

Workgroup: anima Working Group

Internet-Draft:

draft-ietf-anima-constrained-join-proxy-06

Published: 3 December 2021

Intended Status: Standards Track

Expires: 6 June 2022

Authors: M. Richardson

P. van der Stok

Sandelman Software Works      vanderstok consultancy

P. Kampanakis

Cisco Systems

## **Constrained Join Proxy for Bootstrapping Protocols**

### **Abstract**

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

This document extends the work of Bootstrapping Remote Secure Key Infrastructures (BRSKI) by replacing the Circuit-proxy between Pledge and Registrar by a stateless/stateful constrained Join Proxy. It relays join traffic from the Pledge to the Registrar.

### **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 6 June 2022.

### **Copyright Notice**

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

- [1. Introduction](#)
- [2. Terminology](#)
- [3. Requirements Language](#)
- [4. Join Proxy functionality](#)
- [5. Join Proxy specification](#)
  - [5.1. Stateful Join Proxy](#)
  - [5.2. Stateless Join Proxy](#)
  - [5.3. Stateless Message structure](#)
- [6. Comparison of stateless and stateful modes](#)
- [7. Discovery](#)
  - [7.1. Join Proxy discovers Registrar](#)
    - [7.1.1. CoAP discovery](#)
    - [7.1.2. GRASP discovery](#)
    - [7.1.3. 6tisch discovery](#)
  - [7.2. Pledge discovers Registrar](#)
    - [7.2.1. CoAP discovery](#)
    - [7.2.2. GRASP discovery](#)
    - [7.2.3. 6tisch discovery](#)
  - [7.3. Pledge discovers Join Proxy](#)
    - [7.3.1. CoAP discovery](#)
    - [7.3.2. GRASP discovery](#)
    - [7.3.3. 6tisch discovery](#)
- [8. Security Considerations](#)
- [9. IANA Considerations](#)
  - [9.1. Resource Type Attributes registry](#)
  - [9.2. service name and port number registry](#)
- [10. Acknowledgements](#)
- [11. Contributors](#)
- [12. Changelog](#)
  - [12.1. 05 to 06](#)
  - [12.2. 04 to 05](#)
  - [12.3. 03 to 04](#)
  - [12.4. 02 to 03](#)
  - [12.5. 01 to 02](#)
  - [12.6. 00 to 01](#)
  - [12.7. 00 to 00](#)
- [13. References](#)
  - [13.1. Normative References](#)
  - [13.2. Informative References](#)

## 1. Introduction

The Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol described in [\[RFC8995\]](#) provides a solution for a secure zero-touch (automated) bootstrap of new (unconfigured) devices. In the context of BRSKI, new devices, called "Pledges", are equipped with a factory-installed Initial Device Identifier (IDevID) (see [\[ieee802-1AR\]](#)), and are enrolled into a network. BRSKI makes use of Enrollment over Secure Transport (EST) [\[RFC7030\]](#) with [\[RFC8366\]](#) vouchers to securely enroll devices. A Registrar provides the security anchor of the network to which a Pledge enrolls. In this document, BRSKI is extended such that a Pledge connects to "Registrars" via a Join Proxy.

A complete specification of the terminology is pointed at in [Section 2](#).

The specified solutions in [\[RFC8995\]](#) and [\[RFC7030\]](#) are based on POST or GET requests to the EST resources (`/cacerts`, `/simpleenroll`, `/simplereenroll`, `/serverkeygen`, and `/csrattrs`), and the brski resources (`/requestvoucher`, `/voucher_status`, and `/enrollstatus`). These requests use https and may be too large in terms of code space or bandwidth required for constrained devices. Constrained devices which may be part of constrained networks [\[RFC7228\]](#), typically implement the IPv6 over Low-Power Wireless personal Area Networks (6LoWPAN) [\[RFC4944\]](#) and Constrained Application Protocol (CoAP) [\[RFC7252\]](#).

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

DTLS is a client-server protocol relying on the underlying IP layer to perform the routing between the DTLS Client and the DTLS Server. However, the Pledge will not be IP routable until it is authenticated to the network. A new Pledge can only initially use a link-local IPv6 address to communicate with a neighbour on the same link [\[RFC6775\]](#) until it receives the necessary network configuration parameters. However, before the Pledge can receive these configuration parameters, it needs to authenticate itself to the network to which it connects.

During enrollment, a DTLS connection is required between Pledge and Registrar.

Once a Pledge is enrolled, it can act as Join Proxy between other Pledges and the enrolling Registrar.

This document specifies a new form of Join Proxy and protocol to act as intermediary between Pledge and Registrar to relay DTLS messages between Pledge and Registrar. Two versions of the Join Proxy are specified:

- 1 A stateful Join Proxy that locally stores IP addresses during the connection.
- 2 A stateless Join Proxy that where the connection state is stored in the messages.

This document is very much inspired by text published earlier in [[I-D.kumar-dice-dtls-relay](#)]. [[I-D.richardson-anima-state-for-joinrouter](#)] outlined the various options for building a Join Proxy. [[RFC8995](#)] adopted only the Circuit Proxy method (1), leaving the other methods as future work. This document standardizes the CoAP/DTLS (method 4).

## 2. Terminology

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

The term "installation network" refers to all devices in the installation and the network connections between them. The term "installation IP\_address" refers to the set of addresses which are routable over the whole installation network.

## 3. Requirements Language

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

## 4. Join Proxy functionality

As depicted in the [Figure 1](#), the Pledge (P), in an LLN mesh can be more than one hop away from the Registrar (R) and not yet authenticated into the network.

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

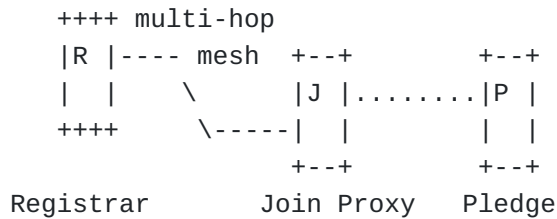


Figure 1: multi-hop enrollment.

Without routing the Pledge (P) cannot establish a secure connection to the Registrar (R) over multiple hops in the network.

Furthermore, the Pledge (P) cannot discover the IP address of the Registrar (R) over multiple hops to initiate a DTLS connection and perform authentication.

To overcome the problems with non-routability of DTLS packets and/or discovery of the destination address of the Registrar, the Join Proxy is introduced. This Join Proxy functionality is configured into all authenticated devices in the network which may act as a Join Proxy for Pledges. The Join Proxy allows for routing of the packets from the Pledge using IP routing to the intended Registrar. An authenticated Join Proxy can discover the routable IP address of the Registrar over multiple hops. The following [Section 5](#) specifies the two Join Proxy modes. A comparison is presented in [Section 6](#).

## 5. Join Proxy specification

A Join Proxy can operate in two modes:

- \*Stateful mode

- \*Stateless mode

A Join Proxy MUST implement one of the two modes. A Join Proxy MAY implement both, with an unspecified mechanism to switch between the two modes.

## 5.1. Stateful Join Proxy

In stateful mode, the Join Proxy forwards the DTLS messages to the Registrar.

Assume that the Pledge does not know the IP address of the Registrar it needs to contact. The Join Proxy has been enrolled via the Registrar and learns the IP address and port of the Registrar, for example by using the discovery mechanism described in [Section 7](#). The Pledge first discovers (see [Section 7](#)) and selects the most appropriate Join Proxy. (Discovery can also be based upon [\[RFC8995\]](#) section 4.1). For service discovery via DNS-SD [\[RFC6763\]](#), this document specifies the service names in [Section 9.2](#). The Pledge initiates its request as if the Join Proxy is the intended Registrar. The Join Proxy receives the message at a discoverable join-port. The Join Proxy constructs an IP packet by copying the DTLS payload from the message received from the Pledge, and provides source and destination addresses to forward the message to the intended Registrar. The Join Proxy maintains a 4-tuple array to translate the DTLS messages received from the Registrar and forward it back to the Pledge.

In [Figure 2](#) the various steps of the message flow are shown, with 5684 being the standard coaps port:

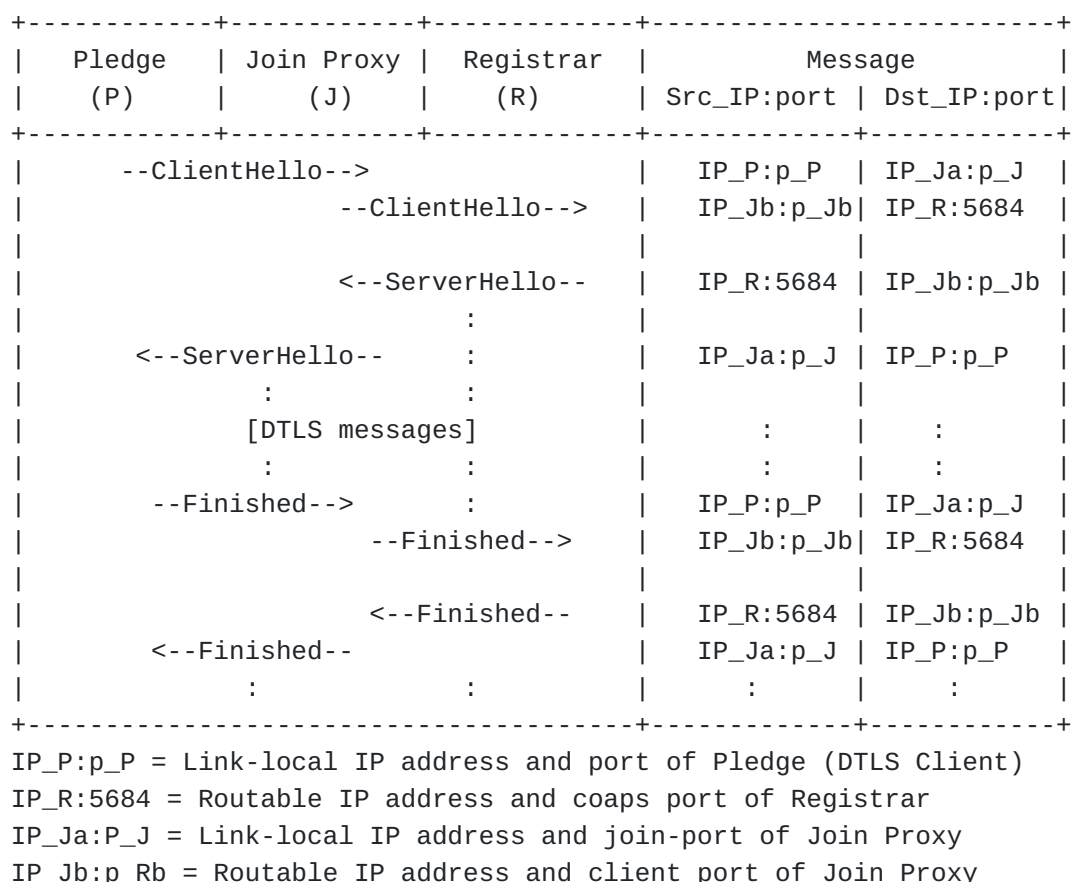


Figure 2: constrained stateful joining message flow with Registrar address known to Join Proxy.

## 5.2. Stateless Join Proxy

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

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

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

The Join Proxy transforms the DTLS message to a JPY message which includes the DTLS data as payload, and sends the JPY message to the join-port of the Registrar.

The JPY message payload consists of two parts:

- \*Header (H) field: consisting of the source link-local address and port of the Pledge (P), and

- \*Contents (C) field: containing the original DTLS payload.

On receiving the JPY message, the Registrar (or proxy) retrieves the two parts.

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

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

In this scenario, both the Registrar and the Join Proxy use discoverable join-ports, for the Join Proxy this may be a default CoAP port.

The [Figure 3](#) depicts the message flow diagram:



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

IP\_P:p\_P = Link-local IP address and port of the Pledge

IP\_R:p\_Ra = Routable IP address and join-port of Registrar

IP\_Ja:p\_Ja = Link-local IP address and join-port of Join Proxy

IP\_Jb:p\_Jb = Routable IP address and port of Join Proxy

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

Figure 3: constrained stateless joining message flow.

### 5.3. Stateless Message structure

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

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

1. header field: containing a CBOR array [[RFC8949](#)] with the Pledge IPv6 Link Local address as a CBOR byte string, the Pledge's UDP port number as a CBOR integer, the IP address family (IPv4/IPv6) as a CBOR integer, and the proxy's ifindex or other identifier for the physical port as CBOR integer. The header field is not DTLS encrypted.
2. Content field: containing the DTLS payload as a CBOR byte string.

The address family integer is defined in [[family](#)] with:

- 1 IP (IP version 4)
- 2 IP6 (IP version 6)

The Join Proxy cannot decrypt the DTLS payload and has no knowledge of the transported media type.

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

Figure 4: CDDL representation of JPY message

The contents are DTLS encrypted. In CBOR diagnostic notation the payload JPY[H(IP\_P:p\_P)], will look like:

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

On reception by the Registrar, the Registrar MUST verify that the number of array elements is larger than or equal to 5, and reject the message when the number of array elements is smaller than 5. The Registrar replaces the 5th "content" element with the DTLS payload of the response message and leaves all other array elements unchanged.

Examples are shown in [Appendix A](#).

When additions are added to the array in later versions of this protocol, any additional array elements (i.e., not specified by current document) MUST be ignored by a receiver if it doesn't know these elements. This approach allows evolution of the protocol while maintaining backwards-compatibility. A version number isn't needed; that number is defined by the length of the array.

## 6. Comparison of stateless and stateful modes

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

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

Figure 5: Comparison between stateful and stateless mode

## 7. Discovery

It is assumed that Join Proxy seamlessly provides a coaps connection between Pledge and Registrar. In particular this section extends section 4.1 of [[RFC8995](#)] for the constrained case.

The discovery follows two steps with two alternatives for step 1:

\*Step 1. Two alternatives exist (near and remote):

-Near: the Pledge is one hop away from the Registrar. The Pledge discovers the link-local address of the Registrar as described in [[I-D.ietf-ace-coap-est](#)]. From then on, it follows the BRSKI process as described in [[I-D.ietf-ace-coap-est](#)] and [[I-D.ietf-anima-constrained-voucher](#)], using link-local addresses.

-Remote: the Pledge is more than one hop away from a relevant Registrar, and discovers the link-local address and join-port

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

\*Step 2. The enrolled Join Proxy discovers the join-port of the Registrar.

The order in which the two alternatives of step 1 are tried is installation dependent. The trigger for discovery in Step 2 in implementation dependent.

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

Three discovery cases are discussed: Join Proxy discovers Registrar, Pledge discovers Registrar, and Pledge discovers Join Proxy. Each discovery case considers three alternatives: CoAP discovery, GRASP discovery, and 6tisch discovery.

## **7.1. Join Proxy discovers Registrar**

In this section, the Join Proxy and Registrar are assumed to communicate via Link-Local addresses. This section describes the discovery of the Registrar by the Join Proxy.

### **7.1.1. CoAP discovery**

The discovery of the coaps Registrar, using coap discovery, by the Join Proxy follows sections 6.3 and 6.5.1 of [[I-D.ietf-anima-constrained-voucher](#)]. The stateless Join Proxy can discover the join-port of the Registrar by sending a GET request to "/.well-known/core" including a resource type (rt) parameter with the value "brski.rjp" [[RFC6690](#)]. Upon success, the return payload will contain the join-port of the Registrar.

REQ: GET coap://[IP\_address]/.well-known/core?rt=brski.rjp

RES: 2.05 Content

<coaps://[IP\_address]:join-port>; rt="brski.rjp"

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

### **7.1.2. 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 Registrar announces itself using 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](#)] .

### **7.1.3. 6tisch discovery**

The discovery of the Registrar by the Join Proxy uses the enhanced beacons as discussed in [[I-D.ietf-6tisch-enrollment-enhanced-beacon](#)] .

## **7.2. Pledge discovers Registrar**

In this section, the Pledge and Registrar are assumed to communicate via Link-Local addresses. This section describes the discovery of the Registrar by the Pledge.

### **7.2.1. CoAP discovery**

The discovery of the coaps Registrar, using coap discovery, by the Pledge follows sections 6.3 and 6.5.1 of [[I-D.ietf-anima-constrained-voucher](#)]..

### **7.2.2. GRASP discovery**

This section is normative for uses with an ANIMA ACP. In the context of autonomic networks, the Pledge uses the DULL GRASP M\_FLOOD mechanism to announce itself. Section 4.1.1 of [[RFC8995](#)] discusses this in more detail. The Registrar announces itself using 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](#)] .

### **7.2.3. 6tisch discovery**

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

## **7.3. Pledge discovers Join Proxy**

In this section, the Pledge and Join Proxy are assumed to communicate via Link-Local addresses. This section describes the discovery of the Join Proxy by the Pledge.

### 7.3.1. CoAP discovery

In the context of a coap network without Autonomic Network support, discovery follows the standard coap policy. The Pledge can discover a Join Proxy by sending a link-local multicast message to ALL CoAP Nodes with address FF02::FD. Multiple or no nodes may respond. The handling of multiple responses and the absence of responses follow section 4 of [[RFC8995](#)].

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

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

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

```
RES: 2.05 Content  
<coaps://[IP_address]:join-port>; rt="brski.jp"
```

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

### 7.3.2. GRASP discovery

This section is normative for uses with an ANIMA ACP. The Pledge MUST listen for GRASP M\_FLOOD [[RFC8990](#)] announcements of the objective: `"AN_Proxy"`. See section 4.1.1 [[RFC8995](#)] for the details of the objective.

### 7.3.3. 6tisch discovery

The discovery of the Join Proxy by the Pledge uses the enhanced beacons as discussed in [[I-D.ietf-6tisch-enrollment-enhanced-beacon](#)].

## 8. Security Considerations

All of the concerns in [[RFC8995](#)] section 4.1 apply. The Pledge can be deceived by malicious Join Proxy announcements. The Pledge will only join a network to which it receives a valid [[RFC8366](#)] voucher [[I-D.ietf-anima-constrained-voucher](#)]. Once the Pledge joined, the payload between Pledge and Registrar is protected by DTLS.

It should be noted here that the contents of the CBOR map used to convey return address information is not DTLS protected. When the communication between JOIN Proxy and Registrar passes over an unsecure network, an attacker can change the CBOR array, causing the Registrar to deviate traffic from the intended Pledge. If such scenario needs to be avoided, then it is reasonable for the Join Proxy to encrypt the CBOR array using a locally generated symmetric key. The Registrar would not be able to examine the result, but it does not need to do so. This is a topic for future work.

Another possibility is to use level 2 protection between Registrar and Join Proxy.

## **9. IANA Considerations**

### **9.1. Resource Type Attributes registry**

This specification registers two new Resource Type (rt=) Link Target Attributes in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry per the [[RFC6690](#)] procedure.

Attribute Value: brski.jp

Description: This BRSKI resource type is used to query and return the supported BRSKI (CoAP over DTLS) port of the constrained Join Proxy.

Reference: [this document]

Attribute Value: brski.rjp

Description: This BRSKI resource type is used to query and return the supported BRSKI JPY protocol port of the Registrar.

Reference: [this document]

### **9.2. service name and port number registry**

This specification registers two service names under the "Service Name and Transport Protocol Port Number" registry.

Service Name: brski-jp  
Transport Protocol(s): udp  
Assignee: IESG <iesg@ietf.org>  
Contact: IESG <iesg@ietf.org>  
Description: Bootstrapping Remote Secure Key Infrastructure  
                  constrained Join Proxy  
Reference: [this document]

Service Name: brski-rjp  
Transport Protocol(s): udp  
Assignee: IESG <iesg@ietf.org>  
Contact: IESG <iesg@ietf.org>  
Description: Bootstrapping Remote Secure Key Infrastructure  
                  Registrar join-port used by stateless constrained  
                  Join Proxy  
Reference: [this document]

## **10. Acknowledgements**

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

## **11. Contributors**

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

## **12. Changelog**

### **12.1. 05 to 06**

- \* RT value change to brski.jp and brski.rjp
- \* new registry values for IANA
- \* improved handling of jpy header array

### **12.2. 04 to 05**

- \* Join Proxy and join-port consistent spelling
- \* some nits removed
- \* restructured discovery
- \* section
- \* rephrased parts of security section

### **12.3. 03 to 04**

- \* mail address and reference



#### 12.4. 02 to 03

- \* Terminology updated
- \* Several clarifications on discovery and routability
- \* DTLS payload introduced

