SIDR Operations
Internet-Draft
Intended status: Informational
Expires: July 18, 2017

O. Muravskiy
T. Bruijnzeels
RIPE NCC

January 14, 2017

# RPKI Certificate Tree Validation by the RIPE NCC RPKI Validator
draft-ietf-sidrops-rpki-tree-validation-00

## Abstract

This document describes the approach to validate the content of the RPKI certificate tree, as used by the RIPE NCC RPKI Validator. This approach is independent of a particular object retrieval mechanism. This allows it to be used with repositories available over the rsync protocol, the RPKI Repository Delta Protocol, and repositories that use a mix of both.

## 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 July 18, 2017.

## Copyright Notice

Table of Contents

## 1. Scope of this document

This document describes how the RIPE NCC RPKI Validator version 2.23 has been implemented. Source code to this software can be found here: [github]. The purpose of this document is to provide transparency to users of (and contributors to) this software tool, as well as serve to be subjected to scrutiny by the SIDR Operations Working Group. It is not intended as a document that describes a standard or best practices on how validation should be done in general.

## 2. Introduction

In order to use information published in RPKI repositories, Relying Parties (RP) need to retrieve and validate the content of certificates, certificate revocation lists (CRLs), and other RPKI signed objects. To validate a particular object, one must ensure that all certificates in the certificate chain up to the Trust Anchor (TA) are valid. Therefore the validation of a certificate tree is performed top-down, starting from the TA certificate and descending down the certificate chain, validating every encountered certificate and its products. The result of this process is a list of all encountered RPKI objects with a validity status attached to each of them. These results may later be used by a Relying Party in taking routing decisions, etc.

Traditionally RPKI data is made available to RPs through the repositories [RFC6481] accessible over [rsync] protocol. Relying parties are advised to keep a local copy of repository data, and perform regular updates of this copy from the repository (Section 5 of [RFC6481]). The RPKI Repository Delta Protocol [I-D.ietf-sidr-delta-protocol] introduces another method to fetch repository data and keep the local copy up to date with the repository.

This document describes how the RIPE NCC RPKI Validator discovers RPKI objects to download, builds certificate paths, and validates RPKI objects, independently from what repository access protocol is used. To achieve this, it puts downloaded RPKI objects in an object store, where each RPKI object can be found by its URI, the hash of its content, value of its Authority Key Identifier (AKI) extension, or a combination of these. It also keeps track of the download and the validation time for every object, to decide which locally stored objects are not used in the RPKI tree validation and could be removed.

# 3. General Considerations

## 3.1. Hash comparisons

This algorithm relies on the properties of the file hash algorithm (defined in [RFC6485]) to compute the hash of repository objects. It assumes that any two objects for which the hash value is the same, are identical.

The hash comparison is used when matching objects in the repository with entries on the manifest (Section 4.2.2), and when looking up objects in the object store (Section 6).

## 3.2. Discovery of RPKI objects issued by a CA

There are several possible ways of discovering products of a CA certificate: one could use all objects located in a repository directory designated as a publication point for a CA, or only objects mentioned on the manifest located at that publication point (see Section 6 of [RFC6486]), or use all objects whose AKI extension matches the Subject Key Identifier (SKI) extension (Section 4.2.1 of [RFC5280]) of a CA certificate.

For publication points whose content is consistent with the manifest and issuing certificate all of these approaches should produce the same result. For inconsistent publication points the results might be different. Section 6 of [RFC6486] leaves the decision on how to deal with inconsistencies to a local policy.

The implementation described here does not rely on content of repository directories, but uses the Authority Key Identifier (AKI) extension of a manifest and a certificate revocation list (CRL) to find in an object store (Section 6) a manifest and a CRL issued by a particular Certification Authority (CA) (see Section 4.2.1). It further uses the hashes of manifest's fileList entries (Section 4.2.1 of [RFC6486]) to find other objects issued by the CA, as described in Section 4.2.2.

