

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

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Scope of this document	3
2. Introduction	3
3. General Considerations	4
3.1. Hash comparisons	4
3.2. Discovery of RPKI objects issued by a CA	4
3.3. Manifest entries versus repository content	4
4. Top-down Validation of a Single Trust Anchor Certificate Tree	5
4.1. Fetching the Trust Anchor Certificate Using the Trust Anchor Locator	5
4.2. CA Certificate Validation	6
4.2.1. Finding the most recent valid manifest and CRL	7
4.2.2. Manifest entries validation	7
4.3. Object Store Cleanup	8
5. Remote Objects Fetcher	9
5.1. Fetcher Operations	9
5.1.1. Fetch repository objects	9
5.1.2. Fetch single repository object	10
6. Local Object Store	11
6.1. Store Operations	11
6.1.1. Store Repository Object	11
6.1.2. Get objects by hash	11
6.1.3. Get certificate objects by URI	11
6.1.4. Get manifest objects by AKI	11
6.1.5. Delete objects for a URI	11
6.1.6. Delete outdated objects	11
6.1.7. Update object's validation time	11
7. Acknowledgements	11
8. IANA Considerations	12
9. Security Considerations	12
9.1. Hash collisions	12
9.2. Mismatch between the expected and the actual location of an object in the repository	12
9.3. Manifest content versus publication point content	13
9.4. Storing of a TA certificate object before its complete validation	13
9.5. Possible denial of service	13
10. References	13
10.1. Normative References	13
10.2. Informative References	14
Authors' Addresses	15

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 its 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

SIDROPS
Internet-Draft
Intended status: Informational
Expires: December 28, 2017

D. Ma
ZDNS
S. Kent
BBN
June 26, 2017

Requirements for Resource Public Key Infrastructure (RPKI) Relying
Parties
draft-madi-sidrops-rp-00

Abstract

This document provides a single reference point for requirements for Relying Party (RP) software for use in the Resource Public Key Infrastructure (RPKI). It cites requirements that appear in several RPKI RFCs, making it easier for implementers to become aware of these requirements that are segmented with orthogonal functionalities.

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 December 28, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Fetching and Caching RPKI Repository Objects	3
2.1. TAL Acquisition and Processing	4
2.2. Locating RPKI Objects Using Authority and Subject Information Extensions	4
2.3. Dealing with Key Rollover	4
2.4. Dealing with Algorithm Transition	4
2.5. Strategies for Efficient Cache Maintenance	5
3. Certificate and CRL Processing	5
3.1. Verifying Resource Certificate and Syntax	5
3.2. Certificate Path Validation	5
3.3. CRL Processing	5
4. Processing RPKI Repository Signed Objects	6
4.1. Basic Signed Object Syntax Checks	6
4.2. Syntax and Validation for Each Type of Signed Object	6
4.2.1. Manifest	6
4.2.2. ROA	6
4.2.3. Ghostbusters	7
4.2.4. Verifying BGPsec Router Certificate	7
4.3. How to Make Use of Manifest Data	7
4.4. What to Do with Ghostbusters Information	8
5. Delivering Validated Cache to BGP Speakers	8
6. Security considerations	8
7. IANA Considerations	8
8. Acknowledgements	8
9. References	8
9.1. Normative References	8
9.2. Informative References	10
Authors' Addresses	10

1. Introduction

The RPKI RP software is used by network operators and others to acquire and verify Internet Number Resource (INR) data stored in the RPKI repository system. RPKI data, when verified, allow an RP to verify assertions about which Autonomous Systems (ASes) are authorized to originate routes for IP address prefixes. RPKI data also establishes binding between public keys and BGP routers, and indicates the AS numbers that each router is authorized to represent.

Noting that the essential requirements imposed on RPs are scattered throughout numerous RFC documents that are protocol specific or provide best practices, as follows:

RFC 6481 (Repository Structure)
RFC 6482 (ROA format)
RFC 6486 (Manifests)
RFC 6487 (Certificate and CRL profile)
RFC 6488 (RPKI Signed Objects)
RFC 6489 (Key Rollover)
RFC 6810 (RPKI to Router Protocol)
RFC 6916 (Algorithm Agility)
RFC 7730 (Trust Anchor Locator)
RFC 7935 (Algorithms)
RFC XXXX (Router Certificates)[ID.sidr-bgpsec-pki-profiles]

This makes it hard for an implementer to be confident that he/she has addressed all of these generalized requirements. Besides, software engineering calls for how to segment the RP system into components with orthogonal functionalities, so that those components could be distributed across the operational timeline of the user. Taxonomy of generalized RP requirements is going to help have 'RP role' well framed.

To consolidate RP requirements in one document, with pointers to all the relevant RFCs, this document outlines a set of baseline requirements imposed on RPs and provides a single reference point for requirements for RP software for use in the RPKI, as segmented with orthogonal functionalities:

- o Fetching and Caching RPKI Repository Objects
- o Processing Certificates and CRLs
- o Processing RPKI Repository Signed Objects
- o Delivering Validated Cache Data to BGP Speakers

This document will be update to reflect new or changed requirements as these RFCs are updated, or new RFCs are written.

2. Fetching and Caching RPKI Repository Objects

RP software uses synchronization mechanisms supported by targeted repositories (e.g., [rsync]) to download all RPKI changed data objects in the repository system and cache them locally. The software validates the RPKI data and uses it to generate authenticated data identifying which ASes are authorized to originate routes for address prefixes, and which routers are authorized to sign BGP updates on behalf of ASes.

2.1. TAL Acquisition and Processing

In the RPKI, each relying party (RP) chooses its own set of trust anchors (TAs). Consistent with the extant INR allocation hierarchy, the IANA and/or the five RIRs are obvious candidates to be default TAs for the RP.

An RP does not retrieve TAs directly. A set of Trust Anchor Locators (TALs) is used by each RP to retrieve and verify the authenticity of each trust anchor.

TAL acquisition and processing are specified in Section 3 of [RFC7730].

2.2. Locating RPKI Objects Using Authority and Subject Information Extensions

The RPKI repository system is a distributed one, consisting of multiple repository instances. Each repository instance contains one or more repository publication points. An RP discovers publication points using the SIA and AIA extensions from (validated) certificates.

Section 5 of [RFC6481] specifies how an RP locates all RPKI objects by using the SIA and AIA extensions. Detailed specifications of SIA and AIA extensions in a resource certificate are described in section 4 of [RFC6487].

2.3. Dealing with Key Rollover

An RP takes the key rollover period into account with regard to its frequency of synchronization with RPKI repository system.

RP requirements in dealing with key rollover are described in section 3 of [RFC6489].

2.4. Dealing with Algorithm Transition

The set of cryptographic algorithms used with the RPKI is expected to change over time. Each RP is expected to be aware of the milestones established for the algorithm transition and what actions are required at every juncture.

RP requirements for dealing with algorithm transition are specified in section 4 of [RFC6916].

2.5. Strategies for Efficient Cache Maintenance

Each RP is expected to maintain a local cache of RPKI objects. The cache needs to be as up to date and consistent with repository publication point data as the RP's frequency of checking permits.

The last paragraph of section 5 of [RFC6481] provides guidance for maintenance of a local cache.

3. Certificate and CRL Processing

The RPKI make use of X.509 certificates and CRLs, but it profiles these standard formats [RFC6487]. The major change to the profile established in [RFC5280] is the mandatory use of a new extension to X.509 certificate [RFC3779].

3.1. Verifying Resource Certificate and Syntax

Certificates in the RPKI are called resource certificates, and they are required to conform to the profile [RFC6487]. An RP is required to verify that a resource certificate adheres to the profile established by [RFC6487]. This means that all extensions mandated by [RFC6487] must be present and value of each extension must be within the range specified by this RFC. Moreover, any extension excluded by [RFC6487] must be omitted.

Section 7.1 of [RFC6487] gives the procedure that the RP should follow to verify resource certificate and syntax.

3.2. Certificate Path Validation

In the RPKI, issuer can only assign and/or allocate public INRs belong to it, thus the INRs in issuer's certificate are required to encompass the INRs in the subject's certificate. This is one of necessary principles of certificate path validation in addition to cryptographic verification i.e., verification of the signature on each certificate using the public key of the parent certificate).

Section 7.2 of [RFC6487] gives the procedure that the RP should follow to perform certificate path validation.

3.3. CRL Processing

The CRL processing requirements imposed on CAs and RP are described in [RFC6487]. CRLs in the RPKI are tightly constrained; only the AuthorityKeyIdentifier and CRLNumber extensions are allowed, and they MUST be present. No other CRL extensions are allowed, and no CRLentry extensions are permitted. RPs are required to verify that

these constraints have been met. Each CRL in the RPI MUST be verified using the public key from the certificate of the CA that issued the CRL.

In the RPKI, RPs are expected to pay extra attention when dealing with a CRL that is not consistent with the Manifest associated with the publication point associated with the CRL.

Processing of a CRL that is not consistent with a manifest is a matter of local policy, as described in the fourth paragraph of Section 6.6 of [RFC6486].

4. Processing RPKI Repository Signed Objects

4.1. Basic Signed Object Syntax Checks

Before an RP can use a signed object from the RPKI repository, the RP is required to check the signed object syntax.

Section 3 of [RFC6488] lists all the steps that the RP is required to execute in order to validate the top level syntax of a repository signed object.

Note that these checks are necessary, but not sufficient. Additional validation checks must be performed based on the specific type of signed object.

4.2. Syntax and Validation for Each Type of Signed Object

4.2.1. Manifest

To determine whether a manifest is valid, the RP is required to perform manifest-specific checks in addition to those specified in [RFC6488].

Specific checks for a Manifest are described in section 4 of [RFC6486]. If any of these checks fails, indicating that the manifest is invalid, then the manifest will be discarded and treated as though no manifest were present.

4.2.2. ROA

To validate a ROA, the RP is required to perform all the checks specified in [RFC6488] as well as the additional ROA-specific validation steps. The IP address delegation extension [RFC3779] present in the end-entity (EE) certificate (contained within the ROA), must encompass each of the IP address prefix(es) in the ROA.

More details for ROA validation are specified in section 2 of [RFC6482].

4.2.3. Ghostbusters

The Ghostbusters Record is optional; a publication point in the RPKI can have zero or more associated Ghostbuster Records. If a CA has at least one Ghostbuster Record, RP is required to verify that this Ghostbusters Record conforms to the syntax of signed object defined in [RFC6488].

The payload of this signed object is a (severely) profiled vCard. An RP is required to verify that the payload of Ghostbusters conforms to format as profiled in [RFC6493].

4.2.4. Verifying BGPsec Router Certificate

A BGPsec Router Certificate is a resource certificate, so it is required to comply with [RFC6487]. Additionally, the certificate must contain an AS Identifier Delegation extension, and must not contain an IP Address Delegation extension. The validation procedure used for BGPsec Router Certificates is identical to the validation procedure described in Section 7 of [RFC6487], but using the constraints applied come from specification of section 7 of [ID.sidr-bgpsec-pki-profiles].

Note that the cryptographic algorithms used by BGPsec routers are found in [ID.sidr-bgpsec-algs]. Currently, the algorithms specified in [ID.sidr-bgpsec-algs] and [RFC7935] are different. BGPsec RPs will need to support algorithms that are used to validate BGPsec signatures as well as the algorithms that are needed to validate signatures on BGPsec certificates, RPKI CA certificates, and RPKI CRLs.

4.3. How to Make Use of Manifest Data

For a given publication point, the RP ought to perform tests to determine the state of the Manifest at the publication point. A Manifest can be classified as either valid or invalid, and a valid Manifest is either current and stale. An RP decides how to make use of a Manifest based on its state, according to local (RP) policy.

If there are valid objects in a publication point that are not present on a Manifest, [RFC6486] does not mandate specific RP behavior with respect to such objects. However, most RP software ignores such objects and this document recommends that this behavior be adopted uniformly.

In the absence of a Manifest, an RP is expected to accept all valid signed objects present in the publication point. If a Manifest is stale (see [RFC6486]) and an RP has no way to acquire a more recent Manifest, the RP is expected to (TBD).

4.4. What to Do with Ghostbusters Information

An RP may encounter a stale Manifest or CRL, or an expired CA certificate or ROA at a publication point. An RP is expected to use the information from the Ghostbusters record to contact the maintainer of the publication point where any stale/expired objects were encountered. The intent here is to encourage the relevant CA and/or repository manager to update the slate or expired objects.

5. Delivering Validated Cache to BGP Speakers

On a periodic basis, BGP speakers within an AS request updated validated origin AS data and router/ASN data from the RP's cache. The RP passes this information to BGP speakers to enable them to verify the authenticity of routing announcements. The specification of the protocol designed to deliver validated cache data from an RP to a BGP Speaker is provided in [RFC6810].

6. Security considerations

TBD

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgements

The authors thank David Mandelberg and Wei Wang for their review, feedback and editorial assistance in preparing this document.

9. References

9.1. Normative References

[ID.sidr-bgpsec-algs]
Turner, S., "BGPsec Algorithms, Key Formats and Signature Formats", work-in-progress, <draft-ietf-sidr-bgpsec-algs>.

- [ID.sidr-bgpsec-pki-profiles] Reynolds, M., Turner, S., and S. Kent, "A Profile for BGPsec Router Certificates, Certificate Revocation Lists, and Certification Requests", work-in-progress, <draft-ietf-sidr-bgpsec-pki-profiles>.
- [RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, DOI 10.17487/RFC3779, June 2004, <<http://www.rfc-editor.org/info/rfc3779>>.
- [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>>.
- [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>>.
- [RFC6489] Huston, G., Michaelson, G., and S. Kent, "Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)", BCP 174, RFC 6489, DOI 10.17487/RFC6489, February 2012, <<http://www.rfc-editor.org/info/rfc6489>>.

- [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>>.
- [RFC6810] Bush, R. and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol", RFC 6810, DOI 10.17487/RFC6810, January 2013, <<http://www.rfc-editor.org/info/rfc6810>>.
- [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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7935>>.

9.2. 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>>.
- [rsync] "rsync web page", <<http://rsync.samba.org/>>.

Authors' Addresses

Di Ma
ZDNS
4 South 4th St. Zhongguancun
Haidian, Beijing 100190
China

Email: madi@zdns.cn

Stephen Kent
BBN
10 Moulton St
Cambridge, MA 02138-1119
USA

Email: kent@alum.mit.edu

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

J. Paillisse
UPC-BarcelonaTech
A. Rodriguez-Natal
V. Ermagan
F. Maino
Cisco Systems
A. Cabellos
UPC-BarcelonaTech
July 03, 2017

An analysis of the applicability of blockchain to secure IP addresses
allocation, delegation and bindings.
draft-paillisse-sidrops-blockchain-00.txt

Abstract

This document analyzes how blockchain technology can be used to secure the allocation, delegation and binding to topological information of the IP address space. The main outcomes of the analysis is that blockchain is suitable in environments with multiple distrusting parties and that Proof of Stake is a potential candidate for a consensus algorithm.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Definition of Terms	3
3. Blockchain in a nutshell	3
3.1. Overview	3
3.1.1. Chain of signatures	4
3.1.2. Consensus algorithm	5
3.2. Features	5
3.3. Description of consensus algorithms	6
3.3.1. Proof of Work (PoW)	6
3.3.2. Proof of Stake (PoS)	7
4. Blockchain for IP addresses	8
4.1. Problem statement	8
4.2. Analysis	9
4.3. A consensus algorithm for IP addresses	9
5. Architecture overview	10
5.1. Pros and cons	12
5.2. Security evaluation	13
5.2.1. Attacks against a PoS-based consensus algorithm	14
5.2.2. Attacks against the P2P network	15
6. Other Considerations	15
6.1. Revocation	15
6.2. Key rollover	15
6.3. Incentives	15
6.4. Storage management	15
6.5. Transaction censorship	15
6.6. Configuration parameters	15
7. Security Considerations	16
8. IANA Considerations	16
9. Acknowledgements	16
10. Informative References	16
Authors' Addresses	17

1. Introduction

Blockchain [Bitcoin] is attracting a lot of attention among the security community since it provides means for exchanging information among a set of distrusting entities without the use of digital certificates and centralized control. Blockchain provides means for

the distrusting parties to reach consensus. Formally, it is regarded as a new solution to the Byzantine Generals problem, well-known in fault-tolerant distributed systems.

Although at the time of this writing the main application of blockchain are financial systems, their use in the field of networking is being explored (e.g., [Hari2016]). Some successful systems exist such as [Blockstack] and [Namecoin], which aim at building a secure DNS.

The main goal of this document is to represent a first step towards the understanding of the properties of blockchains and their applicability in the Internet infrastructure, specifically securing the allocation, delegation and bindings of IP addresses. First, it introduces blockchain, then it analyzes how blockchain could be used to secure the delegation of IP addresses. Finally, it presents an initial design for such an infrastructure. This document also includes a preliminary security analysis of such system. It is important to note that the goal of this document is not to provide a complete architecture that secures IP address allocation, delegation and bindings.

2. Definition of Terms

TBC

3. Blockchain in a nutshell

3.1. Overview

Conceptually, a blockchain is a distributed, secure and trustless database. It can also be regarded as a state machine with rules that clearly state which transitions can be performed. Participants in the blockchain communicate through a P2P network. The smallest data unit of a blockchain is a transaction. Users attach data to a transaction along with its signature and their associated public key. Usually, the attached data is an asset or a token, something that is unique and should not be replicated (e.g., coins in Bitcoin). Then they broadcast this transaction to the other participants. The rest of the nodes in the network store temporarily this transaction. At some fixed intervals in time, one of the nodes takes a set of these transactions and groups them in a block. It then broadcasts this block back to the network. When the other nodes receive this block they verify it, remove the transactions contained in the block from the temporary storage and add it after the previous block, thus creating a chain of blocks. It should be noted that all nodes store the entire blockchain locally. In addition, most blockchains give some sort of reward to nodes that add new blocks, although this is

not strictly necessary. Figure 1 presents an overview of the most common elements in a block.

Block Number	Hash(Previous Block)	Hash(All Block Transactions)	Block Creator Signature
Transaction 1			
Transaction 2			
...			
...			
Transaction N			

Figure 1.- Common structure of a block

Two basic mechanisms are used to protect the chained data: a chain of signatures and a consensus algorithm.

3.1.1. Chain of signatures

The chain of signatures operates at transaction level. Consider the sender and receiver of a token, each with its public-private keypair. To change the owner of a token, the sender signs the data and the receiver's public key. It then puts together its public key, the signature, the data and the hash of the receiver's public key (Figure 2) to form a transaction.

Sender Public Key	Signature Sender Private Key	Data	Hash(Receiver Public Key)
----------------------	---------------------------------	------	------------------------------

Figure 2.- Common transaction structure in a blockchain

In conclusion, the rules of the blockchain enforce that:

- o The owner of the receiver private key has total control over the contents of the transaction. In Bitcoin this translates in a central property: only this owner can spend a coin.
- o When an owner sends a token to the new owner, it irreversibly transfers the control of the contents to the new owner.

3.1.2. Consensus algorithm

The consensus algorithm is the central part of blockchain and it controls the chaining of data blocks. The main role of the algorithm is to provide a set of well-defined rules so that participants agree on a consistent view of the database. For this it has the following main functions. First, forks (multiple chains) can exist, this may happen for instance due to varying network latency among participants. In this case the participants must agree on which is the valid chain. And second, another important function of the consensus algorithm is to determine which participants are allowed to add a new data blocks. Section 3.3 contains more information regarding available consensus algorithms.

It is important to note that regardless of the consensus algorithm, in blockchain data blocks are always added, never deleted nor modified. This creates a tamper-proof, shared ledger among all participants. Transactions can be tracked back by inspecting past blocks, thus enabling the verification of claims by certain parties.

3.2. Features

The following list tries to briefly summarize the main characteristics of the blockchain technology:

Decentralized: No central entity controls the blockchain, it is shared among all participants.

No CAs: No digital certificates, Certification Authorities or CRLs are needed.

Limited prior trust: It is not required to trust other nodes. It is worth noting that some consensus algorithms rely on some limited levels of trust.

Tamper-proof: Since data can be only added but never modified, attempts to alter previous records are detected.

Non-repudiation: All nodes share a common, immutable view on the status of the blockchain, and blockchain provides non-repudiation mechanisms.

Censorship-resistant: Gaining control over a transaction involves having access to the associated private key.

Append-only: Data is always added, but never modified nor deleted.

Privacy: Entities participating in the blockchain can achieve privacy using anonymous keys, i.e. randomly-generated keys not related to their identity. In addition, a new keypair should be generated for each new transaction in order to prevent tracking [Bitcoin], section 10.

Slow updates: New transactions have to be verified, added to a block and received by all nodes. This results in a delay since the transaction is created until it is finally available to all the nodes. This delay will depend on the consensus algorithm and the block creation rate.

Large storage: The size of the blockchain keeps growing forever, because data blocks are always added. This may result in scalability issues.

3.3. Description of consensus algorithms

The two more popular consensus algorithms are: Proof of Work and Proof of Stake.

3.3.1. Proof of Work (PoW)

In Proof of Work nodes have to solve a complex mathematical problem to add a block, thus requiring some computational effort, this is commonly known as mining. For example in Bitcoin the problem is to find a hash starting with a fixed amount of zeroes, the only known way to solve this problem is by brute force. The valid chain is the one with most accumulated computing power, this chain is also the more expensive in terms of computing power to modify. This is because modifying a block going N blocks back from the tip of the chain would require redoing the computations for all these N blocks. As a result, an attacker should have more computational power than the power required to create the N blocks to be able to modify the chain. Overall, it is commonly assumed that if more than half of the nodes are honest the blockchain is considered as secure.

PoW offers relevant features, adding new blocks requires an external resource (CPU power) that has an economical cost. However this also results in some relevant drawbacks:

Risk of overtaking: The security of PoW is entirely based on computation power. This means that if an entity has access to

more than half of the total blockchain's computing power it can control the chain. As a result and in order to keep blockchain secure, the incentive of taking control of the chain must be lower than the cost of acquiring and operating the hardware that provides the equivalent to half of the participants computing power. This is hard to guarantee since the economy of the blockchain and the economy of the required hardware are independent. As an example an attacker can acquire the required hardware and operate it, take control of the blockchain to obtain an economical benefit and finally sell the hardware to reduce the final cost of the attack.

Hardware dependency: Bitcoin automatically increases -over time- the complexity of the mathematical problem that needs to be solved in order to add a block. This is done to account for Moore's law. As a result the community has designed mining specific hardware (ASICs) that provides a competitive advantage. In this context blockchain becomes less democratic, since the cost of participating in it increases.

Energy inefficiency: PoW requires large amounts of energy to perform the computations (e.g., [miningfarm]).

3.3.2. Proof of Stake (PoS)

The main idea behind Proof of Stake is that participants with more assets (or stake) in the blockchain are more likely to add blocks. With this, the control of the chain is given to entities who own more stake. For each new block, a signer is selected randomly from the list of participants typically weighted according to their stake. A fundamental assumption behind PoS is that such entities have more incentives for honest behaviour since they have more assets in the chain.

Proof of Stake is seen as an alternative to PoW. At the time of this writing major players in the blockchain environment such as [Ethereum] are preparing a shift towards PoS, moreover several blockchains based on PoS already exist (eg. [Peercoin]). The main reason behind this paradigm shift is that PoS addresses some of PoW's main drawbacks:

- o It does not require special hardware nor computationally or energy-expensive calculations.
- o An attacker must get hold of a significant part of the assets in order to gain control of the blockchain. As opposed to PoS the investment required to gain control of the chain lies within the chain, and does not involve using external resources.

On the other side, Proof of Stake introduces new sources of attacks:

- o In Proof of Stake the signer is selected randomly among the stakers. In this context attackers can manipulate the source of randomness to sign more blocks and ultimately gain control over the chain.
- o As opposed to PoW, creating forks is very inexpensive, since no computational power is required. The PoS must provide means to select the valid chain, which is typically the longer one.
- o Collusions of high-stakers can create alternate chains which can appear to be valid.

4. Blockchain for IP addresses

4.1. Problem statement

The objective of this section is to analyze if an infrastructure using blockchain can provide a similar degree of security to traditional PKI-based architectures. Specifically we aim to secure:

- o Binding of IP address blocks to the holder (private key holder).
- o The allocations and delegations of IP address blocks among their holders.
- o Binding of IP address blocks to their topological locators (eg. AS numbers allocations).

This information is public and shared among a set of distrusting entities over the Internet. The architecture must be able to:

- o Allow anyone to verify the legitimate holder of a block of addresses
- o Let participating entities allocate address blocks without requiring a trusted third party.
- o Restrict the allocation of a block of addresses to only its legitimate holder.
- o Prevent data modification without the consent of its holder.

4.2. Analysis

The main rationale behind using blockchain to secure IP address allocations is that IPs can be understood as coins, both concepts share some fundamental characteristics:

- o They are unambiguously allocated to entities.
- o Can be transferred between them.
- o Cannot be assigned to two entities at the same time.
- o Can be divided up to a certain limit.

Such similar properties make it possible to envisage a blockchain that allows its participants delegate IP address blocks, similarly to how Bitcoin transfers coins. For example, IANA could write a transaction allocating addresses to the RIRs, which in turn could allocate them to the LIRs, etc. Complex management logic can be defined as needed for example, rejecting a transaction that allocates of a block of addresses originated by an entity that does not hold that block. In addition, transactions accept multiple inputs and outputs, i.e. an arbitrary amount of public keys either as senders or receivers. This means that it is possible to break and merge blocks of addresses as required. Section 5 provides more detailed information about this architecture.

4.3. A consensus algorithm for IP addresses

As stated before, the consensus algorithm is a central part of a blockchain. The first consensus algorithm designed for blockchain was PoW, and it is a common choice for new blockchain implementations. However it presents several drawbacks (Section 3.3.1) for the IP address scenario.

Using computing power as a means to secure blockchains has been proved to work in financial environments. However, the capability to add new blocks and the security of the chain itself depends on the computing power of the participants, which is not always aligned with their interest in the well-being of the blockchain. Depending on the objectives of the attacker, certain attacks can become profitable. Namely, buying a large quantity of hardware to be able to rewrite the blockchain with false data (e.g., incorrect delegations of IP addresses). This is because the incentives of the participants of the IP addresses blockchain are not linked with their computing power.

In contrast, with Proof of Stake the capability to alter the blockchain remains within it. This aspect is of particular importance in the context of securing IP addresses: it would mean that AS domains holding large blocks of IP addresses are more likely to add blocks. These parties have a reduced incentive in tampering the blockchain because they would suffer the consequences: an insecure Internet. Typically ASes that hold large blocks of IP address space have their business within the Internet and as such, have clear incentives in the correct operation and security of the Internet.

Furthermore, in such blockchain the risk of takeover is reduced compared to PoW, the reason is that accumulating a large amount of IP addresses is typically more complex than accumulating computing power. The risk of takeover is also mitigated compared to other PoS-based blockchains. In this blockchain an attacker would buy tokens from the other parties, who receive a monetary compensation to participate in the attack. However, in a blockchain for IP addresses this would mean buying IP addresses from other parties, who do not have a clear incentive to sell their blocks of addresses to the attacker. Because of this, PoS appears to be a firm candidate for a consensus algorithm in a blockchain for securing IP addresses allocations and delegations.

5. Architecture overview

This architecture mimics the hierarchy of IP address allocation present in today's Internet, with IANA on top of it. All nodes trust IANA's public key, which writes a genesis transaction assigning all of the address space to itself (figure 3).

IANA	Signature IANA	Allocate	Hash(IANA
Public Key 1	Private Key 1	0/0	Public Key 2)

Figure 3.- Genesis transaction

It then begins allocating each block of addresses to the IP address holders. Each transaction allocates part of the address space to the legitimate holder, and the rest of space is given back to IANA using a new keypair (figure 4).

IANA Public Key 2	Signature IANA Private Key 2	Rest of space	Hash(IANA Public Key 3)
		Allocate 001/8	Hash(APNIC Public Key 1)

Figure 4.- Example allocation transaction

In turn, all the parties in the hierarchy allocate or delegate address blocks following the current allocation hierarchy. When a party wants to verify the allocation of a block of addresses, it downloads the blockchain and verifies all the blocks and transactions up to the genesis block, for which it has trust. Figure 5 presents an example of allocation of one prefix to each of the RIRs.

IANA Public Key 3	Signature IANA Private Key 3	Rest of space	Hash(IANA Public Key 4)
		Allocate 005/8	Hash(RIPE Public Key 1)
		Allocate 014/8	Hash(APNIC Public Key 2)
		Allocate 023/8	Hash(ARIN Public Key 1)
		Allocate 102/8	Hash(AFRINIC Public Key 1)
		Allocate 200/8	Hash(LACNIC Public Key 1)

Figure 5.- Example multi-output allocation transaction

Inside the blockchain the typical operations to manage blocks of IP addresses can be defined, such as the delegation of prefixes (figure 6). This helps to enforce the rules of IP addresses management. For instance, since this transaction is marked as a delegation, if the

new owner created an allocation transaction it would be rejected by the other nodes, because the parent transaction does not have the privileges to perform it.

APNIC Public Key 1	Signature APNIC Private Key 1	Rest of space	Hash(APNIC Public Key 3)
		Delegate 001.002/16	Hash(Big ISP Public Key 1)

Figure 6.- Example delegation transaction

This chain can define as many operations as required, for instance storing the binding of AS numbers to the IP prefixes they announce (figure 7).

Big ISP Public Key 1	Signature Big ISP Private Key 1	Bind 001.002/16 AS no. 12345	Hash(Big ISP Public Key 2)
-------------------------	---------------------------------------	---------------------------------------	-------------------------------

Figure 7.- Example binding of AS number to prefix

Additional and more complex operations can be defined if the management logic requires it. For instance, several signatures (from different parties) can be required to consider a transaction valid, etc.

5.1. Pros and cons

In this section we analyze the pros and cons of a PoS-based blockchain system compared to traditional PKI infrastructures:

Advantages:

- o Decentralized: No central entity controls the blockchain, it is shared among all participants.

- o No CAs, CRLs or certificates needed: No digital certificates, Certification Authorities or CRLs are needed.
- o Simplified rekeying: A key rollover can easily be performed by issuing a new transaction allocating the prefixes to a new keypair controlled by the same holder. This process can be performed without involving any third-party.
- o Censorship-resistant: since the control of a transaction is completely under the holder of the private key, the revocation of IP addresses without the legitimate holder's permission involves obtaining its private key. Even if the private key of the previous owner was compromised, ownership of the current transaction is still preserved, as opposed to the compromise of a CA's private key (or a misbehaving CA).
- o Limited prior trust: It is not required to trust other nodes. However, in PoS it is necessary to periodically authenticate the chain state out-of-band to prevent some attacks.
- o Simplified management: since CAs are not required, their management overhead is avoided.
- o Auditable: allocations and delegations can be tracked back in the blockchain to determine if they originate from the legitimate holder.

Drawbacks:

- o PoS does not rely on strong cryptographic guarantees: As opposed to PKI-based systems that rely on strong and well-established cryptographic mechanisms, PoS-based infrastructures ultimately rely on the good behaviour of the high-stakers.
- o Slow updates: New transactions have to be verified and added to a block, which adds a delay until nodes recognize them as correct.
- o Costly bootstrapping: When a node is activated it has to download and verify the entire blockchain.
- o Large storage required: The blockchain grows forever as more blocks are added, blocks cannot be removed.

5.2. Security evaluation

5.2.1. Attacks against a PoS-based consensus algorithm

This section presents a list of the most relevant attacks against a Proof of Stake algorithm and how to mitigate them.

5.2.1.1. Stake grinding

Stake grinding refers to the manipulation of the consensus algorithm in order to progressively obtain more stake, with the goal of signing blocks more frequently with the ultimate goal of taking control of the blockchain. It proceeds as follows: when the attacker has to sign a block, it computes all the possible blocks (varying the data inside them) to find a combination that gives the highest possibility of signing another block in the future. It then signs this block and sends it to the network. This procedure is repeated for all the next signing opportunities. Over time, the attacker will sign more and more blocks until the consensus algorithm will always select the attacker to sign all blocks, thereby having taken control of the blockchain.

To prevent this attack, the source of randomness used to select the signers has to be hard to alter or to predict.

5.2.1.2. Nothing at stake

Nothing at stake is one of the fundamental drawbacks of Proof of Stake and requires careful design based on the incentives of the participants. In common PoS designs, the signers of the new block receive an economical incentive (e.g., Ethereum). However this does not hold in the IP address scenario, since participants should not receive any incentive. The incentive is, as stated before, achieving a consistent view of the IP address space and having a secure Internet.

5.2.1.3. Range attacks

A range attack is performed by creating a fork some blocks back from the tip of the chain. It is conceptually similar to the attack named as 'Risk of overtaking' in Section 3.3.1. In this scenario, the attacker has privately fabricated a chain which (according to the consensus algorithm rules) will be selected over the original one. Benefits of this attack include gaining more stake on the blockchain (this attack could be part of a stake grinding attack) or rewriting the transaction history to erase a payment made in the original blockchain.

The simplest solution to this attack is adding a revert limit to the blockchain, forbidding forks going back more than N blocks. This

provides a means to solidify the blockchain. However, nodes that have been offline for more than N blocks will need an external source that indicates the correct chain. It has been proposed to do this out of band. This is why a PoS blockchain is not purely trustless and requires a small amount of trust.

5.2.2. Attacks against the P2P network

5.2.2.1. DDOS attacks

TBC

5.2.2.2. Transaction flooding

TBC

5.2.2.3. Routing attacks

TBC

6. Other Considerations

6.1. Revocation

TBC

6.2. Key rollover

TBC

6.3. Incentives

TBC

6.4. Storage management

TBC

6.5. Transaction censorship

TBC

6.6. Configuration parameters

TBC

7. Security Considerations

This document aims to understand the security implications of using the blockchain technology to secure IP addresses allocation.

8. IANA Considerations

This memo includes no request to IANA.

9. Acknowledgements

TBD

10. Informative References

- [Bitcoin] Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System. <https://bitcoin.org/bitcoin.pdf>", 2008.
- [Blockstack] Ali, et al., M., "Blockstack : A Global Naming and Storage System Secured by Blockchains, USENIX Annual Technical Conference", 2016.
- [Ethereum] The Ethereum project, "<https://www.ethereum.org/>", 2016.
- [Hari2016] Hari, A. and T. Lakshman, "The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet. Fifteenth ACM Workshop on Hot Topics in Networks", 2016.
- [miningfarm] Inside a mining farm, "<http://www.bbc.com/future/story/20160504-we-looked-inside-a-secret-chinese-bitcoin-mine>", 2016.
- [Namecoin] Namecoin, "<https://namecoin.org/>", 2011.
- [Peercoin] The Peercoin cryptocurrency, "<https://peercoin.net/>", 2016.

Authors' Addresses

Jordi Paillisse
UPC-BarcelonaTech
c/ Jordi Girona 1-3
Barcelona, Catalonia 08034
Spain

Email: jordip@ac.upc.edu

Alberto Rodriguez-Natal
Cisco Systems
170 Tasman Drive
San Jose, CA
USA

Email: natal@cisco.com

Vina Ermagan
Cisco Systems
170 Tasman Drive
San Jose, CA
USA

Email: vermagan@cisco.com

Fabio Maino
Cisco Systems
170 Tasman Drive
San Jose, CA
USA

Email: fmaino@cisco.com

Albert Cabellos
UPC-BarcelonaTech
c/ Jordi Girona 1-3
Barcelona, Catalonia 08034
Spain

Email: acabello@ac.upc.edu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2017

T. Bruijnzeels
RIPE NCC
C. Martinez
LACNIC
June 28, 2017

RPKI signed object for TAL
draft-tbruijnzeels-sidrops-signed-tal-00

Abstract

Trust Anchor Locators (TALs) [RFC7730] are used by Relying Parties in the RPKI to locate and validate Trust Anchor certificates used in RPKI validation. This document defines an RPKI signed object [RFC6488] for a Trust Anchor Locator (TAL) that can be published by Trust Anchor to communicate a new TAL to already deployed Relying Parties. The two primary use cases for this are that 1) a Trust Anchor may wish to change the locations where its TA certificate may be found, and 2) a Trust Anchor may wish to perform a planned migration to a new key. Note that unplanned key rolls are considered out of scope for this document.

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 December 30, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements notation	2
2. Introduction	2
3. Signed TAL definition	3
3.1. The Signed TAL Content Type	3
3.2. The Signed TAL eContent	3
3.3. Signed TAL Validation	4
4. Signed TAL Generation	4
5. Signed TAL Publication	5
6. Supporting a TA Key Roll	5
6.1. Preparing a new TA key	5
6.2. Staging period - Using both the old and the new TA key	5
6.3. Preserving the Signed TAL	6
6.4. Retiring the old key	6
7. Supporting changing TA certificate publication point(s)	6
7.1. Adding a publication point	6
7.2. Withdrawing a publication point	6
7.3. Withdrawing the Signed TAL	7
8. Relying Party Use	7
9. IANA Considerations	7
9.1. OID	7
9.2. File Extension	7
10. Security Considerations	7
11. Acknowledgements	7
12. Normative References	8
Authors' Addresses	9

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

Trust Anchor Locator (TAL) files [RFC7730] are used in the Resource Public Key Infrastructure (RPKI) to help Relying Parties locate and verify a trust anchor certificate. A TAL file consists of:

- o One or more rsync URIs [RFC5781]

- o A subjectPublicKeyInfo [RFC5280] in DER format [X.509], encoded in Base64

The TAL can be distributed out-of-band to Relying Parties (RP), and it allows the RP to retrieve the most recent version of the Trust Anchor (TA) certificate from the cited location, and verify that public key of this certificate matches the TAL. This is useful as it allows selected data in the trust anchor to change, without needing to effect redistribution of the trust anchor per se. In particular the Internet Number Resources (INRs) extension [RFC3779] and the publication points defined in the Subject Information Access [RFC6487] may be updated this way.

The assumption is that both the URIs and key of the TA certificate remain stable. However, an organisation operating a TA may wish to change either of these properties, because of a need to:

- o change one or more URIs
- o perform a planned key roll

In this document we describe a method for TA operators to publish a an updated TAL in a secure a well-defined fashion, so that RPs can be alerted to these changes.

3. Signed TAL definition

A signed TAL is an RPKI signed object, as specified in [RFC6488]. The RPKI signed object template requires specification of the following data elements in the context of the manifest structure.

3.1. The Signed TAL Content Type

This document requests an OID for signed-Tal as follows:

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

This OID MUST appear both within the eContentType in the encapContentInfo object as well as the content-type signed attribute in the signerInfo object (see [RFC6488]).

3.2. The Signed TAL eContent

The content of a Signed TAL is ASN.1 encoded using the Distinguished Encoding Rules (DER) [X.690], and is defined as follows:

```
SignedTalContent ::= IA5String
```

The "SignedTalContent" contains the content of the new TAL encoded in Base64 [RFC4648].

3.3. Signed TAL Validation

Before a Relying Party can use a Signed TAL, the relying party MUST first validate the Signed TAL. To validate a Signed TAL, the relying party MUST perform all the validation checks specified in [RFC6488] as well as the following additional specific validation step.

- o The eContentType in the EncapsulatedContentInfo has OID 1.2.840.113549.1.9.16.1.TBD.
- o The EE certificate of this Signed TAL is signed by a known Trust Anchor
- o The decoded TAL content conforms to the format defined in [RFC7730]

If the above procedure indicates that the manifest is invalid, then the Signed TAL MUST be discarded and treated as though no Signed TAL were present.

4. Signed TAL Generation

A TA MAY choose to generate a single Signed TAL object to publish in its TA certificate publication point(s) in the RPKI. The TA MUST perform the following steps to generate the Signed TAL:

- o Generate a key pair for a "one-time-use" EE certificate to use for the Signed TAL
- o Generate a one-time-use EE certificate for the Signed TAL
- o This EE certificate MUST have an SIA extension access description field with an accessMethod OID value of id-ad-signedobject, where the associated accessLocation references the publication point of the Signed TAL as an object URL.
- o This EE certificate MUST describe its Internet Number Resources (INRs) using the "inherit" attribute, rather than explicit description of a resource set (see [RFC3779]).
- o This EE certificate MUST have a "notBefore" time that is before the moment that the Signed TAL will be published.

- o This EE certificate MUST have a "notAfter" time that reflects the intended staging period, which MUST be at least 24 hours after the moment that the Signed TAL will be published.

5. Signed TAL Publication

A TA MAY publish a single Signed TAL object directly under its TA certificate publication point(s) in the RPKI. The object base filename SHOULD use a similar strategy as the base filename that is used to determine the CRL and Manifest filenames for this TA certificate, and the extension part of the filename MUST be ".tal".

6. Supporting a TA Key Roll

A Signed TAL MAY be used to communicate a planned key roll for the TA.

6.1. Preparing a new TA key

Prior to publishing the Signed TAL for the new key the TA MUST perform the following steps:

- o Generate a new key pair for the new TA certificate
- o Generate a new TA Certificate, using a Subject Information Access for CA certificates (see section 4.8.8.1 of [RFC6487]) that references the URIs that will be used by the new key to publish objects, that are different from the URIs used by the TA certificate for the current key.
- o ALL current signed certificates and other objects, with the exception of the old CRL, Manifest and Signed TAL, must be re-issued by the new key and published under the new publication point(s).
- o The new TA certificate itself MUST be published in a (number of) new location(s) that are different from where the TA certificate for the current key is published.

After these steps are performed a new Signed TAL MUST be generated as described in Section 4, and published as described in Section 5.

6.2. Staging period - Using both the old and the new TA key

The staging period is initiated by the initial publication of a Signed TAL for the new key and must last at least 24 HOURS.

During the staging period the TA MUST continue to operate both the old and the new TA key.

