Workgroup: ANIMA WG
Internet-Draft: draft-ietf-anima-brski-prm-11
Published: 20 November 2023
Intended Status: Standards Track
Expires: 23 May 2024
Authors: S. Fries T. Werner E. Lear
Siemens Siemens Cisco Systems
M. Richardson
Sandelman Software Works

BRSKI with Pledge in Responder Mode (BRSKI-PRM)

Abstract

This document defines enhancements to Bootstrapping a Remote Secure Key Infrastructure (BRSKI, RFC8995) 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 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. The approach defined here is agnostic to the enrollment protocol that connects the domain registrar to the domain CA.

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 23 May 2024.

Copyright Notice

Table of Contents

1. Introduction

BRSKI as defined in [RFC8995] specifies a solution for secure zero-touch (automated) bootstrapping of devices (pledges) in a (customer site) domain. 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 exchanged and authenticated utilizing voucher-request and voucher-response artifacts as defined in [RFC8995]. Vouchers are signed objects from the Manufacturer's Authorized Signing Authority (MASA). The MASA issues the voucher and provides it via the domain registrar to the pledge. [RFC8366] specifies the format of the voucher artifacts. [I-D.ietf-anima-rfc8366bis] further enhances the format of the voucher artifacts and also the voucher-request.

For the certificate enrollment of devices, BRSKI relies on EST [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 EST server.

BRSKI addresses scenarios in which the 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 to generate bootstrapping requests in the customer domain. For this approach, this document:

* introduces the Registrar-Agent as new component to facilitate the communication between the pledge and the domain registrar. The Registrar-Agent may be implemented as an integrated functionality of a commissioning tool or be co-located with the 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 registrar.

* specifies the interaction (data exchange and data objects) between a pledge acting as server, the Registrar-Agent acting as client, and the domain registrar.

* enables the usage of arbitrary transports between the pledge and the domain registrar via the Registrar-Agent; security is addressed at the application layer, and both IP-based and non-IP connectivity can be used between pledge and Registrar-Agent.

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

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]. For the interaction with the domain registrar, the Registrar-Agent will use existing BRSKI [RFC8995] endpoints as well as additional endpoints defined in this document. To utilize the EST server endpoints on the domain registrar, the Registrar-Agent will act as client toward the registrar.

The Registrar-Agent also acts as a client when communicating with a pledge in responder mode. Here, TLS with server-side, certificate-based authentication is only optionally supported. If TLS is optionally used between the Registrar-Agent and the pledge, the Registrar-Agent needs to identify the pledge based on its product-serial-number rather than the hostname as this is not set in an IDevID certificate.

BRSKI-PRM is designed to rely on object security to support also for alternative transports for which TLS may not be available, e.g., Bluetooth or NFC. This is achieved through an additional signature wrapping of the exchanged data objects involving the Registrar-Agent for transport.

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

There may be pledges which 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.

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document relies on the terminology defined in [RFC8995], section 1.2. The following terms are defined additionally:

authenticated self-contained object: Describes an object, which is cryptographically bound to the end entity (EE) certificate (IDevID certificate or LDEVID 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, issues certificates.

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

CSR: Certificate Signing Request.

EE: End Entity.

endpoint: term equivalent to resource in HTTP [RFC9110] and CoAP [RFC7252]. Endpoints are accessible via .well-known URIs.

mTLS: mutual Transport Layer Security.

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

off-site: Describes a component or service or functionality not available on-site. It may be at a central site or an internet resident "cloud" service. The on-site to off-site connection may also be temporary and, e.g., only available at times when workers are present on a construction side, for instance.

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

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

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

PVR: Pledge Voucher-Request is a request for a voucher sent to the domain registrar. The PVR is signed by the Pledge.

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].

RER:
Registrar Enrollment-Request is the CSR of a PER sent to the CA by the domain registrar (RA).

RVR: Registrar Voucher-Request is a request for a voucher signed by the domain registrar to the MASA. It may contain the PVR received from the pledge.

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.

This protocol unavoidably has a mix of both base64 encoded data (as is normal for many JSON encoded protocols), and also BASE64URL encoded data, as specified by JWS. The latter is indicated by a string like "BASE64URL(content-name)".

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 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 a RA/CA and therefore this service is transferred to a backend that offers a higher level of operational security.

3.2. Limitations

The mechanism described in this document presumes the ability of the pledge and the Registrar-Agent to communicate with 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

Based on the intended target environment described in Section 3.1, the following requirements 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 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.

- The security of communication between the Registrar-Agent and the pledge must 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 [RFC6125] 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 an assertion with value "proximity" in the voucher, as the pledge was not in direct contact with the registrar for bootstrapping. Therefore, a new "agent-proximity" assertion value 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 a data object, e.g., a voucher-request or an enrollment-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 -responses 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.

* Certification 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 credential of the pledge (IDevID). 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. Architecture

5.1. Overview

For BRSKI with pledge in responder mode, 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 based on JOSE [RFC7515] and further specified in Section 6 to address the requirements stated in Section 4 above. In constrained environments it may be provided based on COSE [RFC9052] and [RFC9053].

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

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 a 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, having been replaced by Registrar-Agent. 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 wifi without giving up their LTE 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, CA certificates, or new status information.

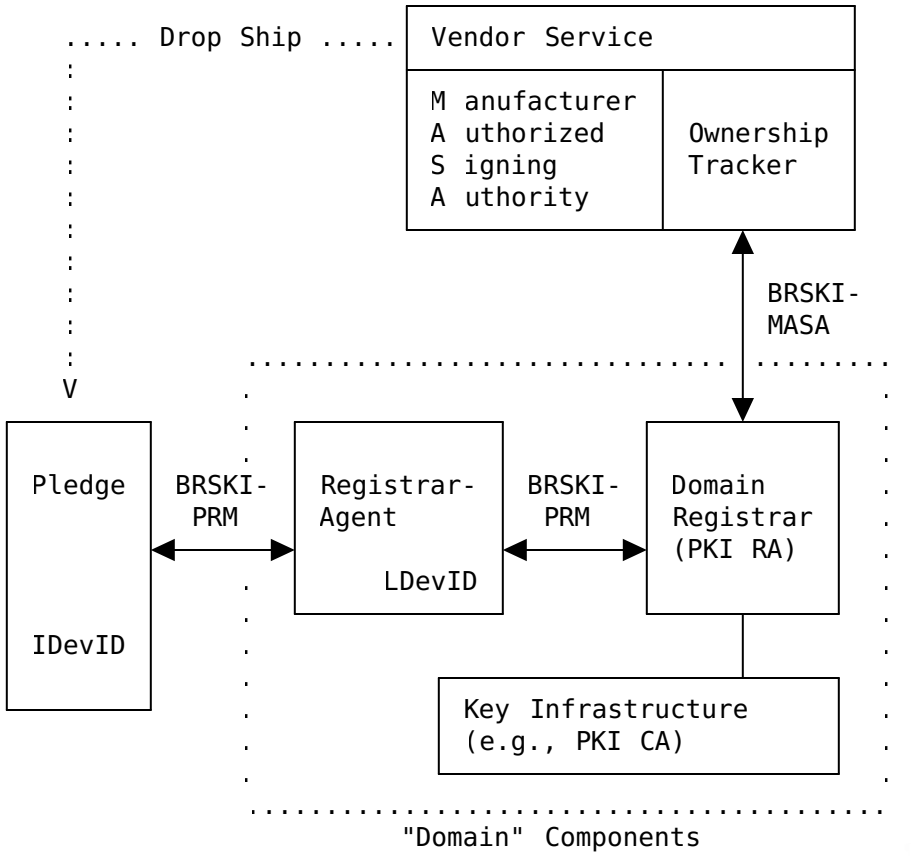

Figure 1: BRSKI-PRM architecture overview using Registrar-Agent

Figure 1 shows the relations between the following main components:

*Pledge: The pledge is expected to respond with the necessary data objects for bootstrapping to the Registrar-Agent. The protocol used between the pledge and the Registrar-Agent is assumed to be HTTP in the context of this document. Any other protocols (including HTTPS) can be used as long as they support the exchange of the necessary data objects. This includes CoAP or protocol to be used over Bluetooth or NFC connections A pledge acting as a server during bootstrapping leads to some differences for BRSKI:

-Discovery of the pledge by the Registrar-Agent **MUST** be possible.

-As the Registrar-Agent **MUST** be able to request data objects for bootstrapping of the pledge, the pledge **MUST** offer corresponding endpoints as defined in Section 5.5.

-The Registrar-Agent **MUST** provide additional data to the pledge in the context of the voucher-request trigger, which the pledge **MUST** include into the PVR as defined in Section 6.1.1 and

Section 6.1.2. This allows the registrar to identify, with which Registrar-Agent the pledge was in contact.

- The order of exchanges in the BRSKI-PRM call flow is different those in BRSKI [RFC8995], as the PVR and PER are collected at once and provided to the registrar. This enables bulk bootstrapping of several devices.

- The data objects utilized for the data exchange between the pledge and the registrar are self-contained authenticated objects (signature-wrapped objects).

* Registrar-Agent: provides a store and forward communication path to exchange data objects between the pledge and the domain registrar. The Registrar-Agent brokers in situations in which the domain registrar is not directly reachable by the pledge. This may be due to a different technology stack or due to missing connectivity. The Registrar-Agent triggers a pledge to create bootstrapping artifacts such as the voucher-request and the enrollment-request on one or multiple pledges. It can then perform a (bulk) bootstrapping based on the collected data. The Registrar-Agent is expected to possess information about the domain registrar: the registrar EE certificate, LDevID(CA) certificate, IP address, either by configuration or by using the discovery mechanism defined in [RFC8995]. 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 registrar. The trust assumption between the Registrar-Agent and the registrar is based on the LDevID of the Registrar-Agent, provided by the PKI responsible for the domain. This allows the Registrar-Agent to authenticate towards the registrar, e.g., in a TLS handshake. Based on this, the registrar is able to distinguish a pledge from a Registrar-Agent during the TLS session establishment and also to verify that this Registrar-Agent is authorized to perform the bootstrapping of the distinct pledge. The Registrar-Agent may be realized as stand-alone component supporting nomadic activities of a service technician moving between different installation sites. Contrary, the Registrar-Agent may also be realized as co-located functionality for a registrar, to support pledges in pledge-responder-mode.

* Join Proxy (not shown): 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, the domain registrar fulfills the same functionality regarding the bootstrapping of the pledge in a (customer site) domain by facilitating the communication of the pledge with the MASA service and the domain PKI service. In contrast to [RFC8995], the domain registrar does not interact with a pledge directly but through the Registrar-Agent. A registrar with combined functionality of BRSKI and BRSKI-PRM detects if the bootstrapping is performed by the pledge directly (BRSKI case) or by the Registrar-Agent (BRSKI-PRM case) based on the utilized credential for authentication (either pledge's IDevID or LDevID from Registrar-Agent), see also Section 6.2.

* The manufacturer provided components/services (MASA and Ownership tracker) are used as defined in [RFC8995]. A MASA is able to support enrollment via Registrar-Agent without changes unless it checks the vouchers proximity indication, in which case it would need to be enhanced to support BRSKI-PRM to also accept the agent-proximity.

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 (Section 3.1.1).

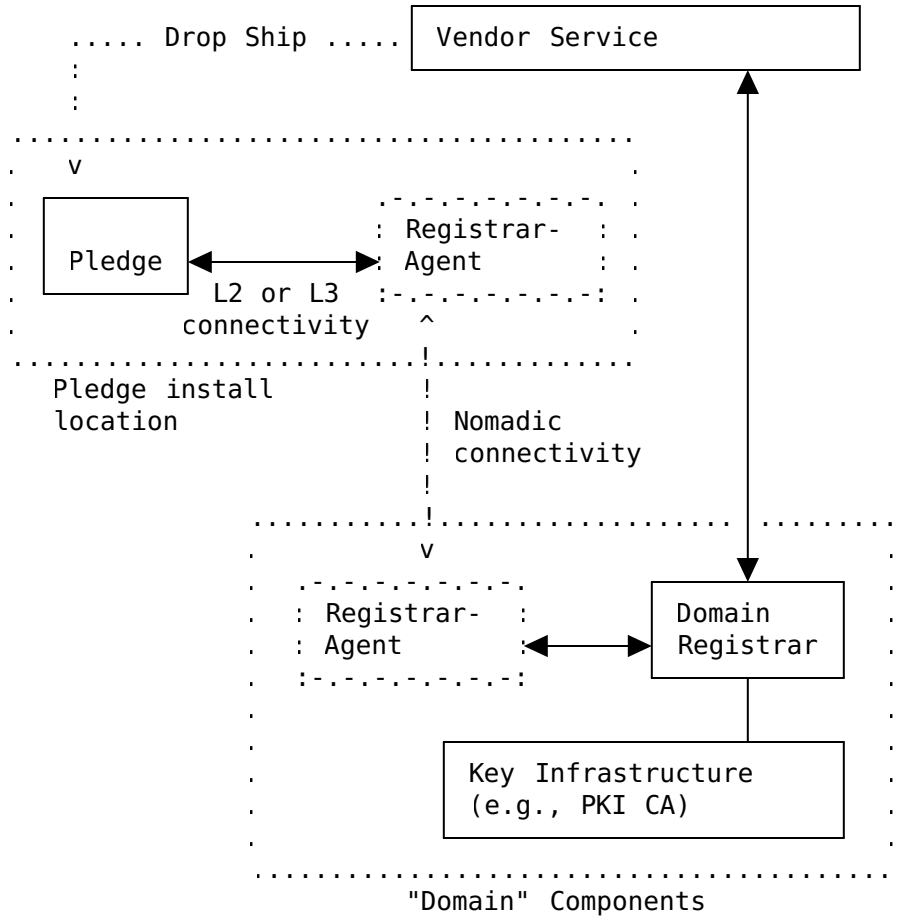

Figure 2: Registrar-Agent nomadic connectivity example

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 install 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 install location: retrieve information about available pledges (IDevID), collect request objects like voucher-requests and enrollment-requests using the BRSKI-PRM approach described in Section 6.1.

3. Connectivity to domain registrar, submit collected pledges' request information, retrieve response objects as voucher and enrollment information using the BRSKI-PRM approach described in Section 6.2.

4. Connectivity to pledge install location, provide retrieved objects to the pledge to enroll pledges and collect status using the BRSKI-PRM approach described in Section 6.3.

5. Connectivity to domain registrar, submit voucher status and enrollment status using the BRSKI-PRM approach described in Section 6.3.6.

Variations of this setup include cases where the Registrar-Agent uses for example WiFi 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. Registrar-Agent co-located with 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 request enrollment from a domain registrar, which may result in undesirable communications pattern and possible overload of a domain registrar.

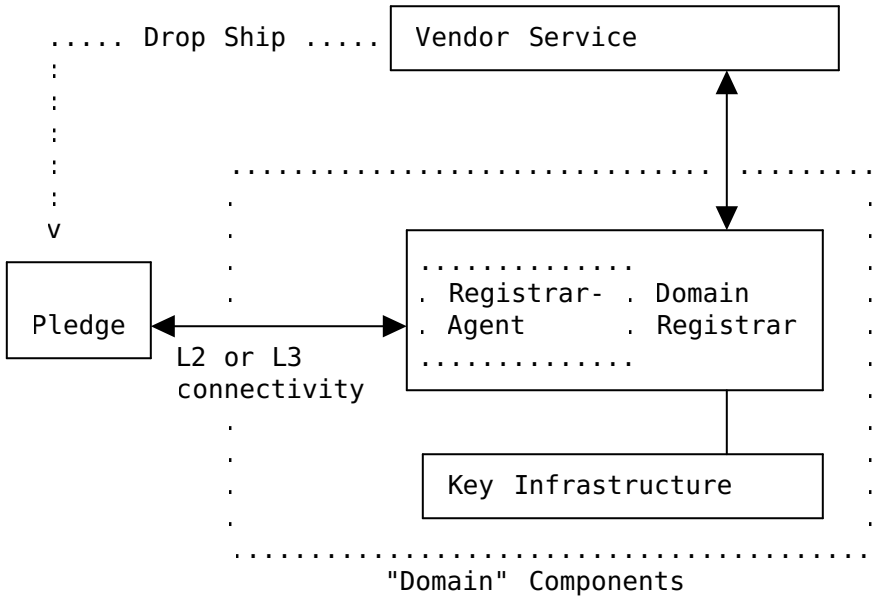

Figure 3: Registrar-Agent integrated into Domain Registrar example

The benefits of BRSKI-PRM can be achieved even without the operational complexity of standalone 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.

5.4. Agent-Proximity Assertion

"Agent-proximity" is a statement in the PVR and in the voucher, that the registrar certificate was provided via the Registrar-Agent as defined in Section 6 and not directly to the pledge. "Agent-proximity" is therefore a different assertion then "proximity", which is defined in section 4 of [RFC8366]. "Agent-proximity" is defined as additional assertion type in [I-D.ietf-anima-rfc8366bis]. This can be verified by the registrar and also by the MASA during the voucher-request processing.

In BRSKI, the pledge verifies POP of the registrar 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 connection is considered provisional until the pledge receives the voucher.

In contrast, in BRSKI-PRM, the pledge has no direct connection to the registrar and must take the Registrar-Agent LDevID provisionally. The Registrar-Agent has included its LDevID, a certificate signed by the domain owner, into the PVR trigger message. The Registrar-Agent identity is therefore included into the Pledge Voucher Request (PVR).

Akin to the BRSKI case, the pledge has provided proximity evidence to the MASA. But additionally, this allows the Registrar to be sure that the PVR collected by the Registrar-Agent was in fact collected by the Registrar-Agent to which the Registrar is connected to.

In a similar fashion, the pledge accepts the registrar certificate provisionally until it receives the voucher as described in Section 6.3. See also Section 5 of [RFC8995] on "PROVISIONAL accept of server cert".

5.5. Behavior of Pledge in Pledge-Responder-Mode

The pledge is triggered by the Registrar-Agent to generate the PVR and PER. It will also be 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. Due to the use of the Registrar-Agent, the interaction with the domain registrar is changed as shown in Figure 5. To enable interaction as responder with the Registrar-Agent, the pledge provides endpoints using the BRSKI defined endpoints based on the "/.well-known/brski" URI tree.

When the Registrar-Agent reaches out to a pledge, for instance with an example URI path "http://pledge.example/.well-known/brski/tpvr", it will in fact send a Host: header of "pledge.example", with a relative path of "/.well-known/brski/tpbr". However in practice the pledge will often be known only by its IP address as returned by a discovery protocol, and that is what will be present in the Host: header.

The pledge **MUST** respond to all queries regardless of what Host: header is provided by the client. [RFC9110], Section 7.2 makes the Host: header mandatory, so it will always be present.

The following operations are defined for the *pledge* in this document, describing their endpoints and their corresponding URIs. The endpoints are defined with short names to also accommodate for the constraint case.

Operation	Endpoint	Details
Trigger pledge voucher-request creation - Returns PVR	/tpvr	Section 6.1
Trigger pledge enrollment-request - Returns PER	/tper	Section 6.1
Supply voucher to pledge - Returns pledge voucher-status	/svr	Section 6.3
Supply enrollment-response to pledge - Returns pledge enrollment-status	/ser	Section 6.3
Supply CA certs to pledge	/scac	Section 6.3
Query status of pledge - Returns pledge-status	/qps	Section 6.4

Table 1: Endpoints on the pledge

5.6. Behavior of Registrar-Agent

The Registrar-Agent as a new component provides a message passing service between the pledge and the domain registrar. It facilitates the exchange of data between the pledge and the domain registrar, which are the voucher-request/response, the enrollment-request/response, as well as related telemetry and status information.

For the communication with the pledge the Registrar-Agent utilizes communication endpoints provided by the pledge. The transport in this specification is based on HTTP but may also be done using other transport mechanisms.

The communication between the Registrar-Agent and the pledge **MAY** be protected using TLS as outlined in Section 6.1. The details of doing TLS validation are Appendix B.

For the communication with the registrar, the Registrar-Agent uses the endpoints of the domain registrar side already specified in [RFC8995] (derived from EST [RFC7030]) where suitable. These endpoints do not expect signature wrapped-objects, which are used b BRSKI-PRM. This specifically applies for the enrollment-request and the provisioning of CA certificates. To accommodate the use of signature-wrapped object, the following additional endpoints are defined for the *registrar*. Operations and their corresponding URIs:

Operation	Endpoint	Details
Supply PER to registrar - Returns enrollment-response	/requestenroll	Section 6.2.6
Request (wrapped) CA certificates - Returns wrapped CA Certificates	/ wrappedcacerts	Section 6.2.7

Table 2: Additional endpoints on the registrar

For authentication to the domain registrar, the Registrar-Agent uses its EE (RegAgt) certificate. The provisioning of the Registrar-Agent LDevID is out of scope for this document, but may be done in advance using a separate BRSKI run or by other means like configuration. It is recommended to use short lived Registrar-Agent LDevIDs in the range of days or weeks as outlined in Section 10.3.

