

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 2, 2017

T. Bruijnzeels
O. Muravskiy
RIPE NCC
B. Weber
Cobenian
R. Austein
Dragon Research Labs
September 29, 2016

RPKI Repository Delta Protocol
draft-ietf-sidr-delta-protocol-04

Abstract

In the Resource Public Key Infrastructure (RPKI), certificate authorities publish certificates, including end entity certificates, Certificate Revocation Lists (CRL), and RPKI signed objects to repositories. Relying Parties (RP) retrieve the published information from those repositories. This document specifies a delta protocol which provides relying parties with a mechanism to query a repository for incremental updates, thus enabling the RP to keep its state in sync with the repository.

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 2, 2017.

Copyright Notice

Copyright (c) 2016 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.	Requirements notation	2
2.	Introduction	2
3.	RPKI Repository Delta Protocol Implementation	3
3.1.	Informal Overview	3
3.2.	Certificate Authority Use	4
3.3.	Repository Server Use	5
3.3.1.	Initialisation	5
3.3.2.	Publishing Updates	5
3.4.	Relying Party Use	7
3.4.1.	Processing the Update Notification File	7
3.4.2.	Processing a Snapshot File	8
3.4.3.	Processing Delta Files	8
3.4.4.	Polling the Update Notification File	9
3.5.	File Definitions	9
3.5.1.	Update Notification File	9
3.5.2.	Snapshot File	11
3.5.3.	Delta File	12
3.5.4.	XML Schema	14
4.	HTTPS considerations	16
5.	Security Considerations	16
6.	IANA Considerations	17
7.	Acknowledgements	17
8.	Normative References	17
	Authors' Addresses	19

[1.](#) Requirements notation

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](#)].

[2.](#) Introduction

In the Resource Public Key Infrastructure (RPKI), Certificate Authorities (CAs) publish certificates [[RFC6487](#)], RPKI signed objects [[RFC6488](#)], manifests [[RFC6486](#)], and CRLs to repositories. CAs may have an embedded mechanism to publish to these repositories, or they may use a separate repository server and publication protocol. RPKI

repositories are currently accessible using the rsync protocol, allowing Relying Parties (RPs) to synchronise a local copy of the RPKI repository used for validation with the remote repositories [[RFC6481](#)].

This document specifies an alternative repository access protocol based on notification, snapshot and delta files that a RP can retrieve over the HTTPS protocol. This allows RPs to perform either a full (re-)synchronisation of their local copy of the repository using snapshot files, or use delta files to keep their local repository updated after initial synchronisation.

This protocol is designed to be consistent (in terms of data structures) with the publication protocol [[I-D.ietf-sidr-publication](#)] and treats publication events of one or more repository objects as discrete events that can be communicated to relying parties. This approach helps to minimize the amount of data that traverses the network and thus helps minimize the amount of time until repository convergence occurs. This protocol also provides a standards based way to obtain consistent, point in time views of a single repository, eliminating a number of consistency related issues. Finally, this approach allows these discrete events to be communicated as immutable files, so that caching infrastructure can be used to reduce the load on a repository server when a large number of relying parties are querying it.

[3. RPKI Repository Delta Protocol Implementation](#)

[3.1. Informal Overview](#)

Certification Authorities (CA) in the RPKI use a repository server to publish their RPKI products, such as manifests, CRLs, signed certificates and RPKI signed objects. This repository server may be remote, or embedded in the CA engine itself. Certificates in the RPKI that use a repository server that supports this delta protocol include a special Subject Information Access (SIA) pointer referring to a notification file.

The notification file includes a globally unique session_id in the form of a version 4 UUID, and serial number that can be used by the Relying Party (RP) to determine if it and the repository are synchronised. Furthermore it includes a link to the most recent complete snapshot of current objects that are published by the repository server, and a list of links to delta files, for each revision starting at a point determined by the repository server, up to the current revision of the repository.

A RP that learns about a notification file location for the first time can download it, and then proceed to download the latest snapshot file, and thus create a local copy of the repository that is in sync with the repository server. The RP should remember the location of this notification file, the session_id and current serial number.

RPs are encouraged to re-fetch this notification file at regular intervals, but not more often than once per minute. After re-fetching the notification file, the RP may find that there are one or more delta files available that allow it to synchronise its local repository with the current state of the repository server. If no contiguous chain of deltas from RP's serial to the latest repository serial is available, or if the session_id has changed, the RP should perform a full resynchronisation instead.

