Workgroup: anima Working Group
Internet-Draft:
draft-ietf-anima-constrained-voucher-22
Updates: 8366, 8995 (if approved)
Published: 21 November 2023
Intended Status: Standards Track
Expires: 24 May 2024
Authors: M. Richardson
Sandelman Software Works
P. van der Stok
vanderstok consultancy
P. Kampanakis
Cisco Systems
E. Dijk
IoTconsultancy.nl

Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)

Abstract

This document defines the Constrained Bootstrapping Remote Secure Key Infrastructure (Constrained BRSKI) protocol, which provides a solution for secure zero-touch bootstrapping of resource-constrained (IoT) devices into the network of a domain owner. This protocol is designed for constrained networks, which may have limited data throughput or may experience frequent packet loss. Constrained BRSKI is a variant of the BRSKI protocol, which uses an artifact signed by the device manufacturer called the "voucher" which enables a new device and the owner's network to mutually authenticate. While the BRSKI voucher is typically encoded in JSON, Constrained BRSKI uses a compact CBOR-encoded voucher. The BRSKI voucher definition is extended with new data types that allow for smaller voucher sizes. The Enrollment over Secure Transport (EST) protocol, used in BRSKI, is replaced with EST-over-CoAPS; and HTTPS used in BRSKI is replaced with DTLS-secured CoAP (CoAPS). This document Updates RFC 8366 and RFC 8995.

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-constrained-voucher/.

Discussion of this document takes place on the anima Working Group mailing list (mailto:anima@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/anima/. Subscribe at https://www.ietf.org/mailman/listinfo/anima/.

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

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 24 May 2024.

Copyright Notice

Table of Contents

1. Introduction

Secure enrollment of new nodes into constrained networks with constrained nodes presents unique challenges. As explained in [RFC7228], such networks may have limited data throughput or may experience frequent packet loss. In addition, its nodes may be constrained by energy availability, memory space, and code size.

The Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol described in [RFC8995] provides a solution for secure zero-touch (automated) bootstrap of new (unconfigured) devices. In it, these new devices are called "pledges", equipped with a factory-installed Initial Device Identifier (IDevID) (see [ieee802-1AR]). Using the IDevID the pledges are securely enrolled into a network.

The BRSKI solution described in [RFC8995] was designed to be modular, and this document describes a version scaled to the constraints of IoT deployments.

Therefore, this document uses the constrained version of the voucher and voucher request artifacts as described in [RFC8366bis], along with a constrained version of the BRSKI protocol. This Constrained BRSKI protocol makes use of the CoAP-based version of EST (EST-coaps from [RFC9148]) rather than the EST over HTTPS [RFC7030]. Constrained BRSKI is itself scalable to multiple resource levels through the definition of optional functions. Appendix E illustrates this.

In BRSKI, the [RFC8366] voucher is by default serialized to JSON with a signature in CMS [RFC5652]. This document uses the new CBOR [RFC8949] voucher serialization, as defined by [RFC8366bis], and applies a new COSE [RFC9052] signature format as defined in Section 9.

This COSE-signed CBOR-encoded voucher is transported using both secured CoAP and HTTPS. The CoAP connection (between Pledge and Registrar) is to be protected by DTLS (CoAPS). The HTTP connection (between Registrar and MASA) is to be protected using TLS (HTTPS).

Section 4 to Section 10 define the default Constrained BRSKI protocol, by means of additions to and modifications of regular BRSKI. Section 11 considers some variations of the protocol, specific to particular deployments or IoT networking technologies. Next in Section 12, some considerations for the design and implementation of Constrained BRSKI components are provided.

Section 13 introduces a variant of Constrained BRSKI for the most-constrained Pledges: the use of Raw Public Keys (RPK). This variant achieves smaller sizes of data objects and avoids doing certain costly PKIX verification operations on the Pledge.

2. Terminology

The following terms are defined in [RFC8366bis], and are used identically as in that document: artifact, domain, imprint, Join Registrar/Coordinator (JRC), Manufacturer Authorized Signing Authority (MASA), Pledge, Registrar, Trust of First Use (TOFU), and Voucher.

The following terms from [RFC8995] are used identically as in that document: Domain CA, enrollment, IDevID, Join Proxy, LDevID, manufacturer, nonced, nonceless, PKIX.

The term Pledge Voucher Request, or acronym PVR, is introduced to refer to the voucher request between the Pledge and the Registrar.

The term Registrar Voucher Request, or acronym RVR, is introduced to refer to the voucher request between the Registrar and the MASA.

This document uses the term "PKIX Certificate" to refer to the X.509v3 profile described in [RFC5280].

In code examples, the string "<CODE BEGINS>" denotes the start of a code example and "<CODE ENDS>" the end of the code example. Four dots ("....") in a CBOR diagnostic notation byte string denotes a further sequence of bytes that is not shown for brevity.

3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Overview of Protocol

[RFC8366bis] defines a voucher that can assert proximity, authenticates the Registrar, and can offer varying levels of anti-replay protection. The proximity proof provided by a voucher is an assertion that the Pledge and the Registrar are believed to be close together, from a network topology point of view. Similar to BRSKI [RFC8995], proximity is proven by making a DTLS connection between a Pledge and a Registrar. The Pledge initiates this connection using a link-local source address.

The secure DTLS connection is then used by the Pledge to make a Pledge Voucher Request (PVR). The Registrar then includes the PVR into its own Registrar Voucher Request (RVR), sent to an agent (MASA) of the Pledge's manufacturer. The MASA verifies the PVR and RVR and issues a signed voucher. The voucher provides an authorization

statement from the manufacturer indicating that the Registrar is the intended owner of the Pledge. The voucher refers to the Registrar through pinning of the Registrar's identity.

After verification of the voucher, the Pledge enrolls into the Registrar's domain by obtaining a certificate using the EST-coaps [RFC9148] protocol, suitable for constrained devices. Once the Pledge has obtained its domain identity (LDevID) in this manner, it can use this identity to obtain network access credentials, to join the local IP network. The method to obtain such credentials depends on the particular network technology used and is outside the scope of this document.

This document does not make any extensions to the semantic meaning of vouchers, though a new signature method based on COSE [RFC9052] is defined to optimize for constrained devices and networks.

The two main parts of the BRSKI protocol are named separately in this document: BRSKI-EST (Section 6) for the protocol between Pledge and Registrar, and BRSKI-MASA (Section 7) for the protocol between the Registrar and the MASA.

Time-based vouchers are supported, but given that constrained devices are unlikely to have accurate time, their use will be uncommon. Most Pledges using constrained vouchers will be online during enrollment and will use live nonces to provide anti-replay protection rather than expiry times.

[RFC8366bis] defines the two CBOR-encoded artifacts of a constrained voucher and a constrained voucher request, which are used by Constrained BRSKI.

The constrained voucher request MUST be signed by the Pledge. COSE [RFC9052] is used for signing as defined in Section 9.2. It signs using the private key associated with its IDevID certificate.

The constrained voucher MUST be signed by the MASA. Also in this case, COSE is used for signing.

For the constrained voucher request (PVR) the default method for the Pledge to identify the Registrar is using the Registrar's full PKIX certificate. But when operating PKIX-less as described in Section 13, the Registrar's Raw Public Key (RPK) is used for this.

For the constrained voucher the default method to indicate ("pin") a trusted domain identity is the domain's PKIX CA certificate, but when operating PKIX-less instead the RPK of the Registrar is pinned.

The BRSKI architectures mandates that the MASA be aware of the capabilities of the Pledge. This is not a drawback as a Pledge is

constructed by a manufacturer which also arranges for the MASA to be aware of the inventory of devices. The MASA therefore knows if the Pledge supports PKIX operations, or if it is limited to RPK operations only. Based upon this, the MASA can select which attributes to use in the voucher for certain operations, like the pinning of the Registrar or domain identity.

5. Updates to *[RFC8366bis]* and RFC8995

This section details the ways in which this document updates other RFCs. The terminology for Updates is taken from [I-D.kuehlewind-update-tag].

This document Updates [RFC8366bis]. It Extends [RFC8366bis] with a COSE signature format, and new mechanisms to identify Registrar proximity and to pin a Raw Public Key (RPK).

This document Updates [RFC8995]. It Amends [RFC8995]

- by clarifying how pinning is done,
- adopts clearer explanation of the TLS Server Name Indicator (SNI), see Section 6.1.4 and Section 7.3,
- clarifies when new trust anchors should be retrieved (Section 6.6.1),
- clarifies what kinds of Extended Key Usage attributes are appropriate for each certificate (Section 6.1.5, Section 7.4).

It Extends [RFC8995] as follows:

- defines the use of CoAP with the BRSKI protocol,
- makes some messages optional if the results can be inferred from other validations (Section 6.6),
- provides the option to return trust anchors in a simpler format (Section 6.7),
- extends the BRSKI-MASA protocol (Section 7) to carry the new voucher-cose+cbor format.

6. BRSKI-EST Protocol

This section describes the extensions to both BRSKI [RFC8995] and EST-coaps [RFC9148] protocol operations between Pledge and Registrar. The extensions are targeting low-resource networks with small packets, based on CoAP and DTLS.

6.1. DTLS Connection

A DTLS connection is established between the Pledge and the Registrar, similar to the TLS connection described in Section 5.1 of [RFC8995]. This may occur via a Join Proxy as described in Section 6.2. Regardless of the Join Proxy presence or particular mechanism used, the DTLS connection should operate identically. The Constrained BRSKI and EST-coaps requests and responses for bootstrapping are carried over this DTLS connection.

6.1.1. DTLS Version

DTLS version 1.3 [RFC9147] SHOULD be used in any implementation of this specification. An exception case where DTLS 1.2 [RFC6347] MAY be used is in a Pledge that uses a software platform where a DTLS 1.3 client is not available (yet). This may occur for example if a legacy device gets software-upgraded to support Constrained BRSKI. For this reason, a Registrar MUST by default support both DTLS 1.3 and DTLS 1.2 client connections. However, for security reasons the Registrar MAY be administratively configured to support only a particular DTLS version or higher.

An EST-coaps server [RFC9148] (as a separate entity from above Registrar) that implements this specification also MUST support both DTLS 1.3 and DTLS 1.2 client connections by default. However, for security reasons the EST-coaps server MAY be administratively configured to support only a particular DTLS version or higher.

6.1.2. TLS Client Certificates: IDevID authentication

As described in Section 5.1 of [RFC8995], the Pledge makes a connection to the Registrar using a TLS Client Certificate for authentication. This is the Pledge's IDevID certificate.

Subsequently the Pledge will send a Pledge Voucher Request (PVR). Further elements of Pledge authentication may be present in the PVR, as detailed in Section 9.2.

6.1.3. DTLS Handshake Fragmentation Considerations

DTLS includes a mechanism to fragment handshake messages. This is described in Section 4.4 of [RFC9147]. Constrained BRSKI will often be used with a Join Proxy, described in Section 6.2, which relays each DTLS message to the Registrar. A stateless Join Proxy will need some additional space to wrap each DTLS message inside a CoAP request, while the wrapped result needs to fit in the maximum IPv6 MTU guaranteed on 6LoWPAN networks, which is 1280 bytes.

For this reason it is RECOMMENDED that a PMTU of 1024 bytes be assumed for the DTLS handshake and appropriate DTLS fragmentation is

used. It is unlikely that any Packet Too Big indications ([RFC4443]) will be relayed by the Join Proxy back to the Pledge.

During the operation of the EST-coaps protocol, the CoAP Blockwise transfer mechanism will be automatically used when message sizes exceed the PMTU. A Pledge/EST-client on a constrained network MUST use the (D)TLS maximum fragment length extension ("max_fragment_length") defined in Section 4 of [RFC6066] with the maximum fragment length set to a value of either 2^9 or 2^10, when operating as a DTLS 1.2 client.

A Pledge/EST-client operating as DTLS 1.3 client, MUST use the (D)TLS record size limit extensions ("record_size_limit") defined in Section 4 of [RFC8449], with RecordSizeLimit set to a value between 512 and 1024.

6.1.4. Registrar and the Server Name Indicator (SNI)

The SNI issue described below affects [RFC8995] as well, and is reported in errata: https://www.rfc-editor.org/errata/eid6648

As the Registrar is discovered by IP address, and typically connected via a Join Proxy, the name of the Registrar is not known to the Pledge. The Pledge will not know what the hostname for the Registrar is, so it cannot do DNS-ID validation ([RFC9525]) on the Registrar's certificate. Instead, it must do validation using the voucher.

As the Pledge does not know the name of the Registrar, the Pledge cannot put any reasonable value into the [RFC6066] Server Name Indicator (SNI). Threfore the Pledge SHOULD omit the SNI extension as per Section 9.2 of [RFC8446].

In some cases, particularly while testing BRSKI, a Pledge may be given the hostname of a particular Registrar to connect to directly. Such a bypass of the discovery process may result in the Pledge taking a different code branch to establish a DTLS connection, and may result in the SNI being inserted by a library. The Registrar MUST ignore any SNI it receives from a Pledge.

A primary motivation for making the SNI ubiquitous in the public web is because it allows for multi-tenant hosting of HTTPS sites on a single (scarce) IPv4 address. This consideration does not apply to the server function in the Registrar because:

* it uses DTLS and CoAP, not HTTPS

* it typically uses IPv6, often [RFC4193] Unique Local Address, which are plentiful

*the server port number is typically discovered, so multiple tenants can be accomodated via unique port numbers.

6.1.5. Registrar Server Certificate Requirements

As per Section 3.6.1 of [RFC7030], the Registrar certificate MUST have the Extended Key Usage (EKU) id-kp-cmcRA. This certificate is also used as a TLS Server Certificate, so it MUST also have the EKU id-kp-serverAuth.

See Appendix C.2.2 for an example of a Registrar certificate with these EKUs set. See Section 6.1.5 for Registrar client certificate requirements.

6.2. Join Proxy

[I-D.ietf-anima-constrained-join-proxy] specifies the details for a stateful and stateless constrained Join Proxy which is equivalent to the Proxy defined in [RFC8995], Section 4.

6.3. Request URIs, Resource Discovery and Content Formats

Constrained BRSKI operates using CoAP over DTLS, with request URIs using the coaps scheme. The Pledge operates in CoAP client role. To keep the protocol messages small the EST-coaps and Constrained BRSKI request URIs are shorter than the respective EST and BRSKI URIs.

During the BRSKI bootstrap on an IPv6 network these request URIs have the following form:

```
coaps://[<link-local-ipv6>]:<port>/.well-known/brski/<short-name>
coaps://[<link-local-ipv6>]:<port>/.well-known/est/<short-name>
```

where <link-local-ipv6> is the discovered link-local IPv6 address of a Join Proxy, and <port> is the discovered port of the Join Proxy that is used to offer the BRSKI proxy functionality.

<short-name> is the short resource name for the Constrained BRSKI and EST-coaps resources. For EST-coaps, Table 1 in Section 5.1 of [RFC9148] defines the CoAP <short-name> resource names. For Constrained BRSKI, Table 1 defines the CoAP <short-name> resource names based on the [RFC8995] long HTTP resource names.

BRSKI resource	Constrained BRSKI resource <short-name>
/requestvoucher	/rv
/voucher_status	/vs
/enrollstatus	/es

Table 1: BRSKI URI paths mapping to Constrained BRSKI URI paths

Section 11 details how the Pledge discovers a Join Proxy link-local address and port in different deployment scenarios.

The request URI formats defined above enable the Pledge to perform bootstrap and enrollment without requiring to perform any discovery of the available bootstrap options, voucher formats, BRSKI/EST resources, enrollment protocols, and so on. This is helpful for a majority of constrained Pledges that would support only a single set of options. However, for Pledges that do support multiple options, sending CoAP discovery queries to the Registrar is supported as defined in Section 14.

Because a Pledge only has indirect access to the Registrar via a single port on the Join Proxy, the Registrar MUST host all BRSKI/EST-coaps resources on the same (UDP) server IP address and port. This is the address and port where a Join Proxy would relay DTLS records from the Pledge to.

Although the request URI templates include IP address, scheme and port, in practice the CoAP request sent over the secure DTLS connection only encodes the request URI. For example, a Pledge that skips resource discovery operations just sends the initial CoAP voucher request as follows:

```
REQ: POST /.well-known/brski/rv
  Content-Format: 836
  Payload       : (COSE-signed Pledge Voucher Request, PVR)
```

Note that only Content-Format 836 ("application/voucher-cose+cbor") is defined in this document for the voucher request resource (/rv). Content-Format 836 MUST be supported by the Registrar for the /rv resource and it MAY support additional formats. The Pledge MAY also indicate in the request the desired format of the (voucher) response, using the Accept Option. An example of using this option in the request is as follows:

```
REQ: POST /.well-known/brski/rv
  Content-Format: 836
  Accept        : 836
  Payload       : (COSE-signed Pledge Voucher Request, PVR)
```

If the Accept Option is omitted in the request, the response format follows from the request payload format (which is 836).

Note that this specification allows for voucher-cose+cbor format requests and vouchers to be transmitted over HTTPS, as well as for voucher-cms+json and other formats yet to be defined over CoAP. The burden for this flexibility is placed upon the Registrar. A Pledge on constrained hardware is expected to support a single format only.

The Pledge and MASA need to support one or more formats (at least format 836) for the voucher and for the voucher request. The MASA needs to support all formats that the Pledge supports.

6.3.1. RFC8995 Telemetry Returns

[RFC8995] defines two telemetry returns from the Pledge which are sent to the Registrar. These are the BRSKI Status Telemetry [RFC8995], Section 5.7 and the Enrollment Status Telemetry [RFC8995], Section 5.9.4. These are two CoAP POST request made the by Pledge at two key steps in the process.