The Registrar-Agent will use its EE certificate when establishing a TLS session with the domain registrar for TLS client authentication. The EE (RegAgt) certificate **MUST** include a SubjectKeyIdentifier (SKID), which is used as reference in the context of an agent-signed-data object as defined in Section 6.1. Note that this is an additional requirement for issuing the certificate, as [IEEE-802.1AR] only requires the SKID to be included for intermediate CA certificates. [RFC8995] makes a similar requirement. In BRSKI-PRM, the SKID is used in favor of providing the complete EE (RegAgt) certificate to accommodate also constraint environments and reduce bandwidth needed for communication with the pledge. In addition, it follows the recommendation from BRSKI to use SKID in favor of a certificate fingerprint to avoid additional computations.

Using an LDevID for TLS client authentication of the Registrar-Agent is a deviation from [RFC8995], in which the pledge's IDevID credential is used to perform TLS client authentication. The use of the EE (RegAgt) certificate allows the domain registrar to distinguish, if bootstrapping is initiated from a pledge or from a Registrar-Agent and to adopt different internal handling accordingly. If a registrar detects a request that originates from a Registrar-Agent it is able to 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 EE (RegAgt) certificate. Alternatively, the domain may feature a CA specifically for issuing

Registrar-Agent LDevID certificates. This allows the registrar to detect Registrar-Agents based on the issuing CA.

As BRSKI-PRM uses authenticated self-contained data objects between the pledge and the domain registrar, the binding of the pledge identity to the requests is provided by the data object signature employing the pledge's IDevID. The objects exchanged between the pledge and the domain registrar used in the context of this specifications are JOSE objects.

In addition to the EE (RegAgt) certificate, the Registrar-Agent is provided with the product-serial-number(s) of the pledge(s) to be bootstrapped. This is necessary to allow the discovery of pledge(s) by the Registrar-Agent using DNS-SD with mDNS (see Section 5.6.2) The list may be provided by prior administrative means or the Registrar-Agent may get the information via an interaction with the pledge. For instance, [RFC9238] describes scanning of a QR code, the product-serial-number would be initialized from the 12N B005 Product Serial Number.

According to [RFC8995] section 5.3, the domain registrar performs the pledge authorization for bootstrapping within his domain based on the pledge voucher-request object. This behavior is retained also in BRSKI-PRM.

The following information **MUST** be available at the Registrar-Agent before interaction with a pledge:

* EE (RegAgt) certificate and corresponding private key: own operational key pair (to 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 (to query discovery of specific pledges based on the product-serial-number).

5.6.1. Discovery of Registrar by Registrar-Agent

As a Registrar-Agent acts as representative of the domain registrar towards the pledge or may even be collocated with the domain registrar, a separate discovery of the registrar is likely not needed as Registrar-Agent and registrar are domain components and have a trust relation. Moreover, 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, as the standard discovery described in section 4 and the appendix A.2 of [RFC8995] does not support of registrars with an enhanced feature set

(like the support of BRSKI-PRM), this 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. This may be provided in [I-D.eckert-anima-brski-discovery].

5.6.2. Discovery of Pledge by Registrar-Agent

The discovery of the pledge by Registrar-Agent in the context of this document describes the minimum discovery approach to be supported. A more general discovery mechanism, also supporting GRASP besides DNS-SD with mDNS may be provided in [I-D.eckert-anima-brski-discovery].

Discovery in BRSKI-PRM uses DNS-based Service Discovery [RFC6763] over Multicast DNS [RFC6762] to discover the pledge. Note that [RFC6762] Section 9 provides support for conflict resolution in situations when an DNS-SD with mDNS responder receives a 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 local host name based on device local information (product-serial-number), which results in "product-serial-number._brski-pledge._tcp.local". The product-serial-number composition is 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.

In the absence of a more general discovery as defined in [I-D.eckert-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 his product-serial-number contained. This allows a commissioning tool to discover pledges to be bootstrapped in the domain. The manufacturer support this functionality as outlined in Section 10.4.

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 WiFi 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.

5.7. Behavior of Pledge with Combined Functionality

Pledges **MAY** support both initiator or 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-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 enrollment-request (PER) or to provide an enrollment-response can be used for further certificates.

6. Bootstrapping Data Objects and Corresponding Exchanges

The interaction of the pledge with the Registrar-Agent may be accomplished using different transport means (protocols and/or network technologies). This specification utilizes HTTP as transport. Alternative transport channels may be CoAP, Bluetooth Low Energy (BLE), or Nearfield Communication (NFC). These transport means may differ from, and are independent of, the ones used between the Registrar-Agent and the registrar. Transport channel independence is realized by data objects, which are not bound to specific transport security and stay the same across the communication from the pledge via the Registrar-Agent to the registrar.. Therefore, authenticated self-contained objects (here: signature-wrapped objects) are applied for data exchanges between the pledge and the registrar.

The Registrar-Agent provides the domain registrar certificate (registrar LDevID certificate) to the pledge to be included in the PVR leaf "agent-provided-proximity-registrar-certificate". This enables the registrar to verify that it is the desired registrar for handling the request.

The registrar certificate may be configured at the Registrar-Agent or may be fetched by the Registrar-Agent based on a prior TLS connection with this domain registrar. In addition, the Registrar-Agent provides agent-signed-data containing the pledge product-serial-number, signed

with the private key corresponding to the EE (RegAgt) certificate, as described in Section 6.1. This enables the registrar to verify and log, which Registrar-Agent was in contact with the pledge, when verifying the PVR.

The registrar **MUST** provide the EE (RegAgt) certificate identified by the SubjectKeyIdentifier (SKID) in the header of the agent-signed-data from the PVR in its RVR (see also Section 6.2.2.

The MASA in turn verifies the registrar LDevID certificate is included in the PVR (contained in the "prior-signed-voucher-request" field of RVR) in the "agent-provided-proximity-registrar-certificate" leaf and may assert the PVR as "verified" or "logged".

In addition, the MASA **MAY** issue the assertion "agent-proximity" as follows: The MASA verifies the signature of the agent-signed-data contained in the prior-signed-voucher-request, based on the provided EE (RegAgt) certificate in the "agent-sign-cert" leaf of the RVR. If both can be verified successfully, the MASA can assert "agent-proximity" in the voucher. The assertion of "agent-proximity" is similar to the proximity assertion by the MASA when using BRSKI. Note that the different assertions do not provide a metric of strength as the security properties are not comparable.

Depending on the MASA verification policy, it may also respond with a suitable 4xx or 5xx error status code as described in section 5.6 of [RFC8995]. When successful, the voucher will then be supplied via the registrar to the Registrar-Agent.

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

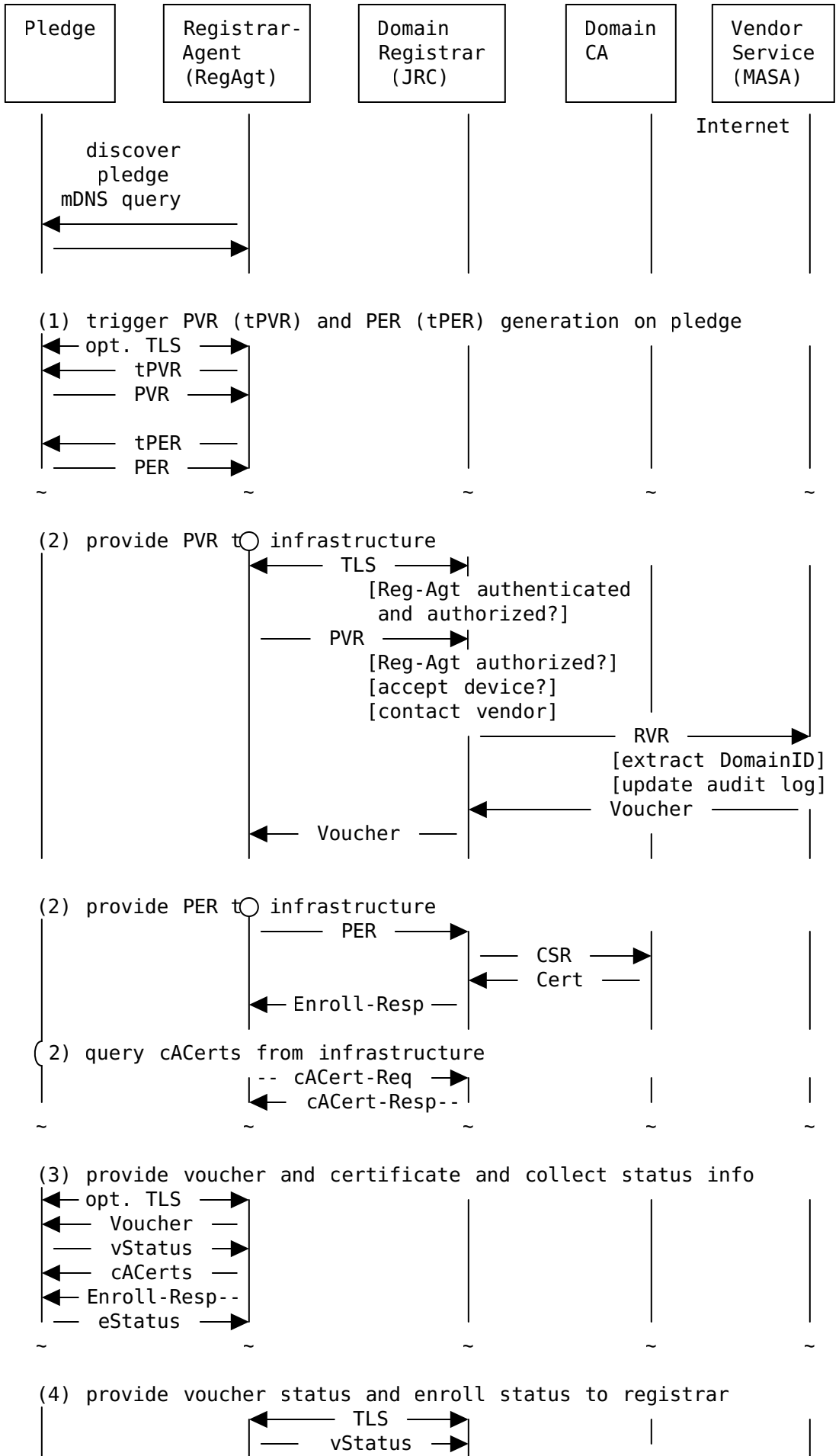
Pledge
Registrar-Agent (RegAgt)
Domain Registrar (JRC)
Domain CA
Vendor Service (MASA)
Internet
discover pledge mDNS query
(1) trigger PVR (tPVR) and PER (tPER) generation on pledge
opt. TLS
tPVR
PVR
tPER
PER
(2) provide PVR to infrastructure
TLS
[Reg-Agt authenticated and authorized?]
PVR
[Reg-Agt authorized?]
[accept device?]
[contact vendor]
RVR
[extract DomainID]
[update audit log]
Voucher
Voucher
(2) provide PER to infrastructure
PER
CSR
Cert
Enroll-Resp
(2) query cACerts from infrastructure
-- cACert-Req
cACert-Resp--
(3) provide voucher and certificate and collect status info
opt. TLS
Voucher
vStatus
cACerts
Enroll-Resp--
eStatus
(4) provide voucher status and enroll status to registrar
TLS
vStatus

Figure 4: Overview pledge-responder-mode exchanges

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

1. Section 6.1 describes the request object acquisition by the Registrar-Agent from pledge.

2. Section 6.2 describes the request object processing initiated by the Registrar-Agent to the registrar and also the interaction of the registrar with the MASA and the domain CA including the response object processing by these entities.

3. Section 6.3 describes the supply of response objects between the Registrar-Agent and the pledge including the status information.

4. Section 6.4 describes the general status handling and addresses corresponding exchanges between the Registrar-Agent and the registrar.

6.1. Request Objects Acquisition by Registrar-Agent from Pledge

The following description assumes that the Registrar-Agent has already discovered the pledge. This may be done as described in Section 5.6.2 and Figure 4 based on DNS-SD or similar.

The focus is on the exchange of signature-wrapped objects using endpoints defined for the pledge in Section 5.5.

Preconditions:

*Pledge: possesses IDevID

*Registrar-Agent:

-possesses own credentials (EE (RegAgt) certificate and corresponding private key) for the registrar domain.

-**MAY** possess the IDevID CA certificate of the pledge vendor/ manufacturer to validate IDevID certificate on returned PVR or in case of TLS usage for pledge communication. The distribution of IDevID CA certificates to the Registrar-Agent is out of scope of this document and may be done by a manual configuration. In addition, the Registrar-Agent **SHOULD** know the product-serial-number(s) of the pledge(s) to be bootstrapped. The Registrar-Agent **MAY** be provided with the product-serial-number(s) in different ways:

oconfigured, e.g., as a list of pledges to be bootstrapped via QR code scanning

odiscovered by using standard approaches like DNS-SD with mDNS as described in Section 5.6.2

odiscovered by using a manufacturer specific approach, e.g., RF beacons. If the product-serial-number(s) are not known in advance, the Registrar-Agent cannot perform a distinct triggering of pledges but and triggers all pledges discovered .

The Registrar-Agent **SHOULD** have synchronized time.

*Registrar (same as in BRSKI): possesses/trusts IDevID CA certificate and has own registrar EE credentials.

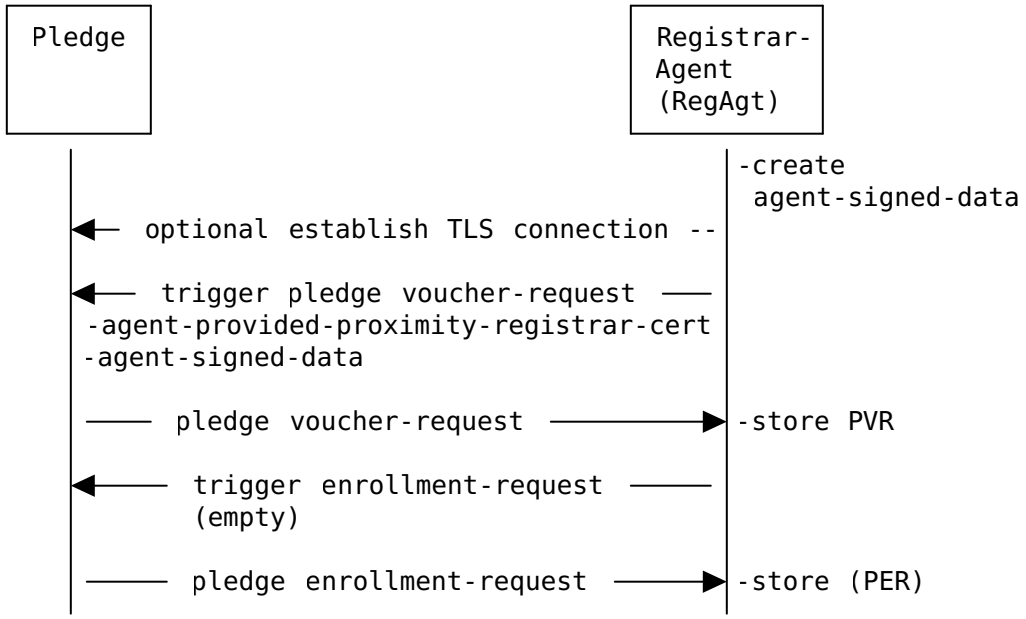

Figure 5: Request collection (Registrar-Agent - pledge)

TLS **MAY** be optionally used to provide privacy for the interaction between the Registrar-Agent and the pledge, see Appendix B.

Note: The Registrar-Agent may trigger the pledge for the PVR or the PER or both. It is expected that this will be aligned with a service technician workflow, visiting and installing each pledge.

6.1.1. Pledge-Voucher-Request (PVR) - Trigger

Triggering the pledge to create the PVR is done using HTTP POST on the defined pledge endpoint: "/.well-known/brski/tpvr"

The Registrar-Agent PVR trigger Content-Type header is: application/json. Following parameters are provided in the JSON object:

- *agent-provided-proximity-registrar-cert: base64-encoded registrar EE TLS certificate.

- *agent-signed-data: base64-encoded JSON-in-JWS object.

The trigger for the pledge to create a PVR is depicted in the following figure:

```
{
  "agent-provided-proximity-registrar-cert": "base64encodedvalue==",
  "agent-signed-data": "base64encodedvalue=="
}
```

Figure 6: Representation of trigger to create PVR

Note that at the time of receiving the PVR trigger, the pledge cannot verify the registrar LDevID certificate and has no proof-of-possession of the corresponding private key for the certificate. The pledge therefore accepts the registrar LDevID certificate provisionally until it receives the voucher as described in Section 6.3.

The pledge will also be unable to verify the agent-signed-data itself as it does not possess the EE (RegAgt) certificate and the domain trust has not been established at this point of the communication. Verification **SHOULD** be done, after the voucher has been received.

The agent-signed-data is a JSON-in-JWS object and contains the following information:

The header of the agent-signed-data contains:

- *alg: algorithm used for creating the object signature.

- *kid: **MUST** contain the base64-encoded bytes of the SubjectKeyIdentifier (the "KeyIdentifier" OCTET STRING value) of the EE (RegAgt) certificate.

The body of the agent-signed-data contains an "ietf-voucher-request:agent-signed-data" element (defined in [I-D.ietf-anima-rfc8366bis]):

- *created-on: **MUST** contain the creation date and time in yang:date-and-time format.

*serial-number: **MUST** contain the product-serial-number as type string as defined in [RFC8995], section 2.3.1. The serial-number corresponds with the product-serial-number contained in the X520SerialNumber field of the IDevID certificate of the pledge.

```
# The agent-signed-data in General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-voucher-request-prm:agent-signed-data)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload representation in JSON syntax of
  "ietf-voucher-request-prm:agent-signed-data"

"ietf-voucher-request-prm:agent-signed-data": {
  "created-on": "2021-04-16T00:00:01.000Z",
  "serial-number": "callee4711"
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "kid": "base64encodedvalue=="
}
```

Figure 7: Representation of agent-signed-data in General JWS Serialization syntax

6.1.2. Pledge-Voucher-Request (PVR) - Response

Upon receiving the voucher-request trigger, the pledge **SHALL** construct the body of the PVR as defined in [RFC8995]. It will contain additional information provided by the Registrar-Agent as specified in the following. This PVR becomes a JSON-in-JWS object as defined in [I-D.ietf-anima-jws-voucher]. If the pledge is unable to construct the PVR it **SHOULD** respond with a HTTP error code to the Registrar-Agent to indicate that it is not able to create the PVR.

The following client error responses **MAY** be used:

*400 Bad Request: if the pledge detected 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 PVR media type was set to application/json.

*403 Forbidden: if the pledge detected that one or more security parameters from the trigger message to create the PVR were not valid, e.g., the LDevID (Reg) certificate.

The header of the PVR **SHALL** contain the following parameters as defined in [RFC7515] to support JWS signing of the object:

*alg: algorithm used for creating the object signature.

*x5c: contains the base64-encoded pledge IDevID certificate. It **MAY** optionally contain the certificate chain for this certificate. If the certificate chain is not included it **MUST** be available at the registrar for verification of the IDevID certificate.

The payload of the PVR **MUST** contain the following parameters as part of the ietf-voucher-request-prm:voucher as defined in [RFC8995]:

*created-on: **SHALL** contain the current date and time in yang:date-and-time format. If the pledge does not have synchronized time, it **SHALL** use the created-on time from the agent-signed-data, received in the trigger to create a PVR.

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

*serial-number: **SHALL** contain the pledge product-serial-number as X520SerialNumber.

*assertion: **SHALL** contain the requested voucher assertion "agent-proximity" (different value as in RFC 8995)..

The ietf-voucher-request:voucher is extended with additional parameters:

*agent-provided-proximity-registrar-cert: **MUST** be included and contains the base64-encoded registrar LDevID certificate (provided as PVR trigger parameter by the Registrar-Agent).

*agent-signed-data: **MUST** contain the base64-encoded agent-signed-data (as defined in Figure 7) and provided as a PVR trigger parameter by the Registrar-Agent.

The enhancements of the YANG module for the ietf-voucher-request with these new leaves are defined in [I-D.ietf-anima-rfc8366bis].

The PVR is signed using the pledge's IDevID credential contained as x5c parameter of the JOSE header.

```
# The PVR in General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-voucher-request-prm:voucher)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded Payload "ietf-voucher-request-prm:voucher"
  representation in JSON syntax
"ietf-voucher-request-prm:voucher": {
   "created-on": "2021-04-16T00:00:02.000Z",
   "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
   "serial-number": "callee4711",
   "assertion": "agent-proximity",
   "agent-provided-proximity-registrar-cert": "base64encodedvalue==",
   "agent-signed-data": "base64encodedvalue=="
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
    "alg": "ES256",
    "x5c": [
      "base64encodedvalue==",
      "base64encodedvalue=="
    ],
    "typ": "voucher-jws+json"
}
```

Figure 8: Representation of PVR

The PVR Media-Type is defined in [I-D.ietf-anima-jws-voucher] as application/voucher-jws+json.