As soon as the RP fetches new content in this way it should start a validation process. An example of a reason why a RP may not do this immediately is because it has learned of more than one notification location and it prefers to complete all its updates before validating.

The repository server may use caching infrastructure to reduce its load. It should be noted that snapshots and deltas for any given session_id and serial number contain an immutable record of the state of the repository server at a certain point in time. For this reason these files can be cached indefinitely. Notification files are polled by RPs to discover if updates exist, and for this reason notification files may not be cached for longer than one minute.

3.2. Certificate Authority Use

Certificate Authorities that use this delta protocol MUST include an instance of an SIA AccessDescription extension in resource certificates they produce, in addition to the ones defined in [\[RFC6487\]](#),

```
AccessDescription ::= SEQUENCE {  
    accessMethod OBJECT IDENTIFIER,  
    accessLocation GeneralName }
```

This extension MUST use an accessMethod of id-ad-rpkiNotify, see: [\[IANA-AD-NUMBERS\]](#),

```
id-ad OBJECT IDENTIFIER ::= { id-pkix 48 }  
id-ad-rpkiNotify OBJECT IDENTIFIER ::= { id-ad 13 }
```


The accessLocation MUST be an HTTPS URI as defined in [[RFC2818](#)], that will point to the update notification file for the repository server that publishes the products of this CA certificate.

Relying Parties that do not support this delta protocol MUST NOT reject a CA certificate merely because it has an SIA extension containing this new kind of AccessDescription.

[3.3.](#) Repository Server Use

[3.3.1.](#) Initialisation

When the repository server initialises it must perform the following actions:

The server MUST generate a new random version 4 UUID to be used as the session_id

The server MUST then generate a snapshot file for serial number ONE for this new session that includes all currently known published objects that the repository server is responsible for. Note that this snapshot file MAY contain zero publish elements at this point if no objects have been submitted for publication yet.

This snapshot file MUST be made available at a URL that is unique to this session_id and serial number, so that it can be cached indefinitely.

The format and caching concerns for snapshot files are explained in more detail in [Section 3.5.2](#).

After the snapshot file has been published the repository server MUST publish a new notification file that contains the new session_id, has serial number ONE, has one reference to the snapshot file that was just published, and that contains no delta references.

The format and caching concerns for update notification files are explained in more detail in [Section 3.5.1](#).

[3.3.2.](#) Publishing Updates

Whenever the repository server receives updates from a CA it SHOULD generate new snapshot and delta files. However, if a publication server services a large number of CAs it MAY choose to combine updates from multiple CAs. If a publication server combines updates in this way, it MUST NOT postpone publishing for longer than one minute.

Updates must be processed as follows:

- o The new repository serial number MUST be one greater than the current repository serial number.
- o A new delta file MUST be generated for this new serial. This delta file MUST include all new, replaced and withdrawn objects for multiple CAs if applicable, as a single change set.
- o This delta file MUST be made available at a URL that is unique to the current session_id and serial number, so that it can be cached indefinitely.
- o The format and caching concerns for delta files are explained in more detail in [Section 3.5.3](#).
- o The repository server MUST also generate a new snapshot file for this new serial. This file MUST contain all "publish" elements for all current objects.
- o The snapshot file MUST be made available at a URL that is unique to this session and new serial, so that it can be cached indefinitely.
- o The format and caching concerns for snapshot files are explained in more detail in [Section 3.5.2](#).
- o The update notification file SHOULD be kept small, and in order to do so the repository server needs to make a decision about which delta files to support. Any older delta files that, when combined with all more recent delta files, will result in total size of deltas exceeding the size of the snapshot, MUST be excluded.
- o The server MAY also exclude more recent delta files if it finds that their usage by a small number of RPs that would be forced to perform a full synchronisation is outweighed by the performance penalty for all RPs in having a large update notification file. However the repository server SHOULD include all deltas for the last two hours.
- o A new notification file MUST now be created by the repository server. This new notification file MUST include a reference to the new snapshot file, and all delta files selected in the previous steps.
- o The format and caching concerns for update notification files are explained in more detail in [Section 3.5.1](#).

If the repository server is not capable of performing the above for some reason, then it MUST perform a full re-initialisation, as explained above in [Section 3.3.1](#).

3.4. Relying Party Use

3.4.1. Processing the Update Notification File

When a Relying Party (RP) performs RPKI validation and learns about a valid certificate with an SIA entry for the RRDP protocol, it SHOULD prefer to use this protocol as follows.