[RFC8995] defines the content of these POST operations in CDDL, which are serialized as JSON. This document extends this with an additional CBOR format, derived using the CDDL rules from [RFC8610].

The new CBOR format has CoAP Content-Format 60 ("application/cbor") and MUST be supported by the Registrar for both the /vs and /es resources. The existing JSON format has CoAP Content-Format 50 ("application/json") and also MUST be supported by the Registrar. A Pledge MUST support at least the new CBOR format and it MAY support the JSON format.

6.4. CoAP Resources Overview

This document extends [RFC9148], and it inherits the functions described in that document: specifically, the mandatory Simple (Re-)Enrollment (/sen and /sren) and Certification Authority certificates request (/crts). Support for CSR Attributes Request (/att) and server-side key generation (/skg, /skc) remains optional for the EST server.

Collecting the resource definitions from both [RFC8995], [RFC7030], and [RFC9148] results in the following shorter forms of URI paths for the commonly used resources:

BRSKI + EST	Constrained BRSKI + EST	Well-known URI namespace
/enrollstatus	/es	brski
/requestvoucher	/rv	brski
/voucher_status	/vs	brski
/cacerts	/crts	est
/csrattrs	/att	est
/simpleenroll	/sen	est
/simplereenroll	/sren	est

Table 2: BRSKI/EST resource name mapping to Constrained BRSKI/EST short resource name

6.5. CoAP Responses

[RFC8995], Section 5 defines a number of HTTP response codes that the Registrar is to return when certain conditions occur.

The 401, 403, 404, 406 and 415 response codes map directly to CoAP codes 4.01, 4.03, 4.04, 4.06 and 4.15.

The 202 Retry process which occurs in the voucher request, is to be handled in the same way as the Section 5.7 of [RFC9148] process for Delayed Responses.

6.6. Extensions to EST-coaps

6.6.1. Pledge Extensions

This section defines optimizations for the EST-coaps protocol as used by Pledge. These aim to reduce payload sizes and the number of messages (round-trips) required for the initial EST enrollment.

A Pledge SHOULD NOT perform the optional EST-coaps "CSR attributes request" (/att). Instead, the Pledge selects the attributes to include in the CSR as specified below.

One or more Subject Distinguished Name fields MUST be included. If the Pledge has no specific information on what attributes/fields are desired in the CSR, which is the common case, it MUST use the Subject Distinguished Name fields from its IDevID unmodified. Note that a Pledge may optionally receive such specific information via the voucher (encoded in a vendor-specific way) or via some other, out-of-band means.

A Pledge uses the following optimized EST-coaps procedure:

1. If the voucher, that validates the current Registrar, contains a single pinned domain CA certificate, the Pledge provisionally considers this certificate as the EST trust anchor, as if it were the result of "CA certificates request" (/crts) to the Registrar.

2. Using this CA certificate as trust anchor it proceeds with EST simple enrollment (/sen) to obtain its provisionally trusted LDevID certificate.

3. If the Pledge validates that the trust anchor CA was used to sign its LDevID certificate, the Pledge accepts the pinned domain CA certificate as the legitimate trust anchor CA for the Registrar's domain and accepts the associated LDevID certificate.

4. If the trust anchor CA was NOT used to sign its LDevID certificate, the Pledge MUST perform a "CA certificates request" (/crts) to the EST server to obtain the EST CA trust anchor(s) since these can differ from the (temporary) pinned domain CA.

5. When doing this /crts request, the Pledge MAY use a CoAP Accept Option with value 287 ("application/pkix-cert") to limit the number of returned EST CA trust anchors to only one. A constrained Pledge MAY support only this format in a /crts response, per Section 5.3 of [RFC9148]. Implementing only this format reduces the code and memory requirements on the Pledge.

6. If the Pledge cannot obtain the single CA certificate or the finally validated CA certificate cannot be chained to the LDevID certificate, then the Pledge MUST abort the enrollment process and report the error using the enrollment status telemetry (/es).

Note that even though the Pledge can avoid performing any /crts request using the above EST-coaps procedure during bootstrap, it SHOULD support retrieval of the trust anchor CA periodically as detailed in the next section.

6.6.2. EST-client Extensions

This section defines extensions to EST-coaps clients, used after the BRSKI bootstrap procedure is completed. (Note that such client is not called "Pledge" in this section, since it is already enrolled into the domain. It has become an EST-coaps client.) A constrained EST-coaps client MAY support only the Content-Format 287 ("application/pkix-cert") in a /crts response, per Section 5.3 of [RFC9148]. In this case, it can only store one trust anchor of the domain.

An EST-coaps client that has an idea of the current time (internally, or via NTP) SHOULD consider the validity time of the trust anchor CA, and MAY begin requesting a new trust anchor CA using the /crts request when the CA has 50% of it's validity time (notAfter - notBefore) left. A client without access to the current time cannot decide if the trust anchor CA has expired, and SHOULD poll periodically for a new trust anchor using the /crts request at an interval of approximately 1 month. An EST-coaps server SHOULD include the CoAP ETag Option in every response to a /crts request, to enable clients to perform low-overhead validation whether their trust anchor CA is still valid. The EST-coaps client SHOULD store the ETag resulting from a /crts response in memory and SHOULD use this value in an ETag Option in its next GET /crts request.

The above-mentioned limitation that an EST-coaps client may support only one trust anchor CA is not an issue in case the domain trust

anchor remains stable. However, special consideration is needed for cases where the domain trust anchor can change over time. Such a change may happen due to relocation of the client device to a new domain, or due to a key update of the trust anchor as described in [RFC4210], Section 4.4.

From the client's viewpoint, a trust anchor change typically happens during EST re-enrollment: since a change of domain CA requires all devices operating under the old domain CA to acquire a new LDevID issued by the new domain CA. A client's re-enrollment may be triggered by various events, such as an instruction to re-enroll sent by a domain entity, or an imminent expiry of its LDevID certificate. How the re-enrollment is explicitly triggered on the client by a domain entity, such as a commissioner or a Registrar, is out of scope of this specification.

The mechanism described in [RFC4210], Section 4.4 for Root CA key update requires four certificates: OldWithOld, OldWithNew, NewWithOld, and NewWithNew. The OldWithOld certificate is already stored in the EST client's trust store. The NewWithNew certificate will be distributed as the single certificate in a /crts response, during EST re-enrollment. Since the EST client can only accept a single certificate in a /crts response it implies that the EST client cannot obtain the certificates OldWithNew and NewWithOld in this way, to perform the complete verification of the new domain CA. Instead, the client only verifies the EST-coaps server using its old domain CA certificate in its trust store as detailed below, and based on this trust in the active and valid DTLS connection it automatically trusts the new (NewWithNew) domain CA certificate that the EST-coaps server provides in the /crts response.

In this manner, even during rollover of trust anchors, it is possible to have only a single trust anchor provided in a /crts response.

During the period of the certificate renewal, it is not possible to create new communication channels between devices with NewCA certificates devices with OldCA certificates. One option is that devices should avoid restarting existing DTLS or OSCORE connections during the interval in which new certificates are being deployed. The recommended period for a certificate renewal procedure is 24 hours or less. For re-enrollment, the constrained EST-coaps client MUST support the following EST-coaps procedure, where optional re-enrollment to a new domain is under control of the EST-coaps server:

1. The client connects with DTLS to the EST-coaps server, and authenticates with its present domain certificate (LDevID certificate) as usual. The EST-coaps server authenticates itself with its domain certificate that is trusted by the client, i.e. it chains to the single trust anchor that the client has stored.

This is the "old" trust anchor, the one that will be eventually replaced in case the server decides to re-enroll the client into a new domain. The client also checks that the server is a Registration Authority (RA) of the domain as required by Section 3.6.1 of [RFC7030].

2. The client performs the simple re-enrollment request (/sren) and upon success it obtains a new LDevID.

3. The client verifies the new LDevID against its (single) existing domain trust anchor. If it chains successfully, this means the trust anchor did not change and the client MAY skip retrieving the current CA certificate using the "CA certificates request" (/crts). If it does not chain successfully, this means the trust anchor was changed/updated and the client then MUST retrieve the new domain trust anchor using the "CA certificates request" (/crts).

4. If the client retrieved a new trust anchor in step 3, then it MUST verify that the new trust anchor chains with the new LDevID certificate it obtained in step 2. If it chains successfully, the client stores both, accepts the new LDevID certificate and stops using it prior LDevID certificate. If it does not chain successfully, the client MUST NOT update its LDevID certificate, it MUST NOT update its (single) domain trust anchor, and the client MUST abort the enrollment process and MUST attempt to report the error to the EST-coaps server using enrollment status telemetry (/es).

Note that even though the EST-coaps client may skip the /crts request in step 3, it SHOULD support retrieval of the trust anchor CA periodically as detailed earlier in this section.

Note that an EST-coaps server that is also a Registrar will already support the enrollment status telemetry resource (/es) in step 4, while an EST-coaps server that purely implements [RFC9148], and not the present specification, will not support this resource.

6.7. Registrar Extensions

The Content-Format 60 (application/cbor) MUST be supported by the Registrar for the /vs and /es resources.

Content-Format 836 MUST be supported by the Registrar for the /rv resource for CoAP POST requests, both as request payload and as response payload.

When a Registrar receives a "CA certificates request" (/crts) request with a CoAP Accept Option with value 287 ("application/pkix-cert") it MUST return only the single CA certificate that is the envisioned or

actual authority for the current, authenticated Pledge making the request.

If the Pledge included in its request an Accept Option for only the 287 ("application/pkix-cert") Content Format, but the domain has been configured to operate with multiple CA trust anchors only, then the Registrar returns a 4.06 Not Acceptable error to signal that the Pledge needs to use the Content Format 281 ("application/pkcs7-mime; smime-type=certs-only") to retrieve all the certificates.

If the current authenticated client is an EST-coaps client that was already enrolled in the domain, and the Registrar is configured to assign this client to a new domain CA trust anchor during the next EST re-enrollment procedure, then the Registrar MUST respond with the new domain CA certificate in case the client performs the "CA Certificates request" (/crts) with an Accept Option for 287 only. This signals the client that a new domain is assigned to it. The client follows the procedure as defined in Section 6.6.2.

7. BRSKI-MASA Protocol

This section describes extensions to and clarifications of the BRSKI-MASA protocol between Registrar and MASA.

7.1. Protocol and Formats

Section 5.4 of [RFC8995] describes a connection between the Registrar and the MASA as being a normal TLS connection using HTTPS. This document does not change that. The Registrar MUST use the format "application/voucher-cose+cbor" in its voucher request to MASA, when the Pledge uses this format in its request to the Registrar.

The MASA only needs to support formats for which it has constructed Pledges that use that format.

The Registrar MUST use the same format for the RVR as the Pledge used for its PVR. The Registrar indicates the voucher format it wants to receive from MASA using the HTTP Accept header. This format MUST be the same as the format of the PVR, so that the Pledge can parse it.

At the moment of writing the creation of coaps based MASAs is deemed unrealistic. The use of CoAP for the BRSKI-MASA connection can be the subject of another document. Some consideration was made to specify CoAP support for consistency, but:

* the Registrar is not expected to be so constrained that it cannot support HTTPS client connections.

* the technology and experience to build Internet-scale HTTPS responders (which the MASA is) is common, while the experience doing the same for CoAP is much less common.

* a Registrar is likely to provide onboarding services to both constrained and non-constrained devices. Such a Registrar would need to speak HTTPS anyway.

* a manufacturer is likely to offer both constrained and non-constrained devices, so there may in practice be no situation in which the MASA could be CoAP-only. Additionally, as the MASA is intended to be a function that can easily be oursourced to a third-party service provider, reducing the complexity would also seem to reduce the cost of that function.

* security-related considerations: see Section 15.6.

7.2. Registrar Voucher Request

If the PVR contains a proximity assertion, the Registrar MUST propagate this assertion into the RVR by including the "assertion" field with the value "proximity". This conforms to the example in Section 3.3 of [RFC8995] of carrying the assertion forward.

7.3. MASA and the Server Name Indicator (SNI)

A TLS/HTTPS connection is established between the Registrar and MASA.

Section 5.4 of [RFC8995] explains this process, and there are no externally visible changes. A MASA that supports the unconstrained voucher formats should be able to support constrained voucher formats equally well.

There is no requirement that a single MASA be used for both constrained and unconstrained voucher requests: the choice of MASA is determined by the id-mod-MASAURLExtn2016 extension contained in the IDevID.

The Registrar MUST do DNS-ID checks ([RFC9525]) on the contents of the certificate provided by the MASA.

In constrast to the Pledge/Registrar situation, the Registrar always knows the name of the MASA, and MUST always include an [RFC6066] Server Name Indicator. The SNI is optional in TLS1.2, but common. The SNI it considered mandatory with TLS1.3.

The presence of the SNI is needed by the MASA, in order for the MASA's server to host multiple tenants (for different customers).

7.4. Registrar Client Certificate Requirement

The Registrar SHOULD use a TLS Client Certificate to authenticate to the MASA per Section 5.4.1 of [RFC8995]. If the certificate that the Registrar uses is marked as a id-kp-cmcRA certificate, via Extended Key Usage, then it MUST also have the id-kp-clientAuth EKU attribute set.

In summary for typical Registrar use, where a single Registrar certificate is used, then the certificate MUST have EKU of: id-kp-cmcRA, id-kp-serverAuth, id-kp-clientAuth.

8. Pinning in Voucher Artifacts

The voucher is a statement by the MASA for use by the Pledge that provides the identity of the Pledge's owner. This section describes how the owner's identity is determined and how it is specified within the voucher.

8.1. Registrar Identity Selection and Encoding

Section 5.5 of [RFC8995] describes BRSKI policies for selection of the owner identity. It indicates some of the flexibility that is possible for the Registrar, and recommends the Registrar to include only certificates in the voucher request (CMS) signing structure that participate in the certificate chain that is to be pinned.

The MASA is expected to evaluate the certificates included by the Registrar in its voucher request, forming them into a chain with the Registrar's (signing) identity on one end. Then, it pins a certificate selected from the chain. For instance, for a domain with a two-level certification authority (see Figure 1), where the voucher request has been signed by "Registrar", its signing structure includes two additional CA certificates. The arrows in the figure indicate the issuing of a certificate, i.e. author of (1) issued (2) and author of (2) issued (3).

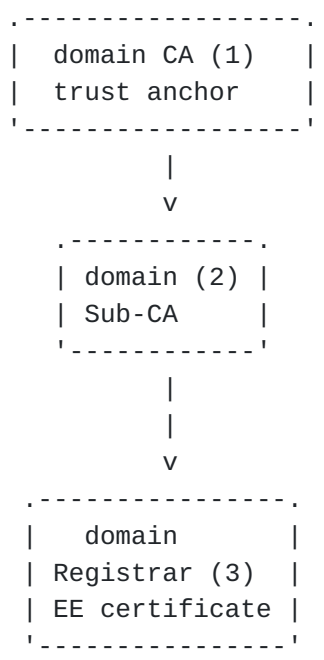

Figure 1: Two-Level CA PKI

When the Registrar is using a COSE-signed constrained voucher request towards MASA, instead of a regular CMS-signed voucher request, the COSE_Sign1 object contains a protected and an unprotected header. The Registrar MUST place all the certificates needed to validate the signature chain from the Registrar on the RVR in an "x5bag" attribute in the unprotected header as defined in [RFC9360].

The "x5bag" attribute is very important as it provides the required signals from the Registrar to control what identity is pinned in the resulting voucher. This is explained in the next section.

8.2. MASA Pinning Policy

The MASA, having assembled and verified the chain in the signing structure of the voucher request needs to select a certificate to pin. (For the case that only the Registrar's End-Entity certificate is included, only this certificate can be selected and this section does not apply.) The BRSKI policy for pinning by the MASA as described in Section 5.5.2 of [RFC8995] leaves much flexibility to the manufacturer.

The present document adds the following rules to the MASA pinning policy to reduce the network load on the constrained network side:

1. for a voucher containing a nonce, it SHOULD select the most specific (lowest-level) CA certificate in the chain.

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

The rationale for 1. is that in case of a voucher with nonce, the voucher is valid only in scope of the present DTLS connection between Pledge and Registrar anyway, so there is no benefit to pin a higher-level CA. By pinning the most specific CA the constrained Pledge can validate its DTLS connection using less crypto operations. The rationale for pinning a CA instead of the Registrar's End-Entity certificate directly is based on the following benefit on constrained networks: the pinned certificate in the voucher can in common cases be re-used as a Domain CA trust anchor during the EST enrollment and during the operational phase that follows after EST enrollment, as explained in Section 6.6.1.

The rationale for 2. follows from the flexible BRSKI trust model for, and purpose of, nonceless vouchers (Sections 5.5.* and 7.4.1 of [RFC8995]).

Refering to Figure 1 of a domain with a two-level certification authority, the most specific CA ("Sub-CA") is the identity that is pinned by MASA in a nonced voucher. A Registrar that wished to have only the Registrar's End-Entity certificate pinned would omit the "domain CA" and "Sub-CA" certificates from the voucher request.

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

8.3. Pinning of Raw Public Keys

Specifically for the most-constrained use cases, the pinning of the raw public key (RPK) of the Registrar is also supported in the constrained voucher, instead of a PKIX certificate. This is used by the RPK variant of Constrained BRSKI described in Section 13, but it can also be used in the default Constrained BRSKI flow as a means to reduce voucher size.

For both cases, if an RPK is pinned, it MUST be the RPK of the Registrar.

When the Pledge is known by MASA to support the RPK variant only, the voucher produced by the MASA pins the RPK of the Registrar in either the "pinned-domain-pubk" or "pinned-domain-pubk-sha256" field of a

voucher. This is described in more detail in [RFC8366bis] and Section 13.

When the Pledge is known by MASA to support PKIX certificates, the "pinned-domain-cert" field present in a voucher normally pins a domain certificate. That can be either the End-Entity certificate of the Registrar, or the certificate of a domain CA of the Registrar's domain as specified in Section 8.2. However, if the Pledge is known by MASA to also support RPK pinning and the MASA intends to pin the Registrar in the voucher (and not the CA), then MASA SHOULD pin the RPK (RPK3 in Figure 2) of the Registrar instead of the Registrar's End-Entity certificate to save space in the voucher.

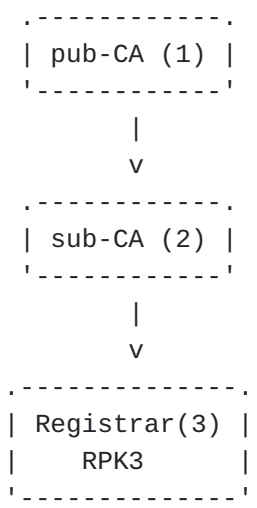

Figure 2: Raw Public Key (RPK) pinning

8.4. Considerations for use of IDevID-Issuer

[RFC8366bis] and [RFC8995] define the idevid-issuer attribute for voucher and voucher-request (respectively), but they summarily explain when to use it.

The use of idevid-issuer is provided so that the serial-number to which the issued voucher pertains can be relative to the entity that issued the devices' IDevID. In most cases there is a one to one relationship between the trust anchor that signs vouchers (and is trusted by the pledge), and the Certification Authority that signs the IDevID. In that case, the serial-number in the voucher must refer to the same device as the serial-number that is in the IDevID certificate.

However, there situations where the one to one relationship may be broken. This occurs whenever a manufacturer has a common MASA, but different products (on different assembly lines) are produced with identical serial numbers. A system of serial numbers which is just a

simple counter is a good example of this. A system of serial numbers where there is some prefix relating the product type does not fit into this, even if the lower digits are a counter.

It is not possible for the Pledge or the Registrar to know which situation applies. The question to be answered is whether or not to include the idevid-issuer in the PVR and the RVR. A second question arises as to what the format of the idevid-issuer contents are.

Analysis of the situation shows that the pledge never needs to include the idevid-issuer in it's PVR, because the pledge's IDevID certificate is available to the Registrar, and the Authority Key Identifier is contained within that IDevID certificate. The pledge therefore has no need to repeat this.

For the RVR, the Registrar has to examine the pledge's IDevID certificate to discover the serial number for the Registrar's Voucher Request (RVR). This is clear in Section 5.5 of [RFC8995]. That section also clarifies that the idevid-issuer is to be included in the RVR.

Concerning the Authority Key Identifier, [RFC8366bis] specifies that the entire object i.e. the extnValue OCTET STRING is to be included: comprising the AuthorityKeyIdentifier, SEQUENCE, Choice as well as the OCTET STRING that is the keyIdentifier.

9. Artifacts

The YANG ([RFC7950]) module and CBOR serialization for the constrained voucher as used by Constrained BRSKI are described in [RFC8366bis]. That document also assigns SID values to YANG elements in accordance with [I-D.ietf-core-sid]. The present section provides some examples of these artifacts and defines a new signature format for these, using COSE.

Compared to the first voucher request definition done in [RFC8995], the constrained voucher request adds the fields proximity-registrar-pubk and proximity-registrar-pubk-sha256. One of these is optionally used to replace the proximity-registrar-cert field, for a smaller voucher request - useful for the most constrained cases.

The constrained voucher adds the fields pinned-domain-pubk and pinned-domain-pubk-sha256. One of these is optionally used instead of the pinned-domain-cert field, for a smaller voucher object.

9.1. Example Artifacts

9.1.1. Example Pledge voucher request (PVR) artifact

Below, an example constrained voucher request (PVR) from a Pledge to a Registrar is shown in CBOR diagnostic notation. Long CBOR byte strings have been shortened (with "....") for readability. The enum value of the assertion field is 2 for the "proximity" assertion as defined in Section 6.3 of [RFC8366bis].

```
{
 2501: {          / SID=2501, ietf-voucher-request:voucher|voucher /
    1: 2,                     / SID=2502, assertion 2 = "proximity"/
    7: h'831D5198A6CA2C7F',   / SID=2508, nonce                    /
   12: h'30593013....D29A54', / SID=2513, proximity-registrar-pubk /
   13: "JADA123456789"        / SID=2514, serial-number           /
 }
}
```

The Pledge has included the item proximity-registrar-pubk which carries the public key of the Registrar, instead of including the full Registrar certificate in a proximity-registrar-cert item. This is done to reduce the size of the PVR. Also note that the Pledge did not include the created-on field since it lacks an internal real-time clock and has no knowledge of the current time at the moment of performing the bootstrapping.

9.1.2. Example Registrar voucher request (RVR) artifact

Next, an example constrained voucher request (RVR) from a Registrar to a MASA is shown in CBOR diagnostic notation. The Registrar has created this request triggered by the reception of the Pledge voucher request (PVR) of the previous example. Again, long CBOR byte strings have been shortened for readability.

```
{
 "ietf-request-voucher:voucher": {
    "assertion":     2,
    "created-on":    "2022-12-05T19:19:19.536Z",
    "nonce":         h'831D5198A6CA2C7F',
    "idevid-issuer": h'04183016....1736C3E0',
    "serial-number": "JADA123456789",
    "prior-signed-voucher-request": h'A11909....373839'
 }
}
```

Note that the Registrar uses here the string data type for all key names, instead of the more compact SID integer keys. This is fine for

any use cases where the network between Registrar and MASA is an unconstrained network where data size is not critical. The constrained voucher request format supports both the string and SID key types.

9.1.3. Example voucher artifacts

Below, an example constrained voucher is shown in CBOR diagnostic notation. It was created by a MASA in response to receiving the Registrar Voucher Request (RVR) shown in Section 9.1.2. The enum value of the assertion field is set to 2, to acknowledge to both the Pledge and the Registrar that the proximity of the Pledge to the Registrar is considered proven.

```
{
 2451: {                / SID = 2451, ietf-voucher:voucher|voucher /
    1: 2,                       / SID = 2452, assertion "proximity" /
    2: "2022-12-05T19:19:23Z", / SID = 2453, created-on             /
    3: false,        / SID = 2454, domain-cert-revocation-checks    /
    7: h'831D5198A6CA2C7F',     / SID = 2508, nonce                 /
    8: h'308201F8....8CFF',     / SID = 2459, pinned-domain-cert    /
   11: "JADA123456789"          / SID = 2462, serial-number         /
 }
}
```

While the above example voucher includes the nonce from the PVR, the next example is a nonce-less voucher. Instead of a nonce, it includes an expires-on field with the date and time on which the voucher expires. Because the MASA did not verify the proximity of the Pledge and Registrar in this case, the assertion field contains a weaker assertion of "verified" (0). This indicates that the MASA verified the domain's ownership of the Pledge via some other means. The enum value of the assertion field for "verified" is calculated to be 0 by following the algorithm described in section 9.6.4.2 of [RFC7950].

```
{
 2451: {                / SID = 2451, ietf-voucher:voucher|voucher /
    1: 0,                       / SID = 2452, assertion "verified"  /
    2: "2022-12-06T10:15:32Z", / SID = 2453, created-on             /
    3: false,        / SID = 2454, domain-cert-revocation-checks    /
    4: "2022-12-13T10:15:32Z", / SID = 2455, expires-on             /
    8: h'308201F8....8CFF',     / SID = 2459, pinned-domain-cert    /
   11: "JADA123456789"          / SID = 2462, serial-number         /
 }
}
```

The voucher is valid for one week. To verify the voucher's validity, the Pledge would either need an internal real-time clock or some

external means of obtaining the current time, such as Network Time Protocol (NTP) or a radio time signal receiver.

9.2. Signing voucher and voucher request artifacts with COSE

The COSE_Sign1 structure is discussed in Section 4.2 of [RFC9052]. The CBOR object that carries the body, the signature, and the information about the body and signature is called the COSE_Sign1 structure. It is used when only one signature is used on the body.

Support for ECDSA with SHA2-256 using curve secp256r1 (aka prime256k1) is RECOMMENDED. Most current low power hardware has support for acceleration of this algorithm. Future hardware designs could omit this in favour of a newer algorithms. This is the ES256 keytype from Table 1 of [RFC9053]. Support for curve secp256k1 is OPTIONAL.

Support for EdDSA using Curve 25519 is RECOMMENDED in new designs if hardware support is available. This is keytype EDDSA (-8) from Table 2 of [RFC9053]. A "crv" parameter is necessary to specify the Curve, which from Table 18. The 'kty' field MUST be present, and it MUST be 'OKP'. (Table 17)

A transition towards EdDSA is occurring in the industry. Some hardware can accelerate only some algorithms with specific curves, other hardware can accelerate any curve, and still other kinds of hardware provide a tool kit for acceleration of any eliptic curve algorithm.

In general, the Pledge is expected to support only a single algorithm, while the Registrar, usually not constrained, is expected to support a wide variety of algorithms: both legacy ones and up-and-coming ones via regular software updates.

An example of the supported COSE_Sign1 object structure containing a Pledge Voucher Request (PVR) is shown in Figure 3.

```
18(                   / tag for COSE_Sign1                        /
  [
    h'A10126',        / protected COSE header encoding: {1: -7}  /
                      /            which means { "alg": ES256 }  /
    {},               / unprotected COSE header parameters       /
    h'A119....3839', / voucher-request binary content (in CBOR) /
    h'4567....1234'   / voucher-request binary Sign1 signature   /
  ]
)
```

Figure 3: COSE_Sign1 PVR example in CBOR diagnostic notation

The [COSE-registry] specifies the integers/encoding for the "alg" field in Figure 3. The "alg" field restricts the key usage for verification of this COSE object to a particular cryptographic algorithm.

9.2.1. Signing of Registrar Voucher Request (RVR)

A Registrar MUST include a COSE "x5bag" structure in the RVR as explained in Section 8.1. Figure 4 shows an example Registrar Voucher Request (RVR) that includes the x5bag as an unprotected header parameter (32). The bag contains two certificates in this case.

```
18(                   / tag for COSE_Sign1                        /
  [
    h'A10126',        / protected COSE header encoding: {1: -7}  /
                      /            which means { "alg": ES256 }  /
    {
      32: [h'308202....9420AE', h'308201....E08CFF']  / x5bag   /
    },
    h'A178....7FED', / voucher-request binary content (in CBOR) /
    h'E1B7....2925'  / voucher-request binary Sign1 signature   /
  ]
)
```

Figure 4: COSE_Sign1 RVR example in CBOR diagnostic notation

A "kid" (key ID) field is optionally present in the unprotected COSE header parameters map of a COSE object. If present, it identifies the public key of the key pair that was used to sign the COSE message. The value of the key identifier "kid" parameter may be in any format agreed between signer and verifier. Usually a hash of the public key is used to identify the public key; but the choice of key identifier method is vendor-specific. If "kid" is not present, then a verifying party needs to use other context information to retrieve the right public key to verify the COSE_Sign1 object against.

By default, a Registrar does not include a "kid" parameter in the RVR since the signing key is already identified by the signing certificates included in the COSE "x5bag" structure. A Registrar nevertheless MAY use a "kid" parameter in its RVR to identify its signing key/identity.

The method of generating such "kid" value is vendor-specific and SHOULD be configurable in the Registrar to support commonly used methods. In order to support future business cases and supply chain integrations, a Registrar using the "kid" field MUST be configurable, on a per-manufacturer basis, to select a particular method for generating the "kid" value such that it is compatible with the method

that the manufacturer expects. Note that the "kid" field always has a CBOR byte string (bstr) format.

9.2.2. Signing of Pledge Voucher Request (PVR)

Like in the RVR, a "kid" (key ID) field is also optionally present in the PVR. It can be used to identify the signing key/identity to the MASA.

A Pledge by default SHOULD NOT use a "kid" parameter in its PVR, because its signing key is already identified by the Pledge's unique serial number that is included in the PVR and (by the Registrar) in the RVR. This achieves the smallest possible PVR data size while still enabling the MASA to verify the PVR. Still, when required the Pledge MAY use a "kid" parameter in its PVR to help the MASA identify the right public key to verify against. This can occur for example if a Pledge has multiple IDevID identities. The "kid" parameter in this case may be an integer byte identifying one out of N identities present, or it may be a hash of the public key, or anything else the Pledge vendor decides. A Registrar normally SHOULD ignore a "kid" parameter used in a received PVR, as this information is intended for the MASA. In other words, there is no need for the Registrar to verify the contents of this field, but it may include it in an audit log.

The example in Figure 5 shows a PVR with the "kid" parameter present.

```
18(                 / tag for COSE_Sign1                        /
  [
    h'A10126',      / protected COSE header encoding: {1: -7}  /
                    /            which means { "alg": ES256 }  /
    {
       4: h'59AB3E' / COSE "kid" header parameter              /
    },
    h'A119....3839', / voucher-request binary content (in CBOR) /
    h'5678....7890'  / voucher-request binary Sign1 signature   /
  ]
)
```

Figure 5: COSE_Sign1 PVR example with "kid" field present

The Pledge SHOULD NOT use the "x5bag" structure in the PVR. A Registrar that processes a PVR with an "x5bag" structure MUST ignore it, and MUST use only the TLS Client Certificate extension for authentication of the Pledge.

A situation where the Pledge MAY use the x5bag structure is for communication of certificate chains to the MASA. This would arise in

some vendor- specific situations involving outsourcing of MASA functionality, or rekeying of the IDevID certification authority.

In Appendix C further examples of signed PVRs are shown.

9.2.3. Signing of voucher by MASA

The MASA SHOULD NOT use a "kid" parameter in the voucher response, because the MASA's signing key is already known to the Pledge. Still, where needed the MASA MAY use a "kid" parameter in the voucher response to help the Pledge identify the right MASA public key to verify against. This can occur for example if a Pledge has multiple IDevID identities.

The MASA SHOULD NOT include an x5bag attribute in the voucher response - the exception is if the MASA knows that the Pledge doesn't pre-store the signing public key and certificate, and thus the MASA needs to provide a cert or cert chain that will enable linking the signing identity to the pre-stored Trust Anchor (CA) in the Pledge. This approach is not recommended, because including certificates in the x5bag attribute will significantly increase the size of the voucher which impacts operations on constrained networks.

If the MASA signing key is based upon a PKI (see [I-D.richardson-anima-masa-considerations] Section 2.3), and the Pledge only pre-stores a manufacturer (root) CA identity in its Trust Store which is not the identity that signs the voucher, then a certificate chain needs to be included with the voucher in order for the Pledge to validate the MASA signing CA's signature by validating the chain up to the CA in its Trust Store.

In BRSKI CMS signed vouchers [RFC8995], the CMS structure has a place for such certificates. In the COSE-signed constrained vouchers described in this document, the x5bag attribute [RFC9360] is used to contain the needed certificates to form the chain. A Registrar MUST NOT remove the x5bag attribute from the unprotected COSE header parameters when sending the voucher back to the Pledge.

In Figure 6 an example is shown of a COSE-signed voucher. This example shows the common case where the "x5bag" attribute is not used.

```
18(                   / tag for COSE_Sign1                        /
  [
    h'A10126',        / protected COSE header encoding: {1: -7}  /
                      /            which means { "alg": ES256 }  /
    {},               / unprotected COSE header parameters       /
    h'A119....3839', / voucher binary content (in CBOR)         /
    h'2A2C....7FBF'  / voucher binary Sign1 signature by MASA   /
  ]
)
```

Figure 6: COSE_Sign1 signed voucher in CBOR diagnostic notation

10. Extensions to Discovery

It is assumed that a Join Proxy (Section 6.2) seamlessly provides a relayed DTLS connection between the Pledge and the Registrar. To use a Join Proxy, a Pledge needs to discover it. For Pledge discovery of a Join Proxy, this section extends Section 4.1 of [RFC8995] for the constrained BRSKI case.

In general, the Pledge may be one or more hops away from the Registrar, where one hop means the Registrar is a direct link-local neighbor of the Pledge. The case of one hop away can be considered as a degenerate case, because a Join Proxy is not really needed then.

The degenerate case would be unusual in constrained wireless network deployments, because a Registrar would typically not have a wireless network interface of the type used for constrained devices. Rather, it would have a high-speed network interface. Nevertheless, the situation where the Registrar is one hop away from the Pledge could occur in various cases like wired IoT networks or in wireless constrained networks where the Pledge is in radio range of a 6LoWPAN Border Router (6LBR) and the 6LBR happens to host a Registrar.

In order to support the degenerate case, the Registrar SHOULD announce itself as if it were a Join Proxy -- though it would actually announce its real (stateful) Registrar CoAPS endpoint. No actual Join Proxy functionality is then required on the Registrar.

So, a Pledge only needs to discover a Join Proxy, regardless of whether it is one or more than one hop away from a relevant Registrar. It first discovers the link-local address and the join-port of a Join Proxy. The Pledge then follows the constrained BRSKI procedure of initiating a DTLS connection using the link-local address and join-port of the Join Proxy.

Once enrolled, a Pledge itself may function as a Join Proxy. The decision whether or not to provide this functionality depends upon many factors and is out of scope for this document. Such a decision

might depend upon the amount of energy available to the device, the network bandwidth available, as well as CPU and memory availability.

The process by which a Pledge discovers the Join Proxy, and how a Join Proxy discovers the location of the Registrar, are the subject of the remainder of this section. Further details on both these topics are provided in [I-D.ietf-anima-constrained-join-proxy].

