Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 16, 2014

S. Weiler
SPARTA, Inc.
A. Sonalker
Battelle Memorial Institute
R. Austein
Dragon Research Labs

February 12, 2014

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

draft-ietf-sidr-publication-05

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 August 16, 2014.

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 stringly 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 2.4. The basic XML message format looks like this:

```
<msg
    type="query"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <!-- Zero or more PDUs -->
</msg>

<msg
    type="reply"
    version="3"
    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 3.

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

A query PDU may be one of two types: publish_query, or withdraw_query.

A reply PDU may be one of three types: 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.

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/> 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.

2.3. Error handling

Errors are 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.

2.4. XML Schema

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

```
# $Id: rpki-publication.rnc 2698 2013-12-13 23:33:07Z sra $
# RelaxNG schema for RPKI publication protocol.

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

# This is version 3 of the protocol.
```

```
   version = "3"

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

   start = element msg {
     attribute version { version } ,
     ( ( attribute type { "query" }, query_elt* ) |
       ( attribute type { "reply" }, reply_elt* ) )
   }

   # PDUs allowed in  queries and replies.

   query_elt = publish_query | withdraw_query
   reply_elt = 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 URIs.

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

   # Handles on remote objects (replaces passing raw SQL IDs).

   object_handle = xsd:string {
      maxLength = "255"
      pattern="[\-_A-Za-z0-9/]*"
   }

   # Error codes.

   error = xsd:token { maxLength="1024" }

   # <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

report_error_reply = element report_error {
  tag?,
  attribute error_code { error },
  xsd:string { maxLength="512000" }?
}
```

3. Examples

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

3.1. <publish/> Query

```
<msg
    type="query"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <publish
      uri="rsync://wombat.example/Alice/blCrcCp9ltyPDNzYKPfxc.cer">
      MIIE+jCCA+KgAwIBAgIBDTANBgkqhkiG9w0BAQsFADAzMTEwLwYDVQQDEyhE
      RjRBODAxN0U2NkE5RTkxNzJFNDYxMkQ4Q0Y0QzgzRjIzOERFMkEzMB4XDTA4
      MDUyMjE4MDUxMloXDTA4MDUyNDE3NTQ1M1owMzExMC8GA1UEAxMoOEZCODIx
      OEYwNkU1MEFCNzAyQTdEOTZEQzhGMENEQ0Q4MjhGN0YxNzCCASIwDQYJKoZI
      hvcNAQEBBQADggEPADCCAQoCggEBAMeziKp0k5nP7v6SZoNsXIMQYRgNtC6F
      r/9Xm/1yQHomiPqHUk47rHhGojYiK5AhkrwoYhkH4UjJl2iwklDYczXuaBU3
      F5qrKlZ4aZnjIxdlP7+hktVpeApL6yuJTUAYeC3UIxnLDVdD6phydZ/FOQlu
      ffiNDjzteCCvoyOUatqt8WB+oND6LToHp028g1YUYLHG6mur0dPdcHOVXLSm
      UDuZ1HDz1nDuYvIVKjB/MpH9aW9XeaQ6ZFIlZVPwuuvI2brR+ThH7Gv27GL/
      o8qFdC300VQfoTZ+rKPGDE8K1cI906BL4kiwx9z0oiDcE96QCz+B0vsjc9mG
      aA1jgAxlXWsCAwEAAaOCAhcwggITMB0GA1UdDgQWBBSPuCGPBuUKtwKn2W3I
      8M3Ngo9/FzAfBgNVHSMEGDAWgBTfSoAX5mqekXLkYS2M9Mg/I43iozBVBgNV
      HR8ETjBMMEqgSKBGhkRyc3luYzovL2xvY2FsaG9zdDo0NDAwL3Rlc3RiZWQv
      UklSLzEvMzBxQUYtWnFucEZ5NUdFdGpQVElQeU9ONHFNLmNybDBFBggrBgEF
      BQcBAQQ5MDcwNQYIKwYBBQUHMAKGKXJzeW5jOi8vbG9jYWxob3N0OjQ0MDAv
      dGVzdGJlZC9XT01CQVQuY2VyMBgGA1UdIAEB/wQOMAwwCgYIKwYBBQUHDgIw
      DwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwgZsGCCsGAQUFBwEL
      BIGOMIGLMDQGCCsGAQUFBzAFhihyc3luYzovL2xvY2FsaG9zdDo0NDAwL3Rl
      c3RiZWQvUklSL1IwLzEvMFMGCCsGAQUFBzAKhkdyc3luYzovL2xvY2FsaG9z
      dDo0NDAwL3Rlc3RiZWQvUklSL1IwLzEvajdnaGp3YmxDcmNDcDlsdHlQRE56
      WUtQZnhjLm1uZjAaBggrBgEFBQcBCAEB/wQLMAmgBzAFAgMA/BUwPgYIKwYB
      BQUHAQcBAf8ELzAtMCsEAgABMCUDAwAKAzAOAwUAwAACAQMFAcAAAiAwDgMF
      AsAAAiwDBQDAAAJkMA0GCSqGSIb3DQEBCwUAA4IBAQCEhuH7jtI2PJY6+zwv
      306vmCuXhtu9Lr2mmRw2ZErB8EMcb5xypMrNqMoKeu14K2x4a4RPJkK4yATh
      M81FPNRsU5mM0acIRnAPtxjHvPME7PHN2w2nGLASRsZmaa+b8A7SSOxVcFUR
      azENztppsolHeTpm0cpLItK7mNpudUg1JGuFo94VLf1MnE2EqARG1vTsNhel
      /SM/UvOArCCOBvf0Gz7kSuupDSZ7qx+LiDmtEsLdbGNQBiYPbLrDk41PHrxd
      x28qIj7ejZkRzNFw/3pi8/XK281h8zeHoFVu6ghRPy5dbOA4akX/KG6b8XIx
      0iwPYdLiDbdWFbtTdPcXBauY
    </publish>
</msg>
```

3.2. <publish/> Reply

```
<msg
    type="reply"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <publish
      uri="rsync://wombat.example/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
</msg>
```

3.3. <withdraw/> Query

```
<msg
    type="query"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <withdraw
      uri="rsync://wombat.example/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
</msg>
```

3.4. <withdraw/> Reply

```
<msg
    type="reply"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <withdraw
      uri="rsync://wombat.example/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
</msg>
```

3.5. <report_error/> With Text

```
<msg
    type="reply"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <report_error
      error_code="your_hair_is_on_fire">
    Shampooing with sterno again, are we?
  </report_error>
</msg>
```

3.6. <report_error/> Without Text

```
<msg
    type="reply"
    version="3"
    xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <report_error
      error_code="your_hair_is_on_fire"/>
</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 rcynic 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 of the RPKI publication protocol messages 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.

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.

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

Email: weiler@tislabs.com

Anuja Sonalker
Battelle Memorial Institute

Email: sonalkera@battelle.org

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net