The RP SHOULD download the update notification file, unless an update notification file was already downloaded and processed from the same location in this validation run.

The RP MAY use a "User-Agent" header explained in [section 5.5.3. of \[RFC7231\]](#) to identify the name and version of the RP software used. This is not required, but would be useful to help track capabilities of Relying Parties in the event of changes to the RPKI standards.

When the RP downloads an update notification file it MUST verify the file format and validation steps described in [section Section 3.5.1.3](#). If this verification fails, the file MUST be rejected.

The RP MUST verify whether the session_id in this update notification file matches the last known session_id for this update notification file location. If the session_id matches the last known session_id, then an RP MAY download and process missing delta files as described in [section Section 3.4.3](#), provided that all delta files for serial numbers between the last processed serial number and the current serial number in the notification file can be processed this way.

If the session_id was not previously known, or if delta files could not be used, then the RP MUST update its last known session_id to this session_id and download and process snapshot file on the update notification file as described in [section Section 3.4.2](#).

If neither update notification file and one snapshot file or delta files could be processed this way, the RP MUST issue an operator error, and SHOULD use an alternate repository retrieval mechanism if it is available.

3.4.2. Processing a Snapshot File

When the RP downloads a snapshot file it MUST verify the file format and validation steps described in [Section 3.5.2.3](#). If this verification fails, the file MUST be rejected.

Furthermore the RP MUST verify that the hash of the contents of this file matches the hash on the update notification file that referenced it. In case of a mismatch of this hash, the file MUST be rejected.

If an RP retrieved a snapshot file that is valid according to the above criteria, it should perform the following actions:

The RP MUST verify that the session_id matches the session_id of the notification file. If the session_id values do not match the file MUST be rejected.

The RP MUST verify that the serial number of this snapshot file is greater than the last processed serial number for this session_id. If this fails the file MUST be rejected.

The RP SHOULD then add all publish elements to a local storage and update its last processed serial number to the serial number of this snapshot file.

3.4.3. Processing Delta Files

If an update notification file contains a contiguous chain of links to delta files from the last processed serial number to the current serial number, then RPs MUST attempt to download and process all delta files in order of serial number as follows.

When the RP downloads a delta file it MUST verify the file format and perform validation steps described in [Section 3.5.3.3](#). If this verification fails, the file MUST be rejected.

Furthermore the RP MUST verify that the hash of the contents of this file matches the hash on the update notification file that referenced it. In case of a mismatch of this hash, the file MUST be rejected.

If an RP retrieved a delta file that is valid according to the above criteria, it should perform the following actions:

The RP MUST verify that the session_id matches the session_id of the notification file. If the session_id values do not match the file MUST be rejected.

The RP MUST verify that the serial number of this delta file is exactly one greater than the last processed serial number for this session_id, and if not this file MUST be rejected.

The RP SHOULD add all publish elements to a local storage and update its last processed serial number to the serial number of this snapshot file.

The RP SHOULD NOT remove objects from its local storage solely because it encounters a "withdraw" element, because this would enable a publication server to withdraw any object without the signing Certificate Authority consent. Instead it is RECOMMENDED that a RP uses additional strategies to determine if an object is still relevant for validation before removing it from its local storage.

3.4.4. Polling the Update Notification File

Once a Relying Party has learned about the location, session_id and last processed serial number of repository that uses the RRDP protocol, the RP MAY start polling the repository server for updates. However the RP MUST NOT poll for updates more often than once every 1 minute, and in order to reduce data usage RPs MUST use the "If-Modified-Since" header explained in [section 3.3 of \[RFC7232\]](#) in requests.

If an RP finds that updates are available it SHOULD download and process the file as described in [Section 3.4.1](#), and initiate a new validation process. A detailed description of the validation process itself is out of scope of this document.

3.5. File Definitions

3.5.1. Update Notification File

3.5.1.1. Purpose

The update notification file is used by RPs to discover whether any changes exist between the state of the repository and the RP's cache. It describes the location of the files containing the snapshot and incremental deltas which can be used by the RP to synchronise with the repository.

3.5.1.2. Cache Concerns

A repository server MAY use caching infrastructure to cache the notification file and reduce the load of HTTPS requests. However, since this file is used by RPs to determine whether any updates are

available the repository server MUST ensure that this file is not cached for longer than 1 minute. An exception to this rule is that it is better to serve a stale notification file, then no notification file.