10.1. Discovery operations by Pledge

The Pledge must discover the address/port and optionally the protocol with which to communicate. The present document only defines coaps (CoAP over DTLS) as the default protocol.

Note that identifying the format of the voucher request and the voucher is not a required part of the Pledge's discovery operation. It is assumed that all Registrars support all relevant voucher(-request) formats, while the Pledge only supports a single format. A Pledge that makes a voucher request to a Registrar that does not support that format will receive a CoAP 4.06 Not Acceptable status code and the bootstrap attempt will fail.

10.1.1. GRASP discovery

This section is normative for uses with an ANIMA ACP. In the context of autonomic networks, the Join Proxy uses the DULL GRASP M_FLOOD mechanism to announce itself. Section 4.1.1 of [RFC8995] discusses this in more detail.

The following changes are necessary with respect to figure 10 of [RFC8995]:

*The transport-proto is IPPROTO_UDP

*the objective is AN_Proxy

The Registrar announces itself in the ACP instance of GRASP using M_FLOOD messages. Autonomic Network Join Proxies MUST support GRASP discovery of a Registrar as described in section 4.3 of [RFC8995] .

Here is an example M_FLOOD announcing the Join Proxy at fe80::1, on standard coaps port 5684, using DTLS.

```
[M_FLOOD, 12340815, h'fe80000000000000000000000000001', 180000,
[["AN_Proxy", 4, 1, "DTLS"],
[O_IPv6_LOCATOR,
h'fe80000000000000000000000000001', IPPROTO_UDP, 5684]]]
```

Figure 7: Example of Join Proxy announcement message

Note that a Join Proxy that supports also supports RFC8995 onboarding using HTTPS may announce more than one objective. Objectives with an empty objective-value (whether CBOR NULL or an empty string) refer to [RFC8995] defaults.

Here is an example of an announcement that offers both constrained and unconstrained onboarding:

```
  [M_FLOOD, 12340851, h'fe800000000000000000000000000001', 180000,
  [["AN_Proxy", 4, 1, ""],
   [O_IPv6_LOCATOR,
    h'fe800000000000000000000000000001', IPPROTO_TCP, 4443],
   ["AN_Proxy", 4, 1, "DTLS"],
   [O_IPv6_LOCATOR,
    h'fe800000000000000000000000000001', IPPROTO_UDP, 5684]]
```

Figure 8: Example of Join Proxy announcing two bootstrap methods

10.1.2. CoAP Discovery

The details on CoAP discovery of a Join Proxy by a Pledge are provided in Section 5.2.1 of [I-D.ietf-anima-constrained-join-proxy]. In this section some examples of CoAP discovery interactions are given.

Below, a typical example is provided showing the Pledge's CoAP request and the Join Proxy's CoAP response. The Join Proxy responds with a link-local source address, which is the same address as indicated in the URI-reference element ([RFC6690]) in the discovery response payload. The Join Proxy has a dedicated port 8485 opened for DTLS connections of Pledges.

```
REQ: GET coap://[ff02::fd]/.well-known/core?rt=brski.jp

RES: 2.05 Content
<coaps://[fe80::c78:e3c4:58a0:a4ad]:8485>;rt=brski.jp
```

The next example shows a Join Proxy that uses the default CoAPS port 5684 for DTLS connections of Pledges. In this case, the Join Proxy host is not using port 5684 for any other purposes.

```
REQ: GET coap://[ff02::fd]/.well-known/core?rt=brski.jp

RES: 2.05 Content
<coaps://[fe80::c78:e3c4:58a0:a4ad]>;rt=brski.jp
```

In the following example, two Join Proxies respond to the multicast query. The Join Proxies each use a slightly different CoRE Link Format 'rt' value encoding. While the first encoding is more compact,

both encodings are allowed per [RFC6690]. The Pledge may now select one of the two Join Proxies for initiating its DTLS connection.

```
REQ: GET coap://[ff02::fd]/.well-known/core?rt=brski*

RES: 2.05 Content
<coaps://[fe80::c78:e3c4:58a0:a4ad]:8485>;rt=brski.jp

RES: 2.05 Content
<coaps://[fe80::d359:3813:f382:3b23]:63245>;rt="brski.jp"
```

10.2. Discovery operations by Join Proxy

The Join Proxy needs to discover a Registrar, at the moment it needs to relay data (of a Pledge) towards the Registrar, or prior to that moment. For example, it may start Registrar discovery as soon as it is requested to be enabled as a Join Proxy.

10.2.1. GRASP Discovery

This section is normative for uses with an ANIMA ACP. In the context of autonomic networks, the Registrar announces itself to a stateful Join Proxy using the ACP instance of GRASP using M_FLOOD messages. Section 4.3 of [RFC8995] discusses this in more detail.

The following changes are necessary with respect to figure 10 of [RFC8995]:

- *The transport-proto is IPPROTO_UDP

- *the objective is AN_join_registrar, identical to [RFC8995].

- *the objective name is "BRSKI_JP".

The Registrar announces itself using the ACP instance of GRASP using M_FLOOD messages. Autonomic Network Join Proxies MUST support GRASP discovery of Registrar as described in section 4.3 of [RFC8995].

Here is an example M_FLOOD announcing the Registrar on example port 5684, which is the standard CoAPS port number.

```
[M_FLOOD, 51804321, h'fda379a6f6ee00000200000064000001', 180000,
[["AN_join_registrar", 4, 255, "BRSKI_JP"],
 [O_IPv6_LOCATOR,
  h'fda379a6f6ee00000200000064000001', IPPROTO_UDP, 5684]]]
```

Figure 9: Example of Registrar announcement message

The Registrar uses a routable address that can be used by enrolled constrained Join Proxies. The address will typically be a Unique Local Address (ULA) as in the example, but could also be a Global Unicast Address (GUA).

10.2.2. CoAP discovery

Further details on CoAP discovery of the Registrar by a Join Proxy are provided in Section 5.1.1 of [I-D.ietf-anima-constrained-join-proxy].

11. Deployment-specific Discovery Considerations

This section details how discovery of a Join Proxy is done by the Pledge in specific deployment scenarios.

11.1. 6TSCH Deployments

In 6TISCH networks, the Constrained Join Proxy (CoJP) mechanism is described in [RFC9031]. Such networks are expected to use [I-D.ietf-lake-edhoc] for key management. This is the subject of future work. The Enhanced Beacon has been extended in [RFC9032] to allow for discovery of the Join Proxy.

11.2. Generic networks using GRASP

Section 10.1.1 defines a mechanism for the Pledge to discover a Join Proxy by listening for [RFC8990] GRASP messages. This mechanism can be used on any network which does not have another more specific mechanism. This mechanism supports mesh networks, and can also be used over unencrypted Wi-Fi.

11.3. Generic networks using mDNS

[RFC8995] defines a mechanism for the Pledge to discover a Join Proxy by sending mDNS [RFC6762] queries. This mechanism can be used on any network which does not have another recommended mechanism. This mechanism does not easily support mesh networks. It can be used over unencrypted Wi-Fi.

11.4. Thread networks using Mesh Link Establishment (MLE)

Thread [Thread] is a wireless mesh network protocol based on 6LoWPAN [RFC6282] and other IETF protocols. In Thread, a new device discovers potential Thread networks and Thread routers to join by using the Mesh Link Establishment (MLE) [I-D.ietf-6lo-mesh-link-establishment] protocol. MLE uses the UDP port number 19788. The new device sends discovery requests on different IEEE 802.15.4 radio channels, to which routers (if any present) respond with a discovery response containing information about their respective network. Once a

suitable router is selected the new device initiates a DTLS transport-layer secured connection to the network's commissioning application, over a link-local single radio hop to the selected Thread router. This link is not yet secured at the radio level: link-layer security will be set up once the new device is approved by the commissioning application to join the Thread network, and it gets provisioned with network access credentials.

The Thread router acts here as a Join Proxy. The MLE discovery response message contains UDP port information to signal the new device which port to use for its DTLS connection. The IPv6 source address of the MLE response message indicates the address of the Join Proxy.

12. Design and Implementation Considerations

12.1. Voucher Format and Encoding

The design considerations for vouchers from Section 8 of [RFC8366bis] apply. Specifically for CBOR encoding of vouchers, one key difference with JSON encoding is that the names of the leaves in the YANG definition do not affect the size of the resulting CBOR, as the SID ([I-D.ietf-core-sid]) translation process assigns integers to the names.

To obtain the lowest code size and RAM use on the Pledge, it is recommended that a Pledge is designed to only process/generate these SID integers and not the lengthy strings. The MASA in that case is required to generate the voucher for that Pledge using only SID integers. Yet, this MASA implementation MUST still support both SID integers and strings, to be able to process the field names in the RVR.

Any POST request to the Registrar with resource /vs or /es returns a 2.04 Changed response with empty payload. The client should be aware that the server may use a piggybacked CoAP response (ACK, 2.04) but may also respond with a separate CoAP response, i.e. first an (ACK, 0.0) that is an acknowledgement of the request reception followed by a (CON, 2.04) response in a separate CoAP message. See [RFC7252] for details.

12.2. Use of Constrained BRSKI with HTTPS

This specification contains two extensions to [RFC8995]: a constrained voucher format (COSE), and a constrained transfer protocol (CoAP).

On constrained networks with constrained devices, it make senses to use both together. However, this document does not mandate that this be the only way.

A given constrained device design and software may be re-used for multiple device models, such as a model having only an IEEE 802.15.4 radio, or a model having only an IEEE 802.11 (Wi-Fi) radio, or a model having both these radios. A manufacturer of such device models may wish to have code only for the use of the constrained voucher format (COSE), and use it on all supported radios including the IEEE 802.11 radio. For this radio, the software stack to support HTTP/TLS may be already integrated into the radio module hence it is attractive for the manufacturer to reuse this. This type of approach is supported by this document. In the case that HTTPS is used, the regular long [RFC8995] resource names are used, together with the new "application/voucher-cose+cbor" media type described in this document. For status telemetry requests, the Pledge may use either one or both of the formats defined in Section 6.3.1. A Registrar MUST support both formats.

Other combinations are possible, but they are not enumerated here. New work such as [I-D.ietf-anima-jws-voucher] provides new formats that may be useable over a number of different transports. In general, sending larger payloads over constrained networks makes less sense, while sending smaller payloads over unconstrained networks is perfectly acceptable.

The Pledge will in most cases support a single voucher format, which it uses without negotiation i.e. without discovery of formats supported. The Registrar, being unconstrained, is expected to support all voucher formats. There will be cases where a Registrar does not support a new format that a new Pledge uses, and this is an unfortunate situation that will result in lack of interoperation.

The responsability for supporting new formats is on the Registrar.

13. Raw Public Key Variant

13.1. Introduction and Scope

This section introduces a Constrained BRSKI variant to reduce the data volume and complexity of the BRSKI bootstrap. The use of a raw public key (RPK) in the pinning process can significantly reduce the number of bytes sent over the wire and the number of round trips, and reduce the code footprint in a Pledge. But it comes with a few significant operational limitations.

One simplification that comes with RPK use is that a Pledge can avoid doing PKIX certificate operations, such as certificate chain validation.

13.2. The Registrar Trust Anchor

When the Pledge first connects to the Registrar, the connection to the Registrar is provisional, as explained in Section 5.6.2 of [RFC8995]. The Registrar normally provides its public key in a TLSServerCertificate, and the Pledge uses that to validate that integrity of the (D)TLS connection, but it does not validate the identity of the provided certificate.

As the TLSServerCertificate object is never verified directly by the Pledge, sending it can be considered superfluous. So instead of using a (TLSServer)Certificate of type X509 (see section 4.4.2 of [RFC8446]), a RawPublicKey object (as defined by [RFC7250]) is used.

A Registrar operating in a mixed environment can determine whether to send a Certificate or a Raw Public Key to the Pledge: this is signaled by the Pledge. In the case it needs an RPK, it includes the server_certificate_type of RawPublicKey. This is shown in section 5 of [RFC7250].

The Pledge MUST send a client_certificate_type of X509 (not an RPK), so that the Registrar can properly identify the Pledge and distill the MASA URI information from its IDevID certificate.

13.3. The Pledge Voucher Request

The Pledge puts the Registrar's public key into the proximity-registrar-pubk field of the Pledge Voucher Request (PVR). (The proximity-registrar-pubk-sha256 can alternatively be used for efficiency, if the 32-bytes of a SHA256 hash turns out to be smaller than a typical ECDSA key.)

As the format of this pubk field is identical to the TLS RawPublicKey data object, no manipulation at all is needed to insert this field into the PVR. This approach reduces the size of the PVR significantly.

13.4. The Voucher Response

A returned voucher will have a pinned-domain-pubk field with the identical key as was found in the proximity-registrar-pubk field above, as well as being identical to the Registrar's RPK in the currently active DTLS connection. (Or alternatively the MASA may include the "pinned-domain-pubk-sha256" field if it knows the Pledge supports this field.)

Validation of this key by the Pledge is what takes the DTLS connection out of the provisional state; see Section 5.6.2 of [RFC8995] for more details.

The received voucher needs to be validated first by the Pledge. The Pledge needs to have a public key to validate the signature from the MASA on the voucher.

The MASA's public key counterpart of the (private) MASA signing key MUST be already installed in the Pledge at manufacturing time. Otherwise, the Pledge cannot validate the voucher's signature.

13.5. The Enrollment Phase

A Pledge that does not support PKIX operations cannot use EST to enroll; it has to use another method for enrollment without certificates and the Registrar has to support this method also. For example, an enrollment process that records an RPK owned by the Pledge as a legitimate entity that is part of the domain.

It is possible that the Pledge will not enroll after obtaining a valid voucher, but instead will do only a network join operation (see for example [RFC9031]). How the Pledge discovers this method and details of such enrollment methods are out of scope of this document.

14. Pledge Discovery of Bootstrap and Enrollment Options

The functionality in this section is optional for a Pledge to implement. In typical cases, for a constrained Pledge that only supports a single bootstrap and enrollment method, this functionality is not needed.

14.1. Pledge Discovery Query for All BRSKI Resources

A Pledge that wishes to discover the available BRSKI bootstrap options/formats can do a discovery operation using the CoAP resource discovery method of Section 4 of [RFC6690], by sending a discovery query to the Registrar over the secured DTLS connection.

For example, if the Registrar supports a short BRSKI URL (/b) instead of just the longer "/.well-known" resources, and supports only the voucher format "application/voucher-cose+cbor" (836), and status reporting in both CBOR and JSON formats, a CoAP resource discovery request and response may look as follows:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</b>;rt=brski,
</b/rv>;rt=brski.rv;ct=836,
</b/vs>;rt=brski.vs;ct="50 60",
</b/es>;rt=brski.es;ct="50 60"
```

The Registrar is under no obligation to provide shorter URLs, and MAY respond to this query with only the "/.well-known/brski/<short-name>" resources for the short names as defined in Table 1. This case is shown in the below interaction:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</.well-known/brski>;rt=brski,
</.well-known/brski/rv>;rt=brski.rv;ct=836,
</.well-known/brski/vs>;rt=brski.vs;ct="50 60",
</.well-known/brski/es>;rt=brski.es;ct="50 60"
```

However, for efficiency reasons it would be better if the Registrar would return shorter URIs instead.

When responding to a discovery request for BRSKI resources, the Registrar MAY return the full resource paths for all <short-name> resources and the content types which are supported by these resources (using ct attributes) as shown in the above examples. This is useful when multiple content types are specified for a particular resource on the Registrar and the discovering Pledge also supports multiple.

Registrars that have implemented shorter URLs MUST also respond in equivalent ways to a request on the corresponding "/.well-known/brski/<short-name>" URLs, and MUST NOT distinguish between them. In particular, a Pledge MAY use the longer (e.g. well-known) and shorter URLs in any combination.

14.2. Pledge Discovery Query for the Root BRSKI Resource

In case the client queries for only rt=brski type resources, the Registrar responds with only the root path for the BRSKI resources (rt=brski, resource /b in earlier examples) and no others. (So, a query for rt=brski, without the wildcard character.) This is shown in the below example. The Pledge in this case requests only the BRSKI root resource of type rt=brski to check if BRSKI is supported by the Registrar and if short names are supported or not. In this case, the Pledge is not interested to check what voucher request formats, or status telemetry formats -- other than the mandatory default formats -- are supported. The compact response then shows that the Registrar indeed supports a short-name BRSKI resource at /b:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</b>;rt=brski
```

The Pledge can now start using any of the BRSKI resources /b/<short-name>. In above example, the well-known resource present under /.well-known/brski is not returned because this is assumed to be well-known to the Pledge and would not require discovery anyway.

As a follow-up example, the Pledge can now start the bootstrap by sending its PVR:

```
REQ: POST /b/rv
Content-Format: 836
Accept: 836
Payload: (binary COSE-signed PVR)
```

14.3. Usage of ct Attribute

The return of multiple content-types in the "ct" attribute by the Registrar allows the Pledge to choose the most appropriate one for a particular operation, and allows extension with new voucher formats. Note that only Content-Format 836 ("application/voucher-cose+cbor") is defined in this document for the voucher request resource (/rv), both as request payload and as response payload.

Content-Format 836 MUST be supported by the Registrar for the /rv resource. If the "ct" attribute is not indicated for the /rv resource in the link format description, this implies that at least format 836 is supported.

Note that this specification allows for voucher-cose+cbor format requests and vouchers to be transmitted over HTTPS, as well as for voucher-cms+json and other formats yet to be defined over CoAP. The burden for this flexibility is placed upon the Registrar. A Pledge on constrained hardware is expected to support a single format only.

The Pledge and MASA need to support one or more formats (at least format 836) for the voucher and for the voucher request. The MASA needs to support all formats that the Pledge supports.

In the below example, a Pledge queries specifically for the brski.rv resource type to learn what voucher formats are supported:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</b/rv>;rt=brski.rv;ct="836 65123 65124"
```

The Registrar returns 3 supported voucher formats: 836, 65123, and 65124. The first is the mandatory "application/voucher-cose+cbor". The other two are numbers from the Experimental Use number range of the CoAP Content-Formats sub-registry, which are used as mere examples. The Pledge can now make a selection between the supported formats.

