Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

S. Weiler
Parsons
A. Sonalker
TowerSec
R. Austein
Dragon Research Labs

March 21, 2016

A Publication Protocol for the Resource Public Key Infrastructure (RPKI)

draft-ietf-sidr-publication-08

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

Copyright Notice

Table of Contents

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. [RFC6480]

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. In some cases, outsourcing operation of the repository will be an explicit goal: some resource holders who strongly wish to control their own RPKI private keys may lack the resources to operate a 24x7 repository, or may simply not wish to do so.

The operator of an RPKI publication repository may well be an Internet registry which issues certificates to its customers, but it need not be; conceptually, operation of a an RPKI publication repository is separate from operation of RPKI CA.

This document defines an RPKI publication protocol which allows publication either within or across organizational boundaries, and which makes fairly minimal demands on either the CA engine or the publication service.

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. We use the term "Business PKI" here because an Internet registry might already have a PKI for authenticating its clients and might wish to reuse that PKI for this protocol. There is, however, no requirement to reuse such a PKI.

2. Protocol Specification

The publication protocol uses XML messages wrapped in signed CMS messages, carried over HTTP transport.

The publication protocol uses 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 [RFC6492].

2.1. Common XML Message Format

The XML schema for this protocol is below in Section 2.6. The basic XML message format looks like this:

```
<msg
    type="query"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <!-- Zero or more PDUs -->
</msg>

<msg
    type="reply"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <!-- Zero or more PDUs -->
</msg>
```

Common attributes:

version: The value of this attribute is the version of this protocol. This document describes version 4.

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

A query PDU may be one of three types: <publish/>, <withdraw/>, or <list/>.

A reply PDU may be one of three types: <success/>, <list/>, or <report_error/>.

The <publish/> and <withdraw/> PDUs include a tag to facilitate bulk operation.

2.2. Publication and Withdrawal

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

Both the <publish/> and <withdraw/> PDUs also have a "hash" attribute, which carries a hash of an existing object at the specified repository URI. For <withdraw/> PDUs, the hash is mandatory, as this operation makes no sense if there is no existing object to withdraw. For <publish/> PDUs, the hash MUST be present if the publication operation is overwriting an existing object, and MUST be omitted if this publication operation is writing to a new URI where no prior object exists. Presence of an object when no "hash" attribute is specified is an error, as is absence of the "hash" attribute or an incorrect hash value when an object is present. Any such errors MUST be reported using the <report_error/> PDU.

The hash algorithm is SHA-256 [SHS], to simplify comparison of publication protocol hashes with RPKI manifest hashes.

The intent behind the "hash" attribute is to allow the client and server to detect any disagreements about the effect that a <publish/> or <withdraw/> PDU will have on the repository.

Note that every publish and withdraw action requires a new manifest, thus every publish or withdraw action will involve at least two objects.

Processing of a query message is handled atomically: either the entire query succeeds or none of it does. When a query message contains multiple PDUs, failure of any PDU may require the server to roll back actions triggered by earlier PDUs.

When a query messages containing <publish/> and/or <withdraw/> PDUs succeeds, a single <success/> reply is returned.

When a query fails, one or more <report_error/> reply PDUs are generated. Typically, only one <report_error/> reply is generated, corresponding to the first query PDU that failed. Servers are permitted to return multiple <report_error/> PDUs.

2.3. Listing the repository

The <list/> operation allows the client to ask the server for a complete listing of objects which the server believes the client has published. This is intended primarily to allow the client to recover

upon detecting (probably via use of the "hash" attribute, see Section 2.2) that they have somehow lost synchronization.

The <list/> query consists of a single PDU. A <list/> query must be the only PDU in a query - it may not be combined with any <publish/> or <withdraw/> queries.

The <list/> reply consists of zero or more PDUs, one per object published in this repository by this client, each PDU conveying the URI and hash of one published object.

2.4. **Error handling**

Errors are handled at two levels.

Errors that make it impossible to decode a query or encode a response are handled at the HTTP layer. 4xx and 5xx HTTP response codes indicate that something bad happened.