How this is achieved exactly depends on the caching infrastructure used. In general a repository server may find certain HTTP headers to be useful, such as: Cache-Control: max-age=60. Another approach can be to have the repository server push out new versions of the notification file to the caching infrastructure when appropriate.

Relying Parties SHOULD NOT cache the notification file for longer than 1 minute, regardless of the headers set by the repository server or CDN.

3.5.1.3. File Format and Validation

Example notification file:

```
<notification xmlns="http://www.ripe.net/rpki/rrdp"
  version="1"
  session_id="9df4b597-af9e-4dca-bdda-719cce2c4e28"
  serial="3">
  <snapshot uri="https://host/9d-8/3/snapshot.xml" hash="AB"/>
  <delta serial="3" uri="https://host/9d-8/3/delta.xml" hash="CD"/>
  <delta serial="2" uri="https://host/9d-8/2/delta.xml" hash="EF"/>
</notification>
```

Note: URIs and hash values in this example are shortened because of formatting.

The following validation rules must be observed when creating or parsing notification files:

- o A RP MUST reject any update notification file that is not well-formed, or which does not conform to the RELAX NG schema outlined in [Section 3.5.4](#) of this document.
- o The XML namespace MUST be <http://www.ripe.net/rpki/rrdp>
- o The encoding MUST be US-ASCII
- o The version attribute in the notification root element MUST be 1
- o The session_id attribute MUST be a random version 4 UUID unique to this session

- o The serial attribute must be an unbounded, unsigned positive integer in decimal format indicating the current version of the repository.
- o The notification file MUST contain exactly one 'snapshot' element for the current repository version.
- o If delta elements are included they MUST form a contiguous sequence of serial numbers starting at a revision determined by the repository server, up to the serial number mentioned in the notification element.
- o The hash attribute in snapshot and delta elements must be the hexadecimal encoding of the SHA-256 hash of the referenced file. The RP MUST verify this hash when the file is retrieved and reject the file if the hash does not match.

3.5.2. Snapshot File

3.5.2.1. Purpose

A snapshot is intended to reflect the complete and current contents of the repository for a specific session and version. Therefore it MUST contain all objects from the repository current as of the time of the publication.

3.5.2.2. Cache Concerns

A snapshot reflects the content of the repository at a specific point in time, and for that reason can be considered immutable data. Snapshot files MUST be published at a URL that is unique to the specific session and serial.

Because these files never change, they MAY be cached indefinitely. However, in order to prevent that these files use a lot of space in caching infrastructure it is RECOMMENDED that a limited interval is used in the order of hours or days.

To avoid race conditions where an RP downloads a notification file moments before it's updated, Repository Servers SHOULD retain old snapshot files for at least 5 minutes after a new notification file is published.

3.5.2.3. File Format and Validation

Example snapshot file:


```
<snapshot xmlns="http://www.ripe.net/rpki/rrdp"
  version="1"
  session_id="9df4b597-af9e-4dca-bdda-719cce2c4e28"
  serial="2">
  <publish uri="rsync://rpki.ripe.net/Alice/Bob.cer">
    ZXhbbXBsZTE=
  </publish>
  <publish uri="rsync://rpki.ripe.net/Alice/Alice.mft">
    ZXhbbXBsZTI=
  </publish>
  <publish uri="rsync://rpki.ripe.net/Alice/Alice.crl">
    ZXhbbXBsZTM=
  </publish>
</snapshot>
```

The following rules must be observed when creating or parsing snapshot files:

- o A RP MUST reject any snapshot file that is not well-formed, or which does not conform to the RELAX NG schema outlined in [Section 3.5.4](#) of this document.
- o The XML namespace MUST be <http://www.ripe.net/rpki/rrdp>.
- o The encoding MUST be US-ASCII.
- o The version attribute in the notification root element MUST be 1
- o The session_id attribute MUST match the expected session_id in the reference in the notification file.
- o The serial attribute MUST match the expected serial in the reference in the notification file.
- o Note that the publish element is defined in the publication protocol [[I-D.ietf-sidr-publication](#)]

[3.5.3. Delta File](#)

[3.5.3.1. Purpose](#)

An incremental delta file contains all changes for exactly one serial increment of the repository server. In other words a single delta will typically include all the new objects, updated objects and withdrawn objects that a Certification Authority sent to the repository server. In its simplest form the update could concern only a single object, but it is recommended that CAs send all changes

for one of their key pairs: i.e. updated objects as well as a new manifest and CRL as one atomic update message.