6.3. Preserving the Signed TAL

The TA MAY preserve a Signed TAL for the old key after the staging period as a hint for RPs that missed the key roll. The following process can be used to achieve this:

- o Produce a new long-lived CRL that revokes all previously signed certificates
- o Produce a new long-lived Signed TAL
- o Produce a new long-lived manifest that includes the CRL and Signed TAL
- o Publish the CRL, MFT and Signed TAL
- o Destroy the old TA key

6.4. Retiring the old key

The TA MAY retire and delete its old key after the staging period is over.

7. Supporting changing TA certificate publication point(s)

A signed TAL MAY be used to communicate an addition or removal of one or more publication locations where the TA certificate can be found.

7.1. Adding a publication point

When adding a publication point for a TA certificate, the TA MUST publish the certificate in the new location(s) prior to publication of the Signed TAL.

7.2. Withdrawing a publication point

When removing a publication point for TA certificate, the TA SHOULD observe a staging period of at least 24 Hours. The staging period is initiated by the publication of an updated Signed TAL where the publication point has been removed. During the staging period the TA SHOULD keep the old publication point up to date and available.

7.3. Withdrawing the Signed TAL

The TA MUST withdraw the Signed TAL after the chosen staging period, of at least 24 hours, is over.

8. Relying Party Use

When an RP discovers a new valid TAL signed under a trust anchor, it SHOULD substitute the current TAL immediately.

RP software MAY start using the new TAL thus found automatically without operator intervention, but it is RECOMMENDED that the software informs the operator of this event, and keeps a back-up of the old TAL.

Furthermore, it is RECOMMENDED that the RP software informs the operator whether the new TAL represents a key roll, or a change in URIs only.

9. IANA Considerations

9.1. OID

IANA is to add the following to the "RPKI Signed Objects" registry:

Decimal	Description	References
TBD	signed-Tal	[section 3.1]

9.2. File Extension

IANA is to add an item for the Signed TAL file extension to the "RPKI Repository Name Scheme" created by [RFC6481] as follows:

Extension	RPKI Object	Reference

.tal	Signed TAL	[this document]

10. Security Considerations

TBD

11. Acknowledgements

TBD

12. Normative References

- [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>>.
- [RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, DOI 10.17487/RFC3779, June 2004, <<http://www.rfc-editor.org/info/rfc3779>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [X.509] ITU-T Recommendation X.509 (2000), "Recommendation X.509: The Directory - Authentication Framework", 2000.

[X.690] ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002,
"Information technology - ASN.1 encoding rules:
Specification of Basic Encoding Rules (BER), Canonical
Encoding Rules (CER) and Distinguished Encoding Rules
(DER)", 2002.

Authors' Addresses

Tim Bruijnzeels
RIPE NCC

Email: tim@ripe.net

Carlos Martinez
LACNIC

Email: carlos@lacnic.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

R. Bush
Internet Initiative Japan
July 3, 2017

Origin Validation Clarifications
draft-ymbk-sidrops-ov-clarify-00

Abstract

Deployment of RPKI-based BGP origin validation is hampered by, among other things, vendor mis-implementations in two critical areas, which routes are validated and whether policy is applied when not specified by configuration. This document is meant to clarify possible misunderstandings causing those mis-implementatons.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119] only when they appear in all upper case. They may also appear in lower or mixed case as English words, without normative meaning.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

Deployment of RPKI-based BGP origin validation is hampered by, among other things, vendor mis-implementations in two critical areas, which routes are validated and whether policy is applied when not specified by configuration. This document is meant to clarify possible misunderstandings causing those mis-implementations.

When a route is distributed into BGP, origin validation marks the announcement as NotFound, Valid, or Invalid per [RFC6811]. Operational testing has shown that the specifications of that RFC must be unclear. This document attempts to clarify two areas seeming to cause confusion.

The implementation issues seem not to be about how to validate, i.e., how to decide if a route is NotFound, Valid, or Invalid. The issues seem to be which routes to mark and whether to apply policy without operator configuration.

2. Suggested Reading

It is assumed that the reader understands BGP, [RFC4271], the RPKI, [RFC6480], Route Origin Authorizations (ROAs), [RFC6482], and RPKI-based Prefix Validation, [RFC6811].

3. Mark ALL Prefixes

An operator should not have to hear from their neighbor that they announced an Invalid route. Their routers should tell them before announcing the Invalid route to a neighbor. For this reason, [RFC6811] says

"When a BGP speaker receives an UPDATE from a neighbor, it SHOULD perform a lookup as described above for each of the Routes in the UPDATE message. The lookup SHOULD also be applied to routes that are redistributed into BGP from another source, such as another protocol or a locally defined static route."

[RFC6811] goes on to say "An implementation MAY provide configuration options to control which routes the lookup is applied to." In the absence of the operator applying such policy, ALL routes in BGP MUST be marked.

This means that, on a router, all routes in BGP, absent operator configuration otherwise, MUST have been marked because they were either received via BGP (whether eBGP or iBGP), redistributed from an IGP, static, or directly connected, or any other distribution into BGP.

4. Marking not Acting

Once routes are marked, the operator should be in complete control of any policy applied based the markings. Absent operator configuration, policy MUST NOT be applied.

Automatic policy actions such as those described in [RFC8097], BGP Prefix Origin Validation State Extended Community, MUST NOT be carried out or otherwise applied unless specifically configured by the operator.

5. Security Considerations

This document does not create security considerations beyond those of [RFC6811].

6. IANA Considerations

This document has no IANA Considerations.

7. References

7.1. Normative References

[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>>.

[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>>.

- [RFC6811] Mohapatra, P., Scudder, J., Ward, D., Bush, R., and R. Austein, "BGP Prefix Origin Validation", RFC 6811, DOI 10.17487/RFC6811, January 2013, <<http://www.rfc-editor.org/info/rfc6811>>.
- [RFC8097] Mohapatra, P., Patel, K., Scudder, J., Ward, D., and R. Bush, "BGP Prefix Origin Validation State Extended Community", RFC 8097, DOI 10.17487/RFC8097, March 2017, <<http://www.rfc-editor.org/info/rfc8097>>.

7.2. Informative References

- [RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <<http://www.rfc-editor.org/info/rfc4271>>.
- [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>>.

Author's Address

Randy Bush
Internet Initiative Japan
5147 Crystal Springs
Bainbridge Island, Washington 98110
US

Email: randy@psg.com