The pledge **MUST** include this Media-Type header field indicating the included media type for the PVR. The PVR is included by the registrar in its RCR as described in Section 6.2.

6.1.3. Pledge-Enrollment-Request (PER) - Trigger

Once the Registrar-Agent has received the PVR it can trigger the pledge to generate a PER. As in BRSKI the PER contains a PKCS#10, but additionally signed using the pledge's IDevID. Note, as the initial enrollment aims to request a generic certificate, no certificate attributes are provided to the pledge.

Triggering the pledge to create the enrollment-request is done using HTTP POST on the defined pledge endpoint: "/.well-known/brski/tper"

The Registrar-Agent PER trigger Content-Type header is: application/json with an empty body by default. Note that using HTTP POST allows for an empty body, but also to provide additional data, like CSR attributes or information about the enroll type "enroll-generic-cert" or "re-enroll-generic-cert". The "enroll-generic-cert" case is shown in Figure 9.

```
{
  "enroll-type" : "enroll-generic-cert"
}
```

Figure 9: Example of trigger to create a PER

This document specifies the request of a generic 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 RA/CA. How the HTTP POST can be used to provide CSR attributes is out of scope for this specification.

6.1.4. Pledge-Enrollment-Request (PER) - Response

In the following the enrollment is described as initial enrollment with an empty HTTP POST body.

Upon receiving the PER trigger, the pledge **SHALL** construct the PER as authenticated self-contained object. 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, POI is provided. Here, a JOSE object is being created in which the body utilizes the YANG module ietf-ztp-types with the grouping for csr-grouping for the CSR as defined in [I-D.ietf-netconf-sztp-csr].

Depending on the capability of the pledge, it constructs the pledge enrollment-request (PER) as plain PKCS#10. Note, the focus in this use case is placed on PKCS#10 as PKCS#10 can be transmitted in different enrollment protocols in the infrastructure like EST, CMP, CMS, and SCEP. If the pledge has already implemented an enrollment protocol, it may leverage that functionality for the creation of the CSR. Note, [I-D.ietf-netconf-sztp-csr] also allows for inclusion of certification requests in different formats used by CMP or CMC.

The pledge **MUST** construct the PER as PKCS#10. In BRSKI-PRM it **MUST** sign it additionally with its IDevID credentials to provide proof-of-identity bound to the PKCS#10 as described below.

If the pledge is unable to construct the PER it **SHOULD** respond with a HTTP 4xx/5xx error code to the Registrar-Agent to indicate that it is not able to create the PER.

The following 4xx client error codes **MAY** be used:

* 400 Bad Request: if the pledge detected an error in the format of the request or detected invalid JSON even though the PER media type was set to application/json.

* 403 Forbidden: if the pledge detected that one or more security parameters (if provided) from the trigger message to create the PER are not valid.

* 406 Not Acceptable: if the request's Accept header indicates a type that is unknown or unsupported. For example, a type other than application/jose+json.

* 415 Unsupported Media Type: if the request's Content-Type header indicates a type that is unknown or unsupported. For example, a type other than 'application/json'.

A successful enrollment will result in a generic LDevID certificate for the pledge in the new domain, which can be used to request further (application specific) LDevID certificates if necessary for operation. The Registrar-Agent **SHALL** use the endpoints specified in this document.

[I-D.ietf-netconf-sztp-csr] considers PKCS#10 but also CMP and CMC as certification request format. Note that the wrapping of the PER signature is only necessary for plain PKCS#10 as other request formats like CMP and CMS support the signature wrapping as part of their own certificate request format.

The Registrar-Agent enrollment-request Content-Type header for a signature-wrapped PKCS#10 is: application/jose+json

The header of the pledge enrollment-request **SHALL** contain the following parameter as defined in [RFC7515]:

* alg: algorithm used for creating the object signature.

* x5c: contains the base64-encoded pledge IDevID certificate. It **MAY** optionally contain the certificate chain for this certificate. If the certificate chain is not included it **MUST** be available at the registrar for verification of the IDevID certificate. The body of the pledge enrollment-request **SHOULD** contain a P10 parameter (for PKCS#10) as defined for ietf-ztp-types:p10-csr in [I-D.ietf-netconf-sztp-csr]:

*P10: contains the base64-encoded PKCS#10 of the pledge.

The JOSE object is signed using the pledge's IDevID credential, which corresponds to the certificate signaled in the JOSE header.

While BRSKI-PRM targets the initial enrollment, re-enrollment **SHOULD** be supported as described in a similar way as for enrollment in this document, if no other re-enrollment mechanism is supported. Note that in this case the current LDevID credential is used instead of the IDevID credential to create the signature of the PKCS#10 request.

```
# The PER in General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-ztp-types)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded Payload "ietf-ztp-types" Representation
  in JSON Syntax
"ietf-ztp-types": {
  "p10-csr": "base64encodedvalue=="
}

# Example: Decoded "JWS Protected Header" Representation
  in JSON Syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "crit":["created-on"],
  "created-on": "2022-09-13T00:00:02.000Z"
}
```

Figure 10: Representation of PER

With the collected PVR and PER, the Registrar-Agent starts the interaction with the domain registrar.

The new protected header field "created-on" is introduced to reflect freshness of the PER. The field is marked critical "crit" to ensure that it must be understood and validated by the receiver (here the domain registrar) according to section 4.1.11 of [RFC7515]. It allows

the registrar to verify the timely correlation between the PER and previously exchanged messages, i.e., created-on of PER >= created-on of PVR >= created-on of PVR trigger. The registrar **MAY** consider to ignore any but the newest PER from the same pledge in the case the registrar has at any point in time more than one pending PER from the pledge.

As the Registrar-Agent is intended to facilitate communication between the pledge and the domain registrar, a collection of requests from more than one pledge is possible. This allows bulk bootstrapping of several pledges using the same connection between the Registrar-Agent and the domain registrar.

6.2. Request Object Handling initiated by the Registrar-Agent on Registrar, MASA and Domain CA

The BRSKI-PRM bootstrapping exchanges between Registrar-Agent and domain registrar resemble the BRSKI exchanges between pledge and domain registrar (pledge-initiator-mode) with some deviations.

Preconditions:

* Registrar-Agent: possesses its own credentials (EE (RegAgt) certificate and corresponding private key) of the domain. In addition, it **MAY** possess the IDevID CA certificate of the pledge vendor/manufacturer to verify the pledge certificate in the received request messages. It has the address of the domain registrar through configuration or by discovery, e.g., DNS-SD with mDNS. The Registrar-Agent has acquired one or more PVR and PER objects.

* Registrar (same as in BRSKI): possesses the IDevID CA certificate of the pledge vendor/manufacturer and its own registrar EE credentials of the domain.

* MASA (same as in BRSKI): possesses its own vendor/manufacturer credentials (voucher signing key and certificate, TLS server certificate and private key) related to pledges IDevID and **MAY** possess the site-specific domain CA certificate.

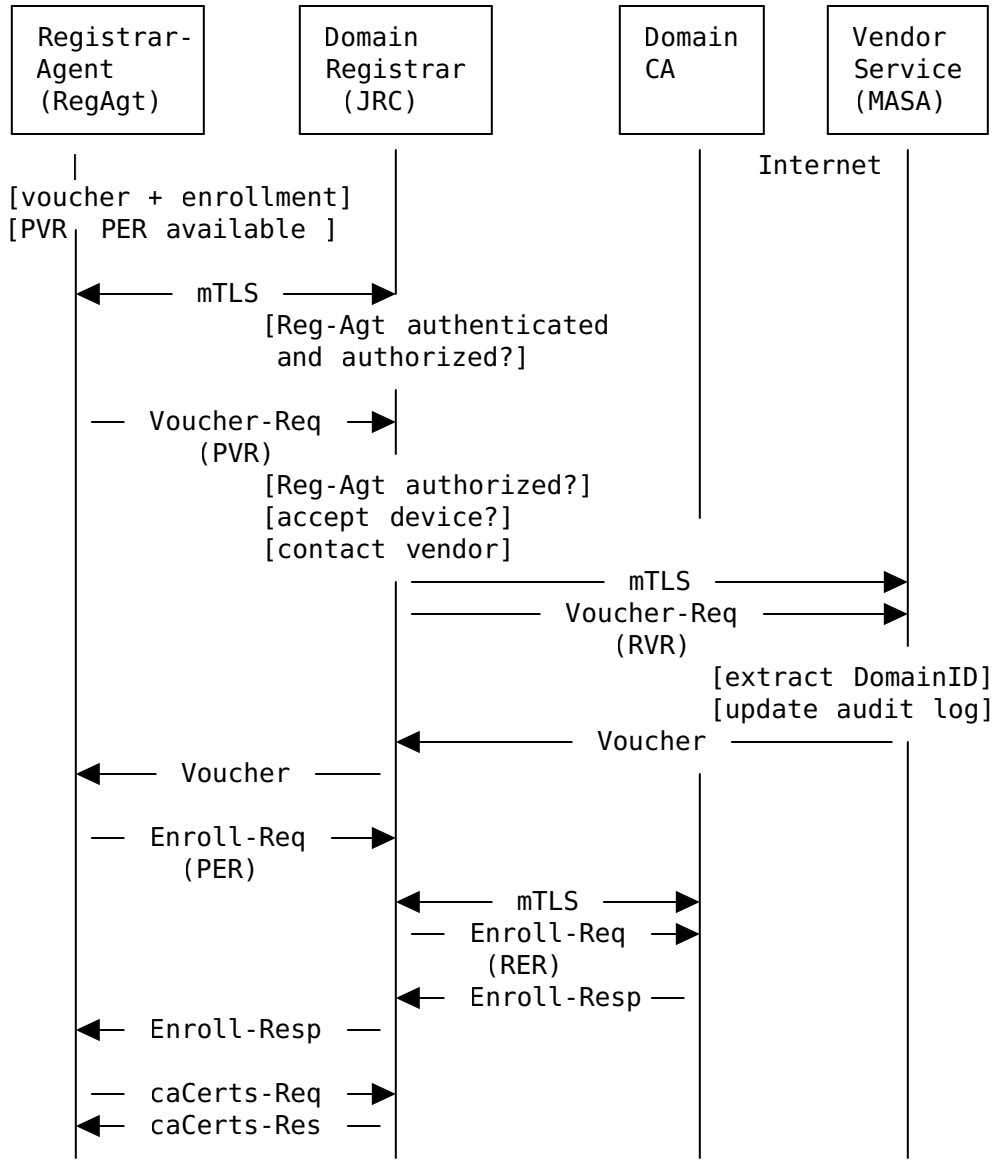

Figure 11: Request processing between Registrar-Agent and bootstrapping services

The Registrar-Agent establishes a TLS connection to the registrar. 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.

6.2.1. Connection Establishment (Registrar-Agent to Registrar)

In contrast to BRSKI [RFC8995] TLS client authentication to the registrar is achieved by using Registrar-Agent EE credentials instead

of pledge IDevID credentials. Consequently BRSKI (pledge-initiator-mode) is distinguishable from BRSKI-PRM (pledge-responder-mode) by the registrar. The registrar **SHOULD** verify that the Registrar-Agent is authorized to establish a connection to the registrar based on the TLS client authentication. If the connection from Registrar-Agent to registrar is established, the authorization **SHOULD** be verified again based on agent-signed-data contained in the PVR. This ensures that the pledge has been triggered by an authorized Registrar-Agent.

With BRSKI-PRM, the pledge generates PVR and PER as JSON-in-JWS objects and the Registrar-Agent forwards them to the registrar. In [RFC8995], the pledge generates PVR as CMS-signed JSON and PER as PKCS#10 or PKCS#7 according to [RFC7030] and inherited by [RFC8995].

For BRSKI-PRM, the Registrar-Agent sends the PVR by HTTP POST to the same registrar endpoint as introduced by BRSKI: "/.well- known/brski/requestvoucher", but with a Content-Type header field for JSON-in-JWS"

The Content-Type header field for JSON-in-JWS PVR is: application/voucher-jws+json (see Figure 8 for the content definition), as defined in [I-D.ietf-anima-jws-voucher].

The Registrar-Agent sets the Accept field in the request-header indicating the acceptable Content-Type for the voucher-response. The voucher-response Content-Type header field is set to application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

6.2.2. Pledge-Voucher-Request (PVR) Processing by Registrar

After receiving the PVR from Registrar-Agent, the registrar **SHALL** perform the verification as defined in section 5.3 of [RFC8995]. In addition, the registrar **SHALL** verify the following parameters from the PVR:

- agent-provided-proximity-registrar-cert: **MUST** contain registrar's own registrar LDevID certificate to ensure the registrar in proximity of the Registrar-Agent is the desired registrar for this PVR.

- agent-signed-data: The registrar **MUST** verify that the Registrar-Agent provided data has been signed with the private key corresponding to the EE (RegAgt) certificate indicated in the "kid" JOSE header parameter. The registrar **MUST** verify that the LDevID(ReAgt) certificate, corresponding to the signature, is still valid. If the certificate is already expired, the registrar **SHALL** reject the request. Validity of used signing certificates at the time of signing the agent-signed-data 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 10.3. The registrar **MUST** fetch the EE (RegAgt) certificate, based on the provided SubjectKeyIdentifier (SKID) contained in the "kid" header parameter of the agent-signed-data, and perform this verification. This requires, that the registrar has access to the EE (RegAgt) certificate data (including intermediate CA certificates if existent) based on the SKID. Note, the registrar may have stored the EE (RegAgt) certificate if used during TLS establishment between Registrar-Agent and registrar or it may be provided via a repository.

If the registrar is unable to validate the PVR it **SHOULD** respond with a HTTP 4xx/5xx error code to the Registrar-Agent.

The following 4xx client error codes **SHOULD** be used:

* 403 Forbidden: if the registrar detected that one or more security related parameters are not valid.

* 404 Not Found status code if the pledge provided information could not be used with automated allowance, as described in section 5.3 of [RFC8995].

* 406 Not Acceptable: if the Content-Type indicated by the Accept header is unknown or unsupported.

If the validation succeeds, the registrar performs pledge authorization according to [RFC8995], Section 5.3 followed by obtaining a voucher from the pledge's MASA according to [RFC8995], Section 5.4 with the modifications described below in Section 6.2.3.

6.2.3. Registrar-Voucher-Request (RVR) Processing (Registrar to MASA)

If the MASA address/URI is learned from the [RFC8995] Section 2.3 IDevID MASA URI extension, then the MASA on that URI **MUST** support the procedures defined in this document if the PVR used JSON-JWS encoding. If the MASA is only configured on the registrar, then a registrar supporting BRKSI-PRM and other voucher encoding formats (such as those in [RFC8995]) **SHOULD** support per-message-format MASA address/URI configuration for the same IDevID trust anchor."

The registrar **SHALL** construct the payload of the RVR as defined in [RFC8995], Section 5.5. The RVR encoding **SHALL** be JSON-in-JWS as defined in [I-D.ietf-anima-jws-voucher].

The header of the RVR **SHALL** contain the following parameter as defined for JWS [RFC7515]:

* alg: algorithm used to create the object signature

*x5c: base64-encoded registrar LDevID certificate(s) (It optionally contains the certificate chain for this certificate)

The payload of the RVR **MUST** contain the following parameter as part of the voucher-request as defined in [RFC8995]:

*created-on: current date and time in yang:date-and-time format of RVR creation

*nonce: copied from the PVR

*serial-number: product-serial-number of pledge. The registrar **MUST** verify that the IDevID certificate subject serialNumber of the pledge (X520SerialNumber) matches the serial-number value in the PVR. In addition, it **MUST** be equal to the serial-number value contained in the agent-signed data of PVR.

*assertion: voucher assertion requested by the pledge (agent-proximity). The registrar provides this information to assure successful verification of Registrar-Agent proximity based on the agent-signed-data.

*prior-signed-voucher-request: PVR as received from Registrar-Agent, see Section 6.1.2

The RVR **MUST** be extended with the following parameter, when the assertion "agent-proximity" is requested, as defined in [I-D.ietf-anima-rfc8366bis]:

*agent-sign-cert: EE (RegAgt) certificate or the EE (RegAgt) certificate including certificate chain. In the context of this document it is a JSON array of base64encoded certificate information and handled in the same way as x5c header objects. If only a single object is contained in the x5c it **MUST** be the base64-encoded EE (RegAgt) certificate. If multiple certificates are included in the x5c, the first **MUST** be the base64-encoded EE (RegAgt) certificate.

The MASA uses this information for verification that the Registrar-Agent is in proximity to the registrar to state the corresponding assertion "agent-proximity".

The object is signed using the registrar LDevID credentials, which corresponds to the certificate referenced in the JOSE header.

```
# The RVR in General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-voucher-request-prm:voucher)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "ietf-voucher-request-prm:voucher"
  representation in JSON syntax
"ietf-voucher-request-prm:voucher": {
   "created-on": "2022-01-04T02:37:39.235Z",
   "nonce": "eDs++/FuDHGUnRxN3E14CQ==",
   "serial-number": "callee4711",
   "assertion": "agent-proximity",
   "prior-signed-voucher-request": "base64encodedvalue==",
   "agent-sign-cert": [
     "base64encodedvalue==",
     "base64encodedvalue==",
     "..."
   ]
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "typ": "voucher-jws+json"
}
```

Figure 12: Representation of RVR

The registrar **SHALL** send the RVR to the MASA endpoint by HTTP POST: "/.well-known/brski/requestvoucher"

The RVR Content-Type header field is defined in [I-D.ietf-anima-jws-voucher] as: application/voucher-jws+json

The registrar **SHOULD** set the Accept header of the RVR indicating the desired media type for the voucher-response. The media type is application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher].

This document uses the JSON-in-JWS format throughout the definition of exchanges and in the examples. Nevertheless, alternative encodings of the voucher as used in BRSKI [RFC8995] with JSON-in-CMS or CBOR-in-COSE_Sign [RFC8152] for constraint environments are possible as well. The assumption is that a pledge typically supports a single encoding variant and creates the PVR in the supported format. To ensure that the pledge is able to process the voucher, the registrar **MUST** use the media type for Accept header in the RVR based on the media type used for the PVR.

Once the MASA receives the RVR it **SHALL** perform the verification as described in Section 5.5 in [RFC8995].

In addition, the following processing **SHALL** be performed for PVR contained in RVR "prior-signed-voucher-request" field:

* agent-provided-proximity-registrar-cert: The MASA **MAY** verify that this field contains the registrar LDevID certificate. If so, it **MUST** correspond to the registrar LDevID credentials used to sign the RVR. Note: Correspond here relates to the case that a single registrar LDevID certificate is used or that different registrar LDevID certificates are used, which are issued by the same CA.

* agent-signed-data: The MASA **MAY** verify this data to issue "agent-proximity" assertion. If so, the agent-signed-data **MUST** contain the pledge product-serial-number, contained in the "serial-number" field of the PVR (from "prior-signed-voucher-request" field) and also in "serial-number" field of the RVR. The EE (RegAgt) certificate to be used for signature verification is identified by the "kid" parameter of the JOSE header. If the assertion "agent-proximity" is requested, the RVR **MUST** contain the corresponding EE (RegAgt) certificate data in the "agent-sign-cert" field of the RVR. It **MUST** be verified by the MASA to the same domain CA as the registrar LDevID certificate. If the "agent-sign-cert" field is not set, the MASA **MAY** state a lower level assertion value, e.g.: "logged" or "verified". Note: Sub-CA certificate(s) **MUST** also be carried by "agent-sign-cert", in case the EE (RegAgt) certificate is issued by a sub-CA and not the domain CA known to the MASA. As the "agent-sign-cert" field is defined as array (x5c), it can handle multiple certificates.

If validation fails, the MASA **SHOULD** respond with an HTTP 4xx client error status code to the registrar. The HTTP error status codes are kept the same as defined in Section 5.6 of [RFC8995] and comprise the codes: 403, 404, 406, and 415.

6.2.4. Voucher Issuance by MASA

The MASA creates a voucher with Media-Type of application/voucher-jws+json as defined in [I-D.ietf-anima-jws-voucher]. If the MASA detects that the Accept header of the PVR does not match application/voucher-jws+json it **SHOULD** respond with the HTTP status code "406 Not Acceptable" as the pledge will not be able to parse the response. The voucher is according to [I-D.ietf-anima-rfc8366bis] but uses the new assertion value specified Section 5.4.

Figure 13 shows an example of the contents of a voucher.

```
# The MASA issued voucher in General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-voucher:voucher)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "ietf-voucher:voucher" representation
  in JSON syntax
"ietf-voucher:voucher": {
  "assertion": "agent-proximity",
  "serial-number": "callee4711",
  "nonce": "base64encodedvalue==",
  "created-on": "2022-01-04T00:00:02.000Z",
  "pinned-domain-cert": "base64encodedvalue=="
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "typ": "voucher-jws+json"
}
```

Figure 13: Representation of MASA issued voucher

The pinned-domain certificate to be put into the voucher is determined by the MASA as described in section 5.5 of [RFC8995]. The MASA returns the voucher-response (voucher) to the registrar.

