMm5JWEgzT3FIS1Z1OHMyUnZpQkdOaXZPS0d\
CSEh0QmRpRkVaWnZiN294SXdFREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdDZ1lJS29aSXp\
qMEVBd0lEU1FBd1JnSWhBSTRQWWJ4dHNzSFAyVkh4XC90elVvUVwvU3N5ZEwzMERRSU5\
FdGNOOW1DVFhQQWlFQXZJYjNvK0ZPM0JUbmNMRnNhSlpSQWtkN3pPdXNuXC9cL1pLT2F\
FS2JzVkRpVT0iXSwiYWxnIjoiRVMyNTYifQ",
    "signature":"0TB5lr-cs1jqka2vNbQm3bBYWfLJd8zdVKIoV53eo2YgSITnKKY\
TvHMUw0wx9wdyuNVjNoAgLysNIgEvlcltBw"
  }]
}
```

Figure 55: Example Voucher-Response from MASA

A.4. Example Voucher, MASA issued Voucher with additional Registrar signature (from MASA to Pledge, via Registrar and Registrar-Agent)

The following is an example voucher-response from MASA to Pledge via Registrar and Registrar-Agent, in "General JWS JSON Serialization". The message size of this Voucher is: 2994 bytes

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

{
  "payload": "eyJpZXRmLXZvdWNoZXI6dm91Y2hlciI6eyJhc3NlcnRpb24iOiJhZ2\
VudC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Njc4OSIsIm5vbmNlIj\
oia2hOeUtwTXRoY2NpYTFyWHc0NC92UT09IiwiY3JlYXRlZC1vbiI6IjIwMjQtMDYtMj\
RUMDk6MDI6MTYuMjQ0WiIsInBpbm5lZC1kb21haW4tY2VydCI6Ik1JSUJwRENDQVVtZ0\
F3SUJBZ0lHQVcwZUx1SCtNQW9HQ0NxR1NNNDlCQU1DTURVeEV6QVJCZ05WQkFvTUNrMT\
VRblZ6YVc1bGMzTXhEVEFMQmdOVkJBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUmxjM1\
JEUVRBZUZ3MHhPVEE1TVRFd01qTTNNekphRncweU9UQTVNVEV3TWpNM016SmFNRFV4RX\
pBUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERUQUxCZ05WQkFjTUJGTnBkR1V4RHpBTk\
JnTlZCQU1NQmxSbGMzUkRRVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSU\
FCT2t2a1RIdThRbFQzRkhKMVVhSTcrV3NIT2IwVVMzU0FMdEc1d3VLUURqaWV4MDYvU2\
NZNVBKaWJ2Z0hUQitGL1FUamdlbEhHeTFZS3B3Y05NY3NTeWFqUlRCRE1CSUdBMVVkRX\
dFQi93UUlNQVlCQWY4Q0FRRXdEZ1lEVlIwUEFRSC9CQVFEQWdJRU1CMEdBMVVkRGdRV0\
JCVG9aSU16UWRzRC9qLytnWC83Y0JKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQk\
dBaUVBdHhRMytJTEdCUEl0U2g0YjlXWGhYTnVocVNQNkgrYi9MQy9mVllEalE2b0NJUU\
RHMnVSQ0hsVnEzeWhCNThUWE1VYnpIOCtPbGhXVXZPbFJEM1ZFcURkY1F3PT0ifX0",
  "signatures": [
    {
      "protected": "eyJ4NWMiOlsiTUlJQmt6Q0NBVGlnQXdJQkFnSUdBV0ZCakNr\
WU1Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RB\
```

```
eEthVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJN\
QjRYRFRFNE1ERXlPVEV3TlRJME1Gb1hEVEk0TURFeU9URXdOVEkwTUZvd1R6RUxNQWtH\
QTFVRUJoTUNRVkV4RlRBVEJnTlZCQW9NREVwcGJtZEthVzVuUTI5eWNERXBNQ2NHQTFV\
RUF3d2dTbWx1WjBwcGJtZERRiM0p3SUZadmRXTm9aWElnVTJsbmJtbHVaeUJMWlhrd1dU\
QVRCZ2NxaGtqT1BRSUJCZ2dxaGtqT1BRTUJCd05DQUFTQzZiZUxBbWVxMVZ3NmlRclJz\
OFIwWlcrNGIxR1d5ZG1XczJHQU1GV3diaXRmMm5JWEgzT3FIS1Z1OHMyUnZpQkdOaXZP\
S0dCSEh0QmRpRkVaWnZiN294SXdFREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdDZ1lJS29a\
SXpqMEVBd0lEU1FBd1JnSWhBSTRQWWJ4dHNzSFAyVkh4L3R6VW9RL1NzeWRMMzBEUUlO\
RXRjTjltQ1RYUEFpRUF2SWIzbytGTzNCVG5jTEZzYUpaUkFrZDd6T3Vzbi8vWktPYUVL\
YnNWRGlVPSJdLCJ0eXAiOiJ2b3VjaGVyLWp3cytqc29uIiwiYWxnIjoiRVMyNTYifQ",
      "signature": "SFtc2xqK8xN2KVqkYKJl7EUU8UJAai3VvCuK8LIfH8HZFvrr\
hqGiY8vK5cbQHQCjVcroFLn7IyhH708XAdstAQ"
    },
    {
      "protected": "eyJ4NWMiOlsiTUlJQjRqQ0NBWWlnQXdJQkFnSUdBWFk3MmJi\
Wk1Bb0dDQ3FHU000OUJBTUNNRFV4RXpBUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERU\
QUxCZ05WQkFjTUJGTnBkR1V4RHpBTkJnTlZCQU1NQmxSbGMzUkRRVEFlRncweU1ERXlN\
RGN3TmpFNE1USmFGdzB6TURFeU1EY3dOakU0TVRKYU1ENHhFekFSQmdOVkJBb01DazE1\
UW5WemFXNWxjM014RFRBTEJnTlZCQWNNQkZOcGRHVXhHREFXQmdOVkJBTU1EMFJ2YldG\
cGJsSmxaMmx6ZEhKaGNqQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VIQTBJQUJC\
azE2Sy9pNzlvUmtLNVliZVBnOFVTUjgvdXMxZFBVaVpITXRva1NkcUtXNWZuV3NCZCtx\
Ukw3V1JmZmVXa3lnZWJvSmZJbGx1cmNpMjV3bmhpT1ZDR2plekI1TUIwR0ExVWRKUVFX\
TUJRR0NDc0dBUVVGQndNQkJnZ3JCZ0VGQlFjREhEQU9CZ05WSFE4QkFmOEVCQU1DQjRB\
d1NBWURWUjBSQkVFd1A0SWRjbVZuYVhOMGNtRnlMWFJsYzNRdWMybGxiV1Z1Y3kxaWRD\
NXVaWFNDSG5KbFoybHpkSEpoY2kxMFpYTjBOaTV6YVdWdFpXNXpMV0owTG01bGREQUtC\
Z2dxaGtqT1BRUURBZ05JQURCRkFpQnhsZEJoWnEwRXY1SkwyUHJXQ3R5UzZoRFlXMXlD\

Ty9SYXVicEM3TWFJRGdJaEFMU0piZ0xuZ2hiYkFnMGRjV0ZVVm8vZ0dOMC9qd3pKWjBT\
bDJoNHhJWGsxIl0sInR5cCI6InZvdWNoZXItandzK2pzb24iLCJhbGciOiJFUzI1NiJ9\
",
      "signature": "0Q7_a7L4ahn2vmfSxxkKg1xsOMMc8_D7B_Ilzqv5DKzCMkc7\
8YeeezDsuh4Z5JNVQUYHPp7LsK_AS_WH8TdVzA"
    }
  ]
}
```


