BMWG
Internet-Draft
Intended status: Informational
Expires: 16 September 2026

L. Qin
Y. Su
Zhongguancun Laboratory
J. Shi
D. Li
Tsinghua University

15 March 2026

RPKI Relying Party Benchmarking Methodology
draft-qin-bmwg-rpki-rp-bench-00

Abstract

This document defines a benchmarking methodology for evaluating RPKI Relying Party (RP) implementations in controlled laboratory environments. The methodology focuses on whether RP implementations correctly perform required validation steps and on the performance of these operations. RP implementations are treated as black boxes, enabling consistent and objective assessment based on externally observable behavior rather than internal design or implementation details.

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 https://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 16 September 2026.

Copyright Notice

Table of Contents

1. Introduction

The Resource Public Key Infrastructure (RPKI) [RFC6480] provides a framework for cryptographically securing Internet routing by allowing Relying Parties (RPs) to validate Route Origin Authorizations (ROAs) and other RPKI objects. Relying Party implementations are expected to comply with the requirements defined in [RFC8897].

Currently, there is no standardized methodology to evaluate whether an RP implementation correctly satisfies these requirements. In addition, the processing performance of RPs, such as the time required to validate objects and generate validated ROA payloads (VRPs), has not been systematically measured.

This document defines a benchmarking methodology for Relying Parties that addresses both functional correctness and processing performance. Specifically, the methodology provides:

1. Functional correctness tests to evaluate compliance with the requirements of [RFC8897].

2. Performance tests to measure the total processing time from object retrieval to VRP generation.

The methodology is intended to support implementers and operators in evaluating and comparing RP behavior under controlled conditions. The remainder of this document is structured as follows:

* Section 4 defines the functional correctness and performance tests.

* Section 5 defines the format for reporting test results.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Relying Party Overview

A Relying Party (RP) in the Resource Public Key Infrastructure (RPKI) is responsible for retrieving, validating, and making available RPKI objects to support secure route validation. [RFC8897] specifies the expected behavior of RPs, which includes:

* Object Retrieval: obtaining RPKI objects from Publication Points (PPs) and keeping them up-to-date.

* Object syntax Validation: checking DER encoding, syntax, and structural correctness of RPKI objects, including certificates, CRLs, ROAs, and manifests.

* Certification Path Validation: constructing and validating certificate chains from the Trust Anchor to each leaf certificate.

* Signed Object Signature Validation: verifying digital signatures of RPKI objects.

* Manifest Processing: ensuring completeness and integrity of published objects.

After validation, the RP produces a Validated Payload for use in routing systems. These functions ensure that only valid and trusted RPKI objects influence routing decisions.

3. Test Setup

This section defines the test setup. The System Under Test (SUT) (i.e., the RP software) is treated as a black box. No internal configuration or implementation behavior of the RP software is mandated. The test setup focuses on providing controlled inputs and observing RP outputs to enable reproducible and comparable measurements.

3.1. Test Environment

In this methodology, the Tester consists of the Servers and the Controller. Together, they generate the test conditions, trigger events, and support observation of the SUT for functional and performance evaluation. The SUT itself is evaluated for correctness and efficiency in processing RPKI objects.

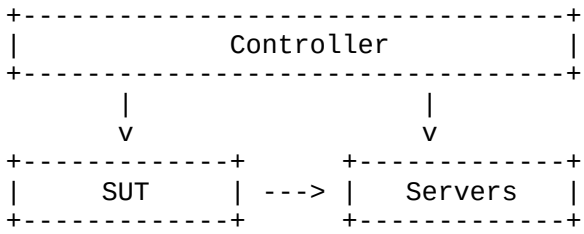

Figure 1: Test environment

To ensure meaningful testing, the environment should include:

* At least one Trust Anchor (TA)

* One or more subordinate certificate authorities (CAs) certificates

The test environment should support multiple RPKI transport protocols for object retrieval, including Rsync [RSYNC], RRDP [RFC8182], or Eric [I-D.ietf-sidrops-rpki-erik-protocol].

3.2. System Under Test and Servers

The System Under Test (SUT) is an RPKI Relying Party implementation that retrieves RPKI objects, validates them, and generates the validated ROA payload (VRP).

The Servers should include:

* Publication Point (PP) servers which host objects signed by CAs. Each PP server contains ROAs, manifests, CRLs, and certificates necessary for the tests.

* Additional components required by specific RPKI transport protocols. For example, Erik Relays [I-D.ietf-sidrops-rpki-erik-protocol].

Objects can be added, modified, or removed. Existing tools (e.g., [Barry]) can be used to implement and manage the test environment.