#### 12.5. 01 to 02

- \*Discovery of Join Proxy and Registrar ports

#### 12.6. 00 to 01

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

#### 12.7. 00 to 00

- \*copied from vanderstok-anima-constrained-join-proxy-05

### 13. References

#### 13.1. Normative References

- [family] "Address Family Numbers", 19 October 2021, <<https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>>.
- [I-D.ietf-ace-coap-est] Stok, P. V. D., Kampanakis, P., Richardson, M. C., and S. Raza, "EST over secure CoAP (EST-coaps)", Work in Progress, Internet-Draft, draft-ietf-ace-coap-est-18, 6 January 2020, <<https://www.ietf.org/archive/id/draft-ietf-ace-coap-est-18.txt>>.
- [I-D.ietf-anima-constrained-voucher] Richardson, M., Stok, P. V. D., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-14, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-anima-constrained-voucher-14.txt>>.

**[ieee802-1AR]**

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

**[RFC2119]**

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

**[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>>.

**[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>>.

**[RFC8949]**

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

**[RFC8990]**

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

**[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>>.

## **13.2. Informative References**

**[I-D.ietf-6tisch-enrollment-enhanced-beacon]** (editor), D. D. and M.

Richardson, "Encapsulation of 6TiSCH Join and Enrollment Information Elements", Work in Progress, Internet-Draft, draft-ietf-6tisch-enrollment-enhanced-beacon-14, 21 February 2020, <<https://www.ietf.org/archive/id/draft-ietf-6tisch-enrollment-enhanced-beacon-14.txt>>.

**[I-D.kumar-dice-dtls-relay]** Kumar, S. S., Keoh, S. L., and O.

Garcia-Morchon, "DTLS Relay for Constrained Environments", Work in Progress, Internet-Draft, draft-

kumar-dice-dtls-relay-02, 20 October 2014, <<https://www.ietf.org/archive/id/draft-kumar-dice-dtls-relay-02.txt>>.

**[I-D.richardson-anima-state-for-joinrouter]**

Richardson, M. C., "Considerations for stateful vs stateless join router in ANIMA bootstrap", Work in Progress, Internet-Draft, draft-richardson-anima-state-for-joinrouter-03, 22 September 2020, <<https://www.ietf.org/archive/id/draft-richardson-anima-state-for-joinrouter-03.txt>>.

**[RFC4944]** Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.

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

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

**[RFC6775]** Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.

**[RFC7030]** Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

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

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

## **Appendix A. Stateless Proxy payload examples**

The examples show the request "GET coaps://192.168.1.200:5965/est/crts" to a Registrar. The header generated between Join Proxy and

Registrar and from Registrar to Join Proxy are shown in detail. The DTLS payload is not shown.

The request from Join Proxy to Registrar looks like:

```
85                                     # array(5)
  50                                 # bytes(16)
    FE800000000000000000000000000000FFFFC0A801C8 #
  19 BDA7                             # unsigned(48551)
  01                                 # unsigned(1) IP
  00                                 # unsigned(0)
  58 2D                             # bytes(45)
<cacrts DTLS encrypted request>
```

In CBOR Diagnostic:

```
[h'FE800000000000000000000000000000FFFFC0A801C8', 48551, 1, 0,
 h'<cacrts DTLS encrypted request>']
```

The response is:

```
85                                     # array(5)
  50                                 # bytes(16)
    FE800000000000000000000000000000FFFFC0A801C8 #
  19 BDA7                             # unsigned(48551)
  01                                 # unsigned(1) IP
  00                                 # unsigned(0)
  59 026A                             # bytes(618)
    <cacrts DTLS encrypted response>
```

In CBOR diagnostic:

```
[h'FE800000000000000000000000000000FFFFC0A801C8', 48551, 1, 0,
 h'<cacrts DTLS encrypted response>']
```

## Authors' Addresses

Michael Richardson  
Sandelman Software Works

Email: [mcr+ietf@sandelman.ca](mailto:mcr+ietf@sandelman.ca)

Peter van der Stok  
vanderstok consultancy

Email: [stokcons@bbhmail.nl](mailto:stokcons@bbhmail.nl)

Panos Kampanakis  
Cisco Systems

Email: [pkampana@cisco.com](mailto:pkampana@cisco.com)