

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2014

R. Austein
Dragon Research Labs
July 12, 2013

An Out-Of-Band Setup Protocol For RPKI Production Services
draft-austein-sidr-rpki-oob-setup-00

Abstract

This note describes a simple out-of-band protocol to ease setup of the RPKI provisioning and publication protocols between two parties. The protocol is encoded in a small number of XML messages, which can be passed back and forth by any mutually agreeable secure means.

This setup protocol is not part of the provisioning or publication protocol, rather, it is intended to simplify configuration of these protocols by setting up relationships and exchanging BPKI keying material.

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 13, 2014.

Copyright Notice

Copyright (c) 2013 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	2
2. Overview of the BPKI	3
3. Protocol Elements	4
3.1. Nomenclature	4
3.2. Common Protocol Elements	5
3.3. Protocol Messages	5
3.3.1. <child_request/>	6
3.3.2. <parent_response/>	6
3.3.3. <publisher_request/>	8
3.3.4. <repository_response/>	9
3.4. <authorization/>	10
3.5. <error/>	11
4. Protocol Walk-Through	12
5. IANA Considerations	16
6. Security Considerations	16
7. Acknowledgements	17
8. Normative References	17
Appendix A. RelaxNG Schema	17
Author's Address	19

1. Introduction

This note describes a small XML-based out-of-band protocol used to set up relationships between parents and children in the RPKI provisioning protocol ([RFC6492]) and between publishers and repositories in the RPKI publication protocol ([I-D.ietf-sidr-publication]).

The basic function of this protocol is public key exchange, in the form of self-signed BPKI X.509 certificates, but workshop experience has demonstrated that it's simpler for the user if we also bundle the other configuration information needed to bring up a new player into the messages used in the key exchange.

The underlying transport for this protocol is deliberately unspecified. It might be a USB stick, a web interface secured with conventional HTTPS, PGP-signed email, a T-shirt printed with a QR code, or a carrier pigeon.

Since much of the purpose of this protocol is key exchange, authentication and integrity of the key exchange MUST be ensured via

external means. Typically such means will tie directly to a new or existing business relationship

2. Overview of the BPKI

Several protocols related to RPKI provisioning use signed CMS messages to authenticate the underlying XML-based protocols. Verification of these CMS messages requires X.509 certificates. The PKI that holds these certificates is distinct from the RPKI, and contains no RFC 3779 resources. We refer to this as the "Business PKI" (BPKI), to distinguish it from the RPKI. The "B" is a hint that the certificate relationships in the BPKI are likely to follow and become part of existing contractual relationships between the issuers and subjects of this PKI.

The RPKI provisioning protocol does not dictate a particular structure for the BPKI, beyond the basic requirement that it be possible for one party to sign and the other party to verify the CMS messages. This allows a certain amount of flexibility to allow an Internet registry to reuse an existing PKI as the BPKI if that makes sense in their context.

In order to keep this protocol simple, we adopt a somewhat constrained model of the BPKI. The first two operations in this protocol are an exchange of public keys between child and parent for use in the provisioning protocol, the latter two operations in this protocol are an exchange of public keys between publisher and repository for use in the publication protocol. In each of these operations, the sending party includes its public key, in the form of a self-signed X.509 CA certificate. The private keys corresponding to the exchanged certificates are not used to sign CMS messages directly; instead, the exchanged CA certificates are the issuers of the BPKI end-entity (EE) certificates which will be included in the CMS messages and can be used, along with the exchanged certificates, to verify the CMS messages.

Details of how to tie the exchanged certificates into an implementation's local BPKI are left to the implementation, but the recommended approach is to cross-certify the received public key and subject name under one's own BPKI, using a Basic Constraints extension with `ca = TRUE`, `pathLenConstraint = 0`, indicating that the cross-certified certificate is a CA certificate which is allowed to issue EE certificates but is not allowed to issue CA certificates. See section 4.2.1.9 of [RFC5280] for more information about the Basic Constraints extension.

For example, suppose that Alice and Bob each have their own self-signed BPKI certificates:

```
Issuer:      CN = Alice CA
Subject:     CN = Alice CA
Public Key:  [Alice CA Public Key]
BasicConstraints: cA = TRUE

Issuer:      CN = Bob CA
Subject:     CN = Bob CA
Public Key:  [Bob CA Public Key]
BasicConstraints: cA = TRUE
```

Alice sends Bob her self-signed BPKI certificate, and Bob cross-certifies its public key and subject name under Bob's own self-signed BPKI certificate:

```
Issuer:      CN = Bob CA
Subject:     CN = Alice CA
Public Key:  [Alice CA Public Key]
BasicConstraints: cA = TRUE, pathLenConstraint = 0
```

Later, when Bob receives a CMS message from Alice, Bob can verify this message via a trust chain back to Bob's own trust anchor:

```
Issuer:      CN = Alice CA
Subject:     CN = Alice EE
Public Key:  [Alice EE Public Key]
```

[[Need some text detailing required and allowed values in the certificates: 2048-bit RSA, what extensions, But once we go there we also have to provide a path for algorithm agility.]]

3. Protocol Elements

Each message in the protocol is a distinct XML element in the "http://www.hactrn.net/uris/rpki/rpki-setup/" XML namespace.

3.1. Nomenclature

All of the protocols configured by this setup protocol have their own terminology for their actors, but in the context of this protocol that terminology becomes somewhat confusing. All of the players in this setup protocol issue certificates, are the subjects of other certificates, operate servers, and, in most cases, act as clients for one protocol or another. Therefore, this note uses its own terms for the actors in this protocol.

Child: An entity acting in the client ("subject") role of the provisioning protocol defined in [RFC6492].

Parent: An entity acting in the server ("issuer") role of the provisioning protocol defined in [RFC6492].

Publisher: An entity acting in the client role of the publication protocol defined in [I-D.ietf-sidr-publication].

Repository: An entity acting in the server role of the publication protocol defined in [I-D.ietf-sidr-publication].

Note that a given entity might act in more than one of these roles; for example, in one of the simplest cases, the child is the same entity as the publisher, while the parent is the same entity as the repository.

3.2. Common Protocol Elements

The first XML attribute in each message is a version field. This document describes version 1 of the protocol.

Most messages contain, among other things, a self-signed BPKI X.509 certificate. These certificates are represented as XML elements whose text value is the Base64 text encoding the DER representation of the X.509 certificate.

A number of attributes contain "handles". A handle in this protocol is a text string in the US-ASCII character set consisting of letters, digits, and the special characters "/", "-", and "_". This protocol places no special semantics on the structure of these handles, although implementations might. Handles are protocol elements, not necessarily meaningful to humans, thus the simplicity of a restricted character set makes more sense than the complex rules which would be needed for internationalized text.

3.3. Protocol Messages

The core of this protocol consists of four message types, representing the basic request and response semantics needed to configure a RPKI engine to talk to its parent and its repository via the provisioning and publication protocols, respectively.

3.3.1. <child_request/>

The <child_request/> message is an initial setup request from a provisioning protocol child to its provisioning protocol parent.

Fields in the <child_request/> message:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

child_handle: The child_handle attribute is what the child calls itself. This is just a hint from the child to the parent, the parent need not honor it.

child_bpki_ta: The <child_bpki_ta/> element is the child's BPKI identity, a self-signed X.509 BPKI certificate, encoded in Base64.

This CA certificate will be the issuer of the BPKI EE certificates corresponding to private keys that the child will use when sending provisioning protocol messages to the parent.

```
-----
<child_request
  child_handle="Bob"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <child_bpki_ta>
    R29kIGlzIHJlYWwgdW5sZXNzIGRlY2xhcmludGVnZXI=
  </child_bpki_ta>
</child_request>
-----
```

3.3.2. <parent_response/>

The <parent_response/> message is a response from a provisioning protocol parent to a provisioning protocol child that had previously sent a <child_request/> message.

Fields in the <parent_response/> message:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

service_uri: The service_uri attribute contains an HTTP URL that the child should contact for up-down ([RFC6492]) service.

child_handle: The `child_handle` attribute is the parent's name for the child. This might or might not match the `child_handle` from the `<child_request/>` message. If they do not match, the parent wins, because the parent gets to dictate the names in the provisioning protocol. This value is the sender field in provisioning protocol request messages and the recipient field in provisioning protocol response messages.

parent_handle: The `parent_handle` attribute is the parent's name for itself. This value is the recipient field in provisioning protocol request messages and the sender field in provisioning protocol response messages.

parent_bpki_ta: The `<parent_bpki_ta/>` element is the parent's BPKI identity, a self-signed X.509 BPKI certificate.

This certificate is the issuer of the BPKI EE certificates corresponding to private keys that the parent will use to sign provisioning protocol messages to the child.

offer: If an `<offer/>` element is present, the parent is offering publication service to the child. The `<offer/>` element, if present, is empty.

referral: If a `<referral/>` element is present, it suggests a third-party publication services that the child might use, and contains:

referrer: A referrer attribute, containing the handle by which the publication repository knows the parent,

contact_uri: An optional `contact_uri` attribute that the child may be able to follow for more information, and

Authorization token: The text of the `<referral/>` element is the Base64 encoding of a signed authorization token granting the child the right to use a portion of the parent's namespace at the publication repository in question. See Section 3.4 for details on the authorization token.

```
-----
<parent_response
  child_handle="Bob-42"
  parent_handle="Alice"
  service_uri="http://alice.example/rpki-up-down/Alice/Bob-42"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
<parent_bpki_ta>
  WW9lIGNhbiBoYWNRIGFueXRoaW5nIHlvdSB3YW50IHdpdGggVEVDTyBhbmQgRERU
```

```

    </parent_bpki_ta>
    <offer/>
  </parent_response>

```

```

<parent_response
  child_handle="Carol"
  parent_handle="Bob"
  service_uri="http://bob.example/rpki-up-down/Bob/Carol"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <parent_bpki_ta>
    R29kIGlziIHJlYWwgdW5sZXNzIGRlY2xhcmVklGludGVnZXI=
  </parent_bpki_ta>
  <referral
    referrer="Alice/Bob-42">
    R28sIGxlbWlpbmdzLCBnbyE=
  </referral>
</parent_response>

```

3.3.3. <publisher_request/>

The <publisher_request/> message is a setup request from a publisher to a repository.

Fields in the <publisher_request/> message:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

publisher_handle: The publisher_handle attribute is the publisher's name for itself. This is just a hint, the repository need not honor it.

publisher_bpki_ta: The <publisher_bpki_ta/> element is the publisher's BPKI identity, a self-signed X.509 BPKI certificate. This certificate is the issuer of the BPKI EE certificates corresponding to private keys that the publisher will use to sign publication protocol messages to the repository.

referral: If a <referral/> element is present, it contains:

referrer: A referrer attribute containing the publication handle of the referring parent, and

Authorization token: The text of the <referral/> element is the Base64 encoding of a signed authorization token granting the publisher the right to use a portion of its parent's namespace at this repository. See Section 3.4 for details on the authorization token.

These fields are copies of values that a parent provided to the child in the <parent_response/> message (see Section 3.3.2). The referrer attribute is present to aid lookup of the corresponding certificate by the repository. Note that the repository operator makes the final decision on whether to grant publication service to the prospective publisher. The <referral/> element just conveys a parent's grant of permission to use a portion of that parent's namespace.

```
-----  
<publisher_request  
  publisher_handle="Bob"  
  version="1"  
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">  
<publisher_bpki_ta>  
  R29kIGlzIHJlYWgdW5sZXNzIGRlY2xhcmVkJGluZGVnZXI=  
</publisher_bpki_ta>  
</publisher_request>  
-----
```

3.3.4. <repository_response/>

The <repository_response/> message is a repository's response to a publisher which has previously sent a <publisher_request/> message.

Fields in the <repository_response/> message:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

service_uri: The service_uri attribute contains an HTTP URL that the publisher should contact for publication service ([I-D.ietf-sidr-publication]).

publisher_handle: The publisher_handle attribute is the repository's name for the publisher. This may or may not match the publisher_handle attribute in the publisher's <publisher_request/> message.

sia_base: The sia_base attribute is the rsync:// URI for the base of the publication space allocated to the publisher.

repository_bpki_ta: The <repository_bpki_ta/> element is the repository's BPKI identity, a self-signed X.509 BPKI certificate.

```
-----
<repository_response
  publisher_handle="Alice/Bob-42"
  service_uri="http://alice.example/rpki-publication/Alice/Bob-42"
  sia_base="rsync://alice.example/rpki/Alice/Bob-42/"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <repository_bpki_ta>
    WW9lIGNhbiBoYWNRIGFueXRoaW5nIHlvdSB3YW50IHdpdGggVEVDTyBhbmQgRERU
  </repository_bpki_ta>
</repository_response>
-----
```

3.4. <authorization/>

The <authorization/> element is a separate message which is signed with CMS, then included as the Base64 content of <referral/> elements in other messages.