In all other cases, errors result in an XML <report_error/> PDU. Like the rest of this protocol, <report_error/> PDUs are CMS-signed XML messages and thus can be archived to provide an audit trail.

<report_error/> PDUs only appear in replies, never in queries.

The "tag" attribute of the <report_error/> PDU associated with a <publish/> or <withdraw/> PDU MUST be set to the same value as the "tag" attribute in the PDU which generated the error. A client can use the "tag" attribute to determine which PDU caused processing of an update to fail.

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 contains two sub-elements:

1. An optional text element <error_text/>, which if present, contains a text string with debugging information intended for human consumption.

2. An optional element <failed_pdu/>, which, if present, contains a verbatim copy of the query PDU whose failure triggered the <report_error/> PDU. The quoted element must be syntactically valid.

See Section 3.7 for examples of a multi-element query and responses.

2.5. Error Codes

These are the defined error codes as well as some discussion of each. Text similar to these descriptions may be sent in an <error_text/> element to help explain the error encountered.

xml_error: Encountered an XML problem. Note that some XML errors may be severe enough to require error reporting at the HTTP layer, instead. Implementations MAY choose to report any or all XML errors at the HTTP layer.

permission_failure: Client does not have permission to update this URI.

bad_cms_signature: Bad CMS signature.

object_already_present: An object is already present at this URI, yet a "hash" attribute was not specified. A "hash" attribute must be specified when overwriting or deleting an object. Perhaps client and server are out of sync?

no_object_present: There is no object present at this URI, yet a "hash" attribute was specified. Perhaps client and server are out of sync?

no_object_matching_hash The "hash" attribute supplied does not match the "hash" attribute of the object at this URI. Perhaps client and server are out of sync?

consistency_problem: Server detected an update that looks like it will cause a consistency problem (e.g. an object was deleted, but the manifest was not updated). Note that a server is not required to make such checks. Indeed, it may be unwise for a server to do so. This error code just provides a way for the server to explain its (in-)action.

other_error: A meteor fell on the server.

2.6. XML Schema

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

```
# $Id: rpki-publication.rnc 3595 2016-03-21 21:31:37Z sra $
# RelaxNG schema for RPKI publication protocol.

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

```
   # This is version 4 of the protocol.

   version = "4"

   # Top level PDU is either a query or a reply.

   start |= element msg {
     attribute version { version },
     attribute type    { "query" },
     query_elt*
   }

   start |= element msg {
     attribute version { version },
     attribute type    { "reply" },
     reply_elt*
   }

   # Tag attributes for bulk operations.

   tag = attribute tag { xsd:token { maxLength="1024" } }

   # Base64 encoded DER stuff.

   base64 = xsd:base64Binary

   # Publication URIs.

   uri = attribute uri { xsd:anyURI { maxLength="4096" } }

   # Digest of an existing object (hexadecimal).

   hash = attribute hash { xsd:string { pattern = "[0-9a-fA-F]+" } }

   # Error codes.

   error |= "xml_error"
   error |= "permission_failure"
   error |= "bad_cms_signature"
   error |= "object_already_present"
   error |= "no_object_present"
   error |= "no_object_matching_hash"
   error |= "consistency_problem"
   error |= "other_error"

   # <publish/> query

   query_elt |= element publish { tag, uri, hash?, base64 }
```

```
   # <withdraw/> query

   query_elt |= element withdraw { tag, uri, hash }

   # <success/> reply

   reply_elt |= element success { empty }

   # <list/> query and reply

   query_elt |= element list { empty }
   reply_elt |= element list { uri, hash }

   # <report_error/> reply

   reply_elt |= element report_error {
     tag?,
     attribute error_code { error },
     element   error_text { xsd:string { maxLength="512000" }}?,
     element   failed_pdu { query_elt }?
   }
```

3. Examples

Following are examples of various queries and the corresponding replies for the RPKI publication protocol.

Note the authors have taken liberties with the Base64, hash, and URI text in these examples in the interest of making the examples fit nicely into RFC text format.

3.1. <publish/> Query, No Existing Object

