Internet Engineering Task Force (IETF)
Request for Comments: 8488
Category: Informational
ISSN: 2070-1721

O. Muravskiy
RIPE NCC
T. Bruijnzeels
NLnet Labs

December 2018

RIPE NCC's Implementation of Resource Public Key Infrastructure (RPKI) Certificate Tree Validation

Abstract

This document describes an approach to validating the content of the Resource Public Key Infrastructure (RPKI) certificate tree, as it is implemented in the RIPE NCC RPKI Validator. This approach is independent of a particular object retrieval mechanism, which allows it to be used with repositories available over the rsync protocol, the RPKI Repository Delta Protocol (RRDP), and repositories that use a mix of both.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8488.

Table of Contents

1. Introduction

This document describes how the RIPE NCC RPKI Validator version 2.25 has been implemented. Source code for this software can be found at [rpki-validator]. The purpose of this document is to provide transparency to users of (and contributors to) this software tool.

In order to use information published in RPKI repositories, Relying Parties (RPs) 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 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 an RP in making routing decisions, etc.

Traditionally, RPKI data is made available to RPs through the repositories [RFC6481] accessible over the rsync protocol [rsync]. RPs are advised to keep a local copy of repository data and perform regular updates of this copy from the repository (see Section 5 of [RFC6481]). The RRDP [RFC8182] 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 of 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, the value of its Authority Key Identifier (AKI) extension, or a combination of these. It also keeps track of the download and validation time for every object, to decide which locally stored objects are not used in the RPKI tree validation and could be removed.

2. General Considerations

2.1. Hash Comparisons

This algorithm relies on the collision resistance properties of the hash algorithm (defined in [RFC7935]) 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 3.2.2) and when looking up objects in the object store (Section 5).

2.2. Discovery of RPKI Objects Issued by a CA

There are several possible ways of discovering potential products of a Certification Authority (CA) certificate: one could 1) use all objects located in a repository directory designated as a publication point for a CA, 2) only use objects mentioned on the manifest located at that publication point (see Section 6 of [RFC6486]), or 3) use all known repository 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 CRL to find in an object store (Section 5) a manifest and a CRL issued by a particular CA (see Section 3.2.1). It further uses the hashes of the manifest's fileList entries (Section 4.2.1 of [RFC6486]) to find other objects issued by the CA, as described in Section 3.2.2.

2.3. Manifest Entries versus Repository Content

Since the current set of RPKI standards (see [RFC6481], [RFC6486], and [RFC6487]) 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 3.2.2). It will also issue warnings for all found mismatches, so that the responsible operators could be made aware of inconsistencies and fix them.

3. Top-Down Validation of a Single Trust Anchor Certificate Tree

When several Trust Anchors are configured, validation of their corresponding certificate trees is performed concurrently and independently from each other. For every configured Trust Anchor, the following steps are performed:

1. The validation of a 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 3.1.

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 the 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 3.2. Otherwise, the validation of the certificate tree is aborted and an error is issued.

4. For each repository object that was validated during this validation run, the validation timestamp is updated in the object store (see Section 5.1.7).

5. Outdated objects are removed from the store as described in Section 3.3. This completes the validation of the TA certificate tree.

3.1. 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 TAL contains a prefetch.uris field, pass the URIs contained in that field to the fetcher (see Section 4.1.1). (This field is a non-standard addition to the TAL format. It helps with 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 4.1.2). 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 are left.

3. From the object store (see Section 5.1.3), retrieve 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 such objects are found or if more than one such objects are found, issue an error and abort the certificate tree validation process with an error. Otherwise, use the single found object as the TA certificate.

3.2. 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 [RFC8182]) instances of an accessMethod are present in the Subject Information Access extension of 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 (that will then fetch all objects available from that repository; see Section 4.1.1).

2. For the CA certificate, find the current manifest and certificate revocation list (CRL) using the procedure described in Section 3.2.1. 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 the validation process using the manifest found in the previous step. (This warning indicates that there is a mismatch between the expected and the actual location of an object in a repository. See Section 7.3 for the explanation of this mismatch and the decision made.)

4. Perform discovery and validation of manifest entries as described in Section 3.2.2.

5. Validate all resource certificate objects found on the manifest using the CRL object:

   * If the strict validation option is enabled by the operator, the validation is performed according to Section 7 of [RFC6487].

   * Otherwise, the validation is performed according to Section 7 of [RFC6487] but with the exception of the resource certification path validation, which is performed according to Section 4.2.4.4 of [RFC8360].

   (Note that this implementation uses the operator configuration to decide which algorithm to use for path validation. It applies the selected algorithm to all resource certificates, rather than applying an appropriate algorithm per resource certificate based on the object identifier (OID) for the Certificate Policy found in that certificate, as specified in [RFC8360].)

6. Validate all Route Origin Authorization (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, recursively, provided that this CA certificate (identified by its SKI) has not yet been validated during current tree validation run.

3.2.1. Finding the Most Recent Valid Manifest and CRL

To find the most recent issued manifest and CRL objects of a particular CA certificate, the following steps are performed:

1. From the store (see Section 5.1.4), fetch 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. Among found objects, find 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 certificate is not in the CRL found above.