3.3. Controller

The controller orchestrates the entire testing process. In Figure 1, the arrows from the Controller to both the SUT and Servers represent the flow of control information.

For the SUT, the controller controls the start and end of tests, and handles the parsing and processing of test results.

For the servers, the controller handles content changes to ensure that the RP sees one set of files during a validation run and a different set during the next run.

4. RPKI Relying Party Benchmarking Tests

4.1. Object Retrieval Correctness

Objective: To evaluate whether the SUT correctly synchronizes RPKI objects from the configured PP servers. This test focuses on verifying successful repository synchronization regardless of the specific transport protocol used by the SUT.

Procedure: The PP servers and all RPKI objects are correctly configured prior to the test. The SUT is configured with the URI of TA, and the repository synchronization update interval is set to a fixed value (e.g., 10 minutes).

The following steps are performed:

1. Initialize the SUT with an empty local repository cache (cold start).

2. Start the SUT and allow it to perform repository synchronization with the configured PP servers.

3. Verify that the SUT retrieves all RPKI objects available from the PP servers.

4. After the initial synchronization completes, update the repository contents on the PP servers (e.g., publish new objects or update existing objects).

5. Wait for one synchronization update interval.

6. Verify that the SUT retrieves the updated repository contents from the PP servers.

Expected results: After the cold start synchronization, the SUT retrieve all RPKI objects available from the configured PP servers. After the repository contents are updated, the SUT synchronize the updated objects within one synchronization update interval.

4.2. Object Syntax Validation Correctness

This section evaluates whether the System Under Test (SUT) correctly performs syntax validation for RPKI objects. The SUT is expected to perform syntax checks according to the relevant specifications and detect objects that do not conform to the defined syntax requirements.

The syntax requirements for different RPKI objects are defined in the following specifications:

* Certificate and CRL: [RFC5280] and [RFC6487]

* Manifest: [RFC6488] and [RFC9286]

* ROA: [RFC6488] and [RFC9592]

For each object type, test objects that violate specific syntax requirements are constructed and published in the repository. The SUT behavior is then observed to determine whether the syntax validation is correctly performed.

4.2.1. DER Decoding

4.2.1.1. Processing Valid DER

Objective: To evaluate whether the SUT correctly decodes RPKI objects that use valid Distinguished Encoding Rules (DER) encoding.

Procedure:

1. Publish a set of RPKI objects with valid DER encoding on the PP servers.
2. Start the SUT and allow it to synchronize the repository contents.
3. Verify that the SUT retrieves the objects and attempts to process them.
4. Verify that the SUT successfully decodes the objects with valid DER encoding.

Expected results: The SUT successfully decode RPKI objects that use valid DER encoding.

4.2.1.2. Handling Invalid DER

Objective: To evaluate whether the SUT correctly detects and rejects RPKI objects that contain invalid DER encoding.

Procedure:

1. Publish a set of RPKI objects containing malformed DER encoding on the PP servers.
2. Start the SUT and allow it to synchronize the repository contents.
3. Verify that the SUT retrieves the malformed objects.
4. Verify that the SUT attempts to decode the objects.

Expected results: The SUT successfully detect DER decoding errors in objects with malformed encoding and reject those objects.

4.2.2. Certificate Syntax Validation

Objective: To evaluate whether the SUT correctly performs syntax validation for certificates according to Section 3.1 of [RFC8897].

Procedure: 1. Construct certificate objects that violate specific syntax requirements defined in Section 7.2 of [RFC6487]. 2. Each test certificate SHOULD violate one syntax requirement at a time (e.g., missing mandatory fields, invalid field values, incorrect extensions, or malformed structures). 3. Publish these malformed certificate objects on the PP servers. 4. Start the SUT and allow it to synchronize the repository contents. 5. Verify that the SUT retrieves the certificate objects and performs syntax validation.

Expected Results: If a certificate violates the syntax requirements, the SUT is able to detect the syntax error.

4.2.3. CRL Syntax Validation

Objective: To evaluate whether the SUT correctly performs syntax validation for CRLs according to Section 3.2 of [RFC8897].

Procedure:

1. Construct CRL objects that violate specific syntax requirements defined in [RFC5280] and [RFC6487].

2. Each test CRL SHOULD violate one syntax requirement at a time.

3. Publish these malformed CRL objects on the PP servers.

4. Start the SUT and allow it to synchronize the repository contents.

5. Verify that the SUT retrieves the CRL objects and performs syntax validation.

Expected Results: If a CRL violates the syntax requirements, the SUT is able to detect the syntax error.