6.2.5. MASA issued Voucher Processing by Registrar

After receiving the voucher the registrar **SHOULD** evaluate it for transparency and logging purposes as outlined in Section 5.6 of [RFC8995]. The registrar **MUST** add an additional signature to the MASA provided voucher using its registrar EE credentials.

The signature is created by signing the original "JWS Payload" produced by MASA and the registrar added "JWS Protected Header" using the registrar EE credentials (see [RFC7515], Section 5.2 point 8. The x5c component of the "JWS Protected Header" **MUST** contain the registrar EE certificate as well as potential subordinate CA certificates up to (but not including) the pinned domain certificate. The pinned domain certificate is already contained in the voucher payload ("pinned-domain-cert").

(For many installations, with a single registrar credential, the registrar credential is what is pinned)

In [RFC8995], the Registrar proved possession of the it's credential when the TLS session was setup. While the pledge could not, at the time, validate the certificate truly belonged the registrar, it did validate that the certificate it was provided was able to authenticate the TLS connection.

In the BRSKI-PRM mode, with the Registrar-Agent mediating all communication, the Pledge has not as yet been able to witness that the intended Registrar really does possess the relevant private key. This second signature provides for the same level of assurance to the pledge, and that it matches the public key that the pledge received in the trigger for the PVR (see Figure 6).

The registrar **MUST** use the same registrar EE credentials used for authentication in the TLS handshake to authenticate towards the Registrar-Agent. This has some operational implications when the registrar may be part of a scalable framework as described in [I-D.richardson-anima-registrar-considerations], Section 1.3.1.

The second signature **MUST** either be done with the private key associated with the registrar EE certificate provided to the Registrar-Agent, or the use of a certificate chain is necessary. This ensures that the same registrar EE certificate can be used to verify the signature as transmitted in the voucher-request as also transferred in the PVR in the "agent-provided-proximity-registrar-cert".

Figure 14 below provides an example of the voucher with two signatures.

```
# The MASA issued voucher with additional registrar signature in
  General JWS Serialization syntax
{
  "payload": "BASE64URL(ietf-voucher:voucher)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header (MASA)))",
      "signature": BASE64URL(JWS Signature)
    },
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header (Reg)))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "ietf-voucher:voucher" representation in
  JSON syntax
"ietf-voucher:voucher": {
   "assertion": "agent-proximity",
   "serial-number": "callee4711",
   "nonce": "base64encodedvalue==",
   "created-on": "2022-01-04T00:00:02.000Z",
   "pinned-domain-cert": "base64encodedvalue=="
}

# Example: Decoded "JWS Protected Header (MASA)" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "typ": "voucher-jws+json"
}

# Example: Decoded "JWS Protected Header (Reg)" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```

Figure 14: Representation of MASA issued voucher with additional registrar signature

Depending on the security policy of the operator, this signature can also be interpreted by the pledge as explicit authorization of the registrar to install the contained trust anchor. The registrar sends the voucher to the Registrar-Agent.

6.2.6. Pledge-Enrollment-Request (PER) Processing (Registrar-Agent to Registrar)

After receiving the voucher, the Registrar-Agent sends the PER to the registrar in the same HTTP-over-TLS connection. Which is similar to the PER processing described in Section 5.2 of [RFC8995]. In case the PER cannot be send in the same HTTP-over-TLS connection the Registrar-Agent may send the PER in a new HTTP-over-TLS connection. The registrar is able to correlate the PVR and the PER based on the signatures and the contained product-serial-number information. Note, this also addresses situations in which a nonceless voucher is used and may be pre-provisioned to the pledge. As specified in Section 6.1.4 deviating from BRSKI the PER is not a raw PKCS#10. As the Registrar-Agent is involved in the exchange, the PKCS#10 is wrapped in a JWS object by the pledge and signed with pledge's IDevID to ensure proof-of-identity as outlined in Figure 10.

EST [RFC7030] standard endpoints (/simpleenroll, /simplereenroll, /serverkeygen, /cacerts) on the registrar cannot be used for BRSKI-PRM. This is caused by the utilization of signature wrapped-objects in BRSKI-PRM. As EST requires to sent a raw PKCS#10 request to e.g., "/.well-known/est/simpleenroll" endpoint, this document makes an enhancement by utilizing EST but with the exception to transport a signature wrapped PKCS#10 request. Therefore a new endpoint for BRSKI-PRM on the registrar is defined as "/.well-known/brski/requestenroll"

The Content-Type header of PER is: application/jose+json.

This is a deviation from the Content-Type header values used in [RFC7030] and results in additional processing at the domain registrar (as EST server). Note, the registrar is already aware that the bootstrapping is performed in a pledge-responder-mode due to the use of the EE (RegAgt) certificate for TLS and the provided PVR as JSON-in-JWS object.

* If the registrar receives a PER with Content-Type header: application/jose+json, it **MUST** verify the wrapping signature using the certificate indicated in the JOSE header.

* The registrar verifies that the pledge's certificate (here IDevID), carried in "x5c" header field, is accepted to join the domain after successful validation of the PVR.

*If both succeed, the registrar utilizes the PKCS#10 request contained in the JWS object body as "P10" parameter of "ietf-sztp-csr:csr" for further processing of the enrollment-request with the corresponding domain CA. It creates a registrar enrollment-request (RER) by utilizing the protocol expected by the domain CA. The domain registrar may either directly forward the provided PKCS#10 request to the CA or provide additional information about attributes to be included by the CA into the requested LDevID certificate. The approach of sending this information to the CA depends on the utilized certificate management protocol between the RA and the CA and is out of scope for this document.

Note while BRSKI-PRM targets the initial enrollment, re-enrollment may be supported in a similar way with the exception that the current LDevID certificate is used instead of the IDevID certificate to verify the wrapping signature of the PKCS#10 request (see also Section 6.1.4).

The Registrar-Agent **SHALL** send the PER to the registrar by HTTP POST to the endpoint: "/.well-known/brski/requestenroll"

The registrar **SHOULD** respond with an HTTP 200 OK in the success case or fail with HTTP 4xx/5xx status codes as defined by the HTTP standard.

A successful interaction with the domain CA will result in a pledge LDevID certificate, which is then forwarded by the registrar to the Registrar-Agent using the Content-Type header: application/pkcs7-mime.

6.2.7. Request Wrapped-CA-certificate(s) (Registrar-Agent to Registrar)

As the pledge will verify it own certificate LDevID certificate when received, it also needs the corresponding CA certificates. This is done in EST [RFC7030] using the "/.well-known/est/cacerts" endpoint, which provides the CA certificates over a TLS protected connection. BRSKI-PRM requires a signature wrapped CA certificate object, to avoid that the pledge can be provided with arbitrary CA certificates in an authorized way. The registrar signed CA certificate object will allow the pledge to verify the authorization to install the received CA certificate(s). As the CA certificate(s) are provided to the pledge after the voucher, the pledge has the required information (the domain certificate) to verify the wrapped CA certificate object.

To support Registrar-Agents requesting a signature wrapped CA certificate(s) object, a new endpoint for BRSKI-PRM is defined on the registrar: "/.well-known/brski/wrappedcacerts"

The Registrar-Agent **SHALL** requests the EST CA trust anchor database information (in form of CA certificates) by HTTP GET.

The Content-Type header of the response **SHALL** be: application/jose+json.

This is a deviation from the Content-Type header values used in EST [RFC7030] and results in additional processing at the domain registrar (as EST server). The additional processing is to sign the CA certificate(s) information using the registrar LDevID credentials. This results in a signed CA certificate(s) object (JSON-in-JWS), the CA certificates are provided as base64 encoded "x5bag" (see definition in [RFC9360]) in the JWS payload.

```
# The CA certificates data with registrar signature in General JWS Seria
{
  "payload": "BASE64URL(certs)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "certs" representation in JSON syntax
{
  "x5bag": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}


# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```

Figure 15: Representation of CA certificate(s) data with registrar signature

6.3. Response Objects supplied by the Registrar-Agent to the Pledge

It is assumed that the Registrar-Agent already obtained the bootstrapping response objects from the domain registrar and can supply them to the pledge:

- *voucher-response - Voucher (from MASA via Registrar)
- *wrapped-CA-certificate(s)-response - CA certificates
- *enrollment-response - LDevID (Pledge) certificate (from CA via registrar)

To deliver these response objects, the Registrar-Agent will re-connect to the pledge. To contact the pledge, it may either discover the pledge as described in Section 5.6.2 or use stored information from the first contact with the pledge.

Preconditions in addition to Section 6.2:

- *Registrar-Agent: obtained voucher and LDevID certificate and optionally IDevID CA certificates. The IDevID CA certificate is necessary, when the connection between the Registrar-Agent and the pledge is established using TLS to enable the Registrar-Agent to validate the pledges' IDevID certificate during the TLS handshake as described in Section 6.1.

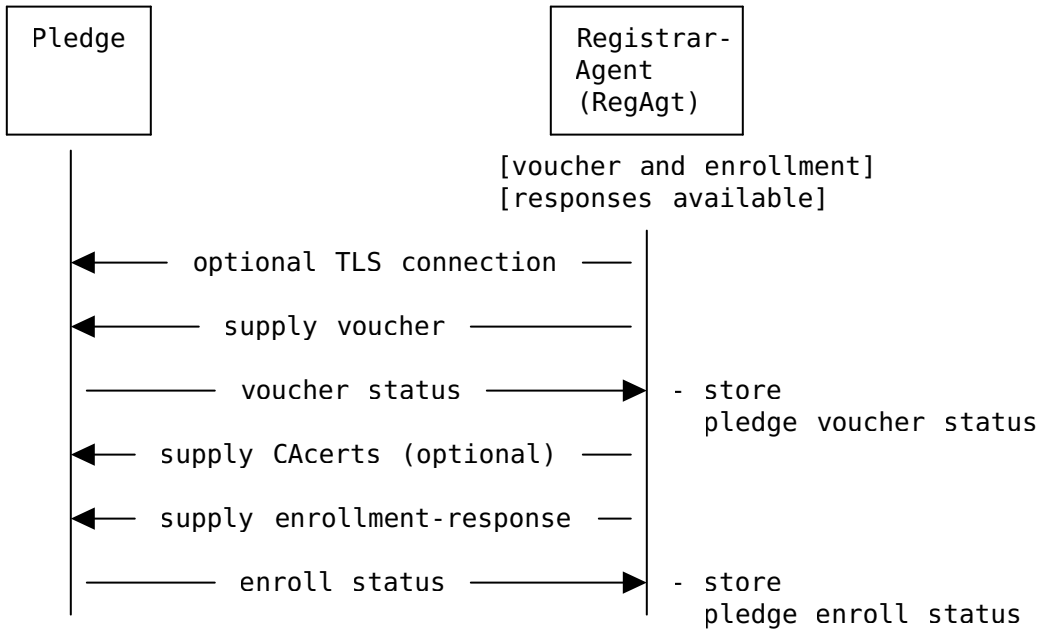

Figure 16: Responses and status handling between pledge and Registrar-Agent

The Registrar-Agent **MAY** optionally use TLS to protect the communication as outlined in Section 6.1.

The Registrar-Agent provides the information via distinct pledge endpoints as following.

6.3.1. Pledge: Voucher-Response Processing

The Registrar-Agent **SHALL** send the voucher-response to the pledge by HTTP POST to the endpoint: "/.well-known/brski/svr".

The Registrar-Agent voucher-response Content-Type header is application/voucher-jws+json and contains the voucher as provided by the MASA. An example is given in Figure 13 for a MASA signed voucher and in Figure 14 for the voucher with the additional signature of the registrar.

A nonceless voucher may be accepted as in [RFC8995] and may be allowed by a manufacture's pledge implementation.

To perform the validation of several signatures on the voucher object, the pledge **SHALL** perform the signature verification in the following order:

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

2. Install trust anchor contained in the voucher ("pinned-domain-cert") provisionally

3. Validate the LDevID(Reg) certificate received in the agent-provided-proximity-registrar-cert in the pledge-voucher-request trigger request (in the field "agent-provided-proximity-registrar-cert")

4. Verify registrar signature of the voucher similar as described in Section 5.6.1 in [RFC8995], but take the registrar certificate instead of the MASA certificate for the verification

Step3 and step 4 have been introduced in BRSKI-PRM to enable verification of LDevID(Reg) certificate and also the proof-of-possession of the corresponding private key by the registrar, which is done in BRSKI based on the established TLS channel. If all steps stated above have been performed successfully, the pledge **SHALL** terminate the "PROVISIONAL accept" state for the domain trust anchor and the registrar LDevID certificate.

If an error occurs during the verification and validation of the voucher, this **SHALL** be reported in the reason field of the pledge voucher status.

6.3.2. Pledge: Voucher Status Telemetry

After voucher verification and validation the pledge **MUST** reply with a status telemetry message as defined in Section 5.7 of [RFC8995]. The pledge generates the voucher-status and provides it as signed JSON-in-JWS object in response to the Registrar-Agent.

The response has the Content-Type application/jose+json and is signed using the IDevID of the pledge as shown in Figure 17. As the reason field is optional (see [RFC8995]), it **MAY** be omitted in case of success.

```
# The "pledge-voucher-status" telemetry in general JWS
  serialization syntax
{
  "payload": "BASE64URL(pledge-voucher-status)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "pledge-voucher-status" representation
  in JSON syntax for success case
{
  "version": 1,
  "status": true,
  "reason": "Voucher successfully processed",
  "reason-context": {
    "pvs-details": "JSON"
  }
}

# Example: Decoded payload "pledge-voucher-status" representation
  in JSON syntax for error case
{
  "version": 1,
  "status": false,
  "reason": "Failed to authenticate MASA certificate because
  it starts in the future (1/1/2023).",
  "reason-context": {
    "pvs-details": "Current date: 1/1/1970"
  }
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```

Figure 17: Representation of pledge voucher status telemetry

If the pledge did not did not provide voucher status telemetry information after processing the voucher, the Registrar-Agent **MAY**

query the pledge status explicitly as described in Section 6.4 and **MAY** resent the voucher depending on the Pledge status following the procedure described in Section 6.3.1.

6.3.3. Pledge: Wrapped-CA-Certificate(s) Processing

The Registrar-Agent **SHALL** provide the set of CA certificates requested from the registrar to the pledge by HTTP POST to the endpoint: "/.well-known/brski/scac".

As the CA certificate provisioning is crucial from a security perspective, this provisioning **SHOULD** only be done, if the voucher-response has been successfully processed by pledge as reflected in the voucher status telemetry.

The CA certificates message has the Content-Type application/jose+json and is signed using the credential of the registrar as shown in Figure 15.

The CA certificates are provided as base64 encoded "x5bag". The pledge **SHALL** install the received CA certificates as trust anchor after successful verification of the registrar's signature.

The verification comprises the following steps the pledge **MUST** perform. Maintaining the order of versification steps as indicated allows to determine, which verification has already been passed:

1. Check content-type of the CA certificates message. If no Content-Type is contained in the HTTP header, the default Content-Type utilized in this document (JSON-in-JWS) is used. If the Content-Type of the response is in an unknown or unsupported format, the pledge **SHOULD** reply with a 415 Unsupported media type error code.

2. Check the encoding of the payload. If the pledge detects errors in the encoding of the payload, it **SHOULD** reply with 400 Bad Request error code.

3. Verify that the wrapped CA certificate object is signed using the registrar certificate against the pinned-domain certificate. This **MAY** be done by comparing the hash that is indicating the certificate used to sign the message is that of the pinned-domain certificate. If the validation against the pinned domain-certificate fails, the client **SHOULD** reply with a 401 Unauthorized error code. It signals that the authentication has failed and therefore the object was not accepted.

4. Verify signature of the the received wrapped CA certificate object. If the validation of the signature fails, the pledge

SHOULD reply with a 406 Not Acceptable. It signals that the object has not been accepted.

5. If the received CA certificates are not self-signed, i.e., an intermediate CA certificate, verify them against an already installed trust anchor, as described in section 4.1.3 of [RFC7030].

6.3.4. Pledge: Enrollment-Response Processing

The Registrar-Agent **SHALL** send the enroll-response to the pledge by HTTP POST to the endpoint: "/.well-known/brski/ser".

The Registrar-Agent enroll-response Content-Type header, when using EST [RFC7030] as enrollment protocol between the Registrar-Agent and the infrastructure is: application/pkcs7-mime. Note: It only contains the LDevID certificate for the pledge, not the certificate chain.

Upon reception, the pledge **SHALL** verify the received LDevID certificate. The pledge **SHALL** generate the enroll status and provide it in the response to the Registrar-Agent. If the verification of the LDevID certificate succeeds, the status property **SHALL** be set to "status": true, otherwise to "status": false

6.3.5. Pledge: Enrollment-Status Telemetry

The pledge **MUST** reply with a status telemetry message as defined in Section 5.9.4 of [RFC8995]. As for the other objects, the enroll-status is signed and results in a JSON-in-JWS object. If the pledge verified the received LDevID certificate successfully it **SHALL** sign the response using its new LDevID credentials as shown in Figure 18. In the failure case, the pledge **SHALL** use the available IDevID credentials. As the reason field is optional, it **MAY** be omitted in case of success.

The response has the Content-Type application/jose+json.

```
# The "pledge-enroll-status" telemetry in General JWS Serialization
  syntax
{
  "payload": "BASE64URL(pledge-enroll-status)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "pledge-enroll-status" representation
  in JSON syntax for success case
{
  "version": 1,
  "status": true,
  "reason": "Enrollment response successfully processed",
  "reason-context": {
    "pes-details": "JSON"
  }
}

# Example: Decoded payload "pledge-voucher-status" representation
  in JSON syntax for error case
{
  "version": 1,
  "status": false,
  "reason": "Enrollment response could not be verified.",
  "reason-context": {
    "pes-details": "no matching trust anchor"
  }
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```

Figure 18: Representation of pledge enroll status telemetry

Once the Registrar-Agent has collected the information, it can connect to the registrar to provide it with the status responses.

6.3.6. Telemetry Voucher Status and Enroll Status Handling (Registrar-Agent to Domain Registrar)

The following description requires that the Registrar-Agent has collected the status information from the pledge. It **SHALL** provide the status information to the registrar for further processing.

Preconditions in addition to Section 6.2:

* Registrar-Agent: obtained voucher status and enroll status from pledge.

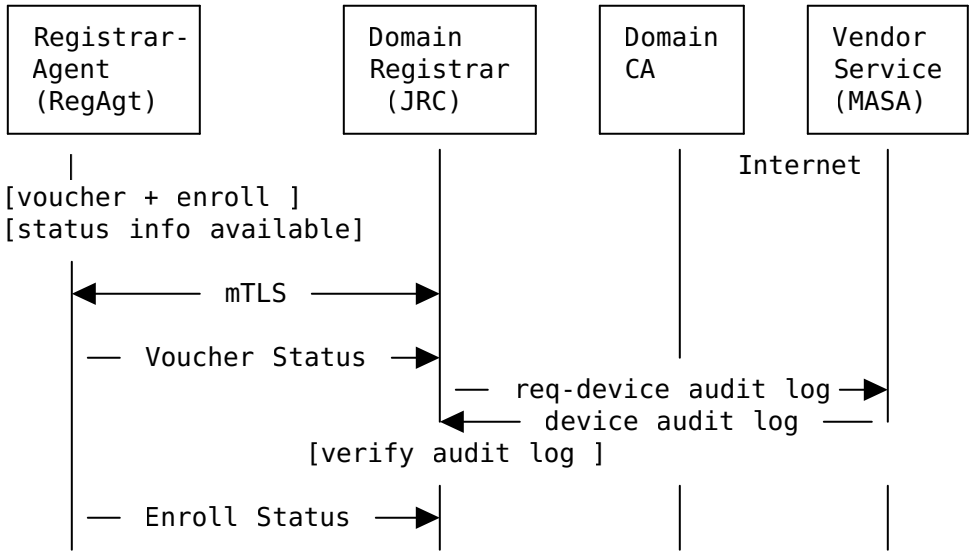

Figure 19: Bootstrapping status handling

The Registrar-Agent **MUST** provide the collected pledge voucher status to the registrar. This status indicates if the pledge could process the voucher successfully or not.

In case the TLS connection to the registrar is already closed, the Registrar-Agent opens a new TLS connection with the registrar as stated in Section 6.2.

The Registrar-Agent sends the pledge voucher status without modification to the registrar with an HTTP-over-TLS POST using the registrar endpoint "/.well-known/brski/voucher_status". The Content-Type header is kept as application/jose+json as described in Figure 16 and depicted in the example in Figure 17.

The registrar **SHOULD** log the transaction provided for a pledge via Registrar-Agent and include the identity of the Registrar-Agent in these logs. For log analysis the following may be considered:

- The registrar knows the interacting Registrar-Agent from the authentication of the Registrar-Agent towards the registrar using LDevID (RegAgt) and can log it accordingly.

- The telemetry information from the pledge can be correlated to the voucher response provided from the registrar to the Registrar-Agent and further to the pledge.

- The telemetry information, when provided to the registrar is provided via the Registrar-Agent and can thus be correlated.

The registrar **SHALL** verify the signature of the pledge voucher status and validate that it belongs to an accepted device of the domain based on the contained "serial-number" in the IDevID certificate referenced in the header of the voucher status.

According to [RFC8995] Section 5.7, the registrar **SHOULD** respond with an HTTP 200 OK in the success case or fail with HTTP 4xx/5xx status codes as defined by the HTTP standard. The Registrar-Agent may use the response to signal success / failure to the service technician operating the Registrar-Agent. Within the server logs the server **SHOULD** capture this telemetry information.

The registrar **SHOULD** proceed with collecting and logging status information by requesting the MASA audit-log from the MASA service as described in Section 5.8 of [RFC8995].

The Registrar-Agent **MUST** provide the pledge's enroll status to the registrar. The status indicates the pledge could process the enroll-response (certificate) and holds the corresponding private key.

The Registrar-Agent sends the pledge enroll status without modification to the registrar with an HTTP-over-TLS POST using the registrar endpoint "/.well-known/brski/enrollstatus". The Content-Type header is kept as application/jose+json as described in Figure 16 and depicted in the example in Figure 18.

The registrar **MUST** verify the signature of the pledge enroll status. Also, the registrar **SHALL** validate that the pledge is an accepted device of the domain based on the contained product-serial-number in the LDevID certificate referenced in the header of the enroll status. The registrar **SHOULD** log this event. In case the pledge enroll status indicates a failure, the pledge was unable to verify the received LDevID certificate and therefore signed the enroll status with its IDevID credential. Note that the signature verification of the status information is an addition to the described handling in Section 5.9.4

of [RFC8995], and is replacing the pledges TLS client authentication by DevID credentials in [RFC8995].

According to [RFC8995] Section 5.9.4, the registrar **SHOULD** respond with an HTTP 200 OK in the success case or fail with HTTP 4xx/5xx status codes as defined by the HTTP standard. Based on the failure case 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 certificate. An example case for the registrar revoking the issued LDevID for the pledge is when the pledge was not able to verify the received LDevID certificate and therefore did send a 406 (Not Acceptable) response. In this case the registrar may revoke the LDevID certificate as the pledge did no accepted it for installation.

The Registrar-Agent may use the response to signal success / failure to the service technician operating the Registrar-Agent. Within the server log the registrar **SHOULD** capture this telemetry information.

6.4. Request Pledge-Status by Registrar-Agent from Pledge

The following assumes that a Registrar-Agent may need to query the status of a pledge. This information may be useful to solve errors, when the pledge was not able to connect to the target domain during the bootstrapping. The pledge **MAY** provide a dedicated endpoint to accept status-requests.

Preconditions:

* Registrar-Agent: possesses LDevID (RegAgt), may have a list of product-serial-number(s) of pledges to be queried and a list of corresponding manufacturer trust anchors to be able to verify signatures performed with the IDevID credential.

* Pledge: may already possess domain credentials and LDevID(Pledge), or may not possess one or both of these.

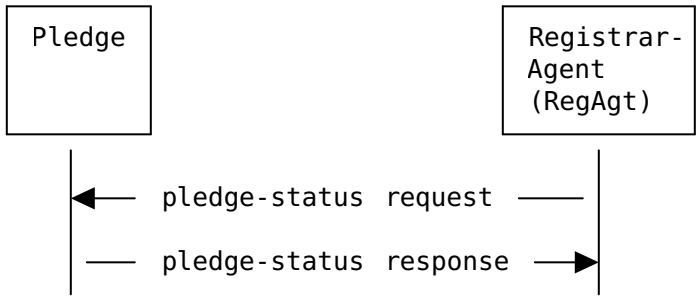

Figure 20: Pledge-status handling between Registrar-Agent and pledge

6.4.1. Pledge-Status - Request (Registrar-Agent to Pledge)

The Registrar-Agent requests the pledge-status via HTTP POST on the defined pledge endpoint: "/.well-known/brski/qps"

The Registrar-Agent Content-Type header for the pledge-status request is: application/jose+json. It contains information on the requested status-type, the time and date the request is created, and the product serial-number of the pledge contacted as shown in Figure 21. The pledge-status request is signed by Registrar-Agent using the private key corresponding to the EE (RegAgt) certificate.

The following Concise Data Definition Language (CDDL) [RFC8610] explains the structure of the format for the pledge-status request. It is defined following the status telemetry definitions in BRSKI [RFC8995]. Consequently, format and semantics of pledge-status requests below are for version 1. The version field is included to permit significant changes to the pledge-status request and response in the future. A pledge or a Registrar-Agent that receives a pledge-status request with a version larger than it knows about **SHOULD** log the contents and alert a human.

```
<CODE BEGINS>
  status-request = {
      "version": uint,
      "created-on": tdate ttime,
      "serial-number": text,
      "status-type": text
  }

<CODE ENDS>
```

Figure 21: CDDL for pledge-status request

The status-type defined for BRSKI-PRM is "bootstrap". This indicates the pledge to provide current status information regarding the bootstrapping status (voucher processing and enrollment of the pledge into the new domain). As the pledge-status request is defined generic, it may be used by other specifications to request further status information, e.g., for onboarding to get further information about enrollment of application specific LDevIDs or other parameters. This is out of scope for this specification.

Figure 22 below shows an example for querying pledge-status using status-type bootstrap.

```
# The Registrar-Agent request of "pledge-status" in general JWS
  serialization syntax
{
  "payload": "BASE64URL(status-request)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "status-request" representation
  in JSON syntax
{
  "version": 1,
  "created-on": "2022-08-12T02:37:39.235Z",
  "serial-number": "pledge-callee4711",
  "status-type": "bootstrap"
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ]
}
```

Figure 22: Example of Registrar-Agent request of pledge-status using status-type bootstrap

6.4.2. Pledge-Status - Response (Pledge - Registrar-Agent)

If the pledge receives the pledge-status request with status-type "bootstrap" it **SHALL** react with a status response message based on the telemetry information described in Section 6.3.

The pledge-status response Content-Type header is application/jose+json.

The following CDDL explains the structure of the format for the status response, which is:

```
<CODE BEGINS>
  status-response = {
    "version": uint,
    "status":
      "factory-default" /
      "voucher-success" /
      "voucher-error" /
      "enroll-success" /
      "enroll-error" /
      "connect-success" /
      "connect-error",
    ?"reason" : text,
    ?"reason-context": { $$arbitrary-map }
  }

<CODE ENDS>
```

Figure 23: CDDL for pledge-status response

Different cases for pledge bootstrapping status may occur, which **SHOULD** be reflected using the status enumeration. This document specifies the status values in the context of the bootstrapping process and credential application. Other documents may enhance the above enumeration to reflect further status information.

The pledge-status response message is signed with IDevID or LDevID, depending on bootstrapping state of the pledge.

* "factory-default": Pledge has not been bootstrapped. Additional information may be provided in the reason or reason-context. The pledge signs the response message using its IDevID(Pledge).

* "voucher-success": Pledge processed the voucher exchange successfully. Additional information may be provided in the reason or reason-context. The pledge signs the response message using its IDevID(Pledge).

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

* "enroll-success": Pledge has processed the enrollment exchange successfully. Additional information may be provided in the reason or reason-context. The pledge signs the response message using its LDevID(Pledge).

* "enroll-error": Pledge enrollment-response processing terminated with error. Additional information may be provided in the reason

or reason-context. The pledge signs the response message using its IDevID(Pledge).

The reason and the reason-context **SHOULD** contain the telemetry information as described in Section 6.3.

As the pledge is assumed to utilize its bootstrapped credentials (LDevID) in communication with other peers, additional status information is provided for the connectivity to other peers, which may be helpful in analyzing potential error cases.

* "connect-success": Pledge could successfully establish a connection to another peer. Additional information may be provided in the reason or reason-context. The pledge signs the response message using its LDevID(Pledge).

* "connect-error": Pledge connection establishment terminated with error. Additional information may be provided in the reason or reason-context. The pledge signs the response message using its LDevID(Pledge).

The pledge-status responses are cumulative in the sense that connect-success implies enroll-success, which in turn implies voucher-success.

Figure 24 provides an example for the bootstrapping-status information.

```
# The pledge "status-response" in General JWS Serialization syntax
{
  "payload": "BASE64URL(status-response)",
  "signatures": [
    {
      "protected": "BASE64URL(UTF8(JWS Protected Header))",
      "signature": BASE64URL(JWS Signature)
    }
  ]
}

# Example: Decoded payload "status-response" representation
  in JSON syntax
{
  "version": 1,
  "status": "enroll-success",
  "reason-context": {
    "additional" : "JSON"
  }
}

# Example: Decoded "JWS Protected Header" representation
  in JSON syntax
{
  "alg": "ES256",
  "x5c": [
    "base64encodedvalue==",
    "base64encodedvalue=="
  ],
  "typ": "jose+json
}
```

Figure 24: Example of pledge-status response

*In case "factory-default" the pledge does not possess the domain certificate resp. the domain trust-anchor. It will not be able to verify the signature of the Registrar-Agent in the bootstrapping-status request.

*In cases "vouchered" and "enrolled" the pledge already possesses the domain certificate (has domain trust-anchor) and can therefore validate the signature of the Registrar-Agent. If validation of the JWS signature fails, the pledge **SHOULD** respond with the HTTP 403 Forbidden status code.

*The HTTP 406 Not Acceptable status code **SHOULD** be used, if the Accept header in the request indicates an unknown or unsupported format.

*The HTTP 415 Unsupported Media Type status code **SHOULD** be used, if the Content-Type of the request is an unknown or unsupported format.

*The HTTP 400 Bad Request status code **SHOULD** be used, if the Accept/Content-Type headers are correct but nevertheless the status-request cannot be correctly parsed.

The pledge **SHOULD** by default only respond to requests from nodes it can authenticate (such as registrar agent), once the pledge is enrolled with CA certificates and a matching domain certificate.

7. Artifacts

7.1. Voucher-Request Artifact

[I-D.ietf-anima-rfc8366bis] extends the voucher-request as defined in [RFC8995] to include additional fields necessary for handling bootstrapping in the pledge-responder-mode. These additional fields are defined in Section 6.1 as:

*agent-signed-data to provide a JSON encoded artifact from the involved Registrar-Agent, which allows the registrar to verify the Registrar-Agent's involvement

*agent-provided-proximity-registrar-cert to provide the registrar certificate visible to the Registrar-Agent, comparable to the registrar-proximity-certificate used in [RFC8995]

*agent-signing certificate to optionally provide the Registrar-Agent signing certificate.

Examples for the application of these fields in the context of a PVR are provided in Section 6.2.

8. IANA Considerations

This document requires the following IANA actions.

8.1. BRSKI .well-known Registry

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

URI	Description	Reference
tpvr	create pledge voucher-request	[THISRFC]
tper	create pledge enrollment-request	[THISRFC]
svr	supply voucher-response	[THISRFC]
ser	supply enrollment-response	[THISRFC]
scac	supply CA certificates to pledge	[THISRFC]
qps	query pledge status	[THISRFC]
requestenroll	supply PER to registrar	[THISRFC]
wrappedcacerts	request wrapped CA certificates	[THISRFC]

8.2. DNS Service Names

IANA has registered the following service names:

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

9. 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 6.1 describes to optional 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 credential 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 LDevID certificate associated with the commissioning device instead of an LDevID certificate associated with the service technician operating the device. This avoids revealing potentially included personal information to Registrar and MASA.

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 unencrypted. 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.

10. 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

10.1. 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 limited 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 resent the PVR previously sent. Note that in case of resending, a contained nonce and also the contained agent-signed-data in the PVR would consequently be reused.

10.2. 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 LDevID 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 IDevID certificate of the pledge. (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 EE (RegAgt) certificate data contained in the PVR.

Misbinding of a pledge by a faked domain registrar is countered as described in BRSKI security considerations [RFC8995] (Section 11.4).

10.3. Misuse of Registrar-Agent Credentials

Concerns of misusage of a Registrar-Agent with a valid EE (RegAgt) 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 EE (RegAgt) certificate may have been acquired by a prior BRSKI run for the Registrar-Agent, if an IDevID is available on Registrar-Agent. Alternatively, the EE (RegAgt) certificate may be acquired by a service technician from the domain PKI system in an authenticated way.

In addition it is required that the EE (RegAgt) 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 EE (RegAgt) 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 EE (RegAgt) 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.

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

To discover a specific pledge a Registrar-Agent may request the service name in combination with the product-serial-number of a specific pledge. The pledge reacts on this if its product-serial-number is part of the request 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 a LDevID or to not support this feature at all, to avoid an enumeration in an operative domain.

10.5. 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 [RFC8995] Section 11.7 (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 traditional 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 [RFC8407] Section 3.7 (Security Considerations Section).

11. Acknowledgments

We would like to thank the various reviewers, in particular Brian E. Carpenter, 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, and Christian Spindler. Special thanks to Esko Dijk for the in deep review and the improving proposals. Support in PoC implementations and comments resulting from the implementation was provided by Hong Rui Li and He Peng Jia.

12. References

12.1. 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-09, 29 August 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-jws-voucher-09>.

[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-10, 22 August 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-rfc8366bis-10>.

[I-D.ietf-netconf-sztp-csr] Watsen, K., Housley, R., and S. Turner, "Conveying a Certificate Signing Request (CSR) in a Secure Zero Touch Provisioning (SZTP) Bootstrapping Request", Work in Progress, Internet-Draft, draft-ietf-netconf-sztp-csr-14, 2 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-sztp-csr-14>.

[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>.

[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>.

[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>.

[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>.

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

[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>.

12.2. 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.eckert-anima-brski-discovery] Eckert, T. T., von Oheimb, D., and E. Dijk, "Discovery for BRSKI variations", Work in Progress, Internet-Draft, draft-eckert-anima-brski-discovery-01, 23 October 2023, <https://

datatracker.ietf.org/doc/html/draft-eckert-anima-brski-discovery-01>.

[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-07, 17 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-brski-ae-07>.

[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-02, 6 August 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-taxonomy-manufacturer-anchors-02>.

[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-07, 11 May 2023, <https://datatracker.ietf.org/doc/html/draft-richardson-anima-registrar-considerations-07>.

[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>.

[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>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/rfc/rfc6125>.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/rfc/rfc8152>.

[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>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[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>.

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: 4649 bytes

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

{
  "payload":
    "eyJpZXRmLXZvdWNoZXItcmVxdWVzdC1wcm06dm91Y2hlciI6eyJhc3NlcnRpb24\
iOiJhZ2VudC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Njc4OSIsIm5\
vbmNlIjoiTDNJSjZocHRIQ0lRb054YWFiOUhXQT09IiwiY3JlYXRlZC1vbiI6IjIwMjI\
tMDQtMjZUMDU6MTY6MTcuNzA5WiIsImFnZW50LXByb3ZpZGVkLXByb3hpbWl0eS1yZWd\
pc3RyYXItY2VydCI6Ik1JSUI0akNDQVlpZ0F3SUJBZ0lHQVhZNzJiYlpNQW9HQ0NxR1N\
NNDlCQU1DTURVeEV6QVJCZ05WQkFvTUNrMTVRblZ6YVc1bGMzTXhEVEFMQmdOVkJBY01\
CRk5wZEdVeER6QU5CZ05WQkFNTUJsUmxjM1JEUVRBZUZ3MHlNREV5TURjd05qRTRNVEp\
hRncwek1ERXlNRGN3TmpFNE1USmFNRDR4RXpBUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzN\
NeERUQUxCZ05WQkFjTUJGTnBkR1V4R0RBV0JnTlZCQU1NRDBSdmJXRnBibEpsWjJsemR\
ISmhjakJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUFCQmsxNksvaTc5b1J\
rSzVZYmVQZzhVU1I4L3VzMWRQVWlaSE10b2tTZHFLVzVmbldzQmQrcVJMN1dSZmZlV2t\
5Z2Vib0pmSWxsdXJjaTI1d25oaU9WQ0dqZXpCNU1CMEdBMVVkSlFRV01CUUdDQ3NHQVF\
VRkJ3TUJCZ2dyQmdFRkJRY0RIREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdTQVlEVlIwUkJ\
FRXdQNElkY21WbmFYTjBjbUZ5TFhSbGMzUXVjMmxsYldWdWN5MWlkQzV1WlhTQ0huSmx\
aMmx6ZEhKaGNpMTBaWE4wTmk1emFXVnRaVzV6TFdKMExtNWxkREFLQmdncWhrak9QUVF\
EQWdOSUFEQkZBaUJ4bGRCaFpxMEV2NUpMMlByV0N0eVM2aERZVzF5Q08vUmF1YnBDN01\
hSURnSWhBTFNKYmdMbmdoYmJBZzBkY1dGVVZvL2dHTjAvand6SlowU2wyaDR4SVhrMSI\
sImFnZW50LXNpZ25lZC1kYXRhIjoiZXlKd1lYbHNiMkZrSWpvaVpYbEtjRnBZVW0xTVd\
GcDJaRmRPYjFwWVNYUmpiVlo0WkZkV2VtUkRNWGRqYlRBMldWZGtiR0p1VVhSak1teHV\
ZbTFXYTB4WFVtaGtSMFZwVDI1emFWa3pTbXhaV0ZKc1drTXhkbUpwU1RaSmFrbDNUV3B\
KZEUxRVVYUk5hbHBWVFVSVk5rMUVZelpPUkVWMVRrUlJORmRwU1hOSmJrNXNZMjFzYUd\
KRE1YVmtWekZwV2xoSmFVOXBTWGROVkVsNlRrUlZNazU2WnpWSmJqRTVJaXdpYzJsbmJ\
tRjBkWEpsY3lJNlczc2ljSEp2ZEdWamRHVmtJam9pWlhsS2NtRlhVV2xQYVVwWlkwaHd\
jMVJWZERSaVNFSkNUbXBvYWxaVVZrZFZWVEZaVmxoYWRWTldVVEpWV0dNNVNXbDNhVmx\
YZUc1SmFtOXBVbFpOZVU1VVdXbG1VU0lzSW5OcFoyNWhkSFZ5WlNJNklrY3pWM2hHU0d\
WMFdGQTRiR3hTVmkwNWRXSnlURmxxU25aUllUWmZlUzFRYWxGWk5FNWhkMW81Y0ZKaGI\
yeE9TbTlFTm1SbFpXdHVTVjlGV0daemVWWlRZbmM0VTBONlRWcE1iakJoUVhWb2FVZFp\
UakJSSW4xZGZRPT0iLCJhZ2VudC1zaWduLWNlcnQiOlsiTUlJQjFEQ0NBWHFnQXdJQkF\
nSUVZbWQ0T1RBS0JnZ3Foa2pPUFFRREFqQStNUk13RVFZRFZRUUtEQXBOZVVKMWMybHV\
aWE56TVEwd0N3WURWUVFIREFSVGFYUmxNUmd3RmdZRFZRUUREQTlVWlhOMFVIVnphRTF\
2WkdWc1EwRXdIaGNOTWpJd05ESTJNRFEwTWpNeldoY05Nekl3TkRJMk1EUTBNak16V2p\
BOU1STXdFUVlEVlFRS0RBcE5lVUoxYzJsdVpYTnpNUTB3Q3dZRFZRUUhEQVJUYVhSbE1\
SY3dGUVlEVlFRRERBNVNaV2RwYzNSeVlYSkJaMlZ1ZERCWk1CTUdCeXFHU000OUFnRUd\
DQ3FHU000OUF3RUhBMElBQkd4bHJOZmozaVJiNy9CUW9kVys1WWlvT3poK2pJdHlxdVJ\
JTy9XejdZb1czaXdEYzNGeGV3TFZmekNyNU52RDEzWmFGYjdmcmFuK3Q5b3RZNVdMaEo\
2alp6QmxNQTRHQTFVZER3RUIvd1FFQXdJSGdEQWZCZ05WSFNNRUdEQVdnQlJ2b1QxdWR\
lMmY2TEVRaFU3SEhqK3ZKL2Q3SXpBZEJnTlZIUTRFRmdRVVhwemxNS3hscEE2OGNVNUZ\
RTVhVdm5JVDZRd3dFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBd0l3Q2dZSUtvWkl6ajB\
FQXdJRFNBQXdSUUlnYzJ5NnhvT3RvUUJsSnNnbE9MMVZ4SEdvc1R5cEVxUmZ6MFF2NFp\
FUHY0d0NJUUNWeWIyRjl6VjNuOTUrb2xnZkZKZ1pUV0V6NGRTYUYzaHpSUWIzWnVCMjl\
RPT0iLCJNSUlCekRDQ0FYR2dBd0lCQWdJRVhYakhwREFLQmdncWhrak9QUVFEQWpBMU1\
STXdFUVlEVlFRS0RBcE5lVUoxYzJsdVpYTnpNUTB3Q3dZRFZRUUhEQVJUYVhSbE1ROHd\
EUVlEVlFRRERBWlVaWE4wUTBFd0hoY05NVGt3T1RFeE1UQXdPRE0yV2hjTk1qa3dPVEV\
4TVRBd09ETTJXakErTVJNd0VRWURWUVFLREFwTmVVSjFjMmx1WlhOek1RMHdDd1lEVlF\
RSERBUlRhWFJsTVJnd0ZnWURWUVFEREE5VVpYTjBVSFZ6YUUxdlpHVnNRMEV3V1RBVEJ\
```