3. If there is an object that matches the above criteria, consider this object to be the valid manifest, and consider the CRL found at the previous step to be 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.

3.2.2. Validating Manifest Entries

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 the entry's hash (see Section 5.1.2).

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 3.2.

Please note that the above steps will not reject objects whose hash matches the hash listed in the manifest but whose URI does not. See Section 7.3 for additional information.

3.3. 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 5.1.6) all objects whose URI attribute matches the URI of one of the encountered objects but whose content's hash does not match the hash of any of the encountered objects. 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 that 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 unused repository objects. However, it will also extend the interval at which unused objects are removed. This creates a risk that such objects will fill up all available disk space if a large enough amount of such objects is published in the repository (either by mistake or with a malicious intent).

4. Remote Objects Fetcher

The fetcher is responsible for downloading objects from remote repositories (described in Section 3 of [RFC6481]) using the rsync protocol [rsync] or RRDP [RFC8182].

4.1. Fetcher Operations

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

4.1.1. 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 follows these steps:

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

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

3. If remote objects cannot be downloaded, issue an error and skip the 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, 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 5.1.1).

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

4.1.2. Fetch Single Repository Object

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

The fetcher follows these steps:

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

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

3. Perform syntactic verification of the 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, 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 5.1.5) whose URI matches the URI given.

6. Put the downloaded object in the object store (Section 5.1.1).

5. Local Object Store

5.1. Store Operations

5.1.1. Store Repository Object

Put the given object in the store 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.

5.1.2. Get Objects by Hash

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

5.1.3. Get Certificate Objects by URI

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

5.1.4. Get Manifest Objects by AKI

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

5.1.5. Delete Objects for a URI

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

5.1.6. Delete Outdated Objects

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

5.1.7. 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.

6. IANA Considerations

This document has no IANA actions.

7. Security Considerations

7.1. Hash Collisions

This implementation will not detect possible hash collisions in the hashes of repository objects (calculated using the file hash algorithm specified in [RFC7935]). It considers objects with same hash values to be identical.

7.2. Algorithm Agility

This implementation only supports hash algorithms and key sizes specified in [RFC7935]. Algorithm agility described in [RFC6916] is not supported.

7.3. Mismatch between the Expected and 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 filenames and content hashes.

However, it is possible that an object whose content hash matches the hash listed in the manifest either has a different filename 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 AKI 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 the case of a mismatch as described above, this implementation will not exclude an object from further validation merely because its actual location or filename does not match the expected location or filename. This decision was made because the actual location of a file in a repository is taken from the repository retrieval mechanism, which, in the case of an rsync repository, does not provide any cryptographic security, and in the 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.

7.4. 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 as the manifest. (This RP behavior is allowed, but not required, by [RFC6486].)

7.5. Possible Denial of Service

The store cleanup procedure described in Section 3.3 tries to minimize 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 (see Section 4.1.1) before they are validated and discarded, leading to out-of-memory or out-of-disk-space conditions and, subsequently, a denial of service.

8. References

8.1. 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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6482>.

[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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6493>.

[RFC6916] Gagliano, R., Kent, S., and S. Turner, "Algorithm Agility Procedure for the Resource Public Key Infrastructure (RPKI)", BCP 182, RFC 6916, DOI 10.17487/RFC6916, April 2013, <https://www.rfc-editor.org/info/rfc6916>.

[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, <https://www.rfc-editor.org/info/rfc7730>.

[RFC7935] Huston, G. and G. Michaelson, Ed., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure", RFC 7935, DOI 10.17487/RFC7935, August 2016, <https://www.rfc-editor.org/info/rfc7935>.

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

[RFC8360] Huston, G., Michaelson, G., Martinez, C., Bruijnzeels, T., Newton, A., and D. Shaw, "Resource Public Key Infrastructure (RPKI) Validation Reconsidered", RFC 8360, DOI 10.17487/RFC8360, April 2018, <https://www.rfc-editor.org/info/rfc8360>.

8.2. Informative References

[rpki-validator] "RIPE-NCC/rpki-validator source code", <https://github.com/RIPE-NCC/rpki-validator>.

[rsync] "rsync", October 2018, <https://rsync.samba.org>.

Acknowledgements

This document describes the algorithm as it is implemented by the software development team at the RIPE NCC, which, over time, included Mikhail Puzanov, Erik Rozendaal, Miklos Juhasz, Misja Alma, Thiago da Cruz Pereira, Yannis Gonianakis, Andrew Snare, Varesh Tapadia, Paolo Milani, Thies Edeling, Hans Westerbeek, Rudi Angela, and Constantijn Visinescu. The authors would also like to acknowledge contributions by Carlos Martinez, Andy Newton, Rob Austein, and Stephen Kent.

Authors' Addresses

Oleg Muravskiy
RIPE NCC

Email: oleg@ripe.net
URI: https://www.ripe.net/

Tim Bruijnzeels
NLnet Labs

Email: tim@nlnetlabs.nl
URI: https://www.nlnetlabs.nl/