Internet Engineering Task Force (IETF)
Request for Comments: 8181
Category: Standards Track
ISSN: 2070-1721

S. Weiler
W3C / MIT
A. Sonalker
STEER Tech
R. Austein
Dragon Research Labs

July 2017

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

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 its 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

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Historical Note 4
  1.2. Terminology 5
2. Protocol Specification 5
  2.1. Common XML Message Format 6
  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 12
  3.1. <publish/> Query, No Existing Object 12
  3.2. <publish/> Query, Overwriting Existing Object 12
  3.3. <withdraw/> Query 13
  3.4. <success/> Reply 13
  3.5. <report_error/> with Optional Elements 13
  3.6. <report_error/> without Optional Elements 14
  3.7. Error Handling with Multi-Element Queries 14
    3.7.1. Multi-Element Query 14
    3.7.2. Successful Multi-Element Response 15
    3.7.3. Failure Multi-Element Response, First Error Only . . 15
    3.7.4. Failure Multi-Element Response, All Errors 16
  3.8. <list/> Query 16
  3.9. <list/> Reply 17
4. IANA Considerations 17
5. Security Considerations 18
6. References 19
  6.1. Normative References 19
  6.2. Informative References 20
Acknowledgements 21
Authors' Addresses 21

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 an RPKI publication repository is separate from operation of an RPKI Certification Authority (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 both the CA engine and 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:

```
          +------+    +------+    +------+
          |  CA  |    |  CA  |    |  CA  |
          +------+    +------+    +------+
              |           |           |    Publication protocol
              |           |           |    business relationship
              +-------+   |  +--------+      perhaps set up by
                      |   |  |                    RFC 8183
                 +----v---v--v-----+
                 |                 |
                 |   Publication   |
                 |   Repository    |
                 |                 |
                 +-----------------+     Distribution protocols
                          |                   rsync or RRDP
           +--------------+----------------+
           |              |                |
   +-------v-----+ +------v------+  +------v------+
   |   Relying   | |   Relying   |  |   Relying   |
   |    Party    | |    Party    |  |    Party    |
   +-------------+ +-------------+  +-------------+
```

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 RPKI Repository Delta Protocol (RRDP) [RFC8182] server.

Operators of certificate engines and publication repositories may find [RFC8183] a useful tool in setting up the pairwise relationships between these servers, but they 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

"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 Cryptographic Message Syntax (CMS) messages, carried over HTTP transport [RFC7230]. The CMS encapsulation is identical to that used in Section 3.1 (and subsections) of RFC 6492 [RFC6492].

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 a 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 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 a signed CMS message. Query messages are signed by the client, and 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, signaling 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 it 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 implementer’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 [RELAX-NG] 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.

```
# RELAX NG 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 that 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">
         SGVsbG8sIG15IG5hbWUgaXMgQWxpY2U=
       </publish>
       </failed_pdu>
     </report_error>
   </msg>
```

3.6. <report_error/> without Optional Elements

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

3.7. Error Handling with Multi-Element Queries

3.7.1. Multi-Element Query

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

3.7.2. Successful Multi-Element Response

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

3.7.3. Failure Multi-Element Response, First Error Only

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

3.7.4. Failure Multi-Element Response, All Errors

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

3.8. <list/> Query

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

3.9. <list/> Reply

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

4. IANA Considerations

IANA has registered the "application/rpki-publication" media type as follows:

Type name: application
Subtype name: rpki-publication
Required parameters: None
Optional parameters: None
Encoding considerations: binary
Security considerations: Carries an RPKI publication protocol message, as defined in RFC 8181.
Interoperability considerations: None
Published specification: RFC 8181
Applications which use this media type: HTTP
Additional information:
   Magic number(s): None
   File extension(s): None
   Macintosh File Type Code(s): None
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 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. References

6.1. Normative References

[RELAX-NG] Clark, J., "RELAX NG Compact Syntax", OASIS Committee Specification, 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", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5781] Weiler, S., Ward, D., and R. Housley, "The rsync URI Scheme", RFC 5781, DOI 10.17487/RFC5781, February 2010, <http://www.rfc-editor.org/info/rfc5781>.

[RFC6492] Huston, G., Loomans, R., Ellacott, B., and R. Austein, "A Protocol for Provisioning Resource Certificates", RFC 6492, DOI 10.17487/RFC6492, February 2012, <http://www.rfc-editor.org/info/rfc6492>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[XML] Cowan, J., "Extensible Markup Language (XML) 1.1", W3C Consortium Recommendation REC-xml11-20060816, October 2002, <http://www.w3.org/TR/2002/CR-xml11-20021015>.

6.2. Informative References

[RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, DOI 10.17487/RFC6480, February 2012, <http://www.rfc-editor.org/info/rfc6480>.

[RFC8182] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "The RPKI Repository Delta Protocol (RRDP)", RFC 8182, DOI 10.17487/RFC8182, July 2017, <http://www.rfc-editor.org/info/rfc8182>.

[RFC8183] Austein, R., "An Out-of-Band Setup Protocol for Resource Public Key Infrastructure (RPKI) Production Services", RFC 8183, DOI 10.17487/RFC8183, July 2017, <http://www.rfc-editor.org/info/rfc8183>.

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.

Authors' Addresses

Samuel Weiler
W3C / MIT

Email: weiler@csail.mit.edu

Anuja Sonalker
STEER Tech

Email: anuja@steer-tech.com

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net