RIPE NCC Response to the Report from Radically Open Security

Table of Contents

1. Executive Summary

1.1 Introduction

Security is always high on our list of priorities for RPKI. Every year, we ask an external party to carry out a security audit of our RPKI systems. This is the first year that we are publishing the security report, in an effort to increase transparency and trust in the RPKI system. Please note that the report also listed several recommendations that should be included in a penetration test. These recommendations have been redacted from the original report, as we will include them in the penetration test scheduled for June 2021. Also, some comments about the proprietary software for the Hardware Security Module have been redacted.

Between 3 August and 25 September2020, Radically Open Security B.V. (ROS) carried out a code audit for the RIPE NCC. The audit was intended to assess compliance with the various RFCs covering the RPKI technology as well as the security of its various components. This article describes the findings of ROS and what RIPE NCC has done to mitigate the risks identified in the report.

1.2 Scope

The scope of the code audit was:

- RPKI Core
- the RPKI Publication Server, which exposes RPKI material to the Relying Parties
- Hardware Security Module (HSM), which is used to generate and store key pairs for the Trust Authority

Regarding the RPKI Core and Publication Server, the following RFCs have been identified. This table also includes the number of findings associated with each RFC.

Component	RFC	Number of Issues Found
RPKI Core	RFC3779	0
RPKI Core	RFC5280	5
RPKI Core	RFC5652	2
RPKI Core	RFC6480	0
RPKI Core	RFC6481	-
RPKI Core	RFC6482	2
RPKI Core	RFC6484	2
RPKI Core	RFC6485	1
RPKI Core	RFC6486	3
RPKI Core	RFC6487	0
RPKI Core	RFC6488	4
RPKI Core	RFC6489	0
RPKI Core	RFC6490	3
Publication	RFC8181	7
Publication	RFC8182	2
HSM	Not bound	4

In this document we list the RFC, followed by the findings from Radically Open Security (ROS) and our response. Where possible, we reference the pull request associated to the changes we made.

2. Results

2.3 RFC5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

2.3.1 Findings

4.1.2.2. Serial number

ROS: There seems to be no validation logic in place that checks for the length of this field or for its sign.

Response: Validation logic has been added.

4.1.2.4. Issuer

RFC: This field is validated correctly upon generation (not null)

ROS: The BouncyCastle implementation seems to be correct but does not seem to explicitly have support for the required set of standard attributes. The RFC specifies that the implementation should be prepared to receive a certain set of attribute types. It does, but it is also prepared to accept attribute types that are not on the list. It is recommended to create a validator that explicitly checks for this.

Response: Our software doesn't set these attributes explicitly and the RPKI Core uses 'issuer' derived from the public key. No personal data is stored in the RPKI Core.

5.2.3 CRL Number

RFC: Given the requirements above, CRL numbers can be expected to contain long integers. CRL verifiers MUST be able to handle `CRLNumber` values up to 20 octets. Conforming CRL issuers MUST NOT use `CRLNumber` values longer than 20 octets.

Response: We have updated our code and no longer generate `CRLNumber` values longer than 20 octets.

5.3.1 CRL Entry Extensions – Reason Code

RFC: The `reasonCode` is a non-critical CRL entry extension that identifies the reason for the certificate revocation. CRL issuers are strongly encouraged to include meaningful reason codes in CRL entries; however, the reason code CRL entry extension SHOULD be absent instead of using the unspecified (0) `reasonCode` value.

ROS: `org.bouncycastle.asn1.x509.V2TBSCertListGenerator::addCRLEntry()` allows `reasonCode` > 10 to be used, which is not specified by the RFC

Response: We only use "0", which results in an absent extension in the generated CRL. Since BouncyCastly is not web-specific, the maintainers did not want to limit the `reasonCode that` BouncyCastle accepts. See: https://github.com/bcgit/bc-java/pull/930#issuecomment-822401620

2.3.2 Section-specific Remarks

4.1.2.4. Issuer

RFC: To indicate that a certificate has no well-defined expiration date, the `notAfter` SHOULD be assigned the `GeneralizedTime` value of 99991231235959Z.

Response: We don't use open-ended certificates, but we have added validation logic as a preventive measure.

4.1.2.7. Subject Public Key Info

ROS: BouncyCastle uses an ASN1Vector instead of a Sequence, as specified by the RFC to implement this feature

Response: Not-applicable. Reviewer possibly missed line 152 in SubjectPublicKeyInfo.java (in bc-git).

8. Security Considerations

ROS: Loss of a CA's private signing key may also be problematic. The CA would not be able to produce CRLs or perform normal key rollover. CAs SHOULD maintain secure backup for signing keys. The security of the key backup procedures is a critical factor in avoiding key compromise.

Response: Key backup procedures are in place and described in the CPS 6.2.4. In addition, backups of the encrypted private key for the Offline CA are stored as well.

2.4 RFC5652: Cryptographic Message Syntax (CMS)

2.4.1 Findings

5.2 EncapsulatedContentInfo Type

RFC: The optional omission of the `eContent` within the `EncapsulatedContentInfo` field makes it possible to construct "external signatures". In the case of external signatures, the content being signed is absent from the `EncapsulatedContentInfo` value included in the signed-data content type. If the `eContent` value within `EncapsulatedContentInfo` is absent, then the `signatureValue` is calculated and the `eContentType` is assigned as though the `eContent` value was present.

ROS:

- `RPKISignedDataGenerator::generate()` implements support for this specification.
- line 95 mentions a todo about validation logix that seems not to be implemented.
- Verification is done in `org.bouncycastle.cms.SignerInformation.doVerify()`

Response: We have submitted a pull request for the BouncyCastle Java library in order to split out the control paths to add a number of functions that have a single responsibility:
https://github.com/bcgit/bc-java/pull/936
We also added validation checks, see https://github.com/RIPE-NCC/rpki-commons/pull/73

ROS: There seems to be a different implementation of CMS lurking in the BouncyCastle code base which is not used by RPKI Core. At first glance, this seems to be a better implementation - better as in 'better organized, commented and readable'. It was not immediately clear how to change to RPKI Core code to make use of this different implementation.

Response: We have extensions on BouncyCastle for RFC6488 that are specific to the CMS implementation used (see PR). We have decided to use the current CMS implementation for now.
https://github.com/RIPE-NCC/rpki-commons/pull/9

As suggested by ROS, we added more tests on CMS partly related to ROA/MFT. We added property-based tests for building (ROA/MFT) CMS, with generated KeyPairs to encode it, parse it back, and check if properties are still consistent. See https://github.com/RIPE-NCC/rpki-commons/pull/68.

We also added a test highlighting and documenting the main difference between RPKISignedDataGenerator and the original `CMSSignedDataGenerator`, see
https://github.com/RIPE-NCC/rpki-commons/pull/63

2.6 RFC 6481 - A Profile for Resource Certificate Repository Structure

2.6.2 General Remarks

ROS: This RFC is better checked for compliance using a pentest. The RFC mentions requirements about the availability of certain files that can only be verified with the files actually present on a live system. In this particular case, a code audit is not the right tool.

Response: Agreed. This will be included in our upcoming penetration test.

2.7 RFC 6482 - A Profile for Route Origin Authorizations (ROAs)

2.7.1 Findings

3.2 `asID`

RFC: The `asID` field contains the AS number that is authorized to originate routes to the given IP address prefixes.

ROS: There is a CMS ROA parser in place
(`net.ripe.rpki.commons.crypto.cms.roa.RoaCmsParser`)
that validates a ROA CMS message. In the function that handles the actual content, the ASN and prefixes are validated (`RoaCