

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

S. Weiler
A. Sonalker
SPARTA, Inc.
R. Austein
ISC
July 11, 2011

A Publication Protocol for the Resource Public Key Infrastructure (RPKI)
[draft-ietf-sidr-publication-01](#)

Abstract

This document defines a protocol for publishing Resource Public Key Infrastructure (RPKI) objects. Even though the RPKI will have many participants issuing certificates and creating other objects, it is operationally useful to consolidate the publication of those objects. This document provides the protocol for doing so.

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 <http://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 January 12, 2012.

Copyright Notice

Copyright (c) 2011 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 (<http://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 Simplified BSD License text as described in [Section 4](#).e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Terminology	3
2.	Context	3
3.	Protocol Specification	4
3.1.	Common Details	4
3.1.1.	Common XML Message Format	4
3.2.	Control Sub-Protocol	5
3.2.1.	Config Object	5
3.2.2.	Client Object	5
3.3.	Publication Sub-Protocol	6
3.4.	Error handling	7
3.5.	XML Schema	7
4.	Operational Considerations	9
5.	IANA Considerations	9
6.	Security Considerations	10
7.	References	10
7.1.	Normative References	10
7.2.	Informative References	10
	Authors' Addresses	11

1. Introduction

This document assumes a working knowledge of the Resource Public Key Infrastructure (RPKI), which is intended to support improved routing security on the Internet. [[I-D.ietf-sidr-arch](#)]

In order to make participation in the RPKI easier, it is helpful to have a few consolidated repositories for RPKI objects, thus saving every participant from the cost of maintaining a new service. Similarly, relying parties using the RPKI objects will find it faster and more reliable to retrieve the necessary set from a smaller number of repositories.

These consolidated RPKI object repositories will in many cases be outside the administrative scope of the organization issuing a given RPKI object. Hence the need for a protocol to publish RPKI objects.

This document defines the RPKI publication protocol, including a sub-protocol for configuring the publication engine.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

"Publication engine" and "publication server" are used interchangeably to refer to the server providing the service described in this document.

"Business Public Key Infrastructure" ("Business PKI" or "BPKI") refers to a PKI, separate from the RPKI, used to authenticate clients to the publication engine.

2. Context

This protocol was designed specifically for the case where an internet registry, already issuing RPKI certificates to its children, also wishes to run a publication service for its children.

We use the term "Business PKI" here because an internet registry might already have a PKI, separate from the RPKI, for authenticating its clients and might wish to reuse that PKI for this protocol. Such reuse is not a requirement.

3. Protocol Specification

In summary, the publication protocol uses XML messages wrapped in CMS, carried over HTTP transport.

The publication protocol consists of two separate subprotocols. The first is a control protocol used to configure a publication engine. The second subprotocol, which we refer to by the overloaded term "publication protocol", is used to request publication of specific objects. The publication engine operates a single HTTP server on a single port. It distinguishes between the two protocols by using different URLs for them.

3.1. Common Details

This section discusses details that the two subprotocols have in common, including the transport and CMS wrappers.

Both protocols use a simple request/response interaction. The client passes a request to the server, and the server generates a corresponding response.

A message exchange commences with the client initiating an HTTP POST with content type of "application/rpki-publication", with the message object as the body. The server's response will similarly be the body of the response with a content type of "application/rpki-publication".

The content of the POST and the server's response will be a well-formed Cryptographic Message Syntax (CMS) [[RFC5652](#)] object with OID = 1.2.840.113549.1.7.2 as described in Section 3.1 of [[I-D.ietf-sidr-rescerts-provisioning](#)].

3.1.1. Common XML Message Format

The XML schema for this protocol (including both subprotocols) is below in [Section 3.5](#). Both subprotocols use the same basic XML message format, which looks like:


```
<?xml version='1.0' encoding='us-ascii'?>
<msg xmlns="http://www.hactrn.net/uris/rpki/publication-spec/"
    version="2"
    type="message type">
    [one or more PDUs]
</msg>
```

version:

The value of this attribute is the version of this protocol.
This document describes version 2.

type:

The possible values of this attribute are "reply" and "query".

A query PDU may be one of four types: config_query, client_query, publish_query, or withdraw_query. The first two are used by the control sub-protocol, the latter two by the publication sub-protocol.