4.2.4. Manifest Syntax Validation

Objective: To evaluate whether the SUT correctly performs syntax validation for Manifest according to Section 4.2.1 of [RFC8897].

Procedure 1. Construct manifest objects that violate specific syntax requirements defined in [RFC6488] and [RFC9286]. 2. Each test manifest SHOULD violate one syntax requirement at a time. 3. Publish

these malformed manifest objects on the PP servers. 4. Start the SUT and allow it to synchronize the repository contents. 5. Verify that the SUT retrieves the manifest objects and performs syntax validation.

Expected Results: If a manifest violates the syntax requirements, the SUT is able to detect the syntax error.

4.2.5. ROA Syntax Validation

Objective: To evaluate whether the SUT correctly performs syntax validation for Manifest according to Section 4.2.2 of [RFC8897].

Procedure:

1. Construct ROA objects that violate specific syntax requirements defined in [RFC6488] and [RFC9582].

2. Each test ROA SHOULD violate one syntax requirement at a time.

3. Publish these malformed ROA objects on the PP servers.

4. Start the SUT and allow it to synchronize the repository contents.

5. Verify that the SUT retrieves the ROA objects and performs syntax validation.

Expected Results: If a manifest violates the syntax requirements, the SUT is able to detect the syntax error.

4.3. Certification Path Validation Correctness

Objective: To evaluate whether the SUT correctly performs certification path validation and detects certification paths that violate the requirements defined in Section 3.2 of [RFC8897].

Procedure:

1. Construct RPKI object sets whose associated certification paths violate specific requirements defined in [RFC6487].

2. Each test case SHOULD introduce one violation at a time in the certification path.

3. Examples of such violations include, but are not limited to:

* Invalid certification path structure, where the subject of a certificate does not match the issuer of the next certificate in the certification path.

* Invalid certificate signature, where the certificate cannot be verified using the issuer's public key.

* Resource extension violation, where the resources listed in a child certificate are not encompassed by the resources listed in the parent certificate.

4. Publish these certificates on the PP servers.

5. Start the SUT and allow it to synchronize the repository contents.

6. Verify that the SUT retrieves the certificates and performs certification path validation.

Expected Results: If the certification path violates the requirements, the SUT is able to detect the validation failure and reject the affected objects.

4.4. Signed Object Signature Validation Correctness

Objective: To evaluate whether the SUT correctly verifies the digital signatures of RPKI signed objects using the corresponding public keys in the associated certificates.

Procedure:

1. Construct a set of valid RPKI signed objects (e.g., Manifests, ROAs) whose signatures correctly match their associated certificates.

2. Publish these objects on the PP servers.

3. Start the SUT and allow it to synchronize the repository contents.

4. Verify that the SUT retrieves the objects and performs signature validation.

5. Construct a set of signed objects whose signatures are intentionally invalid (e.g., the object content is modified after signing or the signature does not match the corresponding certificate).

6. Publish these malformed objects on the PP servers.

7. Trigger repository synchronization on the SUT.

8. Verify that the SUT performs signature validation for the retrieved objects.

Expected Results The SUT verifies the digital signature of each retrieved signed object using the corresponding public key contained in the associated certificate. If the signature is invalid, the SUT is able to detect the signature verification failure and reject the object.

4.5. Manifest Processing Correctness

This section evaluates whether the SUT correctly uses manifests to verify the integrity and completeness of RPKI repository objects.

Manifests are expected to be used to:

* Verify that the content of each RPKI object matches the hash listed in the manifest.

* Verify that all objects listed in the manifest exist in the repository and that there are no extra objects not declared in the manifest.

4.5.1. Manifest Hash-Mismatch Check

Objective: To evaluate whether the SUT detects when the content of a retrieved RPKI object does not match the hash declared in its associated manifest.

Procedure:

1. Construct a set of RPKI objects (e.g., ROAs) and a corresponding manifest.

2. Modify the content of one or more objects after the manifest has been generated, so that the object hash no longer matches the hash listed in the manifest.

3. Publish the manifest and the modified objects on the PP servers.

4. Start the SUT and allow it to synchronize the repository contents.

5. Verify that the SUT retrieves the manifest and the associated objects, and performs the hash-mismatch check.

Expected Results: The SUT is able to detects any objects whose content does not match the hash declared in the manifest.

4.5.2. Manifest Object-Mismatch Check

Objective: To evaluate whether the SUT detects when the objects listed in a manifest are missing from or extra in the repository.

Procedure

1. Construct a manifest that lists a set of RPKI objects.
2. Publish the manifest and a modified set of objects on the PP servers where:
   * Some objects declared in the manifest are missing from the repository, and/or
   * Extra objects exist in the repository that are not declared in the manifest.
3. Start the SUT and allow it to synchronize the repository contents.
4. Verify that the SUT retrieves the manifest and performs object-mismatch checking.