```
nY3Foa2pPUFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVRsRzBmd1QzM29leloxdmtIUWJldGV\
ibWorQm9WK1pGc2pjZlF3MlRPa0pQaE9rT2ZBYnU5YlMxcVppOHlhRVY4b2VyS2wvNlp\
YYmZ4T21CanJScmNYbzJZd1pEQVNCZ05WSFJNQkFmOEVDREFHQVFIL0FnRUFNQTRHQTF\
VZER3RUIvd1FFQXdJQ0JEQWZCZ05WSFNNRUdEQVdnQlRvWklNelFkc0Qvai8rZ1gvN2N\
CSnVjSC9YbWpBZEJnTlZIUTRFRmdRVWI2RTliblh0bitpeEVJVk94eDQvcnlmM2V5TXd\
DZ1lJS29aSXpqMEVBd0lEU1FBd1JnSWhBUG5CMHcxTkN1cmhNeEp3d2ZqejdnRGlpeGt\
VWUxQU1o5ZU45a29oTlFVakFpRUF3NFk3bHR4V2lQd0t0MUo5bmp5ZkROObDVNdUVEQml\
teFIzQ1hvWktHUXJVPSJdfX0",
    "signatures":[{
      "protected":"eyJ4NWMiOlsiTUlJQitUQ0NBYUNnQXdJQkFnSUdBWG5WanNVN\
U1Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RBe\
EthVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJNQ\
0FYRFRJeE1EWXdOREExTkRZeE5Gb1lEems1T1RreE1qTXhNak0xT1RVNVdqQlNNUXN3Q\
1FZRFZRUUdFd0pCVVRFVk1CTUdBMVVFQ2d3TVNtbHVaMHBwYm1kRGIzSndNUk13RVFZR\
FZRUUZFd293TVRJek5EVTJOemc1TVJjd0ZRWURWUVFEREE1S2FXNW5TbWx1WjBSbGRtb\
GpaVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUFCQzc5bGlhUmNCalpjR\
UVYdzdyVWVhdnRHSkF1SDRwazRJNDJ2YUJNc1UxMWlMRENDTGtWaHRVVjIxbXZhS0N2T\
XgyWStTTWdROGZmd0wyM3ozVElWQldqZFRCek1Dc0dDQ3NHQVFVRkJ3RWdCQjhXSFcxa\
GMyRXRkR1Z6ZEM1emFXVnRaVzV6TFdKMExtNWxkRG81TkRRek1COEdBMVVkSXdRWU1CY\
UFGRlFMak56UFwvU1wva291alF3amc1RTVmdndjWWJNQk1HQTFVZEpRUU1NQW9HQ0NzR\
0FRVUZCd01DTUE0R0ExVWREd0VCXC93UUVBd0lIZ0RBS0JnZ3Foa2pPUFFRREFnTkhBR\
EJFQWlCdTN3UkJMc0pNUDVzTTA3MEgrVUZyeU5VNmdLekxPUmNGeVJST2xxcUhpZ0lnW\
ENtSkxUekVsdkQycG9LNmR4NmwxXC91eW1UbmJRRERmSmxhdHVYMlJvT0U9Il0sImFsZ\
yI6IkVTMjU2In0",
      "signature":"Y_ohapnmvbwjLuUicOB7NAmbGM7igBfpUlkKUuSNdG-eDI4BQ\
yuXZ2aw93zZId45R7XxAK-12YKIx6qLjiPjMw"
  }]
}
```


Figure 25: 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: 13257 bytes

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

{
  "payload":
    "eyJpZXRmLXZvdWNoZXItcmVxdWVzdC1wcm06dm91Y2hlciI6eyJhc3NlcnRpb24\
iOiJhZ2VudC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiY2FmZmUtOTg3NDUiLCJ\
ub25jZSI6ImM1VEVPb29NTE5hNEN4L1UrVExoQ3c9PSIsInByaW9yLXNpZ25lZC12b3V\
jaGVyLXJlcXVlc3QiOiJleUp3WVhsc2IyRmtJam9pWlhsS2NGcFlVbTFNV0ZwMlpGZE9\
iMXBZU1hSamJWWjRaRmRXZW1SRE1YZGpiVEEyWkcwNU1Wa3lhR3hqYVVrMlpYbEthR01\
6VG14amJsSndZakkwYVU5cFNtaGFNbFoxWkVNeGQyTnRPVFJoVnpGd1pFaHJhVXhEU25\
wYVdFcHdXVmQzZEdKdVZuUlpiVlo1U1dwdmFWa3lSbTFhYlZWMFQxUm5NMDVFVldsTVE\
wcDFZakkxYWxwVFNUWkpiVTB4VmtWV1VHSXlPVTVVUlRWb1RrVk9ORXd4VlhKV1JYaHZ\
VVE5qT1ZCVFNYTkpiVTU1V2xkR01GcFhVWFJpTWpScFQybEplVTFFU1hsTVZFRjVURlJ\
KZVZaRVFUTlBhazE2VDJwQk5FeHFSVFZPYkc5cFRFTkthRm95Vm5Wa1F6RjNZMjA1TW1\
GWFVteGFRekYzWTIwNU5HRlhNWEJrU0d0MFkyMVdibUZZVGpCamJVWjVURmRPYkdOdVV\
XbFBhVXBPVTFWc1JGSkdVa1JSTUVacFZESmtRbVF3YkVOUlYyUktVakJHV1ZkWVRuZFR\
WMVoyVkZWR2RsSXdUa1JqVldSVVZGUlJOVkZyUms1Uk1ERkhaRE5vUkdWclJrdFJiV1J\
QVm10S1FsZFdVa0poTUZwVFZGWktTbVF3VmtKWFZWSlhWVlpHVEZKRlJuTlViVlpXVkc\
1YWFWZEZTbTlaYlRWeVpVVmFWVkZXVWtOYU1EVlhVV3RHZWxSVlVrWk5WRlpXVFRGYWN\
GbDZTbk5oTWtaWVVtNXNiRlpGVmxGVVZVVjNVakJGZUZaVlZrTmtNMlJJVmtab2MxWkh\
SbGxWYlhoT1ZXdFdNMUpJWkZwU1JscFNWVlZTUlZGWGFFOWFWbHBQWTBkU1NGWnJVbEp\
XUlVac1VtNWpkMlZWTVVWU1dHeE9Va1pHTTFSdWNFcE5WVFZGWVVkR1IyUjZRalpVVlZ\
KR1pWVXhSVlZZWkU5bGEydDRWR3RTYjFsVk1VaFRXR2hFWld0R1MxRnRaRTlXYTBwQ1Y\
xWlNRbUV3V2xOVVZrcEtaREJXUWxkVlVsZFZWa1pNVWtWR2MxUnRWbFpVYmxwcFYwVkt\
iMWx0TlhKbFJWcEZVVlpPUTFvd05WZFJhMFo2VkZWTmQwMVVWbFpOTVZwd1dYcEtjMkV\
4YkZsVGFsWk9WVlJvTTFKR1JscFNSbHBTVlZWb1JWRldjRTlhVmxwUFkwZFNTRlpZYUV\
oU1JVWllVVzFrVDFaclNrSlVWVEZGVFVSRk1WWlVTbk5OUm5CWFUyMTRZVTF0ZURaYVJ\
XaExZVWRPY1ZGc2NFNVJhekZJVVc1c2VGSXhUazVPUkd4Q1dqQldTRkV3VG5oU01VNU9\
Ua1JzUW1Rd1ZrbFJWRUpLVVZWS1NsVklhRmhrVlRGSlVucG9TMVJIV2taVlJVVTFUMWh\
hZDAxdVZrbFNWVFZxVlROc2JWa3pWVE5VUjJSYVRraEJlRTVGUmtOT01GSk9XbFJGTkU\
xSVRrWmxSV1J4VkZOME0yTnJOVFJPTURVMVdWaE9lRXQ2Um5CTlJXUlVVVEZKTlU1VVV\
UTk5SRVp0VWpKV1dGUldSbWxqVjNCWVpXdEtZVlJWU1hkU01FVjRWbGRTUzFWV1JsaFV\
WVXBTVWpCT1JHTXdaRUpWVmxaSFVXNWtUbEZyU201YU0wcERXakJXUjFGc1JtcFNSV2h\
GVVZVNVExb3dOVmRUUmtVMFVXdEdiVTlGVmtOUlZURkVVV3BTUW1Rd2RFSlhWVkpYVld\
wQ1UxRnJUa1prTUdjd1UxZFNhVmRIZURaWlZtaFRZa2RPZEZadE5XaFhSVFIzV1RJeFI\
yVlZlSFJOVkZaYVRXcHNNRmt3WkVka1YxWlVUbGR3YVUxcVFqTlJNbVJhVTFWMGRsZHJ\
iRFpoYWtKR1VWaGtTbEpHVGtKUldHUlRWVlZzYjFGV1FqVlBXRnBOVTFkR01WVnJWbEp\
qYlhnMVlteGtNRlI2VmxOaVZYQXdZVmhTVW1GNlpFOVhSRkpMVlVoV2RWVklRazlVYXp\
BeFZWUnNRbUZWUm5sa1JVNTBWRVprWVZSVmJFMVdiRTEyVFZWc1JWbFhjR2xqTVU1Sll\
tNXdkbUpYYjNkV1F6a3paVmhKY21Nd2RFdGpRM1IxVFhwU1VsQlVNR2xNUTBwb1dqSld\
kV1JETVhwaFYyUjFXbGRSZEZwSFJqQlpVMGsyU1cxV05WTnVaRnBYUjNoNldXcEtSMkV\
3YkhGaU1teGhWMGQ0VEZrd1duZFhWbFowVFZVeFdGSnVRWGxYYTFwclZESkplR05HYkZ\
SWFJrcHhXV3hhWVU1R2NFZGFSbVJzWWxaS1JWUldhR3RoYlVwVlVWUktXRlp0VW5KWmE\
yUkxaRlpXV1ZWdGNFNWlXR2d4VjFjd2VGWXlSWGRsUm1oV1lsZG9jbFZxUWxkalJsRjV\
UbGh3YUZadGREWlZNakUwVjJ4a1IxTnVUbGhoTURFMFdrY3hTMk5HVGxWWGEzQm9ZVEo\
zZWxaR1pIZFNiVkpHVFZaV1UxZEdTazlXYTFwM1ZteFNWbFZyY0U5aGVrVXlWVlpTWVZ\
Sc1NrWlNha1pWVmxaS1ExcEVSbXRqUms1WlZHdHdhV0Y2Vm5wWFZFbDRZekpHU0ZOclV\
rNVhSbHB5Vm01d1IyTkdaSE5oUlhCb1ZsUnNkMVV5TVhkWGJGbDRZMGhTV0dKRk1UTlV\
iRlUxVWxac05sRnJPVlpOUnpneFYyMTRSbUZWZUVSVGJuQm9WakpTTVZkV2FGTk5WMDU\
```