## 3.3. Manifest entries versus repository content

Since the current set of RPKI standards requires use of the manifest [RFC6486] to describe the content of a publication point, this implementation requires strict consistency between the publication point content and manifest content. (This is a more stringent requirement than established in [RFC6486].) Therefore it will not process objects that are found in the publication point but do not match any of the entries of that publication point's manifest (see Section 4.2.2). It will also issue warnings for all found

mismatches, so that the responsible operators could be made aware of inconsistencies and fix them.

# 4. Top-down Validation of a Single Trust Anchor Certificate Tree

1. The validation of a Trust Anchor (TA) certificate tree starts from its TA certificate. To retrieve the TA certificate, a Trust Anchor Locator (TAL) object is used, as described in Section 4.1.

2. If the TA certificate is retrieved, it is validated according to Section 7 of [RFC6487] and Section 2.2 of [RFC7730]. Otherwise the validation of certificate tree is aborted and an error is issued.

3. If the TA certificate is valid, then all its subordinate objects are validated as described in Section 4.2. Otherwise the validation of certificate tree is aborted and an error is issued.

4. For each repository object that was validated during this validation run, its validation timestamp is updated in the object store (see Section 6.1.7).

5. Outdated objects are removed from the store as described in Section 4.3. This completes the validation of the TA certificate tree.

## 4.1. Fetching the Trust Anchor Certificate Using the Trust Anchor Locator

The following steps are performed in order to fetch a Trust Anchor Certificate:

1. (Optional) If the Trust Anchor Locator contains a "prefetch.uris" field, pass the URIs contained in that field to the fetcher (see Section 5.1.1). (This field is a non-standard addition to the TAL format. It helps fetching non-hierarchical rsync repositories more efficiently.)

2. Extract the first TA certificate URI from the TAL's URI section (see Section 2.1 of [RFC7730]) and pass it to the object fetcher (Section 5.1.2). If the fetcher returns an error, repeat this step for every URI in the URI section, until no error is encountered, or no more URIs left.

3. Retrieve from the object store (see Section 6.1.3) all certificate objects, for which the URI matches the URI extracted from the TAL in the previous step, and the public key matches the

subjectPublicKeyInfo extension of the TAL (see Section 2.1 of [RFC7730]).

4. If no, or more than one such objects are found, issue an error and abort certificate tree validation process with an error. Otherwise, use the single found object as the Trust Anchor certificate.

## 4.2. CA Certificate Validation

The following steps describe the validation of a single CA Resource certificate:

1. If both the caRepository (Section 4.8.8.1 of [RFC6487]), and the id-ad-rpkiNotify (Section 3.2 of [I-D.ietf-sidr-delta-protocol]) SIA pointers are present in the CA certificate, use a local policy to determine which pointer to use. Extract the URI from the selected pointer and pass it to the object fetcher (see Section 5.1.1).

2. For the CA certificate, find the current manifest and certificate revocation list (CRL), using the procedure described in Section 4.2.1. If no such manifest and CRL could be found, stop validation of this certificate, consider it invalid, and issue an error.

3. Compare the URI found in the id-ad-rpkiManifest field (Section 4.8.8.1 of [RFC6487]) of the SIA extension of the certificate with the URI of the manifest found in the previous step. If they are different, issue a warning, but continue validation process using this manifest object. (This warning indicates that there is a mismatch between the expected and the actual location of an object in a repository. See Section 9 for the explanation of this mismatch and the decision taken.)

4. Perform manifest entries discovery and validation as described in Section 4.2.2.

5. Validate all resource certificate objects found on the manifest, using the CRL object found on the manifest, according to Section 7 of [RFC6487].

6. Validate all ROA objects found on the manifest, using the CRL object found on the manifest, according to Section 4 of [RFC6482].

7. Validate all Ghostbusters Record objects found on the manifest, using the CRL object found on the manifest, according to Section 7 of [RFC6493].

8. For every valid CA certificate object found on the manifest, apply the procedure described in this section (Section 4.2), recursively, provided that this CA certificate (identified by its SKI) has not yet been validated during current tree validation run.