Expected Results The SUT is able to detect any discrepancies between the objects declared in the manifest and the objects present in the repository.

4.6. Processing Time test

Objective: To measure the end-to-end processing time of the SUT, from the moment it begins processing retrieved RPKI objects to the moment the validated ROA payload is generated. This includes all validation steps such as DER decoding, object syntax validation, certification path validation, signature verification, and manifest usage checks.

Procedure:

1. Allow the SUT to synchronize with a test repository containing a predefined set of RPKI objects.

2. Start the SUT and record the timestamp at the moment it begins processing the repository objects.

3. Allow the SUT to complete the full validation pipeline, including all functional steps.

4. Record the timestamp at the moment the SUT produces the final validated ROA payload (VRP).

5. Calculate the total processing time as the difference between the start and end timestamps.

6. Repeat the test for different repository sizes and object types to evaluate performance under varying load conditions.

5. Report Format

This section defines the format for reporting the results of RPKI Relying Party benchmarking tests. The format is concise and suitable for documenting both functional correctness and performance results.

System Under Test (SUT) Information: - RP implementation name and version

Test Repository Information:

* Number of objects and object types used in the test

Functional Correctness Test Results:

* Object Retrieval Correctness: pass/fail

* DER Decoding Correctness:

  - Valid DER Decoding: pass/fail Invalid DER Handling: pass/fail

* Object Syntax Validation Correctness:

  - Certificate: pass/fail

  - CRL: pass/fail

  - Manifest: pass/fail

  - ROA: pass/fail

* Certification Path Validation Correctness: pass/fail

* Signed Object Signature Validation Correctness: Invalid Signature: pass/fail

* Manifest Usage Correctness:

  - Hash-Mismatch Check: pass/fail

  - Object-Mismatch Check: pass/fail

Performance Test Results:

* Processing Time Performance Test:

  - Repository size

  - Number of objects

  - Total processing time from start of object fetching to generation of validated ROA payload

Notes:

* Functional correctness tests report pass/fail based on whether the SUT correctly performs all validation checks required by the relevant specifications. A test passes only if all applicable validation requirements are correctly enforced.

* Performance tests report total processing time only, as defined above.

6. Security Considerations

This document defines benchmarking methodologies for RPKI RP implementations in controlled laboratory environments using dedicated address space and constrained resources. No additional security considerations are identified within the scope of this document.

7. IANA Considerations

This document has no IANA requests.

Acknowledgements

The authors would like to thank Jorge Cano and the FORT team for their detailed reviews and constructive feedback, which helped clarify and improve the scope and methodology of this work.

References

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, <https://www.rfc-editor.org/info/rfc2119>.

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

Informative References

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

[RFC9286] Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure (RPKI)", RFC 9286, DOI 10.17487/RFC9286, June 2022, <https://www.rfc-editor.org/info/rfc9286>.

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

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

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

[RFC8897] Ma, D. and S. Kent, "Requirements for Resource Public Key Infrastructure (RPKI) Relying Parties", RFC 8897, DOI 10.17487/RFC8897, September 2020, <https://www.rfc-editor.org/info/rfc8897>.

[RFC9582] Snijders, J., Maddison, B., Lepinski, M., Kong, D., and S. Kent, "A Profile for Route Origin Authorizations (ROAs)", RFC 9582, DOI 10.17487/RFC9582, May 2024, <https://www.rfc-editor.org/info/rfc9582>.

[I-D.ietf-sidrops-rpki-erik-protocol]
Snijders, J., Bruijnzeels, T., Harrison, T., and W. Ohgai, "The Erik Synchronization Protocol for use with the Resource Public Key Infrastructure (RPKI)", Work in Progress, Internet-Draft, draft-ietf-sidrops-rpki-erik-protocol-03, 27 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-rpki-erik-protocol-03>.

[RSYNC] "The rsync web pages", n.d., <https://rsync.samba.org>.

[Barry] "BaRRy", n.d., <https://github.com/lacnic/barry>.

Authors' Addresses

Lancheng Qin
Zhongguancun Laboratory
Beijing
China
Email: qinlc@mail.zgclab.edu.cn

Yingying Su
Zhongguancun Laboratory
Beijing
China
Email: suyy@mail.zgclab.edu.cn

Jiayi Shi
Tsinghua University
Beijing
China
Email: sjy23@mails.tsinghua.edu.cn

Dan Li
Tsinghua University
Beijing
China
Email: tolidan@tsinghua.edu.cn