wVm01d1NtRnVRbWxhV0d4TFpESk9kRTlVUW1GV01EUjNWMnhrVW1GVk9YQlRiWGhzVmx\
oQ2RsZFhkR3RoYlVaV1QxaENWR0V4Y0ZkYVYzUnlaVVpTZEdKRmNHcE5SM2d3V2tWb1E\
xbFdSWGRoZWtwVVZqTm9kbFZ0ZEhwbFZsWlpVMnhTYVdKclNrcFdhMVpUVVcxV2MxSnV\
VbFJpVlZwVlZXdGFjbVZzVFhwalJ6bFhUVlpHTmxkWWNFTmhiRWw1V1ROa1ZVMUdSak5\
aVm1SaFZXdHNjR1F5YkdwTmJYaDFXVzB4UjAxSFVsbFRiWGhLWVcwNWNGZEVRazlsYXp\
sV1QxWm9VMkpyY0dGWmJHTTFaV3hOZDA5WVZsSlhSMUpUVjFSR1YyRXhiM2RqUlZaWVV\
qRndUVmxzVWt0WFZUbFhVMnhXVjFJd05URlVSazE0WXpGUmQwNVdRbWxTZWxaMlZqSjB\
SazFGT1VaWGJYUlhZa2hCZDFReFVrOWhNbEY0Vld0d1YxWlVWbGRYUkVwaFYyczVWMWR\
1UWxkaVJHdzFWWHBPVDFaV1JuSmhSa3BRVmpOQ2Vsa3haSHBsVjBsNldUSnNiVlpxUlR\
WSmFYZHBXVmRrYkdKdVVYUmpNbXh1WW1reGFscFlTakJKYW5CaVNXc3hTbE5WVGt0U1J\
VNUVVmRPZUZvd1JqTlRWVXBDV2pCc1JsZEhlSEZSTURGRlVWVjBRMW95WkhoaFIzUnh\
WREZDVWxWVlVrSmhhMHB6VkZaR2VtUXdUbEpYVlZKWFZWWkdTRkpZWkV0UmJGWlZVbFp\
PVGxGclJraFJWRVpXVWxWT2JtUXdjRlZYUjNoRldXcEplR1F4YkZoT1ZGWk9WV3hXTTF\
KWVpGcFNSbHBTVlZWNFJWRllhRTlhVmxwUFRWWnNkVlJ1UW1GU01uaHZXVEkxY21WRlV\
qWlJWVFZEV2pBMVYxRnJSbXBVVlVweVRWUldWazF0ZDNkWGJGSkdXVlV4UTFvd1pFSk5\
WbFpHVVZoa00xVnNVbGxpUmxKb1YwWktjMVpWYUZkbGJVWkdUVmhhWVZJeFducFZWRUp\
HWkRCb2Ixa3dOVTVoYTBZelZGZHdTazVGTVVWWk0zQk9aV3RGZDFZeWFHcFVhekUyVVZ\
oa1RtRnJhekJVVlZKcVpXc3hObEZVUWxoaGEwcDBWRlpHZW1Rd1RsSlhWVkpYVlZaR1N\
GSllaRXRSYkZaVlVsWk9UbEZyUmtoUlZFWldVbFZPYm1Rd2NGVlhSM2hGV1dwSmVHUXh\
iRmhPVkZaT1ZXeFdNMUpZWkZwU1JscFNWVlY0UlZGWWFFOWFWbHBQVFZac2RWUnVRbUZ\
TTW5odldUSTFjbVZGVWpaUlZUVkRXakExVjFGclJtcFVWVXB5VFZSV1ZrMXRkM2RYYkZ\
KR1dXc3hRMkV3WkVKTlZsWkdVVmhrTTFVeFVsbGlSbEpvVjBaS2MxWlZhRmRsYlVaR1R\
WaGFZVkl4V25wVlZtaERaREF4UjJFelpFWmtNV3hKVXpJNVlWTlljSEZOUlU1Q1ZWWnN\
TbE15T1dGVFdIQnhUVVZTUWxWWFRrVlZWMlJDVWxSWmQwMVZPSEppTW5CRVlUTktSVlZ\
1WXpOYU1Hd3lWMnRWTUdGVVRUQmFSMHB2VVROR2NGSjZaSEZpTWprelYyNUJNR1ZJV2p\
aU2JsSk5XbnBhVlZaNlFtOVViVkpKWkd4Q1JWVXhVbnBrVm1oVVpWWmpOV1JJU1hwUld\
HUkVZa2RhUkdJd1VsZFVia1pRWkhwc05WUldaekpVYlRWT1VqRldNMUpIWkZwU1JscFR\
UVVpDUWxWVlozWlJhMFpTVWtWR2JscFZSazVSYW1oSVVWUkdWbHBGYkROVlZteE9VVzF\
HUWxKcmJ6TlRTRkpVWkROQ1RWUklWbEJYYW1ScVlUQkdjMVZWYUZaTk1tUkNWRmRqZGx\
Ock1VTk5SV1JDVFZaV2ExSkhaRkpXTUVwRFZXMU9WVTVVFRCaWF6RmFaR3hTYWxKdVV\
uSmFia295VGpOb1ZrNHdVbkJpVldoeFpXdEdWVkZ0WkU5V2EyaFVWbFZXUlZKRlJreFJ\
iV1J1WTJ0S2JsSlZXa05WVjA1RlVWZHdRbE13U201YU0wWnZZVEp3VUZWR1JsSlNSVVp\
1Vkd0c1FsSkZTa2RSVjJ4R1VWaENTMDR6YUhkVWJGWXlWVlZ3U0UxRk5XOVVSMGwyV2x\
oU2FVMXFRazFTUmxWNFRtMTRkMVV3YUZCT01rWnNZbnBDVjFkWVozZGxTR1JFVTFWRmN\
sUjZWWFpYVkZwRllVTjBhVkZxU1RSTmFsSXhZVmRHVUZWWFJsWlNSRnB1VVZVMWIxZFV\
iRmRTYlZGeVlXNUtlVmt3VmpKVGJsRnBURU5LVGxOVmJFUlNNVkpFVVRCR2FVc3laRUp\
rTUd4RFVWZGtTbEpXYUhOaGEwVjJaV3RHVEZGdFpHNWpWMmh5WVdzNVVWVldSa1ZSVjN\
CRFdUQXhVbU16WkVSVlZteEZWbXhHVWxJd1ZqTlRhMHBXVmtWV1ZGUlZTa0pTTUVWNFZ\
sVldSRm96WkV0V1JtaHpVa2RKZVUxWVpGcFdlbFV4VkZaS1ZtUXdWak5YVlZKWFZWWkd\
UVkpGUmpSVWJWWlhWR3BHV21Kck5YZFhhMlJ6WVVkT2RXRXphRVZsYTBaUFVXMWtUMVp\
yU2tKWk1ERkRZWHBGTVZaVVNuTk5SbkJWVWxaS1RsRlVhRWhSVkVaV1VsVkdNMlF3YkZ\
WWFIzaFZXVlpvVTJKR1JYZFNXR1JKWVVkT1QxUlhjRUprTURGeFUxUlNUbEpIVGpWVWJ\
uQldUbFprYjFrd05VNWxhMFl6VkZkd1NrNUZNVVZaTTJ4UFpXeFZNVll5Y0VOaVJURlN\
Zek5rUkZWV2JFVldiRVpTVWpCV00xTnJTbFpXUlZaVVZGVktRbEl3UlhoV1ZWWkVXak5\
rUzFaR2FITlNSMGw1VFZoa1dsWjZWVEZVVmtwV1pEQldNMWRWVWxkVlZrWk5Va1ZHTkZ\
SdFZsZFVha1phWW1zMWQxZHJaSE5oUjA1MVlUTm9SV1ZyUms5UmJXUlBWbXRLUWxrd01\
VTmhla1V4VmxSS2MwMUdjRlZTVjBaT1VXMWtTRkZVUmxaU1ZVWXpaREZLVlZkSGVGVlp\
WbWhUWWtaV1NWWnVjR2hTVkVZeVYydGtWMk14UlhkU1dHUllWa1ZHVlZGdFpHcGpWMmh\
5WVdzNVVWVlZiRU5SYldSdVkxZG9jbUZyT1ZGVlZURkRVVzVrVDFFd1JrSlZhM0JEVm0\

```
wNWVscEZkRE5YVlRVMFlWWkNORk5JV25CU2JrWk1aV3RTYzA5WFdqQlVTRlpPV1ZjeGQ\
xSnNSbXBYU0dONFRXcGthRlJ0T1ZOWmJrNUpUREJhVG1OdE1UWlJNRVpKVFhwak0wMTZ\
UbXBOYlRscFZVZE9jMlJzVG5sWFZVb3lUVVZPTUZZeFJqQlpWRnBvU3pKT2RrMXNiRE5\
YYTFKQ1ZUQktibFJzV2tsVmF6RkRVVmRaTkZKVlRrVlJWV1JDVlZWbmRsRlhaRVpSVlR\
GQ1RrVmtRazFXVm10U1NHUkdVV2s1TTFWVlZrSmtNR3hFVVd0U1FscHJTbTVVYkZwSlZ\
UQXhSbEl3VWtKV01tUkRWVmh3TkdWdVpIZFZia0pOWlZNNWVWUldWbHBsYlVadlRXNU5\
lRTB5VmxaUFYyUkhaV3RHYTFGdFpFOVdhMmhTVGtWV1Ixb3hSbFppYms1c1RWVjRSR0V\
6VGpGT1JGWjFaRWhzVWxFeFdrSmFSbEpzVVZWR05WSkVhSEprTUU1dVYxVnNUR0l4Y0V\
wbGJXOTNVbFZHTTFOVlVsUlJWVVl6Vld4R1NtRkZSa3BqTVd4eldsWndUR015Y0VkVWE\
wNTZVMnQwYkZWSGVFaFVWVVpOV2xoQ1YxcFViRVpVUkdSUFlqTlJNVTFVVmpObFJ6Rlh\
aRlZ3UTFGWGJFSlpNRlpPVmxaV2IxSldUbnBVUm1SUlRsaG9WRlZXVlhkWFNFWTJWbTV\
GTkZkWVduQlNha1pwVm0wNU5sSXpjRFJPV0hCUFdqSk9lbVI2TURsSmJERTVabEVpTEN\
KemFXZHVZWFIxY21WeklqcGJleUp3Y205MFpXTjBaV1FpT2lKbGVVbzBUbGROYVU5c2M\
ybFVWV3hLVVRCb2NWRXdUa0paTVU1dVVWaGtTbEZyUm01VFZXUkNWMGRvTUUxVVVucGl\
NREZDWWpCa1JGRXpSa2hWTURBd1QxVktRbFJWVGs1U1ZtdzBVVE53UWxOclNtNVViRnB\
EVVZac1ZWRlhkRTlUVlRGVFZGaGtSbFZXYkVWV2JFWlNVekJTUW1OR1VtaFdNVm93VjJ\
4ak1XVnJiRVpTYTJoT1ZWUm9NMUpHUmxwU1JscFNWVlY0UlZGV2NFUldhMDVEVWtaV1I\
xUllhRVpXUlVaUlVXMWtUMVpyU2tKVVZURkVVbXh3YzFsdE1WTmtiVTV5Vkd0S1RsRXd\
SbGxTUmxKS1pVVXhSVlJZYkU5aGEwVXhWRmR3U21WVk5WZGlNV3hGWlcxek1WUXhVbkp\
sUlRGeFZGaG9UbUZyTUhoVU1WSldUbFprY1ZGdGFFNVZXRTR6VVRGR1dsSkdXbEpWVld\
SR1pEQndSVlV3VWtaV1JURkRVbFZrUWsxV1ZrWlJNbVF6VXpGVmVXSkhlR2xXTVZveFd\
UTnNRMUZzU2paU1ZrSk9VVlJDU0ZGVVJsWlNWVTR6WkRCa1VtSkdSbTVWVkVaRFZrVXh\
VMVZZWkVaYU1XeEZWbXhHVWxKclZqTmtSM0JhVmpGd2RGZHNUWGRPVlRsRldYcENUMVp\
GVmxoVVZVcFNVakJGZUZaVlZrSmtNMlJQVmxWYWIxSkZNVFZPVlZwUFpXeFdNRlJXVWt\
Ka01VWlZVV3h3VGxGck1VaFJibXg0VWpGT1RrNUViRUphTUZaSVVUQk9lRkl4VGs1T1J\
HeENaREJXU1ZGVVFrcFJWVXBQVFVSb2NWWXdlSHBOUjBadlV6Qm9XbGR1Vm05aVZ6Rmp\
UREpqTkdScVVsaFRNR2d5VVZoU2FGcHNSazFSVTNSS1pGVXhUMkZITVc1aFZ6RllUakJ\
HVG1OdVJtOWlWMHBWVFRCc2FGVkZUalZoUnpGaFUxWk9kMVI2V20xaVUzTXlVMWhhWTB\
3eldrcGphM1J2VlZaU01WWnRPVXhoYldSYVVWaGtiV0ZyUm5aUmJXUnVZMnRLYmxKVld\
rTlZWMDVEVTFWR1Vsa3dVa05qU0ZKYVYwVTFiMVJHYUZOaVIwMTZWVmhXYWsxdGVITlp\
iR1JYWkZkT05VNVhjR2xOYWtFeVZERlNVazFGTVRaUlZsSkRXakExVjFOR1RsWlNWVkp\
GVVZWMFExb3laSGxSYldSR1VtdEtVbGt3VWtKV1JVWlFVVzFrVDFacmFGSlBSVXBDV21\
wb1JsRnJSazVSTUVrd1VWaGtUVlZXYkVWV2JFbDNWV3RLUkZkWVpFdFRWV3h3V2tWa2I\
ySkZlSFZYYlhocFlsWktNbGt5YXpGaGJHeFlUa2hXYVdKVWEzZFVSekV3WkZkSmVsa3p\
WbXRTTW1oM1dUTnJNVTFzYkZobFJFWmhWa1ZHVEZGdFpHNWpWMmh5WVdzNVVWVldSa1Z\
SVjJSUFUxVkdSVkZyV2tKaFZVazFVVzB4ZUZRemNIRlZWR2hzV1ZkamVWTnVVblprVmx\
KdlVsWm9lVm93T1VOWFZsRjNVWHBvWVdSRGREVlBWemxKVWtad1JWbHNVbEpUVjJoQ1Z\
HMXpNbVJIT1ZOaU1sWkVXVmMxYUZSWGNFNVdSWGgwWWxaV2RXSlhTbkphYWtKNldsaGF\
jbEV3YnpSTmJXc3hWbGhHY1ZWcldsZFZVMHBrVEVOS2FHSkhZMmxQYVVwR1ZYcEpNVTV\
wU2praUxDSnphV2R1WVhSMWNtVWlPaUphWTFwa1dYbzBiMUl3UjJKc09UWnFNWGxZWm5\
kdlRYZGxVVGt6VGpCdFNVUmxjVFkyVTBacWRFdG9lR1pSWjNJMGRUWkpOVEJKWldNMmE\
xWTJhSEV3YVcxdlptTlBhVGs0VW1OSVpXUmpNVzFuZHpCWVp5SjlYWDA9IiwiY3JlYXR\
lZC1vbiI6IjIwMjItMDItMjJUMDc6MzM6MjUuMDIwWiIsImFnZW50LXNpZ24tY2VydCI\
6WyJNSUlDSkRDQ0FjcWdBd0lCQWdJRVhsakNNREFLQmdncWhrak9QUVFEQWpCbE1Rc3d\
DUVlEVlFRR0V3SkJVVEVTTUJBR0ExVUVDZ3dKVFhsRGIyMXdZVzU1TVJVd0V3WURWUVF\
MREF4TmVWTjFZbk5wWkdsaGNua3hEekFOQmdOVkJBY01CazE1VTJsMFpURWFNQmdHQTF\
VRUF3d1JUWGxUYVhSbFVIVnphRTF2WkdWc1EwRXdIaGNOTWpBd01qSTRNRGN6TXpBMFd\
oY05NekF3TWpJNE1EY3pNekEwV2pCbU1Rc3dDUVlEVlFRR0V3SkJVVEVTTUJBR0ExVUV\
DZ3dKVFhsRGIyMXdZVzU1TVJVd0V3WURWUVFMREF4TmVWTjFZbk5wWkdsaGNua3hEekF\
```

```
OQmdOVkJBY01CazE1VTJsMFpURWJNQmtHQTFVRUF3d1NUWGxUYVhSbFVIVnphRTF2Wkd\
Wc1FYQndNRmt3RXdZSEtvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFnQUU2MDFPK29qQ2t\
yRFJ3N2dJdlpFNGkzNGRiaENxaUc3am9vd1pwNHh2ekZ0TGc2VFcwaE5kSHZQRFNUc3V\
YU3lXOXRyM0F3Q2xmQ29EVk5xT3c5eU1YNk5uTUdVd0RnWURWUjBQQVFIL0JBUURBZ2V\
BTUI4R0ExVWRJd1FZTUJhQUZKN0h0U3dwTEx1T1o3Y2tBbFFIVTNnQU1nL0pNQjBHQTF\
VZERnUVdCQlJjVDUzNG5NWXZUY0Z0a2Zydjd4VTdEaW1IanpBVEJnTlZIU1VFRERBS0J\
nZ3JCZ0VGQlFjREFqQUtCZ2dxaGtqT1BRUURBZ05JQURCRkFpRUFwSjd4cE5VdlFKRzB\
OaExiL2V0YjIwTERVMTZscFNITzdhZW8wVll4MHh3Q0lBK081L1k2RGgrYkIyODI0dWl\
hT1FhVUQ2Z0FOaFk5VkZkK2pycmNFdkp0IiwiTUlJQ0dUQ0NBYitnQXdJQkFnSUVYbGp\
BL3pBS0JnZ3Foa2pPUFFRREFqQmNNUXN3Q1FZRFZRUUdFd0pCVVRFU01CQUdBMVVFQ2d\
3SlRYbERiMjF3WVc1NU1SVXdFd1lEVlFRTERBeE5lVk4xWW5OcFpHbGhjbmt4RHpBTkJ\
nTlZCQWNNQmsxNVUybDBaVEVSTUE4R0ExVUVBd3dJVFhsVGFYUmxRMEV3SGhjTk1qQXd\
Nakk0TURjeU56VTVXaGNOTXpBd01qSTRNRGN5TnpVNVdqQmxNUXN3Q1FZRFZRUUdFd0p\
CVVRFU01CQUdBMVVFQ2d3SlRYbERiMjF3WVc1NU1SVXdFd1lEVlFRTERBeE5lVk4xWW5\
OcFpHbGhjbmt4RHpBTkJnTlZCQWNNQmsxNVUybDBaVEVhTUJnR0ExVUVBd3dSVFhsVGF\
YUmxVSFZ6YUUxdlpHVnNRMEV3V1RBVEJnY3Foa2pPUFFJQkJnZ3Foa2pPUFFNQkJ3TkN\
BQVJKQlZvc2RLd1lOeGlQeEh2aUZxS3pEbDlmdEx1TWFtcEZRY1h3MTI3YU5vUmJzSC9\
GTXJtekNBSDM3NzMzYzJvYlBjbHZTcllCdjBDdFdRdGE2YStjbzJZd1pEQVNCZ05WSFJ\
NQkFmOEVDREFHQVFIL0FnRUFNQTRHQTFVZER3RUIvd1FFQXdJQ0JEQWZCZ05WSFNNRUd\
EQVdnQlF6eHp3cFJwTHkvck1VWXphaDJzMTNlVTlnRnpBZEJnTlZIUTRFRmdRVW5zZTF\
MQ2tzdTQ1bnR5UUNWQWRUZUFBeUQ4a3dDZ1lJS29aSXpqMEVBd0lEU0FBd1JRSWhBSXN\
ZbGVaS3NqRk5Dc0pLZVBsR01BTGVwVmU5RUw3Tm90NTE1d3htVnVKQkFpQWNFTVVVaEV\
Tc0xXUDV4U1FVMFhxelZxOFl2aUYxYlZvekd6eDV6Tmdjc3c9PSJdfX0",
  "signatures":[{
    "protected":"eyJ4NWMiOlsiTUlJQjhEQ0NBWmFnQXdJQkFnSUdBWEJoTUtZSU1\
Bb0dDQ3FHU000OUJBTUNNRnd4Q3pBSkJnTlZCQVlUQWtGUk1SSXdFQVlEVlFRS0RBbE5\
lVU52YlhCaGJua3hGVEFUQmdOVkJBc01ERTE1VTNWaWMybGthV0Z5ZVRFUE1BMEdBMVV\
FQnd3R1RYbFRhWFJsTVJFd0R3WURWUVFEREFoTmVWTnBkR1ZEUVRBZUZ3MHlNREF5TWp\
Bd05qQXlNak5hRncwek1EQXlNakF3TmpBeU1qTmFNSGt4Q3pBSkJnTlZCQVlUQWtGUk1\
SSXdFQVlEVlFRS0RBbE5lVU52YlhCaGJua3hGVEFUQmdOVkJBc01ERTE1VTNWaWMybGt\
hV0Z5ZVRFUE1BMEdBMVVFQnd3R1RYbFRhWFJsTVM0d0xBWURWUVFERENWU1pXZHBjM1J\
5WVhJZ1ZtOTFZMmhsY2lCU1pYRjFaWE4wSUZOcFoyNXBibWNnUzJWNU1Ga3dFd1lIS29\
aSXpqMENBUVlJS29aSXpqMERBUWNEUWdBRUJUVFwvc1JmTDlsSnVGbVwvY24zU2pHcWp\
QXC9xdnBrNytoSTIwOE5oVkRvR2hcLzdLUCt2TXpYeVFRQStqUjZrNnJhTTI4ZlwvbHV\
FK1h1aHVwN1Vmem05Q3FNbk1DVXdFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUhBeHd3RGd\
ZRFZSMFBBUUhcL0JBUURBZ2VBTUFvR0NDcUdTTTQ5QkFNQ0EwZ0FNRVVDSUhOK3VBbUp\
ldVhlc1wveWQxd1M0Mlo0RFhKNEptMWErZzNYa1pnZjhUaGxuQWlFQXBVdTZzZnljRWt\
veDdSWlhtZitLOHc0cDZzUldyamExUVJwWTAyNjQxSFk9IiwiTUlJQjhEQ0NBWmVnQXd\
JQkFnSUdBWEJoTUtZQk1Bb0dDQ3FHU000OUJBTUNNRnd4Q3pBSkJnTlZCQVlUQWtGUk1\
SSXdFQVlEVlFRS0RBbE5lVU52YlhCaGJua3hGVEFUQmdOVkJBc01ERTE1VTNWaWMybGt\
hV0Z5ZVRFUE1BMEdBMVVFQnd3R1RYbFRhWFJsTVJFd0R3WURWUVFEREFoTmVWTnBkR1Z\
EUVRBZUZ3MHlNREF5TWpBd05qQXlNak5hRncwek1EQXlNakF3TmpBeU1qTmFNRnd4Q3p\
BSkJnTlZCQVlUQWtGUk1SSXdFQVlEVlFRS0RBbE5lVU52YlhCaGJua3hGVEFUQmdOVkJ\
Bc01ERTE1VTNWaWMybGthV0Z5ZVRFUE1BMEdBMVVFQnd3R1RYbFRhWFJsTVJFd0R3WUR\
WUVFEREFoTmVWTnBkR1ZEUVRCWk1CTUdCeXFHU000OUFnRUdDQ3FHU000OUF3RUhBMEl\
BQkluQ3VoV1ZzZ2NONzFvWmVzMUZHXC9xZFZnTVBva01wZlMyNzFcL2V5SWNcL29EVmJ\
NRkhDYmV2SjVMTTgxOTVOYVpLTlNvSFAzS3dMeXVCaDh2MncwOVp1alJUQkRNQklHQTF\
VZEV3RUJcL3dRSU1BWUJBZjhDQVFFd0RnWURWUjBQQVFIXC9CQVFEQWdJRU1CMEdBMVV\
kRGdRV0JCUXp4endwUnBMeVwvck1VWXphaDJzMTNlVTlnRnpBS0JnZ3Foa2pPUFFRREF\
```

```
nTkhBREJFQWlCZGJIU212YW9qaDBpZWtaSUtOVzhRMGxTbGI1K0RLTlFcL05LY1I3dWx\
6dGdJZ2RwejZiUkYyREZtcGlKb3JCMkd5VmE4YVdkd2xIc0RvRVdZY0k0UEdKYmc9Il0\
sImFsZyI6IkVTMjU2In0",
    "signature":"67t3n8zyEek4IM2Ko3Y_UvE1hzp794QFNTqG-HzTrBQtE4_4-yS\
yyFd3kP6YCn35YYJ7yK35d3styo_yoiPfKA"
  }]
}
```


Figure 26: Example Registrar Voucher-Request - RVR

A.3. Example Voucher-Response (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\
CVG9aSU16UWRzRC9qLytnWC83Y0JKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQkd\
BaUVBdHhRMytJTEdCUEl0U2g0YjlXWGhYTnVocVNQNkgrYi9MQy9mVllEalE2b0NJUUR\
HMnVSQ0hsVnEzeWhCNThUWE1VYnpIOCtPbGhXVXZPbFJEM1ZFcURkY1F3PT0ifX0",
  "signatures":[{
    "protected":"eyJ4NWMiOlsiTUlJQmt6Q0NBVGlnQXdJQkFnSUdBV0ZCakNrWU1\
Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RBeEt\
hVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJNQjR\
YRFRFNE1ERXlPVEV3TlRJME1Gb1hEVEk0TURFeU9URXdOVEkwTUZvd1R6RUxNQWtHQTF\
VRUJoTUNRVkV4RlRBVEJnTlZCQW9NREVwcGJtZEthVzVuUTI5eWNERXBNQ2NHQTFVRUF\
3d2dTbWx1WjBwcGJtZERiM0p3SUZadmRXTm9aWElnVTJsbmJtbHVaeUJMWlhrd1dUQVR\
CZ2NxaGtqT1BRSUJCZ2dxaGtqT1BRTUJCd05DQUFTQzZiZUxBbWVxMVZ3NmlRclJzOFI\
wWlcrNGIxR1d5ZG1XczJHQU1GV3diaXRmMm5JWEgzT3FIS1Z1OHMyUnZpQkdOaXZPS0d\
CSEh0QmRpRkVaWnZiN294SXdFREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdDZ1lJS29aSXp\
qMEVBd0lEU1FBd1JnSWhBSTRQWWJ4dHNzSFAyVkh4XC90elVvUVwvU3N5ZEwzMERRSU5\
FdGNOOW1DVFhQQWlFQXZJYjNvK0ZPM0JUbmNMRnNhSlpSQWtkN3pPdXNuXC9cL1pLT2F\
FS2JzVkRpVT0iXSwiYWxnIjoiRVMyNTYifQ",
    "signature":"0TB5lr-cs1jqka2vNbQm3bBYWfLJd8zdVKIoV53eo2YgSITnKKY\
TvHMUw0wx9wdyuNVjNoAgLysNIgEvlcltBw"
  }]
}
```

Figure 27: Example Voucher-Response from MASA

A.4. Example Voucher-Response, 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: 3006 bytes

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

{
  "payload":"eyJpZXRmLXZvdWNoZXI6dm91Y2hlciI6eyJhc3NlcnRpb24iOiJhZ2V\
udC1wcm94aW1pdHkiLCJzZXJpYWwtbnVtYmVyIjoiMDEyMzQ1Njc4OSIsIm5vbmNlIjo\
iUUJiSXMxNTJzbkFvVzdSeVFMWENvZz09IiwiY3JlYXRlZC1vbiI6IjIwMjItMDktMjl\
UMDM6Mzc6MjYuMzgyWiIsInBpbm5lZC1kb21haW4tY2VydCI6Ik1JSUJwRENDQVVtZ0F\
3SUJBZ0lHQVcwZUx1SCtNQW9HQ0NxR1NNNDlCQU1DTURVeEV6QVJCZ05WQkFvTUNrMTV\
RblZ6YVc1bGMzTXhEVEFMQmdOVkJBY01CRk5wZEdVeER6QU5CZ05WQkFNTUJsUmxjM1J\
EUVRBZUZ3MHhPVEE1TVRFd01qTTNNekphRncweU9UQTVNVEV3TWpNM016SmFNRFV4RXp\
BUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERUQUxCZ05WQkFjTUJGTnBkR1V4RHpBTkJ\
nTlZCQU1NQmxSbGMzUkRRVEJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUF\
CT2t2a1RIdThRbFQzRkhKMVVhSTcrV3NIT2IwVVMzU0FMdEc1d3VLUURqaWV4MDYvU2N\
ZNVBKaWJ2Z0hUQitGL1FUamdlbEhHeTFZS3B3Y05NY3NTeWFqUlRCRE1CSUdBMVVkRXd\
FQi93UUlNQVlCQWY4Q0FRRXdEZ1lEVlIwUEFRSC9CQVFEQWdJRU1CMEdBMVVkRGdRV0J\
CVG9aSU16UWRzRC9qLytnWC83Y0JKdWNIL1htakFLQmdncWhrak9QUVFEQWdOSkFEQkd\
BaUVBdHhRMytJTEdCUEl0U2g0YjlXWGhYTnVocVNQNkgrYi9MQy9mVllEalE2b0NJUUR\
HMnVSQ0hsVnEzeWhCNThUWE1VYnpIOCtPbGhXVXZPbFJEM1ZFcURkY1F3PT0ifX0",
  "signatures":[{
    "protected":"eyJ4NWMiOlsiTUlJQmt6Q0NBVGlnQXdJQkFnSUdBV0ZCakNrWU1\
Bb0dDQ3FHU000OUJBTUNNRDB4Q3pBSkJnTlZCQVlUQWtGUk1SVXdFd1lEVlFRS0RBeEt\
hVzVuU21sdVowTnZjbkF4RnpBVkJnTlZCQU1NRGtwcGJtZEthVzVuVkdWemRFTkJNQjR\
YRFRFNE1ERXlPVEV3TlRJME1Gb1hEVEk0TURFeU9URXdOVEkwTUZvd1R6RUxNQWtHQTF\
VRUJoTUNRVkV4RlRBVEJnTlZCQW9NREVwcGJtZEthVzVuUTI5eWNERXBNQ2NHQTFVRUF\
3d2dTbWx1WjBwcGJtZERiM0p3SUZadmRXTm9aWElnVTJsbmJtbHVaeUJMWlhrd1dUQVR\
CZ2NxaGtqT1BRSUJCZ2dxaGtqT1BRTUJCd05DQUFTQzZiZUxBbWVxMVZ3NmlRclJzOFI\
wWlcrNGIxR1d5ZG1XczJHQU1GV3diaXRmMm5JWEgzT3FIS1Z1OHMyUnZpQkdOaXZPS0d\
CSEh0QmRpRkVaWnZiN294SXdFREFPQmdOVkhROEJBZjhFQkFNQ0I0QXdDZ1lJS29aSXp\
qMEVBd0lEU1FBd1JnSWhBSTRQWWJ4dHNzSFAyVkh4XC90elVvUVwvU3N5ZEwzMERRSU5\
FdGNOOW1DVFhQQWlFQXZJYjNvK0ZPM0JUbmNMRnNhSlpSQWtkN3pPdXNuXC9cL1pLT2F\
FS2JzVkRpVT0iXSwidHlwIjoidm91Y2hlci1qd3MranNvbiIsImFsZyI6IkVTMjU2In0\
",
    "signature":"ShqW8uFRkuAXIzjAhB4bolMMndcY7GYq3Kbo94yvGtjCaxEX3Hp\
6QXZUTEJ_kulQ1G7DnaU4igDPdUGtcV9Lkw"},{
    "protected":"eyJ4NWMiOlsiTUlJQjRqQ0NBWWlnQXdJQkFnSUdBWFk3MmJiWk1\
Bb0dDQ3FHU000OUJBTUNNRFV4RXpBUkJnTlZCQW9NQ2sxNVFuVnphVzVsYzNNeERUQUx\
CZ05WQkFjTUJGTnBkR1V4RHpBTkJnTlZCQU1NQmxSbGMzUkRRVEFlRncweU1ERXlNRGN\
3TmpFNE1USmFGdzB6TURFeU1EY3dOakU0TVRKYU1ENHhFekFSQmdOVkJBb01DazVFTlU\
WemFXNWxjM014RFRBTEJnTlZCQWNNQkZOcGRHVXhHREFXQmdOVkJBTU1EMFJ2YldGcGJ\
sSmxaMmx6ZEhKaGNqQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VIQTBJQUJCazE\
2S1wvaTc5b1JrSzVZYmVQZzhVU1I4XC91czFkUFVpWkhNdG9rU2RxS1c1Zm5Xc0JkK3F\
STDdXUmZmZVdreWdlYm9KZklsbHVyY2kyNXduaGlPVkNHamV6QjVNQjBHQTFVZEpRUVd\
NQlFHQ0NzR0FRVUZCd01CQmdnckJnRUZCUWNESERBT0JnTlZIUThCQWY4RUJBTUNCNEF\
3U0FZRFZSMFJCRUV3UDRJZGNtVm5hWE4wY21GeUxYUmxjM1F1YzJsbGJXVnVjeTFpZEM\
1dVpYU0NIbkpsWjJsemRISmhjaTEwWlhOME5pNXphV1Z0Wlc1ekxXSjBMbTVsZERBS0J\
nZ3Foa2pPUFFRREFnTklBREJGQWlCeGxkQmhacTBFdjVKTDJQcldDdHlTNmhEWVcxeUN\
PXC9SYXVicEM3TWFJRGdJaEFMU0piZ0xuZ2hiYkFnMGRjV0ZVVm9cL2dHTjBcL2p3ekp\
aMFNsMmg0eElYazEiXSwidHlwIjoidm91Y2hlci1qd3MranNvbiIsImFsZyI6IkVTMjU\
2In0",
```

```
    "signature":"N4oXV48V6umsHMKkhdSSmJYFtVb6agjD32uXpIlGx6qVE7Dh0-b\
qhRRyjnxp80IV_Fy1RAOXIIzs3Q8CnMgBgg"
  }]
}
```