Figure 56: Example Voucher-Response from MASA, with additional Registrar signature

Appendix B. HTTP-over-TLS operations between Registrar-Agent and Pledge

The use of HTTP-over-TLS between Registrar-Agent and pledge has been identified as an optional mechanism.

Provided that the key-agreement in the underlying TLS protocol connection can be properly authenticated, the use of TLS provides privacy for the voucher and enrollment operations between the pledge and the Registrar-Agent. The authenticity of the onboarding and enrollment is not dependent upon the security of the TLS connection.

The use of HTTP-over-TLS is not mandated by this document for two main reasons:

1. A certificate is generally required in order to do TLS. While there are other modes of authentication including PSK, various EAP methods, and raw public key, they do not help as there is no previous relationship between the Registrar-Agent and the pledge.

2. The pledge can use its IDevID certificate to authenticate itself, but [RFC9525] DNS-ID methods do not apply, as the pledge does not have a FQDN, and hence cannot be identified by DNS name. Instead a new mechanism is required, which authenticates the X520SerialNumber DN attribute that must be present in every IDevID.

If the Registrar-Agent has a pre-configured list of which product-serial-number(s), from which manufacturers it expects to see, then it can attempt to match this pledge against a list of potential devices.

In many cases only the list of manufacturers is known ahead of time, so at most the Registrar-Agent can show the X520SerialNumber to the (human) operator who may then attempt to confirm that they are standing in front of a device with that product-serial-number. The use of scannable QR codes may help automate this in some cases.

The CA used to sign the IDevID will be a manufacturer private PKI as described in Section 4.1 of [I-D.irtf-t2trg-taxonomy-manufacturer-anchors]. The anchors for this PKI will never be part of the public WebPKI anchors which are distributed with most smartphone operating systems. A Registrar-Agent application will need to use different APIs in order to initiate an HTTPS connection without performing WebPKI verification. The application will then have to do its own certificate chain verification against a store of manufacturer trust anchors. In the Android ecosystem this involves use of a customer TrustManager: many application developers do not create these correctly, and there is significant push to remove this option as it has repeatedly resulted in security failures (see [androidtrustfail]).

Also note that an Extended Key Usage (EKU) for TLS WWW Server authentication cannot be expected in the pledge IDevID certificate. IDevID certificates are intended to be widely usable and EKU does not support that use.

Contributors

Esko Dijk
IoTconsultancy.nl
Email: esko.dijk@iotconsultancy.nl

Toerless Eckert
Futurewei
Email: tte@cs.fau.de

Matthias Kovatsch
Siemens Schweiz AG
Email: ietf@kovatsch.net

Authors' Addresses

Steffen Fries
Siemens AG
Otto-Hahn-Ring 6

81739 Munich
Germany

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

Thomas Werner
Siemens AG
Otto-Hahn-Ring 6
81739 Munich
Germany
Email: thomas-werner@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

Michael C. Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/