Note that if the Registrar only supports the default Content-Formats for each BRSKI resource as specified by this document, it may also omit the ct attributes in the discovery query response. For example as in the following interaction:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</b>;rt=brski,
</b/rv>;rt=brski.rv,
</b/vs>;rt=brski.vs,
</b/es>;rt=brski.es
```

14.4. EST-coaps Resource Discovery

The Pledge can also use discovery to identify enrollment options, for example enrollment using EST-coaps or other methods. The below example shows a Pledge that wants to identify EST-coaps enrollment options by sending a discovery query:

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

RES: 2.05 Content
Content-Format: 40
Payload:
</e/crts>;rt=ace.est.crts;ct="281 287",
</e/sen>;rt=ace.est.sen;ct="281 287",
</e/sren>;rt=ace.est.sren;ct="281 287",
</e/att>;rt=ace.est.att,
</e/skg>;rt=ace.est.skg,
</e/skc>;rt=ace.est.skc
```

The response indicates that EST-coaps enrollment (/sen) and re-enrollment (/sren) is supported, with a choice of two Content-Formats

for the return payload: either a PKCS#7 container of certificates ("application/pkcs7-mime;smime-type=certs-only", type 281) or a single LDevID certificate ("application/pkix-cert", type 287).

Also for the EST cacerts resources (/crts) either a PKCS#7 container with all CA certificates can be returned, or a single (most relevant) CA certificate.

The Pledge can now send a CoAP request to one or more of the discovered resources, with the Accept Option to indicate which return payload format the Pledge wants to receive.

15. Security Considerations

15.1. Duplicate serial-numbers

In the absense of correct use of idevid-issuer by the Registrar as detailed in Section 8.4, it would be possible for a malicious Registrar to use an unauthorized voucher for a device. This would apply only to the case where a Manufacturer Authorized Signing Authority (MASA) is trusted by different products from the same manufacturer, and the manufacturer has duplicated serial numbers as a result of a merge, acquisition or mis-management.

For example, imagine the same manufacturer makes light bulbs as well as gas centrifuges, and said manufacturer does not uniquely allocate product serial numbers. This attack only works for nonceless vouchers. The attacker has obtained a light bulb which happens to have the same serial-number as a gas centrofuge which it wishes to obtain access. The attacker performs a normal BRSKI onboarding for the light bulb, but then uses the resulting voucher to onboard the gas centrofuge. The attack requires that the gas centrofuge be returned to a state where it is willing to perform a new onboarding operation.

This attack is prevented by the mechanism of having the Registrar include the idevid-issuer in the RVR, and the MASA including it in the resulting voucher. The idevid-issuer is not included by default: a MASA needs to be aware if there are parts of the organization which duplicates serial numbers, and if so, include it.

15.2. IDevID security in Pledge

The security of this protocol depends upon the Pledge identifying itself to the Registrar using it's manufacturer installed certificate: the IDevID certificate. Associated with this certificate is the IDevID private key, known only to the Pledge. Disclosure of this private key to an attacker would permit the attacker to impersonate the Pledge towards the Registrar, probably gaining access credentials to that Registrar's network.

If the IDevID private key disclosure is known to the manufacturer, there is little recourse other than recall of the relevant part numbers. The process for communicating this recall would be within the BRSKI-MASA protocol. Neither this specification nor [RFC8995] provides for consultation of a Certification Revocation List (CRL) or Open Certificate Status Protocol (OCSP) by a Registrar when evaluating an IDevID certificate. However, the BRSKI-MASA protocol submits the IDevID from the Registrar to the manufacturer's MASA and a manufacturer would have an opportunity to decline to issue a voucher for a device which they believe has become compromised.

It may be difficult for a manufacturer to determine when an IDevID private key has been disclosed. Two situations present themselves: in the first situation a compromised private key might be reused in a counterfeit device, which is sold to another customer. This would present itself as an onboarding of the same device in two different networks. The manufacturer may become suspicious seeing two voucher requests for the same device from different Registrars. Such activity could be indistinguishable from a device which has been resold from one operator to another, or re-deployed by an operator from one location to another.

In the second situation, an attacker having compromised the IDevID private key of a device might then install malware into the same device and attempt to return it to service. The device, now blank, would go through a second onboarding process with the original Registrar. Such a Registrar could notice that the device has been "factory reset" and alert the operator to this situation. One remedy against the presence of malware is through the use of Remote Attestation such as described in [I-D.ietf-rats-architecture]. Future work will need to specify a background-check Attestation flow as part of the voucher-request/voucher-response process. Attestation may still require access to a private key (e.g. IDevID private key) in order to sign Evidence, so a primary goal should be to keep any private key safe within the Pledge.

In larger, more expensive, systems there is budget (power, space, and bill of materials) to include more specific defenses for a private key. For instance, this includes putting the IDevID private key in a Trusted Platform Module (TPM), or use of Trusted Execution Environments (TEE) for access to the key. On smaller IoT devices, the cost and power budget for an extra part is often prohibitive.

It is becoming more and more common for CPUs to have an internal set of one-time fuses that can be programmed (often they are "burnt" by a laser) at the factory. This section of memory is only accessible in some priviledged CPU state. The use of this kind of CPU is appropriate as it provides significant resistance against key disclosure even when the device can be disassembled by an attacker.

In a number of industry verticals, there is increasing concern about counterfeit parts. These may be look-alike parts created in a different factory, or parts which are created in the same factory during an illegal night-shift, but which are not subject to the appropriate level of quality control. The use of a manufacturer-signed IDevID certificate provides for discovery of the pedigree of each part, and this often justifies the cost of the security measures associated with storing the private key.

15.3. Security of CoAP and UDP protocols

Section 7.1 explains that no CoAPS version of the BRSKI-MASA protocol is proposed. The connection from the Registrar to the MASA continues to be HTTPS as in [RFC8995]. This has been done to simplify the MASA deployment for the manufacturer, because no new protocol needs to be enabled on the server.

The use of UDP protocols across the open Internet is sometimes fraught with security challenges. Denial-of-service attacks against UDP based protocols are trivial as there is no three-way handshake as done for TCP. The three-way handshake of TCP guarantees that the node sending the connection request is reachable using the origin IP address. While DTLS contains an option to do a stateless challenge -- a process actually stronger than that done by TCP -- it is not yet common for this mechanism to be available in hardware at multigigabit speeds. It is for this reason that this document defines using HTTPS for the Registrar to MASA connection.

15.4. Registrar Certificate may be self-signed

The provisional (D)TLS connection formed by the Pledge with the Registrar does not authenticate the Registrar's identity. This Registrar's identity is validated by the [RFC8366bis] voucher that is issued by the MASA, signed with an anchor that was built-in to the Pledge.

The Registrar may therefore use any certificate, including a self-signed one. The only restrictions on the certificate is that it MUST have EKU bits set as detailed in Section 6.1.5 and Section 7.4.

15.5. Use of RPK alternatives to proximity-registrar-cert

In [RFC8366bis], Part voucher-request-artifact two compact alternative fields for proximity-registrar-cert are defined that include an RPK: proximity-registrar-pubk and proximity-registrar-pubk-sha256. The Pledge can use these fields in its PVR to identify the Registrar based on its public key only. Since the full certificate of the proximate Registrar is not included, use of these fields by a Pledge implies that a Registrar could insert another certificate with the same public key identity into the RVR. For

example, an older or a newer version of its certificate. The MASA will not be able to detect such act by the Registrar. But since any 'other' certificate the Registrar could insert in this way still encodes its identity the additional risk of using the RPK alternatives is neglible.

When a Registrar sees a PVR that uses one of proximity-registrar-pubk or proximity-registrar-pubk-sha256 fields, this implies the Registrar must include the certificate identified by these fields into its RVR. Otherwise, the MASA is unable to verify proximity. This requirement is already implied by the "MUST" requirement in Section 8.1.

15.6. MASA support of CoAPS

The use of CoAP for the BRSKI-MASA connection is not in scope of the current document. The following security considerations have led to this choice of scope:

* the technology and experience to build secure Internet-scale HTTPS responders (which the MASA is) is common, while the experience in doing the same for CoAP is much less common.

* in many enterprise networks, outgoing UDP connections are often treated as suspicious, which could effectively block CoAP connections for some firewall configurations.

* reducing the complexity of MASA (i.e. less protocols supported) would also reduce its potential attack surface, which is relevant since the MASA is 24/7 exposed on the Internet and accepting (untrusted) incoming connections.

16. IANA Considerations

16.1. GRASP Discovery Registry

IANA is asked to extend the registration of the "AN_Proxy" (without quotes) in the "GRASP Objective Names" table in the Grasp Parameter registry. This document should also be cited for this existing registration, because Section 10.1.1 defines the new protocol value IPPROTO_UDP for the objective.

IANA is asked to extend the registration of the "AN_join_registrar" (without quotes) in the "GRASP Objective Names" table in the Grasp Parameter registry. This document should also be cited for this existing registration, because Section 10.2.1 adds the objective value "BRSKI_JP" to the objective.

16.2. Resource Type Registry

Additions to the sub-registry "Resource Type Link Target Attribute Values", within the "CoRE Parameters" IANA registry are specified below.

Reference: [This RFC]

Attribute	Description
brski	Root path of Bootstrapping Remote Secure Key Infrastructure (BRSKI) resources
brski.rv	BRSKI request voucher resource
brski.vs	BRSKI voucher status telemetry resource
brski.es	BRSKI enrollment status telemetry resource

Table 3: Resource Type (rt) link target attribute values for IANA registration

16.3. Media Types Registry

This section registers the 'application/voucher-cose+cbor' in the IANA "Media Types" registry. This media type is used to indicate that the content is a CBOR voucher or voucher request signed with a COSE_Sign1 structure [RFC9052].

16.3.1. application/voucher-cose+cbor

Type name: application
Subtype name: voucher-cose+cbor
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: binary (CBOR)
Security considerations: Security Considerations of [This RFC].
Interoperability considerations: The format is designed to be broadly interoperable.
Published specification: [This RFC]
Applications that use this media type: ANIMA, 6tisch, and other zero-touch onboarding systems
Fragment identifier considerations: The syntax and semantics of fragment identifiers specified for application/voucher-cose+cbor are as specified for application/cbor. (At publication of this document, there is no fragment identification syntax defined for application/cbor.)
Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .vch
Macintosh file type code(s): N/A
Person & email address to contact for further information: IETF ANIMA Working Group (anima@ietf.org) or IETF Operations and Management Area Working Group (opsawg@ietf.org)
Intended usage: COMMON
Restrictions on usage: N/A
Author: ANIMA WG
Change controller: IETF
Provisional registration? (standards tree only): NO

16.4. CoAP Content-Format Registry

IANA has allocated ID 836 from the sub-registry "CoAP Content-Formats".

Media type	Encoding	ID	Reference
application/voucher-cose+cbor	-	836	[This RFC]

16.5. Update to BRSKI Parameters Registry

This section updates the BRSKI Well-Known URIs sub-registry of the IANA Bootstrapping Remote Secure Key Infrastructures (BRSKI) Parameters Registry by adding a new column "Short URI". The contents of this field MUST be specified for any newly registered URI as follows:

Short URI: A short name for the "URI" resource that can be used by a Constrained BRSKI Pledge in a CoAP request to the Registrar. In case the "URI" resource is only used between Registrar and MASA, the value "--" is registered denoting that a short name is not applicable.

The initial contents of the sub-registry including the new column are as follows:

URI	Short URI	Description	Reference
requestvoucher	rv	Request voucher: Pledge to Registrar, and Registrar to MASA	[RFC8995], [This RFC]
voucher_status	vs	Voucher status telemetry: Pledge to Registrar	[RFC8995], [This RFC]
requestauditlog	--	Request audit log: Registrar to MASA	[RFC8995]
enrollstatus	es	Enrollment status telemetry: Pledge to Registrar	[RFC8995], [This RFC]

Table 4: Update of the BRSKI Well-Known URI Sub-Registry

17. Acknowledgements

We are very grateful to Jim Schaad for explaining COSE/CMS choices and for correcting early versions of the COSE_Sign1 objects.

Michel Veillette did extensive work on _pyang_ to extend it to support the SID allocation process, and this document was among its first users.

Russ Housley , Daniel Franke and Henk Birkholtz provided review feedback.

The BRSKI design team has met on many Tuesdays and Thursdays for document review. The team includes: Aurelio Schellenbaum , David von Oheimb , Steffen Fries , Thomas Werner and Toerless Eckert .

18. Changelog

-22:
Streamlined text to focus mostly on the default flow, with optional functions moved to their own sections (#273). Editorial updates. Reference rfc6125bis updated to RFC 9525.

-11 to -21:
(For change details see GitHub issues https://github.com/anima-wg/constrained-voucher/issues , related Pull Requests and commits.)

-10:
Design considerations extended; Examples made consistent

-08:
Examples for cose_sign1 are completed and improved.

-06:
New SID values assigned; regenerated examples

-04:
voucher and request-voucher MUST be signed; examples for signed request are added in appendix; IANA SID registration is updated; SID values in examples are aligned; signed cms examples aligned with new SIDs.

-03:
Examples are inverted.

-02:
Example of requestvoucher with unsigned appllication/cbor is added; attributes of voucher "refined" to optional; CBOR serialization of vouchers improved; Discovery port numbers are specified.

-01:
application/json is optional, application/cbor is compulsory; Cms and cose mediatypes are introduced.

-00:
initial version

19. References

19.1. Normative References

[ieee802-1AR] IEEE Standard, "IEEE 802.1AR Secure Device Identifier", 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

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

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/

RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

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

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8449] Thomson, M., "Record Size Limit Extension for TLS", RFC 8449, DOI 10.17487/RFC8449, August 2018, <https://www.rfc-editor.org/info/rfc8449>.

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

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

[RFC9031] Vučinić, M., Ed., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", RFC 9031, DOI 10.17487/RFC9031, May 2021, <https://www.rfc-editor.org/info/rfc9031>.

[RFC9032] Dujovne, D., Ed. and M. Richardson, "Encapsulation of 6TiSCH Join and Enrollment Information Elements", RFC 9032, DOI 10.17487/RFC9032, May 2021, <https://www.rfc-editor.org/info/rfc9032>.

[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/info/rfc9052>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version

1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9148] van der Stok, P., Kampanakis, P., Richardson, M., and S. Raza, "EST-coaps: Enrollment over Secure Transport with the Secure Constrained Application Protocol", RFC 9148, DOI 10.17487/RFC9148, April 2022, <https://www.rfc-editor.org/info/rfc9148>.

[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/info/rfc9360>.

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

19.2. Informative References

[COSE-registry] IANA, "CBOR Object Signing and Encryption (COSE) registry", 2017, <https://www.iana.org/assignments/cose/cose.xhtml>.

[I-D.ietf-6lo-mesh-link-establishment] Kelsey, R., "Mesh Link Establishment", Work in Progress, Internet-Draft, draft-ietf-6lo-mesh-link-establishment-00, 1 December 2015, <https://datatracker.ietf.org/doc/html/draft-ietf-6lo-mesh-link-establishment-00>.

[I-D.ietf-anima-constrained-join-proxy] Richardson, M., Van der Stok, P., and P. Kampanakis, "Join Proxy for Bootstrapping of Constrained Network Elements", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-join-proxy-15, 6 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-constrained-join-proxy-15>.

[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-core-sid] Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-

sid-23, 30 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-23>.

[I-D.ietf-lake-edhoc] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-22, 25 August 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-22>.

[I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-architecture-22>.

[I-D.kuehlewind-update-tag] Kühlewind, M. and S. Krishnan, "Definition of new tags for relations between RFCs", Work in Progress, Internet-Draft, draft-kuehlewind-update-tag-04, 12 July 2021, <https://datatracker.ietf.org/doc/html/draft-kuehlewind-update-tag-04>.

[I-D.richardson-anima-masa-considerations] Richardson, M. and W. Pan, "Operational Considerations for Voucher infrastructure for BRSKI MASA", Work in Progress, Internet-Draft, draft-richardson-anima-masa-considerations-08, 9 May 2023, <https://datatracker.ietf.org/doc/html/draft-richardson-anima-masa-considerations-08>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI

10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <https://www.rfc-editor.org/info/rfc8990>.

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

[Thread] Thread Group, Inc, "Thread support page, White Papers", November 2023, <https://www.threadgroup.org/support#Whitepapers>.

Appendix A. Library Support for BRSKI

For the implementation of BRSKI, the use of a software library to manipulate PKIX certificates and use crypto algorithms is often beneficial. Two C-based examples are OpenSSL and mbedtls. Others more targeted to specific platforms or languages exist. It is important to realize that the library interfaces differ significantly between libraries.

Libraries do not support all known crypto algorithms. Before deciding on a library, it is important to look at their supported crypto algorithms and the roadmap for future support. Apart from availability, the library footprint, and the required execution cycles should be investigated beforehand.

The handling of certificates usually includes the checking of a certificate chain. In some libraries, chains are constructed and verified on the basis of a set of certificates, the trust anchor (usually self signed root CA), and the target certificate. In other libraries, the chain must be constructed beforehand and obey order criteria. Verification always includes the checking of the signatures. Less frequent is the checking the validity of the dates or checking the existence of a revoked certificate in the chain against a set of revoked certificates. Checking the chain on the consistency of the certificate extensions which specify the use of the certificate usually needs to be programmed explicitly.