The eContentType for the signed CMS message is id-ct-xml.

Fields in the <authorization/> element:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

authorized_sia_base: The value of the authorized_sia_base attribute is the rsync:// URI of the base of the namespace which the referrer is delegating.

bpki_ta: The <bpki_ta/> element is the identity of the entity to whom the referrer is delegating the portion of the namespace named in the authorized_sia_base attribute. The identity is represented as a self-signed X.509 BPKI certificate.

```
-----
<authorization
  authorized_sia_base="rsync://alice.example/rpki/Alice/Bob-42/Carol/"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  SSd2ZSBoYWQgZnVuIGJlZm9yZS4gIFRoXMGaXNuJ3QgaXQu
</authorization>
-----
```

3.5. <error/>

The <error/> element is an optional message which can be used in response to any of the core protocol messages described in Section 3.3.

Whether an <error/> element is an appropriate way to signal errors back to the sender of a protocol message depends on details of the implementation which are outside this specification. For example, if this protocol is embedded in a web portal interface which is designed to let a human being upload and download these messages via upload and download forms, a human-readable error message may be more appropriate. On the other hand, a portal intended to be driven by a robotic client might well want to use an <error/> message to signal errors. Similar arguments apply to non-web encapsulations (email, USB stick, ...); the primary factor is likely to be whether the implementation expects the error to be handled by a human being or by a program.

Fields in the <error/> message:

version: The version attribute specifies the protocol version. This note describes protocol version 1.

reason: The reason attribute contains a code indicating what was wrong with the message. This version of the protocol defines the following codes:

syntax-error: Receiver could not parse the offending message.

authentication-failure: Receiver could not authenticate the offending message.

refused: Receiver refused to perform the requested action.

Offending message: The <error/> element contains a verbatim copy of the message to which this error applies.

```
-----
<error
  reason="refused"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <child_request
    child_handle="Carol">
    <child_bpki_ta>
      SSd2ZSBoYWQgZnVuIGJlZm9yZS4gIFRoXMGaXNuJ3QgaXQu
    </child_bpki_ta>
```

```

    </child_request>
  </error>

```

4. Protocol Walk-Through

This section walks through a few simple examples of the protocol in use, and stars our old friends, Alice, Bob, and Carol. In this example, Alice is the root of a RPKI tree, Bob wants to get address and ASN resources from Alice, and Carol wants to get some of those resources in turn from Bob. Alice offers publication service, which is used by all three.

Alice, Bob, and Carol each generates his or her own self-signed BPKI certificate.

Bob constructs a `<child_request/>` message and sends it to Alice:

```

-----
<child_request
  child_handle="Bob"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <child_bpki_ta>
    R29kIGlzIHJlYWwgdW5sZXNzIGRlY2xhcmludGVnZXI=
  </child_bpki_ta>
</child_request>
-----

```

- o Bob's preferred handle is "Bob", so Bob uses that when setting `child_handle`.
- o `<child_bpki_ta/>` is Bob's self-signed BPKI certificate.

Alice replies with a `<parent_response/>` message, but Alice already has 41 other children named Bob, so she calls this one "Bob-42". Alice's provisioning protocol server happens to use a RESTful URL scheme so that it can find the expected validation context for the provisioning protocol CMS message just by looking at the URL, so the service URL she provides to Bob includes both her name and Bob's. Alice offers publication service, so she offers to let Bob use it; Alice doesn't have to do this, she could just omit this and leave Bob to find publication service on his own, but Alice is trying to be helpful to her customer Bob. Bob doesn't have to accept Alice's offer, but may choose to do so.

```

-----
<parent_response
  child_handle="Bob-42"
  parent_handle="Alice"
  service_uri="http://alice.example/rpki-up-down/Alice/Bob-42"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <parent_bpki_ta>
    WW9lIGNhbiBoYWNRIGFueXRoaW5nIHlvdSB3YW50IHdpdGggVEVDTyBhbWQgRERU
  </parent_bpki_ta>
  <offer/>
</parent_response>
-----

```

o <parent_bpki_ta/> is Alice's own self-signed BPKI certificate.

Bob receives Alice's <parent_response/> and extracts the fields Bob's RPKI engine will need to know about (child_handle, parent_handle, service_uri, and <parent_bpki_ta/>). Bob also sees the repository offer, decides to take Alice up on this offer, and constructs a <publisher_request/> message accordingly:

```

-----
<publisher_request
  publisher_handle="Bob"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <publisher_bpki_ta>
    R29kIGlzIHJlYWwgZDw5sZXXNzIGRlY2xhcmludGVnZXI=
  </publisher_bpki_ta>
</publisher_request>
-----

```

Alice receives Bob's request to use Alice's publication service, decides to honor the offer she made, and sends back a <repository_response/> message in response. Alice recognizes Bob as one of her own children, because she's already seen Bob's self-signed BPKI certificate, so she allocates publication space to Bob under her own publication space, so that relying parties who rsync her products will pick up Bob's products automatically without needing an additional fetch operation.

```

-----
<repository_response
  publisher_handle="Alice/Bob-42"
  service_uri="http://alice.example/rpki-publication/Alice/Bob-42"

```

```
sia_base="rsync://alice.example/rpki/Alice/Bob-42/"
version="1"
xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
<repository_bpki_ta>
  WW9lIGNhbiBoYWNRIGFueXRoaW5nIHlvdSB3YW50IHdpdGggVEVDTyBhbmQgRERU
</repository_bpki_ta>
</repository_response>
```

Bob should now have everything he needs to talk to Alice both for provisioning and for publication.

A more interesting case is Bob's child, Carol. Carol wants to get her resources from Bob, and, like Bob, does not particularly want to operate a publication service. Bob doesn't have a publication service of his own to offer, but he can refer Carol to Alice, along with his permission for Carol to use a portion of the namespace that Alice gave him.

Carol's <child_request/> to Bob looks very similar to Bob's earlier request to Alice:

```
<child_request
  child_handle="Carol"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
<child_bpki_ta>
  SSd2ZSBoYWQgZnVuIGJlZm9yZS4gIFRoXMGaXNuJ3QgaXQu
</child_bpki_ta>
</child_request>
```

Bob's <parent_response/> to Carol also looks a lot like Alice's response to Bob, except that Bob includes a <referral/> element instead of an <offer/> element. Carol is an only child, so Bob leaves her name alone:

```
<parent_response
  child_handle="Carol"
  parent_handle="Bob"
  service_uri="http://bob.example/rpki-up-down/Bob/Carol"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
<parent_bpki_ta>
```

```

    R29kIGlziIHJlYWwgdW5sZXNzIGRlY2xhcmVkiGludGVnZXI=
  </parent_bpki_ta>
  <referral
    referrer="Alice/Bob-42">
    R28sIGxlbWlpbmdzLCBnbyE=
  </referral>
</parent_response>

```

Bob's response includes a <referral/> element with a referrer attribute of "Alice/Bob-42", since that's Bob's name to Alice's repository. The Base64-encoded authorization token is an <authorization/> element in a CMS message that can be verified against Bob's self-signed BPKI certificate, using a BPKI EE certificate included in the CMS wrapper. The <authorization/> text is Carol's self-signed BPKI certificate; Bob's signature over this element indicates Bob's permission for Carol to use the indicated portion of Bob's publication space.

```

<authorization
  authorized_sia_base="rsync://alice.example/rpki/Alice/Bob-42/Carol/"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  SSd2ZSBoYWQgZnVuIGJlZm9yZS4gIFRoXMGaXNuJ3QgaXQu
</authorization>

```

Carol, not wanting to have to run a publication service, presents Bob's referral to Alice in the hope that Alice will let Carol use Alice's publication service. So Carol constructs a <publisher_request/> message including the referral information received from Bob, and sends it all to Alice:

```

<publisher_request
  publisher_handle="Carol"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <publisher_bpki_ta>
    SSd2ZSBoYWQgZnVuIGJlZm9yZS4gIFRoXMGaXNuJ3QgaXQu
  </publisher_bpki_ta>
  <referral
    referrer="Alice/Bob-42">
    R28sIGxlbWlpbmdzLCBnbyE=
  </referral>

```

```
</publisher_request>
```

Alice sees the signed authorization token Bob gave to Carol, checks its signature, and unpacks it. When the signature proves valid and the contained BPKI TA matches Carol's, Alice knows that Bob is willing to let Carol use a portion of Bob's namespace. Given this, Alice is willing to provide publication service to Carol in the subtree allocated by Bob for this purpose, so Alice sends back a `<repository_response/>`:

```
<repository_response
  publisher_handle="Alice/Bob-42/Carol"
  service_uri="http://alice.example/rpki-publication/Alice/Bob-42/Carol"
  sia_base="rsync://alice.example/rpki/Alice/Bob-42/Carol/"
  version="1"
  xmlns="http://www.hactrn.net/uris/rpki/rpki-setup/">
  <repository_bpkgi_ta>
    WW9lIGNhbiBoYWNrIGFueXRoaW5nIHlvdSB3YW50IHdpdGggVEVDTyBhbmQgRERU
  </repository_bpkgi_ta>
</repository_response>
```

Once Carol receives this response, Carol should be good to go.

In theory the publication referral mechanism can extend indefinitely (for example, Carol can refer her child Dave to Alice for publication service and it should all work). In practice, this has not yet been implemented, much less tested. In order to keep the protocol relatively simple, we've deliberately ignored perverse cases such as Bob being willing to refer Carol to Alice but not wanting Carol to be allowed to refer Dave to Alice.

5. IANA Considerations

Blah.

6. Security Considerations

As stated in Section 1, the basic function of this protocol is an exchange of public keys to be used as BPKI trust anchors. Integrity and authentication of these exchanges **MUST** be ensured via external mechanisms deliberately left unspecified in this protocol.

7. Acknowledgements

The author would like to thank: Byron Ellacott, George Michaelson, Leif Johansson, Matsuzaki Yoshinobu, Michael Elkins, Randy Bush, Seiichi Kawamura, Tim Bruijnzeels, and anybody else who helped along the way whose name the author has temporarily forgotten.

8. Normative References

- [I-D.ietf-sidr-publication] Weiler, S., Sonalker, A., and R. Austein, "A Publication Protocol for the Resource Public Key Infrastructure (RPKI)", draft-ietf-sidr-publication-03 (work in progress), July 2012.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [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.

Appendix A. RelaxNG Schema

Here is a RelaxNG schema describing the protocol elements.

```
# $Id: rpki-setup.rnc 2408 2013-05-24 13:16:55Z sra $

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

version = "1"

base64 = xsd:base64Binary { maxLength="512000" }
handle = xsd:string { maxLength="255" pattern="[\-_A-Za-z0-9/]*" }
uri = xsd:anyURI { maxLength="4096" }
any = element * { attribute * { text }*, ( any | text )* }

authorization_token = base64
bpki_ta = base64

start | = child_request
start | = parent_response
start | = publisher_request
```

```
start | = repository_response
start | = authorization
start | = error

child_request =
element child_request {
  attribute version { version },
  attribute child_handle { handle },
  element child_bpki_ta { bpki_ta }
}

parent_response =
element parent_response {
  attribute version { version },
  attribute service_uri { uri },
  attribute child_handle { handle },
  attribute parent_handle { handle },
  element parent_bpki_ta { bpki_ta },
  element offer { empty }?,
  element referral {
    attribute referrer { handle },
    attribute contact_uri { uri }?,
    authorization_token
  }*
}

publisher_request =
element publisher_request {
  attribute version { version },
  attribute publisher_handle { handle },
  element publisher_bpki_ta { bpki_ta },
  element referral {
    attribute referrer { handle },
    authorization_token
  }*
}

repository_response =
element repository_response {
  attribute version { version },
  attribute service_uri { uri },
  attribute publisher_handle { handle },
  attribute sia_base { uri },
  element repository_bpki_ta { bpki_ta }
}

authorization =
element authorization {
```

```
    attribute version { version },
    attribute authorized_sia_base { uri },
    bpki_ta
  }

  error =
  element error {
    attribute version { version },
    attribute reason {
      "syntax-error" |
      "authentication-failure" |
      "refused"
    },
    any?
  }
```

Author's Address

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2014

G. Huston
G. Michaelson
APNIC
July 8, 2013

RPKI Validation Reconsidered
draft-huston-rpki-validation-00.txt

Abstract

This document reviews the certificate validation procedure specified in RFC6487 and highlights aspects of operational management of certificates in the RPKI in response to the movement of resources across registries, and the associated actions of Certification Authorities to maintain certification of resources during this movement. The document describes an alternative validation procedure that reduces the operational impact of certificate management during resource movement.

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 9, 2014.

Copyright Notice

Copyright (c) 2013 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.

Table of Contents

1. Introduction	3
2. Operational Considerations	4
3. A Specific Resource RPKI Certificate Validation Process . . .	6
3.1. Resource Transfers and Specific Resource Certificate Validation	7
3.2. A Specification of Specific Resource Validation	8
4. Local Repository Cache Maintenance	10
5. Security Considerations	10
6. IANA Considerations	11
7. Acknowledgements	11
8. References	11
8.1. Normative References	11
8.2. Informative References	11
Authors' Addresses	11

1. Introduction

This document reviews the certificate validation procedure specified in [RFC6487] and highlights aspects of operational management of certificates in the RPKI in response to the movement of resources across registries, and the associated actions of Certification Authorities to maintain certification of resources during this movement. The document describes an alternative validation procedure that reduces the operational impact of certificate management during resource movement.

As currently defined in section 7.2 of [RFC6487], validation of PKIX certificates that conform to the RPKI profile relies on the use of a path validation process where each certificate in the validation path is required to meet the certificate validation criteria. This can be considered to be a recursive validation process where, in the context of an ordered sequence of certificates, as defined by common Issuer and Subject Name pairs, a certificate is defined as valid if it satisfies basic validation criteria relating to the syntactic correctness, currency of validity dates and similar properties of the certificate itself, as described in [RFC5280], and also that it satisfies certain additional criteria with respect to the previous certificate in the sequence, and that this previous certificate is itself a valid certificate using the same criteria. This definition applies recursively to all certificates in the sequence apart from the initial sequence element, which is required to be a Trust Anchor.

For RPKI certificates, the additional criteria relating to the previous certificate in this sequence is that the certificate's number resource set, as defined in [RFC3779], is "encompassed" by the number resource set contained in the previous certificate.

Because [RFC6487] validation demands that all resources in a certificate be valid under the parent (and recursively, to the root), a digitally signed attestation, such as a Route Origin Authorization (ROA) object [RFC6482], which refers only to a subset of RFC3779-specified resources from that certificate chain can be concluded to be invalid, but not by virtue of the relationship between the RFC3779 extensions of the certificates on the putative certificate validation path and the resources in the ROA, but by other resources described in these certificates where the "encompassing" relationship of the resources does not hold. Any such invalidity along the certificate validation path can cause this outcome, not just at the immediate parent of the end entity certificate that attests to the key used to sign the ROA.

For example, in the certificate sequence:

Certificate 1:

Issuer A, Subject B, Resources 192.0.2.0/24, AS64496-AS64500

Certificate 2:

Issuer B, Subject C, Resources 192.0.2.0/24/24, AS64496-AS64511

Certificate 3:

Issuer C, Subject D, Resources 192.0.2.0/24

Certificate 3 is considered to be an invalid certificate, because the resources in Certificate 2 are not encompassed by the resources in Certificate 1, by virtue of certificate 2 holding the resources of the range AS64501 - AS64511 in this RFC3779 resource extension. Obviously, these Autonomous Systems numbers are not related to the IPv4 resources contained in Certificate 3.

2. Operational Considerations

The operational consideration described here relates to the situation where a registry withdraws a resource from the current holder, and the resource is transferred to another registry, to be registered to a new holder in that registry. The reason why this is a consideration in operational deployments of the RPKI lies in the movement of the "home" registry of number resources during cases of mergers, acquisitions, business re-alignments, and resource transfers and the desire to ensure that during this movement all other resources can continue to be validated.

If the original registry's certification actions are simply to issue a new certificate for the current holder with a reduced resource set, and to revoke the original certificate, then there is a distinct possibility of encountering the situation illustrated by the example in the previous section. This is a result of an operational process for certificate issuance by the parent CA being de-coupled from the certificate operations of child CA.

This de-coupled operation of CAs introduces a risk of unintended third party damage: since a CA certificate can refer to holdings which relate to two or more unrelated subordinate certificates, if this CA certificate becomes invalid due to the reduction in the resources allocated to this CA relating to one subordinate resource set, all other subordinate certificates are invalid until the CA certificate is reissued with a reduced resource set.

In the above example, all subordinate certificates issued by CA C are invalid until CA B issues a new certificate for CA C with a reduced resource set.

At the lower levels of the RPKI hierarchy the resource sets affected by such movements of resources may not encompass significantly large pools of resources. However, as one ascends through this hierarchy towards the apex, the larger the resource set that is going to be affected by a period of invalidity by virtue of such uncoordinated certificate management actions. In the case of a Regional Internet Registry (RIR) or National Internet Registry (NIR), the potential risk arising from uncoordinated certification actions relating to a transfer of resources is that the entire set of subordinate certificates that refer to resources administered by the RIR or the NIR cannot be validated during this period.

Avoiding such situations requires that CA's adhere to a very specific ordering of certificate issuance. In this framework, the common registry CA that describes (directly or indirectly) the resources being shifted from one registry to the other, and also contains in subordinate certificates (direct or indirect) the certificates for both registries who are parties to the resource transfer has to coordinate a specific sequence of actions.

This common registry CA has to first issue a new certificate towards the "receiving" registry that adds to the RFC3779 extension resource set the specific resource being transferred into this receiving registry. The common registry CA then has to wait until all registries in the subordinate certificate chain to the receiving registry have also performed a similar issuance of new certificates, and in each case a registry must await the issuance of the immediate superior certificate with the augmented resource set before it, in turn, can issue its own augmented certificate to its subordinate CA. This is a "top down" issuance sequence."

It is possible for the common registry to issue a certificate to the "sending" registry with the reduced resource set at any time, but it should not revoke the previously issued certificate, nor overwrite this previously issued certificate in its repository publication point without specific coordination. Only when the common registry is assured that the top down certificate issuance process to the receiving registry CA chain has been completed can the common registry commence the revocation of the original certificate for the sending registry. However, it should not do so until it is assured that the immediate subordinate registry CA in the path to the sending registry has issued a certificate with a reduced resource set, and so on. This implies that on the sending side the certificate issuance and revocation is a "bottom up" process.

If this process is not carefully followed, then the risk is that some or all of the subordinate certificates of this common registry CA will be unable to be validated until the entire process of

certificate issuance and revocation has been completed. While this sequenced process is intended to preserve validity of certificates in the RPKI, it is a complex and operationally cumbersome process.

The underlying consideration here is that the operational coordination of these certificate issuance and revocation actions to effect a smooth resource transfer across registries is mandated by the nature of the certificate validation process described in [RFC6487].

3. A Specific Resource RPKI Certificate Validation Process

The question considered here is: Is there an alternate definition of RPKI certificate validity that could remove the requirement for such careful orchestration of certification actions across the RPKI to support resource transfers?

The general definition of certificate validity as defined in [RFC5280] assumes a validation question relating to the relying party's (RP's) level of trust in a subject's signed material, given knowledge of a subject's name, the subject's public key, the RP's chosen trust anchor(s) and an overall PKI to define the domain of discourse.

The validation question assumed by the [RFC6487] RPKI certificate validation process relates to a RP's level of trust in the combination of some signed material, a certificate that attests to the public key used to sign this material and the set of all number resources that have been assigned or allocated to the subject of the certificate, given knowledge of the certificate, the RP's chosen trust anchor(s), the RPKI, and the application of the same test applied to the superior certificate in the RPKI hierarchy, and so on to a Trust Anchor.

There is a alternative certificate validation procedure that starts with an attestation containing the subject's signed material and an explicit enumeration of a set of number resources. The associated validation question relates to whether a RPKI validation process can attest to the validity of a subject's signed attestation relating to a particular set of number resources, rather than a signed attestation relating to all number resources held by this subject. We will term this alternate certificate validation process "specific resource" validation.

If the certificate validation procedure is specifically restricted to a question of ascertaining the validity of a particular set of number resources in the context of the RPKI, the RPKI validation procedure

need not be as strict as a recursive "encompassing" condition for the resources contained in each pair of certificates in the validation path. It would be sufficient in the context of this "specific resource" validation procedure to require only that each certificate in the validation path has a number resource extension that "encompasses" the specific resources described in the original validation question. Rather than a validation test for all possible questions, this is a specific validation question in the context of specific resources.

This validation question can be informally described as: Given a certificate and a given resource set, is there an Issuer-Subject ordered sequence of certificates from a Trust Anchor to the certificate being validated, where each certificate on this sequence is well-formed, not revoked by a valid CRL, where the certificate's lifetimes are valid, and where the RFC3779 resource extension in the certificate encompass the given resource set?

In the example from Section 1, using a this alternate certificate validation process, a validation question of certificate 3 and the resource 10.0.1.0/24, the validation outcome would be positive, in that certificates 1, 2 and 3 all encompass the specific resource 10.0.1.0/24, assuming that the certificates are valid in all other respects.

3.1. Resource Transfers and Specific Resource Certificate Validation

When considering the transfer of resources across registries, and the associated certification actions, then if the validation process was one of "specific resource" validation, then there is no requirement for synchronized orchestration of the process of certificate issuance and revocation by the CAs involved in this transfer in order to preserve the validity of resources described in these certificates.

Along the chain of the "sending" registry CA hierarchy each registry CA can issue a certificate with a reduced resource set that removes the resource being transferred, and revoke the previously issued certificate without regard to the specific timing of similar actions by either it's superior or its subordinate registry CA.

Similarly, in the "receiving" registry hierarchy each CA can issue a certificate with an augmented resource set that includes the resource being transferred without particular regard to the timing of similar actions by the other superior or subordinate registry CAs.

Validation questions relating to the migrating resource made against certificates on the "sending registry" will return an invalid outcome as soon as any registry CA in this chain has performed revocation of

the original certificate. Validation questions relating to the migrating resource made against certificates on the "receiving registry" will return an valid outcome only when all the registries in this chain have performed certificate re-issuance and included the resource in the new certificate.

Critically, at all times validation questions relating to any other resource using the "specific resource" validation approach will return the same outcomes throughout this issuance and revocation process. This "specific resource" validation process engenders a more robust outcome in RPKI certificate management. Validation questions relating to resources which are not being transferred from one registry to another cannot be compromised by any failure to adhere to a strict process of issuance and revocation relating to the certification of the resources being transferred.

3.2. A Specification of Specific Resource Validation

The following is a amended specification of certificate validation as described in [RFC6487] that describes the proposed "specific resource" certificate validation process.

Validation of signed resource data using a target resource certificate and a specific set of number resources consists of verifying that the digital signature of the signed resource data is valid, using the public key of the target resource certificate, and also validating the resource certificate in the context of the RPKI, using the path validation process. This path validation process verifies, among other things, that a prospective certification path (a sequence of n certificates) satisfies the following conditions:

1. for all 'x' in $\{1, \dots, n-1\}$, the Subject of certificate 'x' is the Issuer of certificate ('x' + 1);
2. certificate '1' is issued by a trust anchor;
3. certificate 'n' is the certificate to be validated; and
4. for all 'x' in $\{1, \dots, n\}$, certificate 'x' is valid.

Certificate validation entails verifying that all of the following conditions hold, in addition to the Certification Path Validation

criteria specified in Section 6 of [RFC5280]:

1. The certificate can be verified using the Issuer's public key and the signature algorithm
2. The current time lies within the certificate's Validity From and To values.
3. The certificate contains all fields that MUST be present, as defined by this specification, and contains values for selected fields that are defined as allowable values by this specification.
4. No field, or field value, that this specification defines as MUST NOT be present is used in the certificate.
5. The Issuer has not revoked the certificate. A revoked certificate is identified by the certificate's serial number being listed on the Issuer's current CRL, as identified by the CRLDP of the certificate, the CRL is itself valid, and the public key used to verify the signature on the CRL is the same public key used to verify the certificate itself.
6. The resource extension data contained in this certificate "encompasses" the entirety of the resources in the specific resource set.
7. The Certification Path originates with a certificate issued by a trust anchor, and there exists a signing chain across the Certification Path where the Subject of Certificate 'x' in the Certification Path matches the Issuer in Certificate 'x + 1' in the Certification Path, and the public key in Certificate 'x' can verify the signature value in Certificate 'x+1'.

A certificate validation algorithm MAY perform these tests in any chosen order.

4. Local Repository Cache Maintenance

This change in the validation process would have some impact on the operation of a local cache of validated RPKI certificates. Given that the validation process requires the specification of a specific resource set, it would appear that it would not be possible to validate an RPKI certificate without also specifying a specific resource set.

However, using a top-down validation process, and an additional local data structure associated with each locally held validated RPKI certificate, it is possible to maintain a local cache of validated certificates, and the set of valid and invalid resources for each certificate.

