Workgroup: anima Working Group
Internet-Draft:
draft-ietf-anima-constrained-join-proxy-15
Published: 6 November 2023
Intended Status: Standards Track
Expires: 9 May 2024
Authors: M. Richardson, Sandelman Software Works
P. van der Stok, vanderstok consultancy
P. Kampanakis, Cisco Systems

Join Proxy for Bootstrapping of Constrained Network Elements

Abstract

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. The constrained Join Proxy is a mesh neighbor of the Pledge and can relay a DTLS session originating from a Pledge with only link-local addresses to a Registrar which is not a mesh neighbor of the Pledge.

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 constrained Join Proxy.

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-join-proxy/.

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/.

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

Status of This Memo

Copyright Notice

Table of Contents

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 constrained Join Proxy. In particular, the underlying IP network is assumed to be a mesh network as described in [RFC4944], although other IP-over-foo networks are not excluded. An example network is shown in Figure 1.

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 challenged 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) [RFC9147] 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 [RFC9148].

The [I-D.ietf-anima-constrained-voucher] extends [RFC9148] 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 over the mesh network until it is authenticated to the mesh network. A new Pledge can only initially use a link-local IPv6 address to communicate with a mesh neighbor [RFC6775] until it receives the necessary network configuration parameters. The Pledge receives these configuration parameters from the Registrar. When the Registrar is not a direct neighbor of the Registrar but several hops away, the Pledge discovers a neighbor constrained Join Proxy, which transmits the DTLS protected request coming from the Pledge to the Registrar. The constrained Join Proxy must have been enrolled previously into the network, such that the message from the constrained Join Proxy to the Registrar can be routed over one or more hops.

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

An enrolled Pledge can act as constrained Join Proxy between other Pledges and the enrolling Registrar.

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

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

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 constrained Join Proxy. [RFC8995] adopted only the Circuit Proxy method (1), leaving the other methods as future work.

Similar to the difference between storing and non-storing Modes of Operations (MOP) in RPL [RFC6550], the stateful and stateless modes differ in the way that they store the state required to forward the return packet to the pledge. In the stateful method, the return forward state is stored in the Join Proxy. In the stateless method, the return forward state is stored in the network.

2. Terminology

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

The following terms are defined in [RFC8366], and are used identically as in that document: artifact, imprint, domain, Join Registrar/Coordinator, Pledge, and Voucher.

The term "Registrar" is used throughout this document instead of "Join Registrar/Coordinator (JRC)" as defined in [RFC8366].

The term "installation network" refers to all devices in the installation and the network connections between them. The term "installation IP_address" refers to an address out of the set of addresses which are routable over the whole installation network.

The "Constrained Join Proxy" enables a pledge that is multiple hops away from the Registrar, to securely execute the BRSKI protocol [RFC8995] over a secure channel.

The term "Join Proxy" is used interchangeably with the term "constrained Join Proxy" throughout this document.

The [RFC8995] Circuit Proxy is referred to as a TCP circuit Join Proxy.

3. Constrained Join Proxy functionality

As depicted in the Figure 1, the Pledge (P), in a Low-Power and Lossy Network (LLN) mesh [RFC7102] 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 neighbor, the constrained Join Proxy (J) using their link-local IPv6 addresses. However, the Pledge needs to communicate with end-to-end security with a Registrar to authenticate and get the relevant system/network parameters. If the Pledge, knowing the IP address of the Registrar, initiates a DTLS connection to the Registrar, then the packets are dropped at the constrained Join Proxy since the Pledge is not yet admitted to the network or there is no IP routability to Pledge for any returned messages from the Registrar.

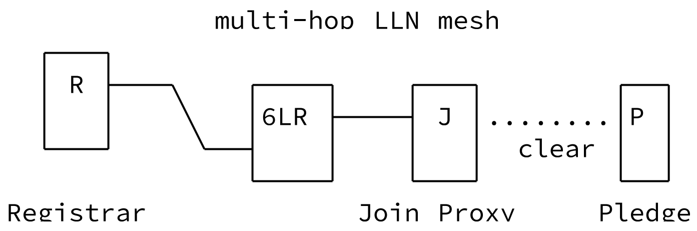

Figure 1: multi-hop enrollment.

Without multi-hop routing, the Pledge cannot establish a secure connection to the Registrar over multiple hops in the network.