A reply PDU may be one of five types: config_reply, client_reply, publish_reply, withdraw_reply, or report_error_reply.

Each of these PDUs may include an optional tag to facilitate bulk operation. If a tag is set in a query PDU, the corresponding reply(s) MUST have the tag attribute set to the same value.

3.2. Control Sub-Protocol

The control sub-protocol is used to configure a publication server. It can set global variables (at the moment, limited to a BPKI CRL) and manage clients who are allowed to publish data on the server.

3.2.1. Config Object

The <config/> object allows configuration of data that apply to the entire publication server rather than a particular client. There is exactly one <config/> object in the publication server, and it only supports the "set" and "get" actions -- it cannot be created or destroyed. Its use is typically restricted to the repository operator.

The <config/> object only has one data element that can be set: the bpkc_crl. This is used by the publication server when authenticating clients.

3.2.2. Client Object

Unlike the <config/> object, the <client/> object represents one client authorized to use the publication server. There may well be

more than one <client/> object on each publication server. Again, its use is typically restricted to the repository operator.

The <client/> object supports five actions: "create", "set", "get", "list", and "destroy". Each client has a "client_handle" attribute, which is used in responses and must be specified in "create", "set", "get", or "destroy" actions.

Payload data which can be configured in a <client/> object include:

- o base_uri (attribute): This attribute represents the base URI below which the client will be allowed to publish data. Additional constraints may be imposed by the publication server in certain cases, for e.g., a child publishing directly under its parent.
- o bpki_cert (element): This represents the X.509 BPki CA certificate for this client. This should be used as part of the certificate chain when validating incoming CMS messages. Two valid approaches exist. If the optional bpki_glue certificate is being used, then the bpki_cert certificate should be issued by the bpki_glue certificate; otherwise, the bpki_cert certificate should be issued by the publication engine's bpki_ta certificate.
- o bpki_glue (element): This is an additional (optional) type of X.509 certificate for this client. It may be used in certain pathological cross-certification cases which require a two-certificate chain due to issuer name conflicts. When being used, issuing order is that the bpki_glue certificate should be the issuer of the bpki_cert certificate. Otherwise, it should be issued by the publication engine's bpki_ta certificate. Since this is an optional use certificate, it may be left unset if not needed.

3.3. Publication Sub-Protocol

The sub-publication protocol requests publication or withdrawal from publication of RPKI objects.

The publication protocol uses a common message format to request publication of any RPKI object. This format was chosen specifically to allow this protocol to accommodate new types of RPKI objects without needing changes to this protocol.

Both the <publish/> and <withdraw/> objects have a payload of an optional tag and a URI. The <publish/> query also contains the DER object to be published, encoded in Base64.

Note that every publish and withdraw action requires a new manifest,

thus every publish or withdraw action will involve at least two objects.

3.4. Error handling

Errors are handled similarly in both subprotocols, and they're handled at two levels.

Since all messages in this protocol are conveyed over HTTP connections, basic errors are indicated via the HTTP response code. 4xx and 5xx responses indicate that something bad happened. Errors that make it impossible to decode a query or encode a response are handled in this way.

Where possible, errors will result in an XML `<report_error/>` message which takes the place of the expected protocol response message. `<report_error/>` messages are CMS-signed XML messages like the rest of this protocol, and thus can be archived to provide an audit trail.

`<report_error/>` messages only appear in replies, never in queries. The `<report_error/>` message can appear in both the control and publication subprotocols.

Like all other messages in this protocol, the `<report_error/>` message includes a "tag" attribute to assist in matching the error with a particular query when using batching. It is optional to set the tag on queries but, if set on the query, it MUST be set on the reply or error.

The error itself is conveyed in the `error_code` (attribute). The value of this attribute is a token indicating the specific error that occurred.

The body of the `<report_error/>` element itself is an optional text string; if present, this is debugging information.