```
   <msg
       type="query"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <publish
         tag="foo"
         uri="rsync://wombat.example/Alice/01a97a70ac477f06.cer">
         SGVsbG8sIG15IG5hbWUgaXMgQWxpY2U=
       </publish>
   </msg>
```

3.2. <publish/> Query, Overwriting Existing Object

```
<msg
    type="query"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <publish
      hash="01a97a70ac477f06"
      tag="foo"
      uri="rsync://wombat.example/Alice/01a97a70ac477f06.cer">
      SGVsbG8sIG15IG5hbWUgaXMgQWxpY2U=
    </publish>
</msg>
```

3.3. <withdraw/> Query

```
<msg
    type="query"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <withdraw
      hash="01a97a70ac477f06"
      tag="foo"
      uri="rsync://wombat.example/Alice/01a97a70ac477f06.cer"/>
</msg>
```

3.4. <success/> Reply

```
<msg
    type="reply"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <success/>
</msg>
```

3.5. <report_error/> With Optional Elements

```
   <msg
       type="reply"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <report_error
         error_code="no_object_matching_hash"
         tag="foo">
       <error_text>
         Can't delete an object I don't have
       </error_text>
       <failed_pdu>
         <publish
             hash="01a97a70ac477f06"
             tag="foo"
             uri="rsync://wombat.example/Alice/01a97a70ac477f06.cer">
         SGVsbG8sIG15IG5hbWUgaXMgQWxpY2U=
       </publish>
       </failed_pdu>
     </report_error>
   </msg>
```

3.6. <report_error/> Without Optional Elements

```
   <msg
       type="reply"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <report_error
         error_code="object_already_present"
         tag="foo"/>
   </msg>
```

3.7. Error Handling With Multi-Element Queries

3.7.1. Multi-Element Query

```
   <msg
       type="query"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <publish
         tag="Alice"
         uri="rsync://wombat.example/Alice/01a97a70ac477f06.cer">
         SGVsbG8sIG15IG5hbWUgaXMgQWxpY2U=
       </publish>
     <withdraw
         hash="f46a4198efa3070e"
         tag="Bob"
         uri="rsync://wombat.example/Bob/f46a4198efa3070e.cer"/>
     <publish
         tag="Carol"
         uri="rsync://wombat.example/Carol/32e0544eeb510ec0.cer">
         SGVsbG8sIG15IG5hbWUgaXMgQ2Fyb2w=
       </publish>
     <withdraw
         hash="421ee4ac65732d72"
         tag="Dave"
         uri="rsync://wombat.example/Dave/421ee4ac65732d72.cer"/>
     <publish
         tag="Eve"
         uri="rsync://wombat.example/Eve/9dd859b01e5c2ebd.cer">
         SGVsbG8sIG15IG5hbWUgaXMgRXZl
       </publish>
   </msg>
```

3.7.2. Successful Multi-Element Response

```
   <msg
       type="reply"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <success/>
   </msg>
```

3.7.3. Failure Multi-Element Response, First Error Only

```
   <msg
       type="reply"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <report_error
         error_code="no_object_matching_hash"
         tag="Dave">
       <failed_pdu>
         <withdraw
             hash="421ee4ac65732d72"
             tag="Dave"
             uri="rsync://wombat.example/Dave/421ee4ac65732d72.cer"/>
       </failed_pdu>
     </report_error>
   </msg>
```

3.7.4. Failure Multi-Element Response, All Errors

```
   <msg
       type="reply"
       version="4"
       xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
     <report_error
         error_code="no_object_matching_hash"
         tag="Dave">
       <failed_pdu>
         <withdraw
             hash="421ee4ac65732d72"
             tag="Dave"
             uri="rsync://wombat.example/Dave/421ee4ac65732d72.cer"/>
       </failed_pdu>
     </report_error>
     <report_error
         error_code="object_already_present"
         tag="Eve">
       <failed_pdu>
         <publish
             tag="Eve"
             uri="rsync://wombat.example/Eve/9dd859b01e5c2ebd.cer">
         SGVsbG8sIG15IG5hbWUgaXMgRXZl
       </publish>
       </failed_pdu>
     </report_error>
   </msg>
```

3.8. <list/> Query