Furthermore, the Pledge cannot discover the IP address of the Registrar 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 constrained Join Proxy is introduced. This constrained Join Proxy functionality is configured into all authenticated devices in the network which may act as a constrained Join Proxy for Pledges. The constrained Join Proxy allows for routing of the packets from the Pledge using IP routing to the intended Registrar. An authenticated constrained Join Proxy can discover the routable IP address of the Registrar over multiple hops. The following Section 4 specifies the two constrained Join Proxy modes. A comparison is presented in Section 6.

When a mesh network is set up, it consists of a Registrar and a set of connected pledges. No constrained Join Proxies are present. The wanted end-state is a network with a Registrar and a set of enrolled devices. Some of these enrolled devices can act as constrained Join Proxies. Pledges can only employ link-local communication until they are enrolled. A Pledge will regularly try to discover a constrained Join Proxy or a Registrar with link-local discovery requests. The Pledges which are neighbors of the Registrar will discover the Registrar and be enrolled following the BRSKI protocol. An enrolled device can act as constrained Join Proxy. The Pledges which are not a neighbor of the Registrar will eventually discover a constrained Join Proxy and follow the BRSKI protocol to be enrolled. While this goes on, more and more constrained Join Proxies with a larger hop distance to the Registrar will emerge. The network should be configured such that at the end of the enrollment process, all pledges have discovered a neighboring constrained Join Proxy or the Registrar, and all Pledges are enrolled.

4. Constrained Join Proxy specification

A Join Proxy can operate in two modes:

*Stateful mode

*Stateless mode

A Join Proxy MUST implement both. A Registrar MUST implement the stateful mode and SHOULD implement the Stateless mode.

A mechanism to switch between modes is out of scope of this document. It is recommended that a Join Proxy uses only one of these modes at any given moment during an installation lifetime.

The advantages and disadvantages of the two modes are presented in Section 6.

4.1. 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, by using a discovery mechanism such as described in Section 5. The Pledge first discovers (see Section 5) and selects the most appropriate Join Proxy. (Discovery can also be based upon [RFC8995] section 4.1). 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 stores the 4-tuple array of the messages received from the Registrar and copies it back to the header of the message returned to the Pledge.

In Figure 2 the various steps of the message flow are shown, with 5684 being the standard coaps port. The columns "SRc_IP:port" and "Dst_IP:port" show the IP address and port values for the source and destination of the message.

```
+------------+------------+-------------+------------------------+
|   Pledge   | Join Proxy |  Registrar  |          Message       |
|    (P)     |     (J)    |    (R)      | Src_IP:port | Dst_IP:port|
+------------+------------+-------------+-------------+----------+
|      --ClientHello-->                 |   IP_P:p_P  | IP_Jl:p_Jl |
|                    --ClientHello-->   |   IP_Jr:p_Jr| IP_R:5684  |
|                                       |             |            |
|                    <--ServerHello--   |   IP_R:5684 | IP_Jr:p_Jr |
|                            :          |             |            |
|       <--ServerHello--     :          |   IP_Jl:p_Jl| IP_P:p_P   |
|              :             :          |             |            |
|             [DTLS messages]           |       :     |    :       |
|              :             :          |       :     |    :       |
|        --Finished-->       :          |   IP_P:p_P  | IP_Jl:p_Jl |
|                      --Finished-->    |   IP_Jr:p_Jr| IP_R:5684  |
|                                       |             |            |
|                      <--Finished--    |   IP_R:5684 | IP_Jr:p_Jr |
|        <--Finished--                  |   IP_Jl:p_Jl| IP_P:p_P   |
|             :              :          |      :      |     :      |
+---------------------------------------+-------------+------------+
IP_P:p_P = Link-local IP address and port of Pledge (DTLS Client)
IP_R:5684 = Routable IP address and coaps port of Registrar
IP_Jl:p_Jl = Link-local IP address and join-port of Join Proxy
IP_Jr:p_Jr = Routable IP address and client port of Join Proxy
```

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

4.2. Stateless Join Proxy

The JPY Encapsulation Protocol allows the stateless Join Proxy to minimize memory requirements on a constrained Join Proxy device. The use of a stateless operation requires no memory in the Join Proxy device because it stores the state in a special encapsulation in the packet. This 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 neighboring (Join Proxy) node. Under normal circumstances, this message would be dropped at the neighbor node since the Pledge is not yet IP routable or is not yet authenticated to send messages through the network. However, if the neighbor 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    | Join Proxy |    Registrar  |        Message       |
|     (P)      |     (J)    |      (R)      |Src_IP:port|Dst_IP:port|
+--------------+-----------+--------------+-----------+-----------+
|      --ClientHello-->                    | IP_P:p_P  |IP_Jl:p_Jl |
|                    --JPY[H(IP_P:p_P),--> | IP_Jr:p_Jr|IP_R:p_Ra  |
|                          C(ClientHello)] |           |           |
|                    <--JPY[H(IP_P:p_P),-- | IP_R:p_Ra |IP_Jr:p_Jr |
|                         C(ServerHello)]  |           |           |
|      <--ServerHello--                    | IP_Jl:p_Jl|IP_P:p_P   |
|              :                           |           |           |
|          [ DTLS messages ]               |     :     |    :      |
|              :                           |     :     |    :      |
|      --Finished-->                       | IP_P:p_P  |IP_Jr:p_Jr |
|                    --JPY[H(IP_P:p_P),--> | IP_Jl:p_Jl|IP_R:p_Ra  |
|                          C(Finished)]    |           |           |
|                    <--JPY[H(IP_P:p_P),-- | IP_R:p_Ra |IP_Jr:p_Jr |
|                         C(Finished)]     |           |           |
|      <--Finished--                       | IP_Jl:p_Jl|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_Jl:p_Jl = Link-local IP address and join-port of Join Proxy
IP_Jr:p_Jr = 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.

4.3. Stateless Message structure

The JPY message is constructed as a payload directly above UDP. There is no CoAP or DTLS layer as both are within the relayed payload.

Header and Contents fields together are consist of one CBOR [RFC8949] array of 2 elements, explained in CDDL [RFC8610]:

1. The context payload. This is a CBOR byte string. It SHOULD be between 8 and 32 bytes in size.

2. Content field: containing the DTLS payload as a CBOR byte string.

```
 JPY_message =
 [
    pledge_context_message : bstr,
    content   : bstr
 ]
```

Figure 4: CDDL representation of JPY message

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

The context payload is to be reflected by the Registrar when sending reply packets to the Join Proxy. The context payload is not standardized. It is to be used by the Join Proxy to record which pledge the traffic came from.

The Join Proxy SHOULD encrypt this context with a symmetric key known only to the Join Proxy. This key need not persist on a long term basis, and MAY be changed periodically. The considerations of Section 5.2 of [RFC8974] apply.

This is intended to be identical to the mechanism described in Section 7.1 of [RFC9031]. However, since the CoAP layer is inside of the DTLS layer (which is between the Pledge and the Registrar), it is not possible for the Join Proxy to act as a CoAP proxy.

For the JPY messages relayed to the Registrar, the Join Proxy SHOULD use the same UDP source port for the JPY messages related to all pledges. A Join Proxy MAY change the UDP source port, but doing so creates more local state. A Join Proxy with multiple CPUs (unlikely in a constrained system, but possible in the future) could, for instance, use different source port numbers to demultiplex connections across CPUs.

4.3.1. Stateless Message structure example construction

A typical context parameter might be constructed with the following CDDL grammar: (This is illustrative only: the contents are not subject to standardization)

```
 pledge_context_message = [
   family:  uint .bits 1,
   ifindex: uint .bits 8,
   srcport: uint .bits 16,
   iid:     bstr .bits 64,
 ]
```

This results in a total of 96 bits, or 12 bytes. The structure stores the srcport, the originating IPv6 Link-Local address, the

IPv4/IPv6 family (as a single bit) and an ifindex to provide the link-local scope. This fits nicely into a single AES128 CBC block for instance, resulting in a 16 byte context message.

The Join Proxy MUST maintain the same context block for all communications from the same pledge. This implies that any encryption key either does not change during the communication, or that when it does, it is acceptable to break any onboarding connections which have not yet completed.

If using a context parameter like defined above, it should be easy for the Join Proxy to meet this requirement without maintaining any local state about the pledge.

Note: when IPv6 is used only the lower 64-bits of the origin IP need to be recorded, because they are all IPv6 Link-Local addresses, so the upper 64-bits are just "fe80::". For IPv4, a Link-Local IPv4 address [RFC3927] would be used, and it would fit into 64-bits. On media where the Interface IDentifier (IID) is not 64-bits, a different arrangement will be necessary.

4.3.2. Processing by Registrar

On reception of a JPY message by the Registrar, the Registrar MUST verify that the number of array elements is 2 or more. The pledge_content field must be provided as input to a DTLS library [RFC9147], which along with the 5-tuple of the UDP connection provides enough context for the Registrar to pick an appropriate context. Note that the socket will need to be used for multiple DTLS flows, which is atypical for how DTLS usually uses sockets. The pledge_context_message can be used to select an appropriate DTLS context, as DTLS headers do not contain any kind of of per session context. The pledge_context_message needs to be linked to the DTLS context, and when DTLS records need to be sent, then the pledge_context_message needs to be prepended to the data that is sent.

Examples are shown in Appendix A.

At the CoAP level, within the Constrained BRSKI and the EST-COAP [RFC9148] level, the block option [RFC7959] is often used. The Registrar and the Pledge MUST select a block size that would allow the addition of the JPY_message header without violating MTU sizes.

5. Discovery

5.1. Discovery operations by Join Proxy

In order to accomodate automatic configuration of the Join Proxy, it must discover the location and a capabilities of the Registar.

Section 10.2 of [I-D.ietf-anima-constrained-voucher] explains the basic mechanism, and this section explains the extensions required to discover whether stateless operation is supported.

5.1.1. CoAP discovery

Section 10.2.2 of [I-D.ietf-anima-constrained-voucher] describes how to use CoAP Discovery. The stateless Join Proxy requires a different end point that can accept the JPY encapsulation.

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 /.well-known/core?rt=brski.rjp

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

In the [RFC6690] link format, and [RFC3986], Section 3.2, the authority attribute can not include a port number unless it also includes the IP address.

The returned join-port is expected to process the encapsulated JPY messages described in section Section 4.3. The scheme remains coaps, as the inside protocol is still CoAP and DTLS.

An EST/Registrar server running at address 2001:db8:0:abcd::52, with the JPY process on port 7634, and the stateful Registrar on port 5683 could reply to a multicast query as follows:

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

RES: 2.05 Content
<coaps+jpy://[2001:db8:0:abcd::52]:7634>;rt=brski.rjp,
<coaps://[2001:db8:0:abcd::52]/.well-known/brski/rv>;rt=brski.rv;ct=83
<coaps://[2001:db8:0:abcd::52]/.well-known/brski/vs>;rt=brski.vs;ct="5
<coaps://[2001:db8:0:abcd::52]/.well-known/brski/es>;rt=brski.es;ct="5
```

The coaps+jpy scheme is registered is defined in Section 8.4, as per [RFC7252], Section 6.2

5.1.2. GRASP discovery

Section 10.2.1 of [I-D.ietf-anima-constrained-voucher] describes how to use GRASP [RFC8990] discovery within the ACP to locate the stateful port of the Registrar.

A Join Proxy which supports a stateless mode of operation using the mechanism described in Section 4.3 must know where to send the encoded content from the pledge. The Registrar announces its willingness to use the stateless mechanism by including an additional objective in it's M_FLOOD'ed AN_join_registrar announcements, but with a different objective value.

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_RJP".

Here is an example M_FLOOD announcing the Registrar on example port 5685, which is a port number chosen by the Registrar.

```
[M_FLOOD, 51804231, h'fda379a6f6ee00000200000064000001', 180000,
[["AN_join_registrar", 4, 255, "BRSKI_RJP"],
 [O_IPv6_LOCATOR,
  h'fda379a6f6ee00000200000064000001', IPPROTO_UDP, 5685]]]
```

Figure 5: Example of Registrar announcement message

Most Registrars will announce both a JPY-stateless and stateful ports, and may also announce an HTTPS/TLS service:

```
[M_FLOOD, 51840231, h'fda379a6f6ee00000200000064000001', 180000,
[["AN_join_registrar", 4, 255, ""],
 [O_IPv6_LOCATOR,
 h'fda379a6f6ee00000200000064000001', IPPROTO_TCP, 8443],
 ["AN_join_registrar", 4, 255, "CMP"],
 [O_IPv6_LOCATOR,
  h'fda379a6f6ee00000200000064000001', IPPROTO_TCP, 8448],
 ["AN_join_registrar", 4, 255, "BRSKI_JP"],
 [O_IPv6_LOCATOR,
  h'fda379a6f6ee00000200000064000001', IPPROTO_UDP, 5684],
 ["AN_join_registrar", 4, 255, "BRSKI_RJP"],
 [O_IPv6_LOCATOR,
  h'fda379a6f6ee00000200000064000001', IPPROTO_UDP, 5685]]]
```

Figure 6: Example of Registrar announcing two services

5.2. Pledge discovers Join Proxy

Regardless of whether the Join Proxy operates in stateful or stateless mode, the Join Proxy is discovered by the Pledge identically. When doing constrained onboarding with DTLS as security, the Pledge will always see an IPv6 Link-Local destination, with a single UDP port to which DTLS messages are to be sent.

5.2.1. 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 [RFC9148]).

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

* The transport-proto is IPPROTO_UDP

* the objective is AN_Proxy

*the objective-value is "DTLS-EST"

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] .

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

```
[M_FLOOD, 12340815, h'fe800000000000000000000000000001', 180000,
[["AN_Proxy", 4, 1, "DTLS-EST"],
[O_IPv6_LOCATOR,
h'fe800000000000000000000000000001', IPPROTO_UDP, 5684]]]
```

Figure 7: Example of Registrar announcement message

5.2.3. 6tisch Discovery

The discovery of CoJP [RFC9031] compatible Join-Proxy by the Pledge uses the enhanced beacons as discussed in [RFC9032]. 6tisch does not use DTLS and so this specification does not apply to it.

The Enhanced Beason discovery mechanism used in 6tisch does not convey a method to the pledge, (equivalent to an objective value, as described above), so only the CoAP/OSCORE mechanism described in [RFC9031] is announced.

A 6tisch network that wanted to use DTLS for security would need a new attribute for the enhanced beacon that announced the availability of a DTLS proxy as described in this document. Future work could provide that capability.

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 operators 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 8: Comparison between stateful and stateless mode

7. Security Considerations

All 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.

A malicious constrained Join Proxy has a number of routing possibilities:

* It sends the message on to a malicious Registrar. This is the same case as the presence of a malicious Registrar discussed in RFC 8995.

* It does not send on the request or does not return the response from the Registrar. This is the case of the not responding or crashing Registrar discussed in RFC 8995.

* It uses the returned response of the Registrar to enroll itself in the network. With very low probability it can decrypt the response because successful enrollment is deemed unlikely.

* It uses the request from the pledge to appropriate the pledge certificate, but then it still needs to acquire the private key of the pledge. This, too, is assumed to be highly unlikely.

* A malicious node can construct an invalid Join Proxy message. Suppose, the destination port is the coaps port. In that case, a Join Proxy can accept the message and add the routing addresses without checking the payload. The Join Proxy then routes it to the Registrar. In all cases, the Registrar needs to receive the message at the join-port, checks that the message consists of two parts and uses the DTLS payload to start the BRSKI procedure. It is highly unlikely that this malicious payload will lead to node acceptance.

* A malicious node can sniff the messages routed by the constrained Join Proxy. It is very unlikely that the malicious node can decrypt the DTLS payload. A malicious node can read the header field of the message sent by the stateless Join Proxy. This ability does not yield much more information than the visible addresses transported in the network packets.

It should be noted here that the contents of the CBOR array 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. These concerns are also expressed in [RFC8974]. It is also pointed out that the encryption in the source is a local matter. Similarly to [RFC8974], the use of AES-CCM [RFC3610] with a 64-bit tag is recommended, combined with a sequence number and a replay window.

If such scenario needs to be avoided, the constrained Join Proxy MUST encrypt the CBOR array using a locally generated symmetric key. The Registrar is not able to examine the encrypted result, but does not need to. The Registrar stores the encrypted header in the return packet without modifications. The constrained Join Proxy can decrypt the contents to route the message to the right destination.

In some installations, layer 2 protection is provided between all member pairs of the mesh. In such an environment encryption of the CBOR array is unnecessary because the layer 2 protection already provide it.

8. IANA Considerations

8.1. Extensions to the "BRSKI AN_Proxy Objective Value" Registry

[I-D.ietf-anima-constrained-voucher] previously registered the objective value DTLS-EST. This document makes use of it, and the registry should be extended to reference this document as well.

8.2. Extensions to the "BRSKI AN_join_registrar Objective Value" Registry

This document registers the objective-value: "BRSKI_RJP"

8.3. 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 resources of the constrained
             Join Proxy.
Reference: [this document]

Attribute Value: brski.rjp
Description: This BRSKI resource type is used for the constrained
             Join Proxy to query and return Join Proxy specific
             BRSKI resources of a Registrar.
Reference: [this document]
```

8.4. CoAPS+JPY Scheme Registration

```
Scheme name: coaps+jpy
Status: permanent
Applications/protocols that use this scheme name: Constrained BRSKI Join
Contact: ANIMA WG
Change controller: IESG
References: [THIS RFC]
Scheme syntax: identical to coaps
Scheme semantics: The encapsulation mechanism described in {{stateless-j
Security considerations: The new encapsulation allows traffic to be retu
   behind a proxy.  The form of the encapsulation can include privacy an
   under the control of the proxy system.
```

8.5. 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]
```

9. Acknowledgements

Many thanks for the comments by Carsten Bormann, Brian Carpenter, Spencer Dawkins, Esko Dijk, Toerless Eckert, Russ Housley, Ines Robles, Rich Salz, Jürgen Schönwälder, Mališa Vučinić, and Rob Wilton.

10. 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.

11. Changelog

11.1. 13 to 12

* jpy message encrypted and no longer standardized

11.2. 12 to 11

* many typos fixes and text re-organized
* core of GRASP and CoAP discovery moved to contrained-voucher document,

11.3. 11 to 10

* Join-Proxy and Registrar discovery merged
* GRASP discovery updated
* ARTART review
* TSVART review

11.4. 10 to 09

* OPSDIR review
* IANA review
* SECDIR review
* GENART review

11.5. 09 to 07

* typos

11.6. 06 to 07

* AD review changes

11.7. 05 to 06

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

11.8. 04 to 05

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

11.9. 03 to 04

* mail address and reference

11.10. 02 to 03

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

11.11. 01 to 02

*Discovery of Join Proxy and Registrar ports

11.12. 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

11.13. 00 to 00

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

12. References

12.1. Normative References

[family] "Address Family Numbers", IANA, 19 October 2021, <https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>.

[I-D.ietf-anima-constrained-voucher] Richardson, M., Van der Stok, P., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-21, 7 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-constrained-voucher-21>.

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

[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>.

[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>.

[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>.

12.2. Informative References

[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://datatracker.ietf.org/doc/html/draft-kumar-dice-dtls-relay-02>.

[I-D.richardson-anima-state-for-joinrouter]
Richardson, M., "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://datatracker.ietf.org/doc/html/draft-richardson-anima-state-for-joinrouter-03>.

[RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <https://www.rfc-editor.org/info/rfc3610>.

[RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <https://www.rfc-editor.org/info/rfc3927>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[RFC6550]
Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

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

[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>.

[RFC7102] Vasseur, JP., "Terms Used in Routing for Low-Power and Lossy Networks", RFC 7102, DOI 10.17487/RFC7102, January 2014, <https://www.rfc-editor.org/info/rfc7102>.

[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>.

[RFC7959]
Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[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>.

[RFC8974] Hartke, K. and M. Richardson, "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", RFC 8974, DOI 10.17487/RFC8974, January 2021, <https://www.rfc-editor.org/info/rfc8974>.

[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>.

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)
      FE800000000000000000FFFFC0A801C8 #
   19 BDA7                           # unsigned(48551)
   01                                # unsigned(1) IP
   00                                # unsigned(0)
   58 2D                             # bytes(45)
<cacrts DTLS encrypted request>
```

In CBOR Diagnostic:

```
 [h'FE800000000000000000FFFFC0A801C8', 48551, 1, 0,
  h'<cacrts DTLS encrypted request>']
```

The response is:

```
   85                                  # array(5)
      50                               # bytes(16)
         FE800000000000000000FFFFC0A801C8 #
      19 BDA7                          # unsigned(48551)
      01                               # unsigned(1) IP
      00                               # unsigned(0)
   59 026A                             # bytes(618)
      <cacrts DTLS encrypted response>
```

In CBOR diagnostic:

```
    [h'FE800000000000000000FFFFC0A801C8', 48551, 1, 0,
    h'<cacrts DTLS encrypted response>']
```

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Peter van der Stok
vanderstok consultancy

Email: stokcons@bbhmail.nl

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com