3.5. XML Schema

The following is a RelaxNG compact form schema describing the Publication Protocol.

```
default namespace = "http://www.hactrn.net/uris/rpki/publication-spec/"

# Top level PDU
start = element msg {
  attribute version { "2" } ,
  ( ( attribute type { "query" }, query_elt*) |
```



```
(attribute type { "reply" }, reply_elt*))
}

# PDUs allowed in a query
query_elt = ( config_query | client_query | publish_query |
  withdraw_query )

# PDUs allowed in a reply
reply_elt = ( config_reply | client_reply | publish_reply |
  withdraw_reply | report_error_reply )

# Tag attributes for bulk operations
tag = attribute tag { xsd:token {maxLength="1024" } }

# Base64 encoded DER stuff
base64 = xsd:base64Binary

# Publication URLs
uri_t = xsd:anyURI { maxLength="4096" }
uri = attribute uri { uri_t }

# Handles on remote objects (replaces passing raw SQL IDs).  NB:
# Unlike the up-down protocol, handles in this protocol allow
# "/" as a hierarchy delimiter.
object_handle = xsd:string {
  maxLength="255" pattern="[\-_A-Za-z0-9/]*" }

# <config/> element (use restricted to repository operator)
# config_handle attribute: create, list, and destroy commands
# omitted deliberately.
config_payload = (element bpkc_crl { base64 }?)
config_query |= element config { attribute action { "set" }, tag?,
  config_payload }
config_reply |= element config { attribute action { "set" }, tag? }
config_query |= element config { attribute action { "get" }, tag? }
config_reply |= element config { attribute action { "get" }, tag?,
  config_payload }

# <client/> element (use restricted to repository operator)
client_handle = attribute client_handle { object_handle }
client_payload = (attribute base_uri { uri_t }?, element bpkc_cert {
  base64 }?, element bpkc_glue { base64 }?)
client_query |= element client { attribute action { "create" },
  tag?, client_handle, client_payload }
client_reply |= element client { attribute action { "create" },
  tag?, client_handle }
client_query |= element client { attribute action { "set" }, tag?,
  client_handle, client_payload }
```



```
client_reply |= element client { attribute action { "set" }, tag?,
  client_handle }
client_query |= element client { attribute action { "get" }, tag?,
  client_handle }
client_reply |= element client { attribute action { "get" }, tag?,
  client_handle, client_payload }
client_query |= element client { attribute action { "list" }, tag? }
client_reply |= element client { attribute action { "list" }, tag?,
  client_handle, client_payload }
client_query |= element client { attribute action { "destroy" },
  tag?, client_handle }
client_reply |= element client { attribute action { "destroy" },
  tag?, client_handle }

# <publish/> element
publish_query |= element publish { tag?, uri, base64 }
publish_reply |= element publish { tag?, uri }

# <withdraw/> element
withdraw_query |= element withdraw { tag?, uri }
withdraw_reply |= element withdraw { tag?, uri }

# <report_error/> element
error = xsd:token { maxLength="1024" }
report_error_reply = element report_error {
  tag?,
  attribute error_code { error },
  xsd:string { maxLength="512000" }?
}
```

4. Operational Considerations

Placeholder section to talk about nesting children under parents in the same repository, to allow for a single rsync to fetch both (observing that the rsync setup times tends to dominate over the sync time). And, more distressingly, talk about the access control impacts of that nesting.

5. IANA Considerations

IANA is asked to register the application/rpki-publication MIME media type as follows:

MIME media type name: application
MIME subtype name: rpki-publication
Required parameters: None
Optional parameters: None
Encoding considerations: binary
Security considerations: Carries an RPKI Publication Protocol
Message, as defined in this document.
Interoperability considerations: None
Published specification: This document
Applications which use this media type: HTTP
Additional information:
Magic number(s): None
File extension(s):
Macintosh File Type Code(s):
Person & email address to contact for further information:
Rob Austein <sra@isc.org>
Intended usage: COMMON
Author/Change controller: Rob Austein <sra@isc.org>

6. Security Considerations

7. References

7.1. Normative References

- [I-D.ietf-sidr-rescerts-provisioning]
Huston, G., Loomans, R., Ellacott, B., and R. Austein, "A Protocol for Provisioning Resource Certificates", [draft-ietf-sidr-rescerts-provisioning-10](#) (work in progress), June 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, [RFC 5652](#), September 2009.

7.2. Informative References

- [I-D.ietf-sidr-arch]
Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", [draft-ietf-sidr-arch-13](#) (work in progress), May 2011.

Authors' Addresses

Samuel Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com

Anuja Sonalker
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: Anuja.Sonalker@sparta.com

Rob Austein
ISC
950 Charter Street
Redwood City, CA 94063
USA

Email: sra@isc.org