Figure 28: 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 dependant upon the security of the TLS connection.

The use of HTTP-over-TLS is not mandated by this document for a number of 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 no help as there is no previous relationship between the Registrar-Agent.

2. The pledge can use it's IDevID certificate to authenticate itself, but [RFC6125] DNS-ID methods do not apply as the pledge does not have a FQDN. Instead a new mechanism is required, which authenticates the X520SerialNumber DN attribute which must be present in every IDevID.

If the Registrar-Agent has a preconfigured 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 QRcodes may help automate this in some cases.

1. The CA used to sign the IDevID will be a manufacturer private PKI as described in [I-D.irtf-t2trg-taxonomy-manufacturer-anchors], Section 4.1. 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 it's own certificate chain verification against a store of manufacturer trust anchors. In the Android ecosystem this involved 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]

2. The use of the Host: (or :authority in HTTP/2) is explained in [RFC9110], Section 7.2. This header is mandatory, and so a compliant HTTPS client is going to insert it. But, the contents of this header will at best be an IP address that came from the discovery process. The pledge **MUST** therefore ignore the Host: header when it processes requests, and the pledge **MUST NOT** do any kind of name-base virtual hosting using the IP address/port combination. Note that there is no requirement for the pledge to operate it's BRSKI-PRM service on port 80 or port 443, so if there is no reason for name-based virtual hosting.

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

Appendix C. History of Changes [RFC Editor: please delete]

Proof of Concept Code available

From IETF draft 10 -> IETF draft 11:

*issue #79, clarified that BRSKI discovery in the context of BRSKI-PRM is not needed in Section 5.6.1.

*issue #103, removed step 6 in verification handling for the wrapped CA certificate provisioning as only applicable after enrollment Section 6.3.3

*issue #128: included notation of nomadic operation of the Registrar-Agent in Section 5, including proposed text from PR #131

*issue #130, introduced DNS service discovery name for brski_pledge to enable discovery by the Registrar-Agent in Section 8

*removed unused reference RFC 5280

*removed site terminology

*deleted duplicated text in Section 5.5

* clarified registrar discovery and relation to BRSKI-Discovery in Section 5.6.1

* clarified discovery of pledges by the Registrar-Agent in Section 5.6.2, deleted reference to GRASP as handled in BRSKI-Discovery

* addressed comments from SECDIR early review

From IETF draft 09 -> IETF draft 10:

* issue #79, clarified discovery in the context of BRSKI-PRM and included information about future discovery enhancements in a separate draft in Section 5.6.1.

* issue #93, included information about conflict resolution in mDNS and GRASP in Section 5.6.2

* issue #103, included verification handling for the wrapped CA certificate provisioning in Section 6.3.3

* issue #106, included additional text to elaborate more the registrar status handling in Section 6.3.6

* issue #116, enhanced DoS description in Section 10.1

* issue #120, included statement regarding pledge host header processing in Section 5.5

* issue #122, availability of product-serial-number information on registrar agent clarified in Section 6.1

* issue #123, Clarified usage of alternative voucher formats in Section 6.2.3

* issue #124, determination of pinned domain certificate done as in RFC 8995 included in Section 6.2.4

* issue #125, remove strength comparison of voucher assertions in Section 5.4 and Section 6

* issue #130, aligned the usage of site and domain throughout the document

* changed naming of registrar certificate from LDevID(RegAgt) to EE (RegAgt) certificate throughout the document

* change x5b to x5bag according to [RFC9360]

* updated JSON examples -> "signature": BASE64URL(JWS Signature)

From IETF draft 08 -> IETF draft 09:

- *issue #80, enhanced Section 5.6.2 with clarification on the product-serial-number and the inclusion of GRASP

- *issue #81, enhanced introduction with motivation for agent_signed_data

- *issue #82, included optional TLS protection of the communication link between Registrar-Agent and pledge in the introduction Section 4, and Section 6.1

- *issue #83, enhanced Section 6.1.4 and Section 6.2.6 with note to re-enrollment

- *issue #87, clarified available information at the Registrar-Agent in Section 6.1

- *issue #88, clarified, that the PVR in Section 6.1.2 and PER in Section 6.1.4 may contain the certificate chain. If not contained it **MUST** be available at the registrar.

- *issue #91, clarified that a separate HTTP connection may also be used to provide the PER in Section 6.2.6

- *resolved remaining editorial issues discovered after WGLC (responded to on the mailing list in Reply 1 and Reply 2) resulting in more consistent descriptions

- *issue #92: kept separate endpoint for wrapped CSR on registrar Section 6.2.7

- *issue #94: clarified terminology (possess vs. obtained)

- *issue #95: clarified optional IDevID CA certificates on Registrar-Agent Section 6.3

- *issue #96: updated Figure 16 to correct to just one CA certificate provisioning

- *issue #97: deleted format explanation in Section 6.3 as it may be misleading

- *issue #99: motivated verification of second signature on voucher in Section 6.3

- *issue #100: included negative example in Figure 17

- *issue #101: included handling if Section 6.3.2 voucher telemetry information has not been received by the Registrar-Agent

- *issue #102: relaxed requirements for CA certs provisioning in Section 6.3.3

- *issue #105: included negative example in Figure 18

- *issue #107: included example for certificate revocation in Section 6.3.6

- *issue #108: renamed heading to Pledge-Status Request of Section 6.4.1

- *issue #111: included pledge-status response processing for authenticated requests in Section 6.4.2

- *issue #112: added "Example key word in pledge-status response in Figure 24

- *issue #113: enhanced description of status reply for "factory-default" in Section 6.4.2

- *issue #114: Consideration of optional TLS usage in Privacy Considerations

- *issue #115: Consideration of optional TLS usage in Privacy Considerations to protect potentially privacy related information in the bootstrapping like status information, etc.

- *issue #116: Enhanced DoS description and mitigation options in security consideration section

- *updated references

From IETF draft 07 -> IETF draft 08:

- *resolved editorial issues discovered after WGLC (still open issues remaining)

- *resolved first comments from the Shepherd review as discussed in PR #85 on the ANIMA github

From IETF draft 06 -> IETF draft 07:

- *WGLC resulted in a removal of the voucher enhancements completely from this document to RFC 8366bis, containing all enhancements and augmentations of the voucher, including the voucher-request as well as the tree diagrams

- *smaller editorial corrections

From IETF draft 05 -> IETF draft 06:

- Update of list of reviewers
- Issue #67, shortened the pledge endpoints to prepare for constraint deployments
- Included table for new endpoints on the registrar in the overview of the Registrar-Agent
- addressed review comments from SECDIR early review (terminology clarifications, editorial improvements)
- addressed review comments from IOTDIR early review (terminology clarifications, editorial improvements)

From IETF draft 04 -> IETF draft 05:

- Restructured document to have a distinct section for the object flow and handling and shortened introduction, issue #72
- Added security considerations for using mDNS without a specific product-serial-number, issue #75
- Clarified pledge-status responses are cumulative, issue #73
- Removed agent-sign-cert from trigger data to save bandwidth and remove complexity through options, issue #70
- Changed terminology for LDevID(Reg) certificate to registrar LDevID certificate, as it does not need to be an LDevID, issue #66
- Added new protected header parameter (created-on) in PER to support freshness validation, issue #63
- Removed reference to CAB Forum as not needed for BRSKI-PRM specifically, issue #65
- Enhanced error codes in section 5.5.1, issue #39, #64
- Enhanced security considerations and privacy considerations, issue #59
- Issue #50 addressed by referring to the utilized enrollment protocol
- Issue #47 MASA verification of LDevID(RegAgt) to the same registrar LDevID certificate domain CA

*Reworked terminology of "enrollment object", "certification object", "enrollment request object", etc., issue #27

*Reworked all message representations to align with encoding

*Added explanation of MASA requiring domain CA cert in section 5.5.1 and section 5.5.2, issue #36

*Defined new endpoint for pledge bootstrapping status inquiry, issue #35 in section Section 6.4, IANA considerations and section Section 5.5

*Included examples for several objects in section Appendix A including message example sizes, issue #33

*PoP for private key to registrar certificate included as mandatory, issues #32 and #49

*Issue #31, clarified that combined pledge may act as client/server for further (re)enrollment

*Issue #42, clarified that Registrar needs to verify the status responses with and ensure that they match the audit log response from the MASA, otherwise it needs drop the pledge and revoke the certificate

*Issue #43, clarified that the pledge shall use the create time from the trigger message if the time has not been synchronized, yet.

*Several editorial changes and enhancements to increasing readability.

From IETF draft 03 -> IETF draft 04:

*In deep Review by Esko Dijk lead to issues #22-#61, which are bein stepwise integrated

*Simplified YANG definition by augmenting the voucher-request from RFC 8995 instead of redefining it.

*Added explanation for terminology "endpoint" used in this document, issue #16

*Added clarification that Registrar-Agent may collect PVR or PER or both in one run, issue #17

*Added a statement that nonceless voucher may be accepted, issue #18

*Simplified structure in section Section 3.1, issue #19

*Removed join proxy in Figure 1 and added explanatory text, issue #20

*Added description of pledge-CAcerts endpoint plus further handling of providing a wrapped CA certs response to the pledge in section Section 6.3; also added new required registrar endpoint (section Section 6.2 and IANA considerations) for the registrar to provide a wrapped CA certs response, issue #21

*utilized defined abbreviations in the document consistently, issue #22

*Reworked text on discovery according to issue #23 to clarify scope and handling

*Added several clarifications based on review comments

From IETF draft 02 -> IETF draft 03:

*Updated examples to state "base64encodedvalue==" for x5c occurrences

*Include link to SVG graphic as general overview

*Restructuring of section 5 to flatten hierarchy

*Enhanced requirements and motivation in Section 4

*Several editorial improvements based on review comments

From IETF draft 01 -> IETF draft 02:

*Issue #15 included additional signature on voucher from registrar in section Section 6.2 and section Section 5.4 The verification of multiple signatures is described in section Section 6.3

*Included representation for General JWS JSON Serialization for examples

*Included error responses from pledge if it is not able to create a pledge voucher-request or an enrollment request in section Section 6.1

*Removed open issue regarding handling of multiple CSRs and enrollment responses during the bootstrapping as the initial target it the provisioning of a generic LDevID certificate. The defined endpoint on the pledge may also be used for management of further certificates.

From IETF draft 00 -> IETF draft 01:

* Issue #15 lead to the inclusion of an option for an additional signature of the registrar on the voucher received from the MASA before forwarding to the Registrar-Agent to support verification of POP of the registrars private key in section Section 6.2 and Section 6.3.

* Based on issue #11, a new endpoint was defined for the registrar to enable delivery of the wrapped enrollment request from the pledge (in contrast to plain PKCS#10 in simple enroll).

* Decision on issue #8 to not provide an additional signature on the enrollment-response object by the registrar. As the enrollment response will only contain the generic LDevID certificate. This credential builds the base for further configuration outside the initial enrollment.

* Decision on issue #7 to not support multiple CSRs during the bootstrapping, as based on the generic LDevID certificate the pledge may enroll for further certificates.

* Closed open issue #5 regarding verification of ietf-ztp-types usage as verified via a proof-of-concept in section {#exchanges_uc2_1}.

* Housekeeping: Removed already addressed open issues stated in the draft directly.

* Reworked text in from introduction to section pledge-responder-mode

* Fixed "serial-number" encoding in PVR/RVR

* Added prior-signed-voucher-request in the parameter description of the registrar-voucher-request in Section 6.2.

* Note added in Section 6.2 if sub-CAs are used, that the corresponding information is to be provided to the MASA.

* Inclusion of limitation section (pledge sleeps and needs to be waked up. Pledge is awake but Registrar-Agent is not available) (Issue #10).

* Assertion-type aligned with voucher in RFC8366bis, deleted related open issues. (Issue #4)

* Included table for endpoints in Section 5.5 for better readability.

* Included registrar authorization check for Registrar-Agent during TLS handshake in section Section 6.2. Also enhanced figure Figure 11 with the authorization step on TLS level.

* Enhanced description of registrar authorization check for Registrar-Agent based on the agent-signed-data in section Section 6.2. Also enhanced figure Figure 11 with the authorization step on pledge-voucher-request level.

* Changed agent-signed-cert to an array to allow for providing further certificate information like the issuing CA cert for the LDevID(RegAgt) certificate in case the registrar and the Registrar-Agent have different issuing CAs in Figure 11 (issue #12). This also required changes in the YANG module in [I-D.ietf-anima-rfc8366bis]

* Addressed YANG warning (issue #1)

* Inclusion of examples for a trigger to create a pledge-voucher-request and an enrollment-request.

From IETF draft-ietf-anima-brski-async-enroll-03 -> IETF anima-brski-prm-00:

* Moved UC2 related parts defining the pledge in responder mode from draft-ietf-anima-brski-async-enroll-03 to this document This required changes and adaptations in several sections to remove the description and references to UC1.

* Addressed feedback for voucher-request enhancements from YANG doctor early review in Section 7.1 as well as in the security considerations (formerly named ietf-async-voucher-request).

* Renamed ietf-async-voucher-request to IETF-voucher-request-prm to to allow better listing of voucher related extensions; aligned with constraint voucher (#20)

* Utilized ietf-voucher-request-async instead of ietf-voucher-request in voucher exchanges to utilize the enhanced voucher-request.

* Included changes from draft-ietf-netconf-sztp-csr-06 regarding the YANG definition of csr-types into the enrollment request exchange.

From IETF draft 02 -> IETF draft 03:

* Housekeeping, deleted open issue regarding YANG voucher-request in Section 6.1 as voucher-request was enhanced with additional leaf.

*Included open issues in YANG model in Section 5 regarding assertion value agent-proximity and csr encapsulation using SZTP sub module).

From IETF draft 01 -> IETF draft 02:

*Defined call flow and objects for interactions in UC2. Object format based on draft for JOSE signed voucher artifacts and aligned the remaining objects with this approach in Section 6.

*Terminology change: issue #2 pledge-agent -> Registrar-Agent to better underline Registrar-Agent relation.

*Terminology change: issue #3 PULL/PUSH -> pledge-initiator-mode and pledge-responder-mode to better address the pledge operation.

*Communication approach between pledge and Registrar-Agent changed by removing TLS-PSK (former section TLS establishment) and associated references to other drafts in favor of relying on higher layer exchange of signed data objects. These data objects are included also in the pledge-voucher-request and lead to an extension of the YANG module for the voucher-request (issue #12).

*Details on trust relationship between Registrar-Agent and registrar (issue #4, #5, #9) included in Section 5.

*Recommendation regarding short-lived certificates for Registrar-Agent authentication towards registrar (issue #7) in the security considerations.

*Introduction of reference to Registrar-Agent signing certificate using SKID in Registrar-Agent signed data (issue #37).

*Enhanced objects in exchanges between pledge and Registrar-Agent to allow the registrar to verify agent-proximity to the pledge (issue #1) in Section 6.

*Details on trust relationship between Registrar-Agent and pledge (issue #5) included in Section 5.

*Split of use case 2 call flow into sub sections in Section 6.

From IETF draft 00 -> IETF draft 01:

*Update of scope in Section 3.1 to include in which the pledge acts as a server. This is one main motivation for use case 2.

*Rework of use case 2 in Section 5 to consider the transport between the pledge and the pledge-agent. Addressed is the TLS

channel establishment between the pledge-agent and the pledge as well as the endpoint definition on the pledge.

* First description of exchanged object types (needs more work)

* Clarification in discovery options for enrollment endpoints at the domain registrar based on well-known endpoints do not result in additional /.well-known URIs. Update of the illustrative example. Note that the change to /brski for the voucher related endpoints has been taken over in the BRSKI main document.

* Updated references.

* Included Thomas Werner as additional author for the document.

From individual version 03 -> IETF draft 00:

* Inclusion of discovery options of enrollment endpoints at the domain registrar based on well-known endpoints in new section as replacement of section 5.1.3 in the individual draft. This is intended to support both use cases in the document. An illustrative example is provided.

* Missing details provided for the description and call flow in pledge-agent use case Section 5, e.g. to accommodate distribution of CA certificates.

* Updated CMP example in to use lightweight CMP instead of CMP, as the draft already provides the necessary /.well-known endpoints.

* Requirements discussion moved to separate section in Section 4. Shortened description of proof of identity binding and mapping to existing protocols.

* Removal of copied call flows for voucher exchange and registrar discovery flow from [RFC8995] in UC1 to avoid doubling or text or inconsistencies.

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

From individual version 02 -> 03:

* Update of terminology from self-contained to authenticated self-contained object to be consistent in the wording and to underline the protection of the object with an existing credential. Note

that the naming of this object may be discussed. An alternative name may be attestation object.

* Simplification of the architecture approach for the initial use case having an offsite PKI.

* Introduction of a new use case utilizing authenticated self-contain objects to onboard a pledge using a commissioning tool containing a pledge-agent. This requires additional changes in the BRSKI call flow sequence and led to changes in the introduction, the application example,and also in the related BRSKI-PRM call flow.

From individual version 01 -> 02:

* Update of introduction text to clearly relate to the usage of IDevID and LDevID.

* Update of description of architecture elements and changes to BRSKI in Section 5.

* Enhanced consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 4.

From individual version 00 -> 01:

* Update of examples, specifically for building automation as well as two new application use cases in Section 3.1.

* Deletion of asynchronous interaction with MASA to not complicate the use case. Note that the voucher exchange can already be handled in an asynchronous manner and is therefore not considered further. This resulted in removal of the alternative path the MASA in Figure 1 and the associated description in Section 5.

* Enhancement of description of architecture elements and changes to BRSKI in Section 5.

* Consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 4.

* New section starting with the mapping to existing enrollment protocols by collecting boundary conditions.

Contributors

Esko Dijk
IoTconsultancy.nl

Email: esko.dijk@iotconsultancy.nl

Toerless Eckert
Futurewei

Email: tte@cs.fau.de

Matthias Kovatsch

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/