3.5.3.2. Cache Concerns

Deltas reflect the difference between two consecutive versions of a repository for a given session. For that reason deltas can be considered immutable data. Delta files MUST be published at a URL that is unique to the specific session and serial.

Because these files never change, they MAY be cached indefinitely. However, in order to prevent these files from using a lot of space in caching infrastructure it is RECOMMENDED that a limited interval is used in the order of hours or days.

To avoid race conditions where an RP downloads a notification file moments before it's updated, Repository Servers SHOULD retain old delta files for at least 5 minutes after they are no longer included in the latest notification file.

3.5.3.3. File Format and Validation

Example delta file:

```
<delta xmlns="http://www.ripe.net/rpki/rrdp"
  version="1"
  session_id="9df4b597-af9e-4dca-bdda-719cce2c4e28"
  serial="3">
  <publish uri="rsync://rpki.ripe.net/repo/Alice/Alice.mft"
    hash="50d8...545c">
    ZXhhbXBsZTQ=
  </publish>
  <publish uri="rsync://rpki.ripe.net/repo/Alice/Alice.crl"
    hash="5fb1...6a56">
    ZXhhbXBsZTU=
  </publish>
  <withdraw uri="rsync://rpki.ripe.net/repo/Alice/Bob.cer"
    hash="caeb...15c1"/>
</delta>
```

Note that a formal RELAX NG specification of this file format is included later in this document. A RP MUST NOT process any delta file that is incomplete or not well-formed.

The following validation rules must be observed when creating or parsing delta files:

- o A RP MUST reject any delta file that is not well-formed, or which does not conform to the RELAX NG schema outlined in [Section 3.5.4](#) of this document.
- o The XML namespace MUST be <http://www.ripe.net/rpki/rrdp>.
- o The encoding MUST be US-ASCII.
- o The version attribute in the delta root element MUST be 1
- o The session_id attribute MUST be a random version 4 UUID unique to this session
- o The session_id attribute MUST match the expected session_id in the reference in the notification file.
- o The serial attribute MUST match the expected serial in the reference in the notification file.
- o Note that the publish and withdraw elements are defined in the publication protocol [[I-D.ietf-sidr-publication](#)]

[3.5.4.](#) XML Schema

The following is a RELAX NG compact form schema describing version 1 of this protocol.

```
#
# RelaxNG schema for RPKI Repository Delta Protocol (RRDP).
#

default namespace = "http://www.ripe.net/rpki/rrdp"

version = xsd:positiveInteger { maxInclusive="1" }
serial  = xsd:nonNegativeInteger
uri     = xsd:anyURI
uuid    = xsd:string           { pattern = "[\0-9a-fA-F]+" }
hash    = xsd:string           { pattern = "[0-9a-fA-F]+" }
base64  = xsd:base64Binary

# Notification file: lists current snapshots and deltas

start |= element notification {
  attribute version { version },
  attribute session_id { uuid },
  attribute serial { serial },
  element snapshot {
    attribute uri { uri },
```



```
    attribute hash { hash }
  },
  element delta {
    attribute serial { serial },
    attribute uri    { uri },
    attribute hash   { hash }
  }*
}

# Snapshot segment: think DNS AXFR.

start |= element snapshot {
  attribute version    { version },
  attribute session_id { uuid },
  attribute serial     { serial },
  element publish      {
    attribute uri { uri },
    base64
  }*
}

# Delta segment: think DNS IXFR.

start |= element delta {
  attribute version    { version },
  attribute session_id { uuid },
  attribute serial     { serial },
  delta_element+
}

delta_element |= element publish {
  attribute uri { uri },
  attribute hash { hash }?,
  base64
}

delta_element |= element withdraw {
  attribute uri { uri },
  attribute hash { hash }
}

# Local Variables:
# indent-tabs-mode: nil
# comment-start: "# "
# comment-start-skip: "#[ \t]*"
# End:
```


4. HTTPS considerations

It is RECOMMENDED that Relying Parties and Publication Servers follow the Best Current Practices outlined in [[RFC7525](#)] on the use of HTTP over TLS (https).

Note that a Man-in-the-Middle (MITM) cannot produce validly signed RPKI data, but they can perform withhold or replay attacks targeting an RP, and keep the RP from learning about changes in the RPKI. Because of this RPs SHOULD do TLS certificate and host name validation when they fetch from an RRDP Publication Server

However, such validation issues are often due to configuration errors, or a lack of a common TLS trust anchor. In these cases it would be better that the RP retrieves the signed RPKI data regardless, and performs validation on it.

Therefore RPs SHOULD log any TLS certificate or host name validation issues they find, so that an operator can investigate the cause. But the RP MUST continue to retrieve the data. The RP MAY choose to log this issue only when fetching the notification update file, but not when it subsequently fetches snapshot or delta files from the same host. Furthermore the RP MAY provide a way for operators to accept untrusted connections for a given host, after the cause has been identified.

5. Security Considerations

RRDP deals exclusively with transfer of RPKI objects from a repository server to a relying party. The trust relation between a CA and its repository server is out of scope for this document. However, it should be noted the from a relying party point of view all RPKI objects (certificates, CRLs, and CMS-wrapped objects) are already covered by object security mechanisms including signed manifests. This allows validation of these objects even though the repository server itself is not trusted. This document makes no change to RPKI validation procedures per se.

The original RPKI transport mechanism is rsync, which offers no channel security mechanism. RRDP replaces the use of rsync by HTTPS; while the channel security mechanism underlying RRDP (HTTPS) is not a cure-all, it does make some forms of denial of service attack more difficult for the attacker. HTTPS issues are discussed in more detail in [Section 4](#).

Supporting both RRDP and rsync necessarily increases the number of opportunities for a malicious RPKI CA to perform denial of service attacks on relying parties, by expanding the number of URIs which the

RP may need to contact in order to complete a validation run. However, other than the relative cost of HTTPS versus rsync, adding RRDP to the mix does not change this picture significantly: with either RRDP or rsync a malicious CA can supply an effectively infinite series of URIs for the RP to follow. The only real solution to this is for the RP to apply some kind of bound to the amount of work it is willing to do. Note also that the attacker in this scenario must be an RPKI CA, since otherwise the normal RPKI object security checks would reject the malicious URIs.

Processing costs for objects retrieved using RRDP may be somewhat different from the same objects retrieved using rsync: because RRDP treats an entire set of changes as a unit (one "delta"), it may not be practical to start processing any of the objects in the delta until the entire delta has been received. With rsync, by contrast, incremental processing may be easy, but the overall cost of transfer may be higher, as may be the number of corner cases in which the RP retrieves some but not all of the updated objects. Overall, RRDP's behavior is closer to a proper transactional system, which (probably) leads to an overall reliability increase.

RRDP is designed to scale much better than rsync. In particular, RRDP is designed to allow use of HTTPS caching infrastructure to reduce load on primary publication servers and increase resilience against denial of service attacks on the RPKI publication service.

6. IANA Considerations

IANA is requested to update the reference for id-ad-rpkiNotify to this document in the PKIX Access Descriptor registry [[IANA-AD-NUMBERS](#)].

7. Acknowledgements

The authors would like to thank David Mandelberg for reviewing this document.

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-09](#) (work in progress), September 2016.

[IANA-AD-NUMBERS]

"SMI Security for PKIX Access Descriptor",
<<http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-1.3.6.1.5.5.7.48>>.

[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>>.

[RFC2818] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.

[RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", [RFC 6481](#), DOI 10.17487/RFC6481, February 2012, <<http://www.rfc-editor.org/info/rfc6481>>.

[RFC6486] Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure (RPKI)", [RFC 6486](#), DOI 10.17487/RFC6486, February 2012, <<http://www.rfc-editor.org/info/rfc6486>>.

[RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", [RFC 6487](#), DOI 10.17487/RFC6487, February 2012, <<http://www.rfc-editor.org/info/rfc6487>>.

[RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", [RFC 6488](#), DOI 10.17487/RFC6488, February 2012, <<http://www.rfc-editor.org/info/rfc6488>>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", [RFC 7231](#), DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", [RFC 7232](#), DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,
"Recommendations for Secure Use of Transport Layer
Security (TLS) and Datagram Transport Layer Security
(DTLS)", [BCP 195](#), [RFC 7525](#), DOI 10.17487/RFC7525, May
2015, <<http://www.rfc-editor.org/info/rfc7525>>.

Authors' Addresses

Tim Bruijnzeels
RIPE NCC

Email: tim@ripe.net

Oleg Muravskiy
RIPE NCC

Email: oleg@ripe.net

Bryan Weber
Cobenian

Email: bryan@cobenian.com

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net