A libary can be used to construct a (D)TLS connection. It is useful to realize that differences beetween (D)TLS implementations will

occur due to the differences in the certicate checks supported by the library. On top of that, checks between client and server certificates enforced by (D)TLS are not always helpful for a BRSKI implementation. For example, the certificates of Pledge and Registrar are usually not related when the BRSKI protocol is started. It must be verified that checks on the relation between client and server certificates do not hamper a succeful DTLS connection establishment.

A.1. OpensSSL

From openssl's apps/verify.c :

```
<CODE BEGINS>
X509 *x = NULL;
int i = 0, ret = 0;
X509_STORE_CTX *csc;
STACK_OF(X509) *chain = NULL;
int num_untrusted;

x = load_cert(file, "certificate file");
if (x == NULL)
    goto end;

csc = X509_STORE_CTX_new();
if (csc == NULL) {
    BIO_printf(bio_err, "error %s: X.509 store context"
               "allocation failed\n",
               (file == NULL) ? "stdin" : file);
    goto end;
}

X509_STORE_set_flags(ctx, vflags);
if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
    X509_STORE_CTX_free(csc);
    BIO_printf(bio_err,
               "error %s: X.509 store context"
               "initialization failed\n",
               (file == NULL) ? "stdin" : file);
    goto end;
}
if (tchain != NULL)
    X509_STORE_CTX_set0_trusted_stack(csc, tchain);
if (crls != NULL)
    X509_STORE_CTX_set0_crls(csc, crls);

i = X509_verify_cert(csc);
X509_STORE_CTX_free(csc);

<CODE ENDS>
```

A.2. mbedTLS

```
<CODE BEGINS>
mbedtls_x509_crt cert;
mbedtls_x509_crt caCert;
uint32_t         certVerifyResultFlags;
...
int result = mbedtls_x509_crt_verify(&cert, &caCert, NULL, NULL,
                            &certVerifyResultFlags, NULL, NULL);

<CODE ENDS>
```

Appendix B. Constrained BRSKI-EST Message Examples

This appendix extends the message examples from Appendix A of [RFC9148] with constrained BRSKI messages. The CoAP headers are only fully worked out for the first example, enrollstatus.

B.1. enrollstatus

A coaps enrollstatus message from Pledge to Registrar can be as follows:

```
REQ: POST coaps://192.0.2.1:8085/b/es
Content-Format: 60
Payload: <binary CBOR encoding of an enrollstatus map>
```

The corresponding CoAP header fields for this request are shown below.

```
Ver = 1
T = 0 (CON)
TKL = 1
Code = 0x02 (0.02 is POST method)
Message ID = 0xab0f
Token = 0x4d
Options
 Option  (Uri-Path)
   Option Delta = 0xb   (option nr = 11)
   Option Length = 0x1
   Option Value = "b"
 Option  (Uri-Path)
   Option Delta = 0x0   (option nr = 11)
   Option Length = 0x2
   Option Value = "es"
 Option  (Content-Format)
   Option Delta = 0x1   (option nr = 12)
   Option Length = 0x1
   Option Value = 60    (application/cbor)
Payload Marker = 0xFF
Payload = A26776657273696F6E0166737461747573F5 (18 bytes binary)

 The Uri-Host and Uri-Port Options are omitted because they coincide
 with the transport protocol (UDP) destination address and port
 respectively.

 The above binary CBOR enrollstatus payload looks as follows in CBOR
 diagnostic notation, for the case of enrollment success:

{
  "version": 1,
  "status": true
 }

 Alternatively the payload could look as follows in case of enrollment
 failure, using the reason field to describe the failure:

Payload = A36776657273696F6E0166737461747573F466726561736F6E782A3C
          496E666F726D617469766520686D616E207265616461626C652065
          72726F72206D6573736167653E    (69 bytes binary)

{
  "version": 1,
  "status": false,
  "reason": "<Informative human readable error message>"
}

 To indicate successful reception of the enrollmentstatus telemetry
 report, a response from the Registrar may then be:
```

```
2.04 Changed
```

Which in case of a piggybacked response has the following CoAP header fields:

```
Ver=1
T=2 (ACK)
TKL=1
Code = 0x44 (2.04 Changed)
Message ID = 0xab0f
Token = 0x4d
```

B.2. voucher_status

A coaps voucher_status message from Pledge to Registrar can be as follows:

```
REQ: POST coaps://[2001:db8::2:1]/.well-known/brski/vs
Content-Format: 60 (application/cbor)
Payload =
  A46776657273696F6E0166737461747573F466726561736F6E7828496E66
  6F726D6174976652068756D616E2D7265616461626C65206572726F7220
  6D6573736167656E726561736F6E2D636F6E74657874A100764164646974
  696F6E616C20696E666F726D6174696F6E
```

The request payload above is binary CBOR but represented here in hexadecimal for readability. Below is the equivalent CBOR diagnostic format.

```
{
  "version": 1,
  "status": false,
  "reason": "Informative human-readable error message",
  "reason-context": { 0: "Additional information" }
}
```

A success response without payload will then be sent by the Registrar back to the Pledge to indicate reception of the telemetry report:

```
2.04 Changed
```

Appendix C. COSE-signed Voucher (Request) Examples

This appendix provides examples of COSE-signed voucher requests and vouchers. First, the used test keys and PKIX certificates are described, followed by examples of a constrained PVR, RVR and voucher.

C.1. Pledge, Registrar and MASA Keys

This section documents the public and private keys used for all examples in this appendix. These keys are not used in any production system, and must only be used for testing purposes.

C.1.1. Pledge IDevID private key

```
<CODE BEGINS>
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIMv+C4dbzeyrEH20qkpFlWIH2FFACGZv9kW7rNWtSlYtoAoGCCqGSM49
AwEHoUQDQgAESH6OUiYFRhfIgWl4GG8jHoj8a+8rf6t5s1mZ/4SePlKom39GQ34p
VYryJ9aHmboLLfz69bzICQFKbkoQ5oaiew==
-----END EC PRIVATE KEY-----
<CODE ENDS>
```

```
<CODE BEGINS>
Private-Key: (256 bit)
priv:
    cb:fe:0b:87:5b:cd:ec:ab:10:7d:b4:aa:4a:45:95:
    62:07:d8:51:40:08:66:6f:f6:45:bb:ac:d5:ad:4a:
    56:2d
pub:
    04:48:7e:8e:52:26:05:46:17:c8:81:69:78:18:6f:
    23:1e:88:fc:6b:ef:2b:7f:ab:79:b3:59:99:ff:84:
    9e:3e:52:a8:9b:7f:46:43:7e:29:55:8a:f2:27:d6:
    87:99:ba:0b:2d:fc:fa:f5:bc:c8:09:01:4a:6e:4a:
    10:e6:86:a2:7b
ASN1 OID: prime256v1
NIST CURVE: P-256

<CODE ENDS>
```

C.1.2. Registrar private key

```
<CODE BEGINS>
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgYJ/MP0dWA9BkYd4W
s6oRY62hDddaEmrAVm5dtAXE/UGhRANCAAQgMIVb6EaRCz7LFcr4Vy0+tWW9xlSh
Xvr27euqi54WCMXJEMk6IIaPyFBNNw8bJvqXWfZ5g7t4hj7amsvqUST2
-----END PRIVATE KEY-----
<CODE ENDS>
```

```
<CODE BEGINS>
Private-Key: (256 bit)
priv:
    60:9f:cc:3f:47:56:03:d0:64:61:de:16:b3:aa:11:
    63:ad:a1:0d:d7:5a:12:6a:c0:56:6e:5d:b4:05:c4:
    fd:41
pub:
    04:20:30:85:5b:e8:46:91:0b:3e:cb:15:ca:f8:57:
    2d:3e:b5:65:bd:c6:54:a1:5e:fa:f6:ed:eb:aa:8b:
    9e:16:08:c5:c9:10:c9:3a:20:86:8f:c8:50:4d:37:
    0f:1b:26:fa:97:59:f6:79:83:bb:78:86:3e:da:9a:
    cb:ea:51:24:f6
ASN1 OID: prime256v1
NIST CURVE: P-256

<CODE ENDS>
```

C.1.3. MASA private key

```
<CODE BEGINS>
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgrbJ1oU+HIJ2SWYAk
DkBTL+YNPxQG+gwsMsZB94N8mZ2hRANCAASS9NVlWJdztwNY81yPlH2UODYWhlYA
ZfsqnEPSFZKnq8mq8gF78ZVbYi6q2FEg8kkORY/rpIU/X7SQsRuD+wMW
-----END PRIVATE KEY-----
<CODE ENDS>

<CODE BEGINS>
Private-Key: (256 bit)
priv:
    ad:b2:75:a1:4f:87:20:9d:92:59:80:24:0e:40:53:
    2f:e6:0d:3f:14:06:fa:0c:2c:32:c6:41:f7:83:7c:
    99:9d
pub:
    04:92:f4:d5:65:58:97:73:b7:03:58:f3:5c:8f:94:
    7d:94:38:36:16:86:56:00:65:fb:2a:9c:43:d2:15:
    92:a7:ab:c9:aa:f2:01:7b:f1:95:5b:62:2e:aa:d8:
    51:20:f2:49:0e:45:8f:eb:a4:85:3f:5f:b4:90:b1:
    1b:83:fb:03:16
ASN1 OID: prime256v1
NIST CURVE: P-256

<CODE ENDS>
```

C.2. Pledge, Registrar, Domain CA and MASA Certificates

All keys and PKIX certificates used for the examples have been generated with OpenSSL - see Appendix D for more details on certificate generation. Below the certificates are listed that

accompany the keys shown above. Each certificate description is followed by the hexadecimal representation of the X.509 ASN.1 DER encoded certificate. This representation can be for example decoded using an online ASN.1 decoder.

C.2.1. Pledge IDevID Certificate

```
<CODE BEGINS>
Certificate:
Data:
 Version: 3 (0x2)
 Serial Number: 32429 (0x7ead)
 Signature Algorithm: ecdsa-with-SHA256
 Issuer: CN = masa.stok.nl, O = vanderstok, L = Helmond,
         C = NL
 Validity
   Not Before: Dec  9 12:50:47 2022 GMT
   Not After : Dec 31 12:50:47 9999 GMT
 Subject: CN = Stok IoT sensor Y-42, serialNumber = JADA123456789
 Subject Public Key Info:
   Public Key Algorithm: id-ecPublicKey
     Public-Key: (256 bit)
     pub:
       04:48:7e:8e:52:26:05:46:17:c8:81:69:78:18:6f:
       23:1e:88:fc:6b:ef:2b:7f:ab:79:b3:59:99:ff:84:
       9e:3e:52:a8:9b:7f:46:43:7e:29:55:8a:f2:27:d6:
       87:99:ba:0b:2d:fc:fa:f5:bc:c8:09:01:4a:6e:4a:
       10:e6:86:a2:7b
     ASN1 OID: prime256v1
     NIST CURVE: P-256
 X509v3 extensions:
   X509v3 Key Usage: critical
     Digital Signature, Non Repudiation, Key Encipherment,
             Data Encipherment
   X509v3 Basic Constraints:
     CA:FALSE
   X509v3 Authority Key Identifier:
     CB:8D:98:CA:74:C5:1B:58:DD:E7:AC:EF:86:9A:94:43:A8:D6:66:A6
   1.3.6.1.5.5.7.1.32:
      hl=2 l=  12 prim: IA5STRING     :masa.stok.nl

Signature Algorithm: ecdsa-with-SHA256
Signature Value:
 30:45:02:20:4d:89:90:7e:03:fb:52:56:42:0c:3f:c1:b1:f1:
 47:b5:b3:93:65:45:2e:be:50:db:67:85:8f:23:89:a2:3f:9e:
 02:21:00:95:33:69:d1:c6:db:f0:f1:f6:52:24:59:d3:0a:95:
 4e:b2:f4:96:a1:31:3c:7b:d9:2f:28:b3:29:71:bb:60:df

<CODE ENDS>
```

Below is the hexadecimal representation of the binary X.509 DER-encoded certificate:

<CODE BEGINS>
308201CE30820174A00302010202027EAD300A06082A8648CE3D040302304B31
15301306035504030C0C6D6173612E73746F6B2E6E6C31133011060355040A0C
0A76616E64657273746F6B3110300E06035504070C0748656C6D6F6E64310B30
09060355040613024E4C3020170D323231323039313235303437 5A180F393939
393132333131323530343 75A3037311D301B06035504030C1453746F6B20496F
542073656E736F7220592D343231163014060355040513 0D4A41444131323334
353637383930593013 06072A8648CE3D020106082A8648CE3D03010703420004
487E8E5226054617C8816978186F231E88FC6BEF2B7FAB79B35999FF849E3E52
A89B7F46437E29558AF227D68799BA0B2DFCFAF5BCC809014A6E4A10E686A27B
A35A3058300E0603551D0F0101FF0404030204F030090603551D130402300030
1F0603551D23041830168014CB8D98CA74C51B58DDE7ACEF869A9443A8D666A6
301A06082B06010505070120040E160C6D6173612E73746F6B2E6E6C300A0608
2A8648CE3D040302034800304502204D89907E03FB5256420C3FC1B1F147B5B3
9365452EBE50DB67858F2389A23F9E022100953369D1C6DBF0F1F6522459D30A
954EB2F496A1313C7BD92F28B32971BB60DF

<CODE ENDS>

C.2.2. Registrar Certificate

```
<CODE BEGINS>
Certificate:
Data:
 Version: 3 (0x2)
 Serial Number:
   c3:f6:21:49:b2:e3:0e:3e
 Signature Algorithm: ecdsa-with-SHA256
 Issuer: CN = Custom-ER Global CA, OU = IT, O = "Custom-ER, Inc.",
         L = San Jose, ST = CA, C = US
 Validity
   Not Before: Dec  9 12:50:47 2022 GMT
   Not After : Dec  8 12:50:47 2025 GMT
 Subject: CN = Custom-ER Registrar, OU = Office dept, O = "Custom-ER,
         Inc.", L = Ottowa, ST = ON, C = CA
 Subject Public Key Info:
   Public Key Algorithm: id-ecPublicKey
     Public-Key: (256 bit)
     pub:
       04:20:30:85:5b:e8:46:91:0b:3e:cb:15:ca:f8:57:
       2d:3e:b5:65:bd:c6:54:a1:5e:fa:f6:ed:eb:aa:8b:
       9e:16:08:c5:c9:10:c9:3a:20:86:8f:c8:50:4d:37:
       0f:1b:26:fa:97:59:f6:79:83:bb:78:86:3e:da:9a:
       cb:ea:51:24:f6
     ASN1 OID: prime256v1
     NIST CURVE: P-256
 X509v3 extensions:
   X509v3 Key Usage: critical
     Digital Signature, Non Repudiation, Key Encipherment,
             Data Encipherment
   X509v3 Basic Constraints:
     CA:FALSE
   X509v3 Subject Key Identifier:
     C9:08:0B:38:7D:8D:D8:5B:3A:59:E7:EC:10:0B:86:63:93:A9:CA:4C
   X509v3 Authority Key Identifier:
     92:EA:76:40:40:4A:8F:AB:4F:27:0B:F3:BC:37:9D:86:CD:72:80:F8
   X509v3 Extended Key Usage: critical
     CMC Registration Authority, TLS Web Server Authentication,
             TLS Web Client Authentication
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
 30:45:02:21:00:d8:4a:7c:69:2f:f9:58:6e:82:22:87:18:f6:
 3b:c3:05:f0:ae:b8:ae:ec:42:78:82:38:79:81:2a:5d:15:61:
 64:02:20:08:f2:3c:13:69:13:b0:2c:e2:63:09:d5:99:4f:eb:
 75:70:af:af:ed:98:cd:f1:12:11:c0:37:f7:18:4d:c1:9d

<CODE ENDS>
```

Below is the hexadecimal representation of the binary X.509 DER-encoded certificate:

<CODE BEGINS>
3082026D30820213A003020102020900C3F62149B2E30E3E300A06082A8648CE
3D0403023072311C301A06035504030C13437573746F6D2D455220476C6F6261
6C204341310B3009060355040B0C024954311830160603550 40A0C0F43757374
6F6D2D45522C20496E632E3111300F06035504070C0853616E204A6F7365310B
300906035504080C024341310B3009060355040613025553301E170D32323132
303931323530343 75A170D32353132303831323530343 75A3079311C301A0603
5504030C13437573746F6D2D4552205265676973747261723114301206035504
0B0C0B4F6666696365206465707431183016060355040A0C0F437573746F6D2D
45522C20496E632E310F300D06035504070C064F74746F7761310B3009060355
04080C024F4E310B3009060355040613024341305930 1306072A8648CE3D0201
06082A8648CE3D030107034200042030855BE846910B3ECB15CAF8572D3EB565
BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC8504D370F1B26FA9759
F67983BB78863EDA9ACBEA5124F6A3818A308187300E0603551D0F0101FF0404
030204F030090603551D1304023000301D0603551D0E04160414C9080B387D8D
D85B3A59E7EC100B866393A9CA4C301F0603551D2304183016801492EA764040
4A8FAB4F270BF3BC379D86CD7280F8302A0603551D250101FF0420301E06082B
0601050507031C06082B0601050507030106082B0601050507030 2300A06082A
8648CE3D0403020348003045022100D84A7C692FF9586E82228718F63BC305F0
AEB8AEEC4278823879812A5D156164022008F23C136913B02CE26309D5994FEB
7570AFAFED98CDF11211C037F7184DC19D