```
<msg
    type="query"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <list/>
</msg>
```

3.9. <list/> Reply

```
<msg
    type="reply"
    version="4"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <list
      hash="eb719b72f0648cf4"
      uri="rsync://wombat.example/Fee/eb719b72f0648cf4.cer"/>
  <list
      hash="c7c50a68b7aa50bf"
      uri="rsync://wombat.example/Fie/c7c50a68b7aa50bf.cer"/>
  <list
      hash="f222481ded47445d"
      uri="rsync://wombat.example/Foe/f222481ded47445d.cer"/>
  <list
      hash="15b94e08713275bc"
      uri="rsync://wombat.example/Fum/15b94e08713275bc.cer"/>
</msg>
```

4. Operational Considerations

There are two basic options open to the repository operator as to how the publication tree is laid out. The first option is simple: each publication client is given its own directory one level below the top of the rsync module, and there is no overlap between the publication spaces used by different clients. For example:

```
rsync://example.org/rpki/Alice/
rsync://example.org/rpki/Bob/
rsync://example.org/rpki/Carol/
```

This has the advantage of being very easy for the publication operator to manage, but has the drawback of making it difficult for relying parties to fetch published objects both safely and as efficiently as possible.

Given that the mandatory-to-implement retrieval protocol for relying parties is rsync, a more efficient repository structure would be one

which minimized the number of rsync fetches required. One such structure would be one in which the publication directories for subjects were placed underneath the publication directories of their issuers: since the normal synchronization tree walk is top-down, this can significantly reduce the total number of rsync connections required to synchronize. For example:

```
rsync://example.org/rpki/Alice/
rsync://example.org/rpki/Alice/Bob/
rsync://example.org/rpki/Alice/Bob/Carol/
```

Preliminary measurement suggests that, in the case of large numbers of small publication directories, the time needed to set up and tear down individual rsync connections becomes significant, and that a properly optimized tree structure can reduce synchronization time by an order of magnitude.

The more complex tree structure does require careful attention to the "base_uri" attribute values when setting up clients. In the example above, assuming that Alice issues to Bob who in turn issues to Carol, Alice has ceded control of a portion of her publication space to Bob, who has in turn ceded a portion of that to Carol, and the "base_uri" attributes in the <client/> setup messages should reflect this.

The details of how the repository operator determines that Alice has given Bob permission to nest Bob's publication directory under Alice's is outside the scope of this protocol.

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@hactrn.net>
Intended usage:  COMMON
Author/Change controller: Rob Austein <sra@hactrn.net>
```

6. Security Considerations

The RPKI publication protocol and the data it publishes use entirely separate PKIs for authentication. The published data is authenticated within the RPKI, and this protocol has nothing to do with that authentication, nor does it require that the published objects be valid in the RPKI. The publication protocol uses a separate Business PKI (BPKI) to authenticate its messages.

Each RPKI publication protocol message is CMS-signed. Because of that protection at the application layer, this protocol does not require the use of HTTPS or other transport security mechanisms.

Although the hashes used in the <publish/> and <withdraw/> PDUs are cryptographic strength, the digest algorithm was selected for convenience in comparing these hashes with the hashes that appear in RPKI manifests. The hashes used in the <publish/> and <withdraw/> PDUs are not particularly security-sensitive, because these PDUs are protected by the CMS signatures.

Compromise of a publication server, perhaps through mismanagement of BPKI keys, could lead to a denial-of-service attack on the RPKI. An attacker gaining access to BPKI keys could use this protocol delete (withdraw) RPKI objects, leading to routing changes or failures. Accordingly, as in most PKIs, good key management practices are important.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, STD 70, September 2009.

[RFC6492] Huston, G., Loomans, R., Ellacott, B., and R. Austein, "A Protocol for Provisioning Resource Certificates", RFC 6492, February 2012.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

7.2. Informative References

[RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.

Authors' Addresses

Samuel Weiler
Parsons

Email: weiler@tislabs.com

Anuja Sonalker
TowerSec Automotive Cyber Security

Email: asonalker@tower-sec.com

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net