The additional data structures are the certificate's validated and invalidated resource set. These sets are defined as follows:

- o If the certificate is used as a Trust Anchor, then the local validated resource set is copied from the certificate's RFC3779 resource set. There is no invalid resource set.
- o Otherwise, the certificate's local validated resource set is defined as the intersection of this certificate's RFC3779 resource set and the parent certificate's local validated resource set. The certificate's invalid resource set is the difference between this set and the certificate's RFC3779 resource set.

If the certificate's validated resource set is empty then the certificate is not valid.

If the invalid resource set is not empty, then any resources that are contained in this invalid resource set cannot be valid by virtue of this certificate.

In all operations on the local repository cache, local applications should use the certificate's local validated resource set in place of the resources described in the certificate's RFC3779 extension.

The invalid resource set can be used as a diagnostic aide in local cache management.

5. Security Considerations

The Security Considerations of [RFC6487] apply to the validation procedure described here.

6. IANA Considerations

No updates to the registries are suggested by this document.

7. Acknowledgements

TBA.

8. References

8.1. Normative References

- [RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, June 2004.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", RFC 6487, February 2012.

8.2. Informative References

- [RFC6482] Lepinski, M., Kent, S., and D. Kong, "A Profile for Route Origin Authorizations (ROAs)", RFC 6482, February 2012.

Authors' Addresses

Geoff Huston
Asia Pacific Network Information Centre (APNIC)
6 Cordelia St
South Brisbane, QLD 4101
Australia

Phone: +61 7 3858 3100
Email: gih@apnic.net

George Michaelson
Asia Pacific Network Information Centre (APNIC)
6 Cordelia St
South Brisbane, QLD 4101
Australia

Phone: +61 7 3858 3100
Email: ggm@apnic.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2014

S. Weiler
SPARTA, Inc.
A. Sonalker
Battelle Memorial Institute
R. Austein
Dragon Research Labs
October 20, 2013

A Publication Protocol for the Resource Public Key Infrastructure (RPKI)
draft-ietf-sidr-publication-04

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 April 23, 2014.

Copyright Notice

Copyright (c) 2013 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	3
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. Examples	11
4.1. <config/> Set Query	11
4.2. <config/> Set Reply	12
4.3. <config/> Get Query	12
4.4. <config/> Get Reply	13
4.5. <client/> Create Query	13
4.6. <client/> Create Reply	14
4.7. <client/> Set Query	14
4.8. <client/> Set Reply	15
4.9. <client/> Get Query	15
4.10. <client/> Get Reply	15
4.11. <client/> List Query	16
4.12. <client/> List Reply	16
4.13. <client/> Destroy Query	17
4.14. <client/> Destroy Reply	17
4.15. <publish/> Query	17
4.16. <publish/> Reply	18
4.17. <withdraw/> Query	18
4.18. <withdraw/> Reply	19
4.19. <report_error/> With Text	19
4.20. <report_error/> Without Text	19
5. Operational Considerations	19
6. IANA Considerations	20
7. Security Considerations	21
8. References	21
8.1. Normative References	21
8.2. Informative References	22
Authors' Addresses	22

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. 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 [RFC6492].

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 `bpmi_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 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. The "create" and "set" actions have an optional flag to clear CMS-timestamp-based replay protection, to allow recovery from misconfigured clocks.

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 publication sub-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.

```
# $Id: rpki-publication.rnc 2601 2013-10-18 19:21:28Z sra $
# RelaxNG schema for RPKI publication protocol.

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

# This is version 2 of the protocol.

version = "2"
```

```
# 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 a query.

query_elt | = config_query
query_elt | = client_query
query_elt | = publish_query
query_elt | = withdraw_query

# PDUs allowed in a reply.

reply_elt | = config_reply
reply_elt | = client_reply
reply_elt | = publish_reply
reply_elt | = withdraw_reply
reply_elt | = 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).

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

# Error codes.

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

# <config/> element (use restricted to repository operator)
```



```
config_payload = (element bpki_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 bpki_cert { base64 }?,
  element bpki_glue { base64 }?
)

client_clear_replay = (
  attribute clear_replay_protection { "yes" }?
)

client_query |= element client {
  attribute action { "create" },
  tag?,
  client_handle,
  client_clear_replay,
  client_payload
}

client_reply |= element client {
  attribute action { "create" },
```

```
    tag?,
    client_handle
  }

  client_query |= element client {
    attribute action { "set" },
    tag?,
    client_handle,
    client_clear_replay,
    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

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

4. Examples

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

4.1. <config/> Set Query

```
<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <config
    action="set">
    <bpki_crl>
      MIIBezBIAgEBMA0GCSqGSIb3DQEBCwUAMCMxITAfBgNVBAMTGFRlc3QgQ2Vy
      dGlmaWNhdGUgcHViZCBURcNMDgwNjAyMjE0OTQlWhcNMDgwNzAyMjE0OTQl
      WqAOMAwwCgYDVROUBAMCAQEwDQYJKoZIhvcNAQELBQADggEBAFWCWgBl4ljV
      qX/Cho+RpqYtvmKMnjPVflMXUB7i28RGP4DAq4l7deDU7Q82xEJyE4TXMWDW
      AV6UG6uUGum0VHWOCj9ohqyiZUGfOsKg2hbwkETm8sAENOsilyNdyKGk6jZl
      6aF5fubxQqZa1pdGCSac1/ZYC5sLLhEz3kmz+B9z9mXFVc5TgAh4dN3Gy5ft
      F8zZAFpDGnS4biCnRVqhGv6R0Lh/5xmii+ZU6kNDhbeMsjJg+ZomtN+wMeHS
      Ibjiy0WuuaZ3k2xSh0C94anrHBZAvvCRhbazjR0Ef5OMZ5lcllw3u08IHuoi
      sHKkehy4Y0GySdj98fV+OuirTH9vt/M=
    </bpki_crl>
  </config>
</msg>
```

4.2. <config/> Set Reply

```
<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <config
    action="set"/>
</msg>
```

4.3. <config/> Get Query

```
<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <config
    action="get"/>
</msg>
```

4.4. <config/> Get Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <config
    action="get">
    <bpki_crl>
      MIIBezBlAgEBMA0GCSqGSIb3DQEBCwUAMCMxITAfBgNVBAMTGFRlc3QgQ2Vy
      dGhmaWNhdGUgcHViZCBURcNMDgwNjAyMjE0OTQ1WhcNMDgwNzAyMjE0OTQ1
      WqAOMAwwCgYDVROUBAMCAQEwDQYJKoZIhvcNAQELBQADggEBAFWCWgBl4ljV
      qX/CHo+RpgYtvmKMnjPVf1MXUB7i28RGP4DAq4l7deDU7Q82xEJyE4TXMWDW
      AV6UG6uUGum0VHWOCj9ohqyiZUGF0sKg2hbwkETm8sAENOsilyNdyKGk6jZ1
      6aF5fubxQqZa1pdGCSac1/ZYC5sLLhEz3kmz+B9z9mXFVc5TgAh4dN3Gy5ft
      F8zZAFpDGnS4biCnRVqhGv6R0Lh/5xmii+ZU6kNDhbeMssjJg+ZomtN+wMeHS
      Ibjiy0Wuuaz3k2xSh0C94anrHBZAvvCRhbazjR0Ef5OMZ5lcllw3u08IHuoi
      sHKkehy4Y0GySdj98fV+OuirTH9vt/M=
    </bpki_crl>
  </config>
</msg>

```

4.5. <client/> Create Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="create"
    base_uri="rsync://wombat.invalid/"
    client_handle="3">
    <bpki_cert>
      MIIDGzCCAgOgAwIBAgIJAKi+/+wUhQ1xMA0GCSqGSIb3DQEBBQUAMCQxIjAg
      BgNVBAMTGVRlc3QgQ2VydGhmaWNhdGUgQm9iIFJvb3QwHhcNMDcwODAxMTk1
      MzEwWWhcNMDcwODMxMTk1MzEwWjAkMSIwIAYDVQQDExlUZXXN0IENlcnRpZmlj
      YXR1IEJvYiBSb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEa
      rKYUtJaM5PH5917SG2ACc7iBYdQ02HYyu8Gb6i9Q2Gxc3cWEX7RTBvgOL79p
      Wf3GI dnoupzMnoZVtY3GUx2G/0WkmLui2TCeDhcfXdQ4rcp8J3V/6ESj+yuE
      PPOG8UN17mUKKgujrch6ZvgCDO9AyOK/uXu+ABQXTPsn2pVe2EVh3V004ShL
      i8GKgVdqb/rW/6GTg0Xb/zLT6WWMuT++6sXTlztJdQYkRamJvKfQDUlnaC8m
      AkGf79Tba0xyBGAUII0GfREY6t4/+NAP2Yyb3xNlBqcJoTov0JfNKHZcCZeP
      r79j7LK/hkZxxip+Na9xDpE+oQRV+DRukCRJdiqg+wIDAQABo1AwTjAMBgNV
      HRMEBTADAQH/MB0GA1UdDgQWBBDTDesXJe6pJaqD4ULlB7+GMDBlimTafBgNV
      HSMEGDAWgBTDesXJe6pJaqD4ULlB7+GMDBlimTANBgkqhkiG9w0BAQUFAAOCA
      QEAWWkNcW6S1tKKqtzJsdfhjJiAAPQmOXJskv0ta/8f6Acgcum1YieNdtT0
      n96P7CUHOWP8QBb9lJzeewR7b6WJLwblOffs3wNq3kk75pJe89r4XY39EZHH
    </bpki_cert>
  </client>
</msg>

```

```

        MW+Dv0PhIKu2CgD4LeyHlFVTQkF/QObGEmkn+s+HTsuzdl12VLwcPlSmsqep
        6LAlFj62qqaIJzNeQ9NVkBqtkygnYlBOKaBTHfQTux3jYNpEo8JJB5e/WFdH
        YyMNRG2xMOtIC7T4+IOHgT8PgrNhaeDg9ctewj0X8Qi9nI9nXeinicLX8vj6
        hdEq3ORv7RZMJNYqv1HQ3wUE2B7fCPFv7EUwzaCds1kgRQ==
    </bpki_cert>
</client>
</msg>

```

4.6. <client/> Create Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="create"
    client_handle="3"/>
</msg>

```

4.7. <client/> Set Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="set"
    client_handle="3">
    <bpki_cert>
      MIIDGzCCAgOgAwIBAgIJAKi+/+wUhQ1xMA0GCSqGSIb3DQEBBQUAMCQxIjAg
      BgNVBAMTGVRlc3QgQ2VydgG1maWNhdGUgQm9iIFJvb3QwHhcNMDCwODAxMTk1
      MzEwWhcNMDCwODMxMTk1MzEwWjAKMSIwIAYDVQQDExlUZjY3Y3QwHhcNMDCw
      YXRlIEJvYiBSb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
      rKYUtJam5PH5917SG2ACC7iBYdQO2HYyu8Gb6i9Q2Gxc3cWEX7RTBvgOL79p
      Wf3GI dnoupzMnoZVtY3GUx2G/0WkmLui2TCeDhcfXdQ4rcp8J3V/6ESj+yuE
      PPOG8UN17mUKKgujrch6ZvgCDO9AyOK/uXu+ABQXTPsn2pVe2EVh3V004ShL
      i8GKgVdqb/rW/6GTg0Xb/zLT6WWMuT++6sXTlztJdQYkRamJvKfQDUlnaC8m
      AkGf79Tba0xyBGAUII0GfREY6t4/+NAP2Yyb3xNlBqcJoTov0JfNKHZcCZeP
      r79j7LK/hkZxxip+Na9xDpE+oQRV+DRukCRJdiqg+wIDAQABolAwTjAMBgNV
      HRMEBTADAQH/MB0GA1UdDgQWBBDTDesXJe6pJAQD4ULlB7+GMDBlimTafBgNV
      HSMEGDAWgBTDesXJe6pJAQD4ULlB7+GMDBlimTANBgkqhkiG9w0BAQUFAAOCA
      AQEAWWkNcW6S1tKKgtzJsdfhjJiAAPQmOXJskv0ta/8f6Acgcum1YieNdtT0
      n96P7CUHOWP8QBb91JzeewR7b6WJLwb1Of fs3wNq3kk75pJe89r4XY39EZHH
      MW+Dv0PhIKu2CgD4LeyHlFVTQkF/QObGEmkn+s+HTsuzdl12VLwcPlSmsqep
      6LAlFj62qqaIJzNeQ9NVkBqtkygnYlBOKaBTHfQTux3jYNpEo8JJB5e/WFdH
      YyMNRG2xMOtIC7T4+IOHgT8PgrNhaeDg9ctewj0X8Qi9nI9nXeinicLX8vj6
    </bpki_cert>
  </client>
</msg>

```

```

        hdEq3ORv7RZMJNYqvlHQ3wUE2B7fCPFv7EUwzaCds1kgRQ==
      </bpki_cert>
    </client>
  </msg>

```

4.8. <client/> Set Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="set"
    client_handle="3"/>
  </client>
</msg>

```

4.9. <client/> Get Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="get"
    client_handle="3"/>
  </client>
</msg>

```

4.10. <client/> Get Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="get"
    base_uri="rsync://wombat.invalid/"
    client_handle="3">
  <bpki_cert>
    MIIDGzCCAgOgAwIBAgIJAKi+/+wUhQ1xMA0GCSqGSIb3DQEBBQUAMCQxIjAg
    BgNVBAMTGVRlc3QgQ2VydGlmaWNhdGUGb3Qm9iIFJvb3QwHhcNMDcwODAxMTk1
    MzEwWhcNMDcwODMxMTk1MzEwWjAKMSIwIAYDVQQDExlUZXXN0IEN1cnRpZmlj
    YXR1IEJvYiBSb290MIIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
    rKYUtJaM5PH5917SG2ACc7iBYdQO2HYyu8Gb6i9Q2Gxc3cWEX7RTBvgOL79p
    Wf3GIIdnoupzMnoZVtY3GUx2G/0WkmLui2TCeDhcfXdQ4rcp8J3V/6ESj+yuE
    PPOG8UN17mUKKgUjrch6ZvgCDO9AyOK/uXu+ABQXTPsn2pVe2EVh3V004ShL
  </bpki_cert>
  </client>
</msg>

```

```

i8GKgVdqb/rW/6GTg0Xb/zLT6WWMuT++6sXTlztJdQYkRamJvKfQDUlnaC8m
AkGf79Tba0xyBGAUII0GfREY6t4/+NAP2Yyb3xNlBqcJoTov0JfNKHZcCZeP
r79j7LK/hkZxxip+Na9xDpE+oQRV+DRukCRJdiqg+wIDAQABolAwTjAMBgNV
HRMEBTADAQH/MB0GA1UdDgQWBBDDESXJe6pjAQD4ULlB7+GMDBlimTafBgNV
HSMEGDAWgBTDESXJe6pjAQD4ULlB7+GMDBlimTANBgkqhkiG9w0BAQUFAAOCA
QEAWWkNcW6SltKKqtzJsdfhjJiAAPQmOXJskv0ta/8f6Acgcum1YieNdtT0
n96P7CUHOWP8QBb9lJzeewR7b6WJLwb1OfFs3wNq3kk75pJe89r4XY39EZHh
MW+Dv0PhIKu2CgD4LeyHlFVTQkF/QObGEmkn+s+HTsuzd1l2VLwcPlSmsqep
6LAlFj62qqaIJzNeQ9NVkBqtkygnYlBokaBTHfQTux3jYNpEo8JJB5e/WFdH
YyMNRg2xMOtIC7T4+IOHgT8PgrNhaeDg9ctewj0X8Qi9nI9nXeinicLX8vj6
hdEq3ORv7RZMJNYqv1HQ3wUE2B7fCPFv7EUwzaCds1kgRQ==
</bpki_cert>
</client>
</msg>

```

4.11. <client/> List Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="list"/>
  </client>
</msg>

```

4.12. <client/> List Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="list"
    client_handle="3">
  <bpki_cert>
MIIDGzCCAgOgAwIBAgIJAKi+/+wUhQ1xMA0GCSqGSIb3DQEBBQUAMCQxIjAg
BgNVBAMTGVRlc3QgQ2VydGlmaWNhdGUgQm9iIFJvb3QwHhcNMDcwODAxMTk1
MzEwWWhcNMDcwODMxMTk1MzEwWjAKMSIwIAYDVQQDExlUZXR0IENlcnRpZmlj
YXR1IEJvYiBSb290MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
rKYUtJam5PH5917SG2ACc7iBYdQO2HYyu8Gb6i9Q2Gxc3cWEX7RTBvgOL79p
Wf3GIIdnoupzMnoZVtY3GUx2G/0WkmLui2TCeDhcfXdQ4rcp8J3V/6ESj+yuE
PPOG8UN17mUKKgujrCh6ZvgCDO9AyOK/uXu+ABQXTPsn2pVe2EVh3V004ShL
i8GKgVdqb/rW/6GTg0Xb/zLT6WWMuT++6sXTlztJdQYkRamJvKfQDUlnaC8m
AkGf79Tba0xyBGAUII0GfREY6t4/+NAP2Yyb3xNlBqcJoTov0JfNKHZcCZeP
r79j7LK/hkZxxip+Na9xDpE+oQRV+DRukCRJdiqg+wIDAQABolAwTjAMBgNV
HRMEBTADAQH/MB0GA1UdDgQWBBDDESXJe6pjAQD4ULlB7+GMDBlimTafBgNV

```



```

    HSMEGDAWgBTDEsXJe6pjAQD4ULlB7+GMDBlimTANBgkqhkiG9w0BAQUFAAOC
    AQEAWWkNcW6S1tKKqtzJsdfhjJiAAPQmOXJskv0ta/8f6Acgcum1YieNdtT0
    n96P7CUHOWP8QBb9lJzeewR7b6WJLwb1OfFs3wNq3kk75pJe89r4XY39EZHh
    MW+Dv0PhIKu2CgD4LeyHlFVTQkF/QObGEmkn+s+HTsuzdl12VLwcPlSmsqep
    6LAlFj62qqaIJzNeQ9NVkBqtkygnYlBOKaBTHfQTux3jYNpEo8JJB5e/WFdH
    YyMNRG2xMOtIC7T4+IOHgT8PgrNhaeDg9ctewj0X8Qi9nI9nXeinicLX8vj6
    hdEq3ORv7RZMJNYqv1HQ3wUE2B7fCPFv7EUwzaCds1kgRQ==
    </bpki_cert>
  </client>
</msg>

```

4.13. <client/> Destroy Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="destroy"
    client_handle="3"/>
</msg>

```

4.14. <client/> Destroy Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <client
    action="destroy"
    client_handle="3"/>
</msg>

```

4.15. <publish/> Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <publish
    uri="rsync://wombat.invalid/Alice/blCrcCp9ltyPDNzYKPfxc.cer">
    MIIe+jCCA+KgAwIBAgIBDTANBgkqhkiG9w0BAQsFADAzMTEwLWYDVQQDEyHE
    RjRBODAxN0U2NkE5RTkxNzJFNDYxMkQ4Q0Y0QzgzRjIzOERFMkEzMB4XDTA4
    MDUyMjE4MDUxMl0XDTA4MDUyNDE3NTQ1Ml0wMzExMC8GA1UEAxMoOEZCODIx
    OEYwNkU1MEFCNzAyQTdEOTZEQzhGMENEQ0Q4MjhGN0YxNzCCASIwDQYJKoZI

```

```

hvcNAQEBBQADggEPADCCAQoCggEBAMEziKp0k5nP7v6SZoNsXIMQYRgNtC6F
r/9Xm/1yQHomiPqHUK47rHhGojYiK5AhkrwoYhkH4UjJl2iwl1DYczXuaBU3
F5qrKlZ4aZnjIxdlP7+hktVpeApL6yuJTUAYeC3UIxnLDVdD6phydZ/FOQ1u
ffiNDjzteCCvoyOUatqt8WB+oND6LToHp028glYUYLHG6mur0dPdcHOVXLsm
UDuZlHDzlnDuYvIVKjB/MpH9aW9XeaQ6ZFIlZVPwuuvI2brR+ThH7Gv27GL/
o8qFdC300VQfoTZ+rKPGDE8K1cI906BL4kiwx9z0oiDcE96QCz+B0vsjc9mG
aA1jgAxlXWsCAwEAAaOCAhcgggITMB0GA1UdDgQWBBSpuCGPBuUKtwKn2W3I
8M3Ngo9/FzAfBgNVHSMEGDAWgBTfSoAX5mqekXLkYS2M9Mg/I43iozBVBgNV
HR8ETjBMMEqgSKBGhkRyc3luYzovL2xvY2FsaG9zdDo0NDAwL3Rlc3RiZWQv
UklSLzEvMzBxQUYtWnFucEZ5NUdFdGpQVElQeU9ONHFNLMNybDBFBggrBgEF
BQcBAQQ5MDcwNQYIKwYBBQUHMAKGKXJzeW5jOi8vbG9jYXRob3N0OjQ0MDAv
dGVzdGJlZC9XT0lCQVQuY2VyMBGAlUdIAEB/wQOMAwWCgYIKwYBBQUHdGJw
DwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYwgZsGCCsGAQUFBwEL
BIGOMIGLMDQGCCsGAQUFBzAFhihyc3luYzovL2xvY2FsaG9zdDo0NDAwL3Rl
c3RiZWQvUklSLlIwLzEvMFMGCCsGAQUFBzAKhkdyd3luYzovL2xvY2FsaG9z
dDo0NDAwL3Rlc3RiZWQvUklSLlIwLzEvaJdnaGp3YmxDcmNDcDlscDhlQRE56
WUTqZnhjLmluZjAaBggrBgEFBQcBCAEB/wQLMAmGbzAFagMA/BUwPgYIKwYB
BQUHAQCBaf8ELzAtMCsEAgABMCUDAwAKAzAOAwUAAACAMFACAAAIawDgMF
AsAAAIwDBQDAAAJkMA0GCSqGSIb3DQEBCwUAA4IBAQCehuH7jtI2PJY6+zwv
306vmCuXhtu9Lr2mmRw2ZErB8EMcb5xypMrNqMoKeu14K2x4a4RPJkK4yATh
M8lFPNRsU5mM0acIRnAPTxxjHvPME7PHN2w2nGLASRsZmaa+b8A7SSOxVcFUR
azENztppsolHeTpm0cpLitK7mNpudUg1JGuFo94VLf1MnE2EqARGlvTsNhel
/SM/UvOArCCOBvf0Gz7kSuupDSZ7qx+LiDmtEsLdbGNQBiYPbLrDk41PHrxd
x28qIj7ejZkRzNFw/3pi8/XK281h8zeHoFVu6ghRPy5dbOA4akX/KG6b8XIX
0iwPYdLiDbdWFbtTdPcXBauY
</publish>
</msg>

```

4.16. <publish/> Reply

```

<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <publish
    uri="rsync://wombat.invalid/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
  </publish>
</msg>

```

4.17. <withdraw/> Query

```

<msg
  type="query"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <withdraw
    uri="rsync://wombat.invalid/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
  </withdraw>
</msg>

```

```
</msg>
```

4.18. <withdraw/> Reply

```
<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <withdraw
    uri="rsync://wombat.invalid/Alice/blCrcCp9ltyPDNzYKPfxc.cer"/>
  </msg>
```

4.19. <report_error/> With Text

```
<msg
  type="reply"
  version="2"
  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>
```

4.20. <report_error/> Without Text