<CODE ENDS>

C.2.3. Domain CA Certificate

The Domain CA certificate is the CA of the customer's domain. It has signed the Registrar (RA) certificate.

```
<CODE BEGINS>
Certificate:
Data:
 Version: 3 (0x2)
 Serial Number: 3092288576548618702 (0x2aea0413a42dc1ce)
 Signature Algorithm: ecdsa-with-SHA256
 Issuer: CN = Custom-ER Global CA, OU = IT, O = "Custom-ER, Inc.",
         L = San Jose, ST = CA, C = US
 Validity
   Not Before: Dec  9 12:50:47 2022 GMT
   Not After : Dec  6 12:50:47 2032 GMT
 Subject: CN = Custom-ER Global CA, OU = IT, O = "Custom-ER, Inc.",
         L = San Jose, ST = CA, C = US
 Subject Public Key Info:
   Public Key Algorithm: id-ecPublicKey
     Public-Key: (256 bit)
     pub:
       04:97:b1:ed:96:91:64:93:09:85:bb:b8:ac:9a:2a:
       f9:45:5c:df:ee:a4:b1:1d:e2:e7:9d:06:8b:fa:80:
       39:26:b4:00:52:51:b3:4f:1c:08:15:a4:cb:e0:3f:
       bd:1b:bc:b6:35:f6:43:1a:22:de:78:65:3b:87:b9:
       95:37:ec:e1:6c
     ASN1 OID: prime256v1
     NIST CURVE: P-256
 X509v3 extensions:
   X509v3 Subject Alternative Name:
     email:help@custom-er.example.com
   X509v3 Key Usage: critical
     Digital Signature, Certificate Sign, CRL Sign
   X509v3 Basic Constraints: critical
     CA:TRUE
   X509v3 Subject Key Identifier:
     92:EA:76:40:40:4A:8F:AB:4F:27:0B:F3:BC:37:9D:86:CD:72:80:F8
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
 30:44:02:20:66:15:df:c3:70:11:f6:73:78:d8:fd:1c:2a:3f:
 bd:d1:3f:51:f6:b6:6f:2d:7c:e2:7a:13:18:21:bb:70:f0:c0:
 02:20:69:86:d8:d2:28:b2:92:6e:23:9e:19:0b:8f:18:25:c9:
 c1:4c:67:95:ff:a0:b3:24:bd:4d:ac:2e:cb:68:d7:13

<CODE ENDS>
```

Below is the hexadecimal representation of the binary X.509 DER-encoded certificate:

<CODE BEGINS>
30820242308201E9A00302010202082AEA0413A42DC1CE300A06082A8648CE3D
0403023072311C301A06035504030C13437573746F6D2D455220476C6F62616C
204341310B3009060355040B0C024954311830160603550 40A0C0F437573746F
6D2D45522C20496E632E3111300F06035504070C0853616E204A6F7365310B30
0906035504080C024341310B300906035504061302555330 59301306072A8648CE3D020106082A8648CE3D03
01070342000497B1ED969164930985BBB8AC9A2AF9455CDFEEA4B11DE2E79D06
8BFA803926B4005251B34F1C0815A4CBE03FBD1BBCB635F6431A22DE78653B87
B99537ECE16CA3693067302506035 51D11041E301C811A68656C704063757374
6F6D2D65722E6578616D706C652E636F6D300E0603551D0F0101FF0404030201
86300F0603551D130101FF040530030101FF301D0603551D0E0416041492EA76
40404A8FAB4F270BF3BC379D86CD7280F8300A06082A8648CE3D040302034700
304402206615DFC37011F67378D8FD1C2A3FBDD13F51F6B66F2D7CE27A131821
BB70F0C002206986D8D228B2926E239E190B8F1825C9C14C6795FFA0B324BD4D
AC2ECB68D713

<CODE ENDS>

C.2.4. MASA Certificate

The MASA CA certificate is the CA that signed the Pledge's IDevID certificate.

```
<CODE BEGINS>
Certificate:
Data:
 Version: 3 (0x2)
 Serial Number:
   e3:9c:da:17:e1:38:6a:0a
 Signature Algorithm: ecdsa-with-SHA256
 Issuer: CN = masa.stok.nl, O = vanderstok, L = Helmond,
         C = NL
 Validity
   Not Before: Dec  9 12:50:47 2022 GMT
   Not After : Dec  6 12:50:47 2032 GMT
 Subject: CN = masa.stok.nl, O = vanderstok, L = Helmond,
         C = NL
 Subject Public Key Info:
   Public Key Algorithm: id-ecPublicKey
     Public-Key: (256 bit)
     pub:
       04:92:f4:d5:65:58:97:73:b7:03:58:f3:5c:8f:94:
       7d:94:38:36:16:86:56:00:65:fb:2a:9c:43:d2:15:
       92:a7:ab:c9:aa:f2:01:7b:f1:95:5b:62:2e:aa:d8:
       51:20:f2:49:0e:45:8f:eb:a4:85:3f:5f:b4:90:b1:
       1b:83:fb:03:16
     ASN1 OID: prime256v1
     NIST CURVE: P-256
 X509v3 extensions:
   X509v3 Subject Alternative Name:
     email:info@masa.stok.nl
   X509v3 Key Usage: critical
     Digital Signature, Certificate Sign, CRL Sign
   X509v3 Basic Constraints: critical
     CA:TRUE, pathlen:3
   X509v3 Subject Key Identifier:
     CB:8D:98:CA:74:C5:1B:58:DD:E7:AC:EF:86:9A:94:43:A8:D6:66:A6
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
 30:46:02:21:00:94:3f:a5:26:51:68:16:38:5b:78:9a:d8:c3:
 af:8e:49:28:22:60:56:26:43:4a:14:98:3e:e1:e4:81:ad:ca:
 1b:02:21:00:ba:4d:aa:fd:fa:68:42:74:03:2b:a8:41:6b:e2:
 90:0c:9e:7b:b8:c0:9c:f7:0e:3f:b4:36:8a:b3:9c:3e:31:0e

<CODE ENDS>
```

Below is the hexadecimal representation of the binary X.509 DER-encoded certificate:

```
<CODE BEGINS>
308201F130820196A003020102020900E39CDA17E1386A0A300A06082A8648CE
3D040302304B31153013060355040304030C0C6D6173612E73746F6B2E6E6C311330
11060355040A0C0A76616E64657273746F6B3110300E06035504070C0748656C
6D6F6E64310B300906035504061302E4C301E170D3232313230393132353034
375A170D3332313230363132353034375A304B31153013060355040304030C0C6D61
73612E73746F6B2E6E6C3113301106035504A0C0A76616E64657273746F6B31
10300E06035504070C0748656C6D6F6E64310B300906035504061302E4C3059
301306072A8648CE3D020106082A8648CE3D030107034200492F4D565589773
B70358F35C8F947D9438361686560065FB2A9C43D21592A7ABC9AAF2017BF195
5B622EAAD85120F2490E458FEBA4853F5FB490B11B83FB0316A3633061301C06
03551D11041530138111696E666F406D6173612E73746F6B2E6E6C300E060355
1D0F0101FF040403020186301206035551D130101FF040830060101FF02010330
1D0603551D0E04160414CB8D98CA74C51B58DDE7ACEF869A9443A8D666A6300A
06082A8648CE3D040302034900304602210094 3FA526516816385B789AD8C3AF
8E492822605626434A14983EE1E481ADCA1B022100BA4DAAFDFA684274032BA8
416BE2900C9E7BB8C09CF70E3FB4368AB39C3E310E

<CODE ENDS>
```

C.3. COSE-signed Pledge Voucher Request (PVR)

In this example, the voucher request (PVR) has been signed by the Pledge using the IDevID private key of Appendix C.1.1, and has been sent to the link-local constrained Join Proxy (JP) over CoAPS to the JP's join port. The join port happens to use the default CoAPS UDP port 5684.

```
REQ: POST coaps://[JP-link-local-address]/b/rv
Content-Format: 836
Payload: <signed_pvr>
```

When the Join Proxy receives the DTLS handshake messages from the Pledge, it will relay these messages to the Registrar. The payload signed_voucher_request is shown as hexadecimal dump (with lf added) below:

```
<CODE BEGINS>
D28443A10126A0587EA11909C5A40102074823BFBBC9C2BCF2130C585B305930
1306072A8648CE3D020106082A8648CE3D030107034200042030855BE846910B
3ECB15CAF8572D3EB565BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868F
C8504D370F1B26FA9759F67983BB78863EDA9ACBEA5124F60D6D4A4144413132
333435363738395840 68987DE8B007F4E9416610BBE2D48E1D7EA1032092B8BF
CE611421950F45B22F17E214820C07E777ADF86175E25D3205568404C25FCEEC
1B817C7861A6104B3D

<CODE ENDS>
```

The representiation of signed_pvr in CBOR diagnostic format (with lf added) is:

<CODE BEGINS>
18([h'A10126', {}, h'A11909C5A40102074823BFBBC9C2BCF2130C585B3059301
306072A8648CE3D020106082A8648CE3D03010703420004203085 5BE846910B3ECB1
5CAF8572D3EB565BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC8504D370
F1B26FA9759F67983BB78863EDA9ACBEA5124F60D6D4A414441313233343536373 83
9', h'68987DE8B007F4E9416610BBE2D48E1D7EA1032092B8BFCE611421950F45B2
2F17E214820C07E777ADF86175E25D3205568404C25FCEEC1B817C7861A6104B3D']
)

<CODE ENDS>

The COSE payload is the PVR, encoded as a CBOR byte string. The diagnostic representation of it is shown below:

<CODE BEGINS>
{2501: {1: 2, 7: h'23BFBBC9C2BCF213', 12: h'3059301306072A8648CE3D02
0106082A8648CE3D03010703420004203085 5BE846910B3ECB15CAF8572D3EB565BD
C654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC8504D370F1B26FA9759F67983
BB78863EDA9ACBEA5124F6', 13: "JADA123456789"}}

<CODE ENDS>

The Pledge uses the "proximity" (key '1', SID 2502, enum value 2) assertion together with an included proximity-registrar-pubk field (key '12', SID 2513) to inform MASA about its proximity to the specific Registrar.

C.4. COSE-signed Registrar Voucher Request (RVR)

In this example the Registrar's voucher request has been signed by the JRC (Registrar) using the private key from Appendix C.1.2. Contained within this voucher request is the voucher request PVR that was made by the Pledge to JRC. Note that the RVR uses the HTTPS protocol (not CoAP) and corresponding long URI path names as defined in [RFC8995]. The Content-Type and Accept headers indicate the constrained voucher format that is defined in the present document. Because the Pledge used this format in the PVR, the JRC must also use this format in the RVR.

```
REQ: POST https://masa.stok.nl/.well-known/brski/requestvoucher
Content-Type: application/voucher-cose+cbor
Accept: application/voucher-cose+cbor
Body: <signed_rvr>
```

The payload signed_rvr is shown as hexadecimal dump (with lf added):

<CODE BEGINS>
D28443A10126A11820825902843082028030820225A003020102020900C3F621
49B2E30E3E300A06082A8648CE3D0403023072311C301A06035504030C134375
73746F6D2D455220476C6F62616C204341310B3009060355040B0C0249543118
3016060355040A0C0F437573746F6D2D45522C20496E632E3111300F06035504
070C0853616E204A6F7365310B300906035504080C024341310B300906035504
0613025553301E170D3232313230363131333735395A170D3235313230353131
333735395A30818D3131302F06035504030C28437573746F6D2D455220436F6D
6D65726369616C204275696C64696E6773205265676973747261723113301106
0355040B0C0A4F6666696365206F7073311830160603550 40A0C0F437573746F
6D2D45522C20496E632E310F300D06035504070C064F74746F7761310B300906
035504080C024F4E310B3009060355040613024341305930130 6072A8648CE3D
020106082A8648CE3D0301070342000420308 55BE846910B3ECB15CAF8572D3E
B565BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC8504D370F1B26FA
9759F67983BB78863EDA9ACBEA5124F6A3818730818430090603551D13040230
00300B0603551D0F0404030204F0301D0603551D0E04160414C9080B387D8DD8
5B3A59E7EC100B866393A9CA4C301F0603551D2304183016801492EA7640404A
8FAB4F270BF3BC379D86CD7280F8302A0603551D250101FF0420301E06082B06
01050507031C06082B0601050507030106082B0601050507030230 0A06082A86
48CE3D0403020349003046022100 91A2033692EB81503D53505FFC8DA326B1EE
7DEA96F29174F0B3341A07812201022100FF7339288108B712F418530A18025A
895408CC45E0BB678B46FBAB37DDB4D36B59024730820243308201E9A0030201
0202082AEA0413A42DC1CE300A06082A8648CE3D0403023072311C301A060355
04030C13437573746F6D2D455220476C6F62616C204341310B3009060355040B
0C0249543118301606035504 0A0C0F437573746F6D2D45522C20496E632E3111
300F06035504070C0853616E204A6F7365310B300906035504080C024341310B
3009060355040613025553301E170D3232313230363131333735395A170D3332
3132303331313337353 95A3072311C301A06035504030C13437573746F6D2D45
5220476C6F62616C204341310B3009060355040B0C0249543118301606035504
0A0C0F437573746F6D2D45522C20496E632E3111300F06035504070C0853616E
204A6F7365310B300906035504080C024341310B3009060355040613025553330
59301306072A8648CE3D020106082A8648CE3D0301070342000497B1ED969164
930985BBB8AC9A2AF9455CDFEEA4B11DE2E79D068BFA803926B4005251B34F1C
0815A4CBE03FBD1BBCB635F6431A22DE78653B87B99537ECE16CA3693067300F
0603551D130101FF040530030101FF30250603551D11041E301C811A68656C70
40637573746F6D2D65722E6578616D706C652E636F6D300E0603551D0F0101FF
040403020186301D0603551D0E0416041492EA7640404A8FAB4F270BF3BC379D
86CD7280F8300A06082A8648CE3D0403020348003045022100D6D813B390BD3A
7B4E85424BCB1ED933AD1E981F2817B59083DD6EC1C5E3FADF02202CEE440619
2BC767E98D7CFAE044C6807481AD8564A7D569DCA3D1CDF1E5E843590124A119
09C5A601020278183230323 22D31322D30365432303A30343A31352E3735345A
05581A041830168014CB8D98CA74C51B58DDE7ACEF869A9443A8D666A6074823
BFBBC9C2BCF2130958C9D28443A10126A0587EA11909C5A40102074823BFBBC9
C2BCF2130C585B3059301306072A8648CE3D020106082A8648CE3D0301070342
00042030855BE846910B3ECB15CAF8572D3EB565BDC654A15EFAF6EDEBAA8B9E
1608C5C910C93A20868FC8504D370F1B26FA9759F67983BB78863EDA9ACBEA51
24F60D6D4A414441313233343536373839584068987DE8B007F4E9416610BBE2
D48E1D7EA1032092B8BFCE611421950F45B22F17E214820C07E777ADF86175E2
5D3205568404C25FCEEC1B817C7861A6104B3D0D6D4A41444131323334353637

```
38395840B1DD40B10787437588AEAC9036899191C16CCDBECA31C197855CCB6B
BA142D709FE329CBC3F76297D6063ACB6759EAB98E96EA4C4AA2135AA48A247B
AC1D6A3F

<CODE ENDS>
```

The representiation of signed_rvr in CBOR diagnostic format (with lf added) is:

<CODE BEGINS>
18([h'A10126', {32: [h'308202830820225A003020102020900C3F62149B2E30
E3E300A06082A8648CE3D0403023072311C301A06035504030C13437573746F6D2D4
55220476C6F62616C204341310B3009060355040B0C024954311830160603550 40A0
C0F437573746F6D2D45522C20496E632E3111300F06035504070C0853616E204A6F7
365310B300906035504080C024341310B300906035504061302555330 1E170D32323
132303631313337353 95A170D323531323035313133373539 5A30818D3131302F060
35504030C28437573746F6D2D455220436F6D6D65726369616C204275696C64696E6
773205265676973747261723113301106035504 0B0C0A4F6666696365206F7073311
83016060355040A0C0F437573746F6D2D45522C20496E632E310F300D06035504070
C064F74746F7761310B300906035504080C024F4E310B300906035504061302434 13
059301306072A8648CE3D020106082A8648CE3D030107034200042030855BE846910
B3ECB15CAF8572D3EB565BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC85
04D370F1B26FA9759F67983BB78863EDA9ACBEA5124F6A3818730818430090603551
D1304023000300B0603551D0F0404030204F0301D0603551D0E04160414C9080B387
D8DD85B3A59E7EC100B866393A9CA4C301F0603551D230418301680 1492EA7640404
A8FAB4F270BF3BC379D86CD7280F8302A0603551D250101FF0420301E06082B06010
50507031C06082B060105050703010 6082B060105050703023 00A06082A8648CE3D0
403020349003046022100 91A2033692EB81503D53505FFC8DA326B1EE7DEA96F2917
4F0B3341A07812201022100FF7339288108B712F418530A18025A895408CC45E0BB6
78B46FBAB37DDB4D36B', h'30820243308201E9A00302010202082AEA0413A42DC1
CE300A06082A8648CE3D0403023072311C301A06035504030C13437573746F6D2D45
5220476C6F62616C204341310B3009060355040B0C024954311830160603550 40A0C
0F437573746F6D2D45522C20496E632E3111300F06035504070C0853616E204A6F73
65310B300906035504080C024341310B300906035504061302555330 1E170D323231
323036313133373539 5A170D333231323033313133373539 5A3072311C301A060355
04030C13437573746F6D2D455220476C6F62616C204341310B3009060355040B0C02
495431183016060355040A0C0F437573746F6D2D45522C20496E632E3111300F0603
5504070C0853616E204A6F7365310B300906035504080C024341310B300906035504
061302555330593 01306072A8648CE3D020106082A8648CE3D0301070342000497B1
ED969164930985BBB8AC9A2AF9455CDFEEA4B11DE2E79D068BFA803926B4005251B3
4F1C0815A4CBE03FBD1BBCB635F6431A22DE78653B87B99537ECE16CA3693067300F
0603551D130101FF040530030101FF30250603551D11041E301C811A68656C704063
7573746F6D2D65722E6578616D706C652E636F6D300E0603551D0F0101FF04040302
0186301D0603551D0E0416041492EA7640404A8FAB4F270BF3BC379D86CD7280F830
0A06082A8648CE3D040302034800304502210 0D6D813B390BD3A7B4E85424BCB1ED9
33AD1E981F2817B59083DD6EC1C5E3FADF02202CEE4406192BC767E98D7CFAE044C6
807481AD8564A7D569DCA3D1CDF1E5E843']}, h'A11909C5A601020278183230323
22D31322D30365432303A30343A31352E3735345A05581A041830168014CB8D98CA7
4C51B58DDE7ACEF869A9443A8D666A6074823BFBBC9C2BCF2130958C9D28443A1012
6A0587EA11909C5A40102074823BFBBC9C2BCF2130C585B3059301306072A8648CE3
D020106082A8648CE3D030107034200042030855BE846910B3ECB15CAF8572D3EB56
5BDC654A15EFAF6EDEBAA8B9E1608C5C910C93A20868FC8504D370F1B26FA9759F67
983BB78863EDA9ACBEA5124F60D6D4A414441313233343536373839584068987DE8B
007F4E9416610BBE2D48E1D7EA1032092B8BFCE611421950F45B22F17E214820C07E
777ADF86175E25D3205568404C25FCEEC1B817C7861A6104B3D0D6D4A41444131323
3343536373839', h'B1DD40B10787437588AEAC9036899191C16CCDBECA31C19785
5CCB6BBA142D709FE329CBC3F76297D6063ACB6759EAB98E96EA4C4AA2135AA48A24
7BAC1D6A3F'])

<CODE ENDS>

C.5. COSE-signed Voucher from MASA

The resulting voucher is created by the MASA and returned to the Registrar:

```
RES: 200 OK
Content-Type: application/voucher-cose+cbor
Body: <signed_voucher>
```

The Registrar then returns the voucher to the Pledge:

```
RES: 2.04 Changed
Content-Format: 836
Body: <signed_voucher>
```

It is signed by the MASA's private key (see Appendix C.1.3) and can be verified by the Pledge using the MASA's public key that it stores.