#### 4.2.1. Finding the most recent valid manifest and CRL

1. Fetch from the store (see Section 6.1.4) all objects of type manifest, whose certificate's AKI extension matches the SKI of the current CA certificate. If no such objects are found, stop processing the current CA certificate and issue an error.

2. Find among found objects the manifest object with the highest manifestNumber field (Section 4.2.1 of [RFC6486]), for which all following conditions are met:

   * There is only one entry in the manifest for which the store contains exactly one object of type CRL, the hash of which matches the hash of the entry.

   * The manifest's certificate AKI equals the above CRL's AKI.

   * The above CRL is a valid object according to Section 6.3 of [RFC5280].

   * The manifest is a valid object according to Section 4.4 of [RFC6486], and its EE certificates is not in the CRL found above.

3. If there is an object that matches above criteria, consider this object to be the valid manifest, and the CRL found at the previous step - the valid CRL for the current CA certificate's publication point.

4. Report an error for every other manifest with a number higher than the number of the valid manifest.

#### 4.2.2. Manifest entries validation

For every entry in the manifest object:

1. Construct an entry's URI by appending the entry name to the current CA's publication point URI.

2. Get all objects from the store whose hash attribute equals entry's hash (see Section 6.1.2).

3. If no such objects are found, issue an error for this manifest entry and progress to the next entry. This case indicates that the repository does not have an object at the location listed in the manifest, or that the object's hash does not match the hash listed in the manifest.

4. For every found object, compare its URI with the URI of the manifest entry.

   * For every object with a non-matching URI issue a warning. This case indicates that the object from the manifest entry is (also) found at a different location in a (possibly different) repository.

   * If no objects with a matching URI are found, issue a warning. This case indicates that there is no object found in the repository at the location listed in the manifest entry (but there is at least one matching object found at a different location).

5. Use all found objects for further validation as per Section 4.2.

Please note that the above steps will not reject objects whose hash matches the hash listed in the manifest, but the URI does not. See Section 9.2 for additional information.

### 4.3. Object Store Cleanup

At the end of every TA tree validation some objects are removed from the store using the following rules:

1. Given all objects that were encountered during the current validation run, remove from the store (Section 6.1.6) all objects whose URI attribute matches the URI of one of the encountered objects, but the content's hash is different. This removes from the store objects that were replaced in the repository by their newer versions with the same URIs.

2. Remove from the store all objects that were last encountered during validation a long time ago (as specified by the local policy). This removes objects that do not appear on any valid manifest anymore (but possibly are still published in a repository).

3. Remove from the store all objects that were downloaded recently (as specified by the local policy), but have never been used in the validation process. This removes objects that have never appeared on any valid manifest.

Shortening the time interval used in step 2 will free more disk space used by the store, at the expense of downloading removed objects again if they are still published in the repository.

Extending the time interval used in step 3 will prevent repeated downloads of repository objects, with the risk that such objects, if created massively by mistake or by an adversary, will fill up local disk space, if they are not cleaned up promptly.

## 5. Remote Objects Fetcher

The fetcher is responsible for downloading objects from remote repositories (described in Section 3 of [RFC6481]) using rsync protocol ([rsync]), or RPKI Repository Delta Protocol (RRDP) ([I-D.ietf-sidr-delta-protocol]).

### 5.1. Fetcher Operations

For every visited URI the fetcher keeps track of the last time a successful fetch occurred.

#### 5.1.1. Fetch repository objects

This operation receives one parameter - a URI. For an rsync repository this URI points to a directory. For an RRDP repository it points to the repository’s notification file.

The fetcher performs following steps:

1. If data associated with the URI has been downloaded recently (as specified by the local policy), skip following steps.

2. Download remote objects using the URI provided (for an rsync repository use recursive mode). If the URI contains schema "https" and download has failed, issue a warning, replace "https" schema in the URI by "http", and try to download objects again, using the resulting URI.