```
<msg
  type="reply"
  version="2"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
  <report_error
    error_code="your_hair_is_on_fire"/>
</msg>
```

5. 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.

6. 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>

7. 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.

8. References

8.1. 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.

8.2. 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2017

S. Weiler
W3C / MIT
A. Sonalker
TowerSec
R. Austein
Dragon Research Labs
March 11, 2017

A Publication Protocol for the Resource Public Key Infrastructure (RPKI)
draft-ietf-sidr-publication-12

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. Even in cases where a certificate issuer runs their own publication repository, it can be useful to run the certificate engine itself on a different machine from the publication repository. This document defines a protocol which addresses these needs.

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 September 12, 2017.

Copyright Notice

Copyright (c) 2017 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	2
1.1. Historical Note	4
1.2. Terminology	4
2. Protocol Specification	5
2.1. Common XML Message Format	5
2.2. Publication and Withdrawal	7
2.3. Listing the repository	8
2.4. Error handling	8
2.5. Error Codes	9
2.6. XML Schema	10
3. Examples	11
3.1. <publish/> Query, No Existing Object	12
3.2. <publish/> Query, Overwriting Existing Object	12
3.3. <withdraw/> Query	12
3.4. <success/> Reply	12
3.5. <report_error/> With Optional Elements	13
3.6. <report_error/> Without Optional Elements	13
3.7. Error Handling With Multi-Element Queries	13
3.7.1. Multi-Element Query	13
3.7.2. Successful Multi-Element Response	14
3.7.3. Failure Multi-Element Response, First Error Only	14
3.7.4. Failure Multi-Element Response, All Errors	15
3.8. <list/> Query	16
3.9. <list/> Reply	16
4. IANA Considerations	16
5. Security Considerations	17
6. Acknowledgements	18
7. References	18
7.1. Normative References	18
7.2. Informative References	19
Authors' Addresses	19

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. See [RFC6480] for an overview of the RPKI.

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.

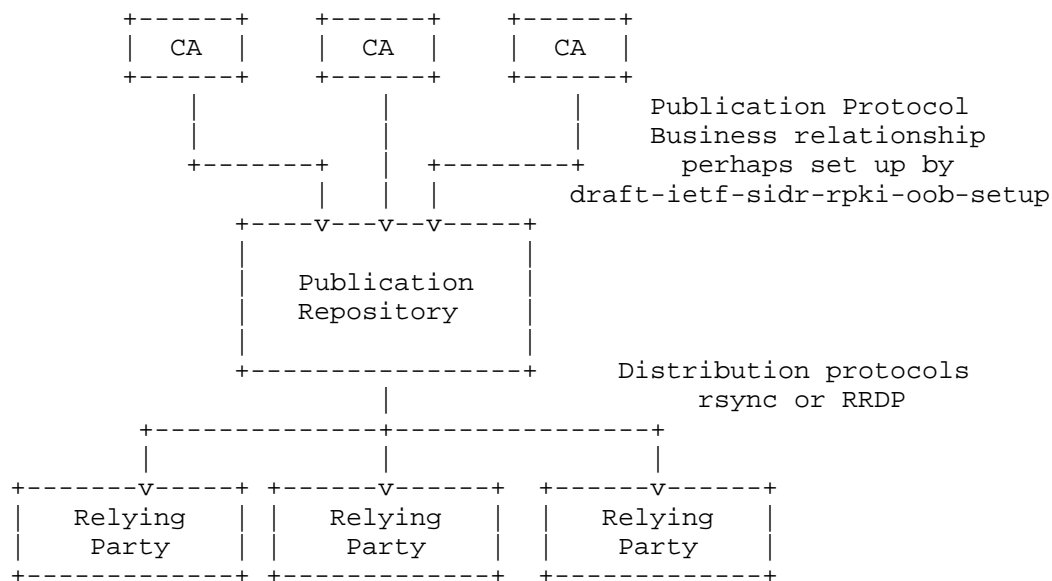
The operator of an RPKI publication repository may well be an Internet registry which issues certificates to its customers, but it need not be; conceptually, operation of a an RPKI publication repository is separate from operation of RPKI CA.

Even in cases where a resource holder operates both a certificate engine and a publication repository, it can be useful to separate the two functions, as they have somewhat different operational and security requirements.

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.

The authentication and message integrity architecture of the publication protocol is essentially identical to the architecture used in [RFC6492], because the participants in this protocol are the same CA engines as in RFC 6492; this allows reuse of the same "Business PKI" ("BPKI", see Section 1.2) infrastructure used to support RFC 6492. As in RFC 6492, authorization is a matter of external configuration: we assume that any given publication repository has some kind of policy controlling which certificate engines are allowed to publish, modify, or withdraw particular RPKI objects, most likely following the recommendation in [RFC6480] Section 4.4, the details of this policy are a private matter between the operator of a certificate engine and the operator of the chosen publication repository.

The following diagram attempts to convey where this publication protocol fits into the overall data flow between the certificate issuers and relying parties:



The publication protocol itself is not visible to relying parties: a relying party sees the public interface of the publication server, which is an rsync or RRDP ([I-D.ietf-sidr-delta-protocol]) server.

Operators of certificate engines and publication repositories may find [I-D.ietf-sidr-rpki-oob-setup] a useful tool in setting up the pairwise relationships between these servers, but are not required to use it.

1.1. Historical Note

This protocol started out as an informal collaboration between several of the early RPKI implementers, and while it was always the designers' intention that the resulting protocol end up on the IETF standards track, it took a few years to get there, because standardization of other pieces of the overall RPKI protocol space was more urgent. The standards track version of this publication protocol preserves the original XML namespace and protocol version scheme in order to maintain backwards compatibility with running code implemented against older versions of the specification.

1.2. 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 ([XML]) messages wrapped in signed CMS messages, carried over HTTP transport ([RFC7230]). The CMS encapsulation is identical to that used in [RFC6492], section 3.1 and subsections.

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].

The CMS signatures are used to protect the integrity of the protocol messages and to authenticate the client and server to each other. Authorization to perform particular operations is a local matter, perhaps determined by contractual agreements between the operators of any particular client-server pair, but in any case is beyond the scope of this specification.

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>
```

As noted above, the outermost XML element is encapsulated in in a signed CMS message. Query messages are signed by the client, reply messages are signed by the server.

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" attribute to facilitate bulk operation. When performing bulk operations, a CA engine will probably find it useful to specify a distinct tag value for each <publish/> or <withdraw/> PDU, to simplify matching an error with the PDU which triggered it. The tag attribute is mandatory, to simplify parsing, but a CA engine which has no particular use for tagging MAY use any syntactically legal value, including simply using the empty string for all tag fields.

This document describes version 4 of this protocol. An implementation which understands only this version of the protocol MUST reject messages with a different protocol version attribute, signalling the error as described in Section 2.4. Since "4" is currently the only value allowed for the version attribute in the schema (Section 2.6), an incorrect protocol version can be detected either by checking the version attribute directly or as a schema validation error. Any future update to this protocol which is either

syntactically or semantically incompatible with the current version will need to increment the protocol version number.

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 an rsync URI ([RFC3986], [RFC5781]). The <publish/> query also contains the DER object to be published, encoded in Base64 ([RFC4648] section 4, with line breaks within the Base64 text permitted but not required).

Both the <publish/> and <withdraw/> PDUs also have a "hash" attribute, which carries a hash of an existing object at the specified repository URI, encoded as a hexadecimal string. For <withdraw/> PDUs, the hash MUST be present, 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 NOT be present if this publication operation is writing to a new URI where no prior object exists. Presence of an object when no "hash" attribute has been specified is an error, as is absence of an object or an incorrect hash value when a "hash" attribute has been specified. 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 message containing <publish/> or <withdraw/> PDUs succeeds, the server returns a single <success/> reply.

When a query fails, the server returns one or more `<report_error/>` reply PDUs. Typically, a server will only generate one `<report_error/>` corresponding to the first query PDU that failed, but servers MAY return multiple `<report_error/>` PDUs at the implementor's discretion.

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.

This schema is normative: in the event of a disagreement between this schema and the document text above, this schema is authoritative.

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/> and <withdraw/> query elements

query_elt |= (
  element publish { tag, uri, hash?, base64 } |
  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. Similarly, these examples do not show the CMS signature wrapper around the XML, just the XML payload.

3.1. <publish/> Query, No Existing Object

```
<msg
  type="query"
  version="4"
  xmlns="http://www.hactrn.net/uris/rpki/publication-spec/">
<!-- body is base64(new-object) -->
<publish
  tag=""
  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/">
<!-- hash is hex(SHA-256(old-object)) -->
<!-- body is base64(new-object) -->
<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/">
<!-- hash is hex(SHA-256(old-object)) -->
<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">
        SGVsbG8sIGl5IG5hbWUgaXMgQWxpY2U=
      </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.hacrn.net/uris/rpki/publication-spec/">
  <list/>
</msg>
```

3.9. <list/> Reply

```
<msg
  type="reply"
  version="4"
  xmlns="http://www.hacrn.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. 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: [[RFCxxxx]]
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: IETF

5. 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 wrapped in a signed CMS message, which provides message integrity protection and an auditable form of message authentication. Because of these protections at the application layer, and because all the data being published are intended to be public information in any case, this protocol does not, strictly speaking, require the use of HTTPS or other transport security mechanisms. There may, however, be circumstances in which a particular publication operator may prefer HTTPS over HTTP anyway, as a matter of (BPKI) CA policy. Use of HTTP versus HTTPS here is, essentially, a private matter between the repository operator and its clients. Note, however, that even if a client/server pair uses HTTPS for this protocol, message authentication for this protocol is still based on the CMS signatures, not HTTPS.

Although the hashes used in the <publish/> and <withdraw/> PDUs are cryptographically strong, 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. Because of this, the most likely

reason for a change to this digest algorithm would be to track a corresponding change in the digest algorithm used in RPKI manifests. If and when such a change happens, it will require incrementing the version number of this publication protocol, but given that the most likely implementation of a publication server uses these hashes as lookup keys in a database, bumping the protocol version number would be a relatively minor portion of the effort of changing the algorithm.

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

6. Acknowledgements

The authors would like to thank: Geoff Huston, George Michaelson, Oleg Muravskiy, Paul Wouters, Randy Bush, Rob Loomans, Robert Kisteleki, Tim Bruijnzeels, Tom Petch, and anybody else who helped along the way but whose name(s) the authors have temporarily forgotten.

7. References

7.1. Normative References

- [RelaxNG] Clark, J., "RELAX NG Compact Syntax", OASIS , November 2002, <<https://www.oasis-open.org/committees/relax-ng/compact-20021121.html>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, STD 66, January 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, STD 70, September 2009.
- [RFC5781] Weiler, S., Ward, D., and R. Housley, "The rsync URI Scheme", RFC 5781, February 2010.

- [RFC6492] Huston, G., Loomans, R., Ellacott, B., and R. Austein, "A Protocol for Provisioning Resource Certificates", RFC 6492, February 2012.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [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>>.
- [XML] Cowan, J., "Extensible Markup Language (XML) 1.1", W3C CR CR-xml11-20021015, October 2002.

7.2. Informative References

- [I-D.ietf-sidr-delta-protocol] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "RPKI Repository Delta Protocol", draft-ietf-sidr-delta-protocol-07 (work in progress), February 2017.
- [I-D.ietf-sidr-rpki-oob-setup] Austein, R., "An Out-Of-Band Setup Protocol For RPKI Production Services", draft-ietf-sidr-rpki-oob-setup-09 (work in progress), February 2017.
- [RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.

Authors' Addresses

Samuel Weiler
W3C / MIT

Email: weiler@csail.mit.edu

Anuja Sonalker
TowerSec Automotive Cyber Security

Email: asonalker@tower-sec.com

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net

Secure Inter-Domain Routing
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2014

S. Kent
D. Mandelberg
BBN Technologies
September 10, 2013

Suspenders: A Fail-safe Mechanism for the RPKI
draft-kent-sidr-suspenders-00

Abstract

The Resource Public Key Infrastructure (RPKI) is an authorization infrastructure that allows the holder of Internet Number Resources (INRs) to make verifiable statements about those resources. The certification authorities (CAs) in the RPKI issue certificates to match their allocation of INRs. These entities are trusted to issue certificates that accurately reflect the allocation state of resources as per their databases. However, there is some risk that a CA will make inappropriate changes to the RPKI, either accidentally or deliberately (e.g., as a result of some form of "government mandate"). The mechanisms described below, and referred to as "Suspenders" are intended to address this risk.

Suspenders enables an INR holder to publish information about changes to objects it signs and publishes in the RPKI repository system. This information is made available via a file that is external to the RPKI repository, so that Relying Parties (RPs) can detect erroneous or malicious changes related to these objects. RPs can then decide, individually, whether to accept changes that are not corroborated by independent assertions by INR holders, or to revert to previously verified RPKI data.

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 March 14, 2014.

Copyright Notice

Copyright (c) 2013 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. Overview	2
2. Terminology	3
3. The LOCK Record and INRD File	4
3.1. LOCK Record Format and Semantics	4
3.2. INRD File Format and Semantics	6
4. Self-checking by RPs	9
5. Detection & Remediation	10
6. INRD Management Scenarios	12
7. IANA Considerations	14
8. Security Considerations	14
9. Acknowledgements	14
10. References	14
10.1. Informative References	14
10.2. Normative References	15
Appendix A. RPKI Object Whacking and Competition	15
Appendix B. Design Criteria (do we still need this section?) . .	16
Authors' Addresses	17

1. Overview

The Resource Public Key Infrastructure (RPKI) is an authorization infrastructure that allows the holder of Internet number resources (INRs) to make verifiable statements about those resources. For example, the holder of a block of IP(v4 or v6) addresses can issue a Route Origination Authorization (ROA) to authorize an autonomous system to originate routes for that block.

The certification authorities (CAs) in the RPKI issue certificates to match their allocation of INRs. These entities are trusted to issue certificates that accurately reflect the allocation state of resources as per their databases. However, there is some risk that a

CA will make inappropriate changes to the RPKI, either accidentally or deliberately (e.g., as a result of some form of "government mandate").

Assertions by INR holders about their resources, and about bindings among resources, are realized by publishing RPKI signed objects via the RPKI repository system [RFC6481]. For example, authorization to originate a route for a prefix is accomplished by issuing a ROA. Changes in the RPKI can have an adverse impact on routing in the Internet, by changing the set of (valid) signed objects for a resource. Invalidating a ROA could cause the origin authorized by the ROA in question to be less preferred; adding a ROA for a more specific prefix could enable an unauthorized party to represent itself as the legitimate origin for traffic for that prefix.

The goal of Suspenders is to minimize the likelihood that changes to the RPKI will adversely affect INR holders, irrespective of whether the changes are inadvertent or malicious. Suspenders should work when an INR holder acts as its own CA (and manages its own publication point), and when the INR holder has outsourced these management functions. Suspenders allows each INR holder to assert a "lock" on selected objects at its publication point, to protect the bindings asserted by these objects. Changes to protected objects are confirmed by the INR holder, via a file published outside the repository system. Changes to the validity of protected objects, effected by changes to any other objects in the RPKI, are presumed to be unauthorized (and thus suspicious), unless independently confirmed by the INR holder.

Detection of potentially adverse changes is carried out by each INR holder for its own resources, and by each RP that elects to implement Suspenders. It is critical that an INR holder be able to quickly detect adverse changes that affect its own resources, so that it can initiate actions to remedy the problem. RPs should be able to detect potentially adverse changes, that are not authorized by INR holders, so that they can (at their discretion) use cached, validated data in lieu of such changes. The model adopted here is to assume that changes to previously-validated data should not be accepted, unless authorized by the relevant INR holder. Thus RPs who detect changes need to be able to verify that these changes are authorized by the INR holder. Because not all INR holders manage their own CAs and publication points, an external mechanism is used to signal authorized changes to RPs.

2. 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].

3. The LOCK Record and INRD File

An INR holder that elects to protect its resources and resource bindings creates a LOCK record in its publication point. The INR holder also generates and signs an Internet Number Resource Declaration (INRD) file, and publishes it at a location independent of the RPKI repository system. The LOCK record consists of a URL that points to the INRD file, and a public key used to verify a signature on the content of that file. (This public key is distinct from any used by the INR holder in the RPKI context.) The INRD file contains the date at which the most recent changes were made, and enumerates those changes. The formats of the LOCK record and INRD file are described below.

3.1. LOCK Record Format and Semantics

The LOCK record conforms to the signed object specification from [RFC6488], which, in turn, uses the CMS [RFC5652] signed-data object format. See [RFC6488] for the top-level signed-data format and the constraints imposed on that format for use in the RPKI context. The LOCK encapsulated content is defined below:

```
EncapsulatedContentInfo ::= SEQUENCE {  
    eContentType ContentType,  
    eContent [0] EXPLICIT OCTET STRING OPTIONAL }
```

```
ContentType ::= OBJECT IDENTIFIER
```

The eContentType for an LOCK record is defined as id-ct-rpkiLOCK and it has the numeric value 1.2.840.113549.1.9.16.1.XX.

```
id-smime OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)  
    rsadsi(113549) pkcs(1) pkcs9(9) 16 }
```

```
id-ct OBJECT IDENTIFIER ::= { id-smime 1 }
```

```
id-ct-rpkiLOCK OBJECT IDENTIFIER ::= { id-ct XX }
```

The eContent for an LOCK record is defined by the following ASN.1

```
LOCK ::= SEQUENCE {  
    version [0] INTEGER DEFAULT 0,
```

```
outsourced  BOOLEAN,  
uRL         IA5String,  
publicKey   SubjectPublicKeyInfo }
```

```
SubjectPublicKeyInfo ::= SEQUENCE {  
    algorithm      AlgorithmIdentifier,  
    subjectPublicKey BIT STRING }
```

The EE certificate embedded in the LOCK record MUST use the inherit flag in the [RFC3779] extensions. (The content of the LOCK is independent of the 3779 extensions in the EE certificate, so it is appropriate to use the inherit flag here.)

The version number of the LOCK record determines the set of RPKI object types that it protects. Version 0 protects the LOCK record itself, ROAs, (subordinate) CA certificates, and router certificates (if present).

The algorithm and subjectPublicKey fields in the publicKey MUST conform to the guidance in Section 3 of [RFC6485].

If an RP elects to process a LOCK record, it verifies the signature on the record using the procedure described in [RFC6488]. If the signature verification fails, it ignores the record. (If the RP has a previously validated LOCK record, it continues to use that record instance.)

If the signature verification succeeds, the RP extracts the version number and verifies that the RP is prepared to process this version of the record. If not, it ignores the record. If it is prepared to process this version, it extracts the URL and public key fields. The URL is used to fetch the corresponding INRD file, and the public key is used to verify the signature on that file.

If the RP has a copy of an INRD file for this publication point, and if the RP detects no material changes to the protected records at the publication point, the RP SHOULD NOT fetch the INRD file. (A material change is one that affects the semantics of the object. For example, for a ROA, only changes to the prefixes and/or ASN are material.) If the RP does not hold a copy of the INRD file, or if a protected record has changed, the RP fetches a new INRD file using the URL, and proceeds as described in Section 3.2.

When an INR holder has outsourced management of its RPKI CA function and publication point, it is susceptible to attacks in which the LOCK record itself is changed. This is because the entity providing these functions could create a new LOCK record containing a new URL and

public key, thus defeating the LOCK/INRD mechanism. An authorized change to the content of a LOCK record should be very rare. A location selected as a home for an INRD file should be stable, and thus the URL should rarely change. The public key used to verify the signature on an INRD file should also be constant for long intervals. The LOCK record contains a flag that indicates whether the INR holder has outsourced CA and publication point management. If this flag is FALSE, an RP will accept changes to the LOCK record (see Section 5) just as it would changes to any other object at a protected publication point. If the flag is TRUE, then any change to a LOCK record is regarded as suspicious by RPs. In such cases the RP delays accepting the new LOCK record and associated INRD file, as discussed in Section 5.

3.2. INRD File Format and Semantics

The INRD file is a DER-encoded ASN.1 file that contains data associated with a single INR holder (publication point owner). The file is encoded using ASN.1, since most of the values it holds will be compared to data from RPKI objects that also are ASN.1 encoded, and because it is a signed object.

```
INRD ::= SEQUENCE {
    tbsINRD      TBSINRD,
    algorithm    AlgorithmIdentifier,
    signature    OCTET STRING
}

TBSINRD ::= SEQUENCE {
    version      [0] INTEGER DEFAULT 0,
    lastChange   UTCTime,
    changeWindow ENUMERATED
        {
            1week (7) DEFAULT
            2week (14)
            4week (28)
        },
    additions    [1] SEQUENCE SIZE (1..MAX) OF
        ProtectedObject OPTIONAL,
    deletions    [2] SEQUENCE SIZE (1..MAX) OF
        ProtectedObject OPTIONAL,
    keyRollover  [3] OCTET STRING OPTIONAL,
    algRollover  [4] OCTET STRING OPTIONAL
}

ProtectedObject ::= CHOICE {
    cmsObject    [0] EncapsulatedContentInfo,
    rtrCert      [1] RtrCertInfo,
```



```
    cACert      [2] CACertInfo
  }

RtrCertInfo ::= SEQUENCE {
    subjKeyId  OCTET STRING,
    aSNum      INTEGER
}

CACertInfo ::= SEQUENCE {
    subjKeyId      OCTET STRING,
    ipAddrBlocks   [0] IPAddrBlocks OPTIONAL,
    aSIdentifiers  [1] ASIdentifiers OPTIONAL
}

-- See [RFC3779] for the definitions of IPAddrBlocks and
-- ASIdentifiers.
```

The lastChange and changeWindow values are used to bound the set of additions and deletions stored in an INRD file. The default is a one week window, but two and four week values also may be expressed. The window determines the oldest time at which changes to protected objects at the publication point are represented in the additions and deletions portions of the file.

The additions element is used by an INR holder to list all protected objects that have been added to the publication point over the interval defined by the change window. If no objects have been added during this interval, the element is omitted. Similarly, the deletions element is used by an INR holder to list all protected objects that have been removed from the publication point over the interval defined by the change window. If no objects have been removed during this interval, the element is omitted.

A LOCK or ROA is listed in the additions and/or deletions fields by putting its EncapsulatedContentInfo in the cmsObject field of a ProtectedObject. A router certificate is listed by putting its SKI and AS number in the rtrCert field. A CA certificate is listed by putting its SKI and [RFC3779] resources in the cACert field. If the outsourced flag in the LOCK record is FALSE, then no CA certificates should be included in the additions or deletions elements. If any CA certificates are included in these elements, they are ignored. RPs SHOULD accept all valid CA certificates issued at this publication point when the outsourced flag is FALSE.

The key rollover element is present only during the time when a key rollover [RFC6489] is taking place. It signals to RPs that an additional set of objects exist that would ordinarily be viewed as

competing with the objects protected by this INRD file. The SKI contained here is that of the CA for the "other" key. During key rollover each CA will have its own LOCK record, that points to its own INRD file. The old CA will list the new CA's SKI here; the new CA will not include this field. Key rollover is a transient condition, so the need for both LOCK records and INRD files ought not be very long.

The algorithm rollover element is present during the time when an algorithm rollover [RFC6916] is taking place. It signals to RPs that an additional set of objects exist that would ordinarily be viewed as competing with the objects protected by this INRD file. The SKI contained here is that of the CA for the "other" algorithm. During algorithm rollover each CA will have its own LOCK record, that points to its own INRD file, and each of them will list the other CA's SKI here. (Note that the SKI value is compared against the SKI in the CA certificate in question. An RP does not compute an SKI. This means that changes to the hash algorithm used to compute an SKI do not affect how an RP processes this field. An RP MUST be prepared to deal with an SKI length other than the 20 octet value in common use today.) Algorithm transition is a long process, so both sets of LOCK records and INRD files will persist for an extended period.

An RP fetches an INRD file using a URL from a LOCK record, as noted above. If the file cannot be located, the RP software logs an error and regards any changes to the publication point as suspicious. If the file is located, the RP verifies the signature on the file using the public key (and indicated algorithms) from the same LOCK record. If the signature fails, the RP software logs an error and regards any changes to the publication point as suspicious. If the signature is valid, the RP extracts the data elements from the INRD file.

If this is the first time that an INRD file is fetched for this publication point, the file is accepted, and its content is used to populate the RP's expanded local cache. If the INRD file is replacing a previously acquired instance for this publication point, the content is used to confirm changes to protected objects at this publication point. If an RP detects changes to protected publication point objects that occurred after the lastChange time, these changes are treated as suspicious.

Authorizing changes to subordinate CA certificates in an INRD file is critical when an INR holder outsources CA and publication point management. Listing these CAs and their associated 3779 extension data enables an RP to detect creation of unauthorized CAs that could then create competing ROAs or router certificates. However, if an INR holder operates its own CA and manages its publication point, it is not necessary to protect against such attacks. To signal this

situation, the "outsourced" flag in the LOCK record is set to FALSE. Under this condition, an RP will not impose change control checks on subordinate CA certificates for the publication point.

Some classes of INR holders need not publish a LOCK record and INRD file. IANA, RIRs, and NIRs, are principally delegators of resources. Each of these RPKI entities SHOULD create one publication point for the resources used by the entity for its own networking needs, and a separate publication point under which all resource delegations take place. The first publication point MAY be protected by a LOCK record, so that ROAs and router certificates associated with those resources can be protected. However, the second publication point OUGHT not include a LOCK record. If this convention is followed, these classes of INR holders need not update an INRD file every time a new subordinate CA is created or modified, as a result of delegation. If an INR holder follows this convention, and includes a LOCK record in its superior publication point, that record, and the associated INRD file, conveys some degree of protection for the subordinate CA resources, even if the INR holders of these resources do not publish LOCK records.

4. Self-checking by RPs

It is easy for an INR holder, acting as an RP, to determine if any of its resource bindings have been undermined via the RPKI. It knows what resources it holds, and what assertions it has made relevant to those resources. Any conflicting RPKI objects represent a problem! It is more difficult for an RP to detect problems with another INR holder's resources, because it lacks the knowledge that the other INR holder has. The mechanisms described in Section 5 are designed to enable RPs to detect these problems. This section describes the procedures each RP executes to detect adverse changes to its own data in the RPKI repository system. Note that the procedures in this section do not require use of the LOCK record or INRD file.

When an INR downloads RPKI data, as it normally does, it SHOULD perform the checks noted below, to detect problems. To enable such checking, each INR holder's RP software MUST be configured with data about the ROAs, and other protected objects, of this INR holder. If any of these objects are missing or fail to validate, then the INR holder has detected a problem, and is notified.

The semantics of ROAs require an additional check; if other ROAs for the same or more specific prefixes are found anywhere in the RPKI repository system, this too indicates a problem, and the INR holder is notified.

The semantics of router certificates, require a separate, additional check. A router certificate binds a public key (and a router ID) to an ASN. Thus, if an INR holder discovers router certificates for its ASN, that it did not authorize, this indicates a problem.

As additional objects are protected via this mechanism, it will be necessary to perform additional checks to detect the latter sort of adverse changes, based on the semantics of the protected objects.

In any case, RP software SHOULD inform the INR holder of the apparent cause and source of the problem, e.g., a revoked or expired certificate or a manifest problem, and guide the INR holder to the responsible CAs (e.g., using Ghostbusters [RFC6493] records).

When an INR holder is alerted to a change adversely affecting its own resources, it is expected to contact the appropriate RPKI entities to rectify the error in a timely fashion. If the changes are determined to be intentional (and not authorized by the INR holder), the INR holder can inform the Internet operations community (via an out of band mechanism), which can then decide, individually, how to respond.

Remedying a problem detected by an INR holder is not likely to be instantaneous, even if the problem is just an error. To avoid adversely affecting routing, the mechanisms described in Section 5 enable RPs to detect a change that adversely affects INR holders, and to reject it, reverting to previously validated INR data. This gives the INR holder time to resolve the problem. Reverting to an earlier version of INR data is conceptually easy for RPs, because they already cache RPKI data. The mechanisms described below require augmenting the RPKI local cache maintained by each RP, to detect adverse changes, making use of information gleaned from LOCK records and INRD files. The next section describes how the LOCK and INRD data is used.

5. Detection & Remediation

The design described in this section assumes that an RP has acquired a snapshot of the RPKI repository, validated and extracted INR holding and binding data, and considers this data to be "good". The detection and remediation algorithm is initialized by acquiring a complete download of RPKI repository data, and by fetching INRD files for all publication points that contain a LOCK record. (Prior to this initialization step, it is not possible for an RP to detect and respond to adverse changes to the RPKI, using the technique described below.)

Each RP already maintains a cache of RPKI data [RFC6480], [RFC6481]; this document extends that cache. For every publication point that

contains a LOCK record, the content of that record, and the corresponding INRD file content, become part of the data maintained by each RP.

An RP acquires and validates all changed RPKI objects as usual. An RP does not update its cache with the changes, until additional checks, described below, are performed. Before accepting any changes the RP MUST process every pub point where there is (or was) a LOCK record. For each of these pub points, if there are changes to protected objects, these changes must be confirmed by the corresponding INRD file before they are accepted. If any of these checks fail, the changes are held in escrow, waiting for a timeout (or an updated INRD file?). After all protected pub point changes have been processed, then changes for unprotected pub point can be accepted. The checks will detect pending changes that would whack or compete with protected objects, and place them in escrow.

After validating all changed objects downloaded from the RPKI repository, an RP performs the following additional checks for every publication point that has (or had) a LOCK record:

- o ROA and router certificate whacking
- o INR sub-delegation changes
- o ROA competition
- o router certificate competition
- o LOCK record changes

A ROA or a router certificate has been whacked (see Appendix A) if it was valid and is now missing or invalid, and if it is not indicated as deleted in the INRD file of its issuer. Any previously valid ROA that is no longer valid (or missing) is checked against the INRD file for the ROA issuer, to determine if the ROA or (router certificate) certificate has been legitimately revoked/removed. If the INRD file confirms the action, the old ROA (or router certificate) is removed from the local cache. If not, the old ROA (or router certificate) is retained, but marked as suspicious.

Changes to INR sub-delegation occur when the INR holder issues a new CA certificate, an existing child CA certificate expires, or any other change affects the status of a child CA certificate. These changes are accepted by an RP only if they are confirmed in the INR holder's INRD file.

A newly issued ROA is in competition (see Appendix A) with an existing ROA if the new ROA specifies the same or a more specific prefix than the older ROA, the new ROA is not issued by one of the existing ROA's issuer's descendants, and the new ROA was not authorized by the INRD file of the existing ROA. A competing ROA is not accepted as valid by an RP.

A newly-issued router certificate competes with an existing router certificate, if the new certificate includes the same ASN and was not authorized by the INRD file covering the existing router certificate. A competing router certificate is not accepted as valid by an RP. (Such a certificate would be accepted if the INRD file of the issuer of the original certificate indicates that the old certificate has been deleted, and not replaced with a new router certificate associated with the same entity. In this case, the newly-issued certificate would not be in competition.)

As noted above, any change to a LOCK record is viewed as suspicious unless the outsourced flag is FALSE. If the record is for a publication point that is not outsourced, then a changed LOCK record is accepted as valid if the corresponding INRD file authorizes the new record. (If the INR holder has changed the public key for the INRD file, it is RECOMMENDED that the URL also change. This allows the INR holder to publish a new INRD file that authorizes the new LOCK record, minimizing the potential race condition between updating an INRD file and a LOCK record.)

If the LOCK record shows that the publication point is outsourced, an RP examines the changes made to the LOCK record. If the URL has changed, but the public key and the outsource flag are unchanged, the new LOCK record may be accepted, if the new INRD file authorizes the change. If not, the new LOCK record is rejected. If the public key has been changed, a delay is imposed on accepting the new LOCK record, even if the INRD file authorizes the change. (should we establish a global delay, or should each INR holder publish its own delay preference in the INRD file?)

Remediation for all of the whacking and competition events consists of NOT making a change in the local cache when an unconfirmed change is encountered.

6. INRD Management Scenarios

Common wisdom notes that we cannot choose our parents, but we can choose our friends, and we should do so wisely. In the RPKI context, and INR holder cannot, generally choose its CA, but Suspenders allows the INR holder to choose its INRD file server. It should do so wisely.

An INRD file is published outside of the RPKI repository system, and is verified using a public key that is also independent of the RPKI. The motivation for these two measures is to insulate this part of the Suspenders system from possible manipulation by an entity to whom CA and publication point services have been outsourced. If an INR holder acts as its own CA, and manages its own publication point, it can publish its INRD file on the same machines as its publication point, but not in the publication point. In this case the independence features are not critical, but they also don't cause harm for this class of INR holder.

Every INR holder needs to choose a location for the INRD file that is highly available. When an INR holder has outsourced CA and publication point management, independent publication of the INRD file is critical. The INR holder needs to choose a location for the INRD file that is highly available. It also is appropriate to consider placing the file outside of the geopolitical region in which the INR holder (and its RIR) operate. Here too the motivation is to insulate the INR holder from a malicious action by the CA service provider, or, perhaps, an RIR above it.

Organizations may arise to offer hosting for INRD files, as a service for INR holders. They could offer just file storage, or they might offer more extensive services. For example, an organization might monitor an INR holder's publication point and create the INRD file data, and even sign it for the INR holder. (In this case the organization would provide the public key to the INR holder for inclusion in the LOCK record.) Various other arrangements between the INR holder and a organization that assists in managing INRD files are possible, and are a local matter between the INR holder and the organization.

A country might elect to mandate use of Suspenders, as a means to protect the INRs of its ISPs and other organizations that run BGP with in the country. The motivation is similar to that cited above, i.e., protecting INRs against errors or malicious actions by RPKI entities. In this case the country itself generally is not an INR holder per se, so the relationship is somewhat different from that discussed above. Nonetheless, the mechanisms described above apply.

For example, Elbonia might mandate that every INR holder within the country make use of Suspenders. Every Elbonian INR holder will be required to include a LOCK record in its publication point, no matter where that publication point is realized. The URL in each LOCK points to a file on a server managed by an Elbonian government organization. Each Elbonian ISP would be required to follow the procedures described in Section 5, when managing its local cache.

7. IANA Considerations

This document registers the following in the "RPKI Signed Object" registry created by [RFC6488]:

Name: LOCK
OID: 1.2.840.113549.1.9.16.1.XX
Reference: [RFCxxxx] (this document)

This document also registers the following three-letter filename extension in the "RPKI Repository Name Schemes" registry created by [RFC6481]:

Filename extension: lck
RPKI Object: LOCK
Reference: [RFCxxxx] (this document)

8. Security Considerations

This document specifies Suspenders, a set of security-focused mechanisms designed to protect INR holders against accidental and malicious changes to RPKI repository data, and to enable RPs to detect and respond to such changes. More text to be provided later.

9. Acknowledgements

Richard Barnes provided the motivation to develop Suspenders, after he identified a problem with the LTAM [I-D.ietf-sidr-ltamgmt] design.

10. References

10.1. Informative References

- [I-D.ietf-sidr-ltamgmt]
Reynolds, M., Kent, S., and M. Lepinski, "Local Trust Anchor Management for the Resource Public Key Infrastructure", draft-ietf-sidr-ltamgmt-08 (work in progress), April 2013.
- [RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.
- [RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", RFC 6481, February 2012.

- [RFC6489] Huston, G., Michaelson, G., and S. Kent, "Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)", BCP 174, RFC 6489, February 2012.
- [RFC6493] Bush, R., "The Resource Public Key Infrastructure (RPKI) Ghostbusters Record", RFC 6493, February 2012.

10.2. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, June 2004.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.
- [RFC6485] Huston, G., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)", RFC 6485, February 2012.
- [RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", RFC 6488, February 2012.
- [RFC6916] Gagliano, R., Kent, S., and S. Turner, "Algorithm Agility Procedure for the Resource Public Key Infrastructure (RPKI)", BCP 182, RFC 6916, April 2013.

Appendix A. RPKI Object Whacking and Competition

There are two ways that an RPKI object can be adversely affected. We term these actions "whacking" and "competition."

Any object in the RPKI can become invalid or inaccessible (to RPs) via various actions by CAs and/or publication point maintainers along the certificate path from the object's EE certificate to a trust anchor (TA). Any action that causes an object to become invalid or inaccessible is termed "whacking". Revocation of the EE certificate for an object whacks it. Revocation of any CA certificate along the certificate path for the object (without reissuance) has the same effect. Reissuance of any CA certificate along the certificate path, with changes in [RFC3779] extensions in any of these certificates to exclude the resources cited in a targeted object, also constitutes whacking. Changing a manifest along the certificate path might whack an object (depending on how RPs deal with manifest changes), and removing an object from the RPKI repository system also potentially

whacks it. Unless an action that causes an object to be whacked is authorized by the creator of an object, whacking is an attack against the INR holder that created the whacked object.

A different form of attack is termed object "competition". The details of object competition are determined by the semantics of the object. In the general case, one object competes with another object (of the same type), if the newer object creates a binding that adversely affects the binding expressed in the original object. So, for example, a newly issued ROA competes with an existing ROA if the new ROA contains the same or more specific prefixes than the older ROA. Competition does not always indicate an attack; the transfer of resources in a "make before break" model implies ROA competition. A newly issued router certificate competes with a previously issued one if the new certificate binds the same ASN to a public key issued by a different entity. (If key rollover or algorithm transition is in progress, such competition is explicitly authorized via the INRD file.)

Competition that is not authorized by the issuer of the original router certificate is viewed as an attack against that certificate.

Appendix B. Design Criteria (do we still need this section?)

Several criteria were employed in developing the mechanisms described in this document.

1. It is anticipated that object whacking and competition, and analogous forms of errors that adversely impact INR holders, will be infrequent. Thus the detection mechanisms employed by RPs to detect such anomalies ought to be efficient (in terms of data fetching, processing, and storage) for the normal case.
2. RPs may elect to ignore/reject adverse changes to objects if they perceive such changes as suspicious. If an RP elects to reject a change to an object it must have access to previously validated objects for the INR holder question.
3. Transfers of "live" address space will occur, although not frequently. INR holders engaged in such transfers must be able to signal to RPs that such transfers are authorized, so that the transfers are not rejected as suspicious.
4. Routes for a prefix may be legitimately originated by more than one AS (MOA). The design MUST enable an INR holder to inform RPs when this situation is authorized.

5. Many INR holders may choose to outsource CA and publication point management functions. INR holders who choose to outsource these functions should be offered equivalent protection against ROA invalidation and competition as INR holders who perform these functions for themselves.
6. Any new RPKI repository objects used with the mechanisms defined here MUST conform to the format specified in [RFC6488].
7. The decision to process any additional data associated with the mechanisms described in this document is local to each RP. RPs that choose to not implement these mechanisms will incur minimal additional data fetching, storage, and processing burdens.
8. The decision to employ the mechanisms described here to protect INR holdings and binding is a local one made by each INR holder. (INR holders who outsource CA and publication point management functions will require the providers of these services to support creation and publication of one new RPKI object. As a result, all such providers must support generation and maintenance of the new RPKI object so that their clients have the option to utilize these capabilities.)
9. Revocation and expiration of RPKI object MUST continue to work as they do currently, for all objects that have not been adversely affected.

Authors' Addresses

Stephen Kent
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: kent@bbn.com

David Mandelberg
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: david@mandelberg.org