Below is the binary signed_voucher, encoded in hexadecimal (with lf added):

<CODE BEGINS>
D28443A10126A0590288A1190993A601020278183230323 22D31322D30365432
303A32333A33302E3730385A03F4074857EED786AD4049070859024730820243
308201E9A00302010202082AEA0413A42DC1CE300A06082A8648CE3D04030230
72311C301A06035504030C13437573746F6D2D455220476C6F62616C20434131
0B3009060355040B0C024954311830160603550 40A0C0F437573746F6D2D4552
2C20496E632E3111300F06035504070C0853616E204A6F7365310B3009060355
04080C024341310B3009060355040613025553301E170D323231323036313133
3735395A170D333231323033313133735395A3072311C301A06035504030C13
437573746F6D2D455220476C6F62616C204341310B3009060355040B0C024954
31183016060355040A0C0F437573746F6D2D45522C20496E632E3111300F0603
5504070C0853616E204A6F7365310B300906035504080C024341310B30090603
55040613025553305930130607 2A8648CE3D020106082A8648CE3D0301070342
000497B1ED969164930985BBB8AC9A2AF9455CDFEEA4B11DE2E79D068BFA8039
26B4005251B34F1C0815A4CBE03FBD1BBCB635F6431A22DE78653B87B99537EC
E16CA3693067300F0603551D130101FF040530030101FF30250603551D11041E
301C811A68656C7040637573746F6D2D65722E6578616D706C652E636F6D300E
0603551D0F0101FF040403020186301D0603551D0E0416041492EA7640404A8F
AB4F270BF3BC379D86CD7280F8300A06082A8648CE3D040302034800304502 21
00D6D813B390BD3A7B4E85424BCB1ED933AD1E981F2817B59083DD6EC1C5E3FA
DF02202CEE4406192BC767E98D7CFAE044C6807481AD8564A7D569DCA3D1CDF1
E5E8430B6D4A41444131323334353637383958 40DF31B21A6AD3F5AC7F4C8B02
6F551BD28FBCE62330D3E262AC170F6BFEDDBA5F2E8FBAA2CAACFED9E8614EAC
5BF2450DADC53AC29DFA30E8787A1400B2E7C832

<CODE ENDS>

The representiation of signed_voucher in CBOR diagnostic format (with lf added) is:

<CODE BEGINS>
18([h'A10126', {}, h'A1190993A601020278183230323 22D31322D30365432303
A32333A33302E3730385A03F4074857EED786AD4049070859024730820243308201E
9A00302010202082AEA0413A42DC1CE300A06082A8648CE3D0403023072311C301A0
6035504030C13437573746F6D2D455220476C6F62616C204341310B3009060355040
B0C024954311830160603550 40A0C0F437573746F6D2D45522C20496E632E3111300
F06035504070C0853616E204A6F7365310B300906035504080C024341310B3009060
3550406130255330593013060 72A8648CE3D0201060
82A8648CE3D0301070342000497B1ED969164930985BBB8AC9A2AF9455CDFEEA4B11
DE2E79D068BFA803926B4005251B34F1C0815A4CBE03FBD1BBCB635F6431A22DE786
53B87B99537ECE16CA3693067300F0603551D130101FF040530030101FF302506035
51D11041E301C811A68656C7040637573746F6D2D65722E6578616D706C652E636F6
D300E0603551D0F0101FF040403020186301D0603551D0E0416041492EA7640404A8
FAB4F270BF3BC379D86CD7280F8300A06082A8648CE3D040302034800304502210 0D
6D813B390BD3A7B4E85424BCB1ED933AD1E981F2817B59083DD6EC1C5E3FADF02202
CEE4406192BC767E98D7CFAE044C6807481AD8564A7D569DCA3D1CDF1E5E8430B6D4
A414441313233343536373839', h'DF31B21A6AD3F5AC7F4C8B026F551BD28FBCE6
2330D3E262AC170F6BFEDDBA5F2E8FBAA2CAACFED9E8614EAC5BF2450DADC53AC29D
FA30E8787A1400B2E7C832'])

<CODE ENDS>

In the above, the third element in the array is the plain CBOR voucher encoded as a CBOR byte string. When decoded, it can be represented by the following CBOR diagnostic notation:

<CODE BEGINS>
{2451: {1: 2, 2: "2022-12-06T20:23:30.708Z", 3: false, 7: h'57EED786
AD404907', 8: h'30820243308201E9A00302010202082AEA0413A42DC1CE300A06
082A8648CE3D0403023072311C301A06035504030C13437573746F6D2D455220476C
6F62616C204341310B3009060355040B0C024954311830160603550 40A0C0F437573
746F6D2D45522C20496E632E3111300F06035504070C0853616E204A6F7365310B30
0906035504080C024341310B3009060355040613025553301E170D32323132303631
31333735395A170D3332313230333131333735395A3072311C301A06035504030C13
437573746F6D2D455220476C6F62616C204341310B3009060355040B0C0249543118
3016060355040A0C0F437573746F6D2D45522C20496E632E3111300F06035504070C
0853616E204A6F7365310B300906035504080C024341310B3009060355040613025 5
533059301306072A8648CE3D020106082A8648CE3D030107034200 0497B1ED969164
930985BBB8AC9A2AF9455CDFEEA4B11DE2E79D068BFA803926B4005251B34F1C0815
A4CBE03FBD1BBCB635F6431A22DE78653B87B99537ECE16CA3693067300F0603551D
130101FF040530030101FF30250603551D11041E301C811A68656C7040637573746F
6D2D65722E6578616D706C652E636F6D300E0603551D0F0101FF040403020186301D
0603551D0E0416041492EA7640404A8FAB4F270BF3BC379D86CD7280F8300A06082A
8648CE3D0403020348003045022100D6D813B390BD3A7B4E85424BCB1ED933AD1E98
1F2817B59083DD6EC1C5E3FADF02202CEE4406192BC767E98D7CFAE044C6807481AD
8564A7D569DCA3D1CDF1E5E843', 11: "JADA123456789"}}

<CODE ENDS>

The largest element in the voucher is identified by key 8, which decodes to SID 2459 (pinned-domain-cert). It contains the complete PKIX (DER-encoded X.509v3) certificate of the Registrar's domain CA. This certificate is shown in Appendix C.2.3.

Appendix D. Generating Certificates with OpenSSL

This informative appendix shows example Bash shell scripts to generate test PKIX certificates for the Pledge IDevID, the Registrar and the MASA. The shell scripts cannot be run stand-alone because they depend on input files which are not all included in this appendix. Nevertheless, these scripts may provide guidance on how OpenSSL can be configured for generating Constrained BRSKI certificates.

The scripts were tested with OpenSSL 3.0.2. Older versions may not work -- OpenSSL 1.1.1 for example does not support all extensions used.

```
<CODE BEGINS>
#!/bin/bash
# File: create-cert-Pledge.sh
# Create new cert for: Pledge IDevID

# days certificate is valid - try to get close to the 802.1AR
# specified 9999-12-31 end date.
SECONDS1=`date +%s` # time now
SECONDS2=`date --date="9999-12-31 23:59:59Z" +%s` # target end time
let VALIDITY="(${SECONDS2}-${SECONDS1})/(24*3600)"
echo "Using validity param -days ${VALIDITY}"

NAME=pledge

# create csr for device
# conform to 802.1AR guidelines, using only CN + serialNumber when
# manufacturer is already present as CA.
# CN is not even mandatory, but just good practice.
openssl req -new -key keys/privkey_pledge.pem -out $NAME.csr -subj \
             "/CN=Stok IoT sensor Y-42/serialNumber=JADA123456789"

# sign csr - it uses faketime only to get endtime to 23:59:59Z
faketime '23:59:59Z' \
openssl x509 -set_serial 32429 -CAform PEM -CA output/masa_ca.pem \
  -CAkey keys/privkey_masa_ca.pem -extfile x509v3.ext -extensions \
  pledge_ext -req -in $NAME.csr -out output/$NAME.pem \
  -days $VALIDITY -sha256

# Note: alternative method using 'ca' command. Currently
# doesn't work without 'country' subject field.
# openssl ca -rand_serial -enddate 99991231235959Z -certform PEM \
#  -cert output/masa_ca.pem -keyfile keys/privkey_masa_ca.pem \
#  -extfile x509v3.ext -extensions pledge_ext -in $NAME.csr \
#  -out $NAME.pem -outdir output

# delete temp files
rm -f $NAME.csr

# convert to .der format
openssl x509 -in output/$NAME.pem -inform PEM -out output/$NAME.der \
             -outform DER

<CODE ENDS>
```

```
<CODE BEGINS>
# File: x509v3.ext
# This file contains all X509v3 extension definitions for OpenSSL
# certificate generation. Each certificate has its own _ext
# section below.

[ req ]
prompt = no

[ masa_ca_ext ]
subjectAltName=email:info@masa.stok.nl
keyUsage = critical,digitalSignature, keyCertSign, cRLSign
basicConstraints = critical,CA:TRUE,pathlen:3
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid

[ pledge_ext ]
keyUsage = critical,digitalSignature, nonRepudiation, \
           keyEncipherment, dataEncipherment
# basicConstraints for a non-CA cert MAY be marked either
# non-critical or critical.
basicConstraints = CA:FALSE
# Don't include subjectKeyIdentifier (SKI) - see 802.1AR-2018
subjectKeyIdentifier = none
authorityKeyIdentifier=keyid
# Include the MASA URI
1.3.6.1.5.5.7.1.32 = ASN1:IA5STRING:masa.stok.nl

[ domain_ca_ext ]
subjectAltName=email:help@custom-er.example.com
keyUsage = critical, keyCertSign, digitalSignature, cRLSign
basicConstraints=critical,CA:TRUE
# RFC 5280 4.2.1.1 : AKI MAY be omitted, and MUST be non-critical;
# SKI MUST be non-critical
subjectKeyIdentifier=hash

[ registrar_ext ]
keyUsage = critical, digitalSignature, nonRepudiation, \
           keyEncipherment, dataEncipherment
basicConstraints=CA:FALSE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
# Set Registrar 'RA' flag along with TLS client/server usage
#  see draft-ietf-anima-constrained-voucher#section-7.3
#  see tools.ietf.org/html/rfc6402#section-2.10
#  see www.openssl.org/docs/man1.1.1/man5/x509v3_config.html
extendedKeyUsage = critical,1.3.6.1.5.5.7.3.28, serverAuth, \
                   clientAuth
```

<CODE ENDS>

<CODE BEGINS>

```
#!/bin/bash
# File: create-cert-Registrar.sh
# Create new cert for: Registrar in a company domain

# days certificate is valid
VALIDITY=1095

# cert filename
NAME=registrar

# create csr
openssl req -new -key keys/privkey_registrar.pem -out $NAME.csr \
 -subj "/CN=Custom-ER Registrar/OU=Office dept/O=Custom-ER, Inc./\
L=Ottowa/ST=ON/C=CA"

# sign csr
openssl x509 -set_serial 0xC3F62149B2E30E3E -CAform PEM -CA \
 output/domain_ca.pem -extfile x509v3.ext -extensions registrar_ext \
 -req -in $NAME.csr -CAkey keys/privkey_domain_ca.pem \
 -out output/$NAME.pem -days $VALIDITY -sha256

# delete temp files
rm -f $NAME.csr

# convert to .der format
openssl x509 -in output/$NAME.pem -inform PEM -out output/$NAME.der \
             -outform DER
```

<CODE ENDS>

```
<CODE BEGINS>
#!/bin/bash
# File: create-cert-MASA.sh
# Create new cert for: MASA CA, self-signed CA certificate

# days certificate is valid
VALIDITY=3650

NAME=masa_ca

# create csr
openssl req -new -key keys/privkey_masa_ca.pem -out $NAME.csr \
            -subj "/CN=masa.stok.nl/O=vanderstok/L=Helmond/C=NL"

# sign csr
mkdir output >& /dev/null
openssl x509 -set_serial 0xE39CDA17E1386A0A  -extfile x509v3.ext \
 -extensions masa_ca_ext -req -in $NAME.csr \
 -signkey keys/privkey_masa_ca.pem -out output/$NAME.pem \
 -days $VALIDITY -sha256

# delete temp files
rm -f $NAME.csr

# convert to .der format
openssl x509 -in output/$NAME.pem -inform PEM -out output/$NAME.der \
             -outform DER

<CODE ENDS>
```

Appendix E. Pledge Device Class Profiles

This specification allows implementers to select between various functional options for the Pledge, yielding different code size footprints and different requirements on Pledge hardware. Thus for each product an optimal trade-off between functionality, development/ maintenance cost and hardware cost can be made.

This appendix illustrates different selection outcomes by means of defining different example "profiles" of constrained Pledges. In the following subsections, these profiles are defined and a comparison is provided.

E.1. Minimal Pledge

The Minimal Pledge profile (Min) aims to reduce code size and hardware cost to a minimum. This comes with some severe functional restrictions, in particular:

* No support for EST re-enrollment: whenever this would be needed, a factory reset followed by a new bootstrap process is required.

* No support for change of Registrar: for this case, a factory reset followed by a new bootstrap process is required.

This profile would be appropriate for single-use devices which must be replaced rather than re-deployed. That might include medical devices, but also sensors used during construction, such as concrete temperature sensors.

E.2. Typical Pledge

The Typical Pledge profile (Typ) aims to support a typical Constrained BRSKI feature set including EST re-enrollment support and Registrar changes.

E.3. Full-featured Pledge

The Full-featured Pledge profile (Full) illustrates a Pledge category that supports multiple bootstrap methods, hardware real-time clock, BRSKI/EST resource discovery, and CSR Attributes request/response. It also supports most of the optional features defined in this specification.

E.4. Comparison Chart of Pledge Classes

The below table specifies the functions implemented in the three example Pledge classes Min, Typ and Full.

Function |====================| Profiles ->	**Min**	**Typ**	**Full**
General	===	===	====
Support Constrained BRSKI bootstrap	Y	Y	Y
Support other bootstrap method(s)	-	-	Y
Real-time clock and cert time checks	-	-	Y
Constrained BRSKI	===	===	====
Discovery for rt=brski*	-	-	Y
Support pinned Registrar public key (RPK)	Y	-	Y
Support pinned Registrar certificate	-	Y	Y
Support pinned Domain CA	-	Y	Y
Constrained EST	===	===	====
Discovery for rt=ace.est*	-	-	Y
GET /att and response parsing	-	-	Y

Function |==========| Profiles ->	Min	Typ	Full
GET /crts format 281 (multiple CA certs)	-	-	Y
GET /crts only format 287 (one CA cert only)	Y	Y	-
ETag handling support for GET /crts	-	Y	Y
Re-enrollment supported	- (1)	Y	Y
6.6.1 optimized procedure	Y	Y	-
Pro-active cert re-enrollment at own initiative	N/A	-	Y
Periodic trust anchor retrieval GET /crts	- (1)	Y	Y
Supports change of Registrar identity	- (1)	Y	Y

Table 5

Notes: (1) is possible only by doing a factory-reset followed by a new bootstrap procedure.

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy

Email: stokcons@bbhmail.nl

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Esko Dijk
IoTconsultancy.nl

Email: esko.dijk@iotconsultancy.nl