3. If remote objects can not be downloaded, issue an error and skip following steps.

4. Perform syntactic verification of fetched objects. The type of every object (certificate, manifest, CRL, ROA, or Ghostbusters

record), is determined based on the object's filename extension (.cer, .mft, .crl, .roa, and .gbr, respectively). The syntax of the object is described in Section 4 of [RFC6487] for resource certificates, step 1 of Section 3 of [RFC6488] for signed objects, and specifically, Section 4 of [RFC6486] for manifests, [RFC5280] for CRLs, Section 3 of [RFC6482] for ROAs, and Section 5 of [RFC6493] for Ghostbusters records.

5. Put every downloaded and syntactically correct object in the object store (Section 6.1.1).

The time interval used in the step 1 should be chosen based on the acceptable delay in receiving repository updates.

#### 5.1.2. Fetch single repository object

This operation receives one parameter - a URI that points to an object in a repository.

The fetcher performs following operations:

1. Download remote object using the URI provided. If the URI contains "https" schema and download failed, issue a warning, replace "https" schema in the URI by "http", and try to download the object using the resulting URI.

2. If the remote object can not be downloaded, issue an error and skip following steps.

3. Perform syntactic verification of fetched object. The type of object (certificate, manifest, CRL, ROA, or Ghostbusters record), is determined based on the object's filename extension (.cer, .mft, .crl, .roa, and .gbr, respectively). The syntax of the object is described in Section 4 of [RFC6487] for resource certificates, step 1 of Section 3 of [RFC6488] for signed objects, and specifically, Section 4 of [RFC6486] for manifests, [RFC5280] for CRLs, Section 3 of [RFC6482] for ROAs, and Section 5 of [RFC6493] for Ghostbusters records.

4. If the downloaded object is not syntactically correct, issue an error and skip further steps.

5. Delete all objects from the object store (Section 6.1.5) whose URI matches the URI given.

6. Put the downloaded object in the object store (Section 6.1.1).

# 6. Local Object Store

## 6.1. Store Operations

### 6.1.1. Store Repository Object

Put given object in the store, along with its type, URI, hash, and AKI, if there is no record with the same hash and URI fields. Note that in the (unlikely) event of hash collision the given object will not replace the object in the store.

### 6.1.2. Get objects by hash

Retrieve all objects from the store whose hash attribute matches the given hash.

### 6.1.3. Get certificate objects by URI

Retrieve from the store all objects of type certificate, whose URI attribute matches the given URI.

### 6.1.4. Get manifest objects by AKI

Retrieve from the store all objects of type manifest, whose AKI attribute matches the given AKI.

### 6.1.5. Delete objects for a URI

For a given URI, delete all objects in the store with matching URI attribute.

### 6.1.6. Delete outdated objects

For a given URI and a list of hashes, delete all objects in the store with matching URI, whose hash attribute is not in the given list of hashes.

### 6.1.7. Update object’s validation time

For all objects in the store whose hash attribute matches the given hash, set the last validation time attribute to the given timestamp.

# 7. Acknowledgements

This document describes the algorithm as it is implemented by the software development team at the RIPE NCC. The authors would also like to acknowledge contributions by Carlos Martinez, Andy Newton, Rob Austein, and Stephen Kent.

## 8. IANA Considerations

This document has no actions for IANA.

## 9. Security Considerations

### 9.1. Hash collisions

This implementation will not detect possible hash collisions in the hashes of repository objects (calculated using the file hash algorithm specified in [RFC6485]). It considers objects with same hash values as identical.

### 9.2. Mismatch between the expected and the actual location of an object in the repository

According to Section 2 of [RFC6481], all objects issued by a particular CA certificate are expected to be located in one repository publication point, specified in the SIA extension of that CA certificate. The manifest object issued by that CA certificate enumerates all other issued objects, listing their file names and content hashes.

However, it is possible that an object whose content hash matches the hash listed in the manifest, has either a different file name, or is located at a different publication point in a repository.

On the other hand, all RPKI objects, either explicitly or within their embedded EE certificate, have an Authority Key Identifier extension that contains the key identifier of their issuing CA certificate. Therefore it is always possible to perform an RPKI validation of the object whose expected location does not match its actual location, provided that the certificate that matches the AKI of the object in question is known to the system that performs validation.

In case of a mismatch described above this implementation will not exclude an object from further validation merely because it’s actual location or file name does not match the expected location or file name. This decision was chosen because the actual location of a file in a repository is taken from the repository retrieval mechanism, which, in case of an rsync repository, does not provide any cryptographic security, and in case of an RRDP repository, provides only a transport layer security, with the fallback to unsecured transport. On the other hand, the manifest is an RPKI signed object, and its content could be verified in the context of the RPKI validation.

### 9.3. Manifest content versus publication point content

This algorithm uses the content of a manifest object to determine other objects issued by a CA certificate. It verifies that the manifest is located in the publication point designated in the CA Certificate's SIA extension. However, if there are other (not listed in the manifest) objects located in the same publication point directory, they are ignored, even if they might be valid and issued by the same CA certificate as the manifest. (This behavior is allowed, but not required, by [RFC6486].)

### 9.4. Storing of a TA certificate object before its complete validation

When fetching and storing a TA certificate to the object store, only a syntactic validation of a downloaded object is performed before newly downloaded object replaces the previously downloaded object in the object store (see Section 5.1.2). If an attacker will be able to replace a genuine TA certificate by a syntactically valid certificate object (either by manipulating the content of a repository, or by a man-in-the-middle attack), this implementation will discard previously downloaded genuine object, and replace it by a false object. Such false object will be detected later, but the validation of the whole RPKI tree under this TA will be aborted, as described in Section 4.

### 9.5. Possible denial of service

The store cleanup procedure described in Section 4.3 tries to minimise removal and subsequent re-fetch of objects that are published in a repository, but not used in the validation. Once such objects are removed from the remote repository, they will be discarded from the local object store after a period of time specified by a local policy. By generating an excessive amount of syntactically valid RPKI objects, a man-in-the-middle attack between a validating tool and a repository could force an implementation to fetch and store those objects in the object store before they are validated and discarded, leading to an out-of-memory or out-of-disk-space conditions, and, subsequently, a denial of service.

## 10. References

### 10.1. Normative References

[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, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[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>.

[RFC6482] Lepinski, M., Kent, S., and D. Kong, "A Profile for Route Origin Authorizations (ROAs)", RFC 6482, DOI 10.17487/RFC6482, February 2012, <http://www.rfc-editor.org/info/rfc6482>.

[RFC6485] Huston, G., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)", RFC 6485, DOI 10.17487/RFC6485, February 2012, <http://www.rfc-editor.org/info/rfc6485>.

[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>.

[RFC6493] Bush, R., "The Resource Public Key Infrastructure (RPKI) Ghostbusters Record", RFC 6493, DOI 10.17487/RFC6493, February 2012, <http://www.rfc-editor.org/info/rfc6493>.

[RFC7730] Huston, G., Weiler, S., Michaelson, G., and S. Kent, "Resource Public Key Infrastructure (RPKI) Trust Anchor Locator", RFC 7730, DOI 10.17487/RFC7730, January 2016, <http://www.rfc-editor.org/info/rfc7730>.

## 10.2. Informative References

[github] "RIPE NCC RPKI Validator on GitHub", <https://github.com/RIPE-NCC/rpki-validator>.

[I-D.ietf-sidr-delta-protocol] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "RPKI Repository Delta Protocol", draft-ietf-sidr-delta-protocol-04 (work in progress), September 2016.

[rsync] "Rsync home page", <https://rsync.samba.org>.

## Authors' Addresses

Oleg Muravskiy
RIPE NCC

Email: oleg@ripe.net

Tim Bruijnzeels
RIPE NCC

Email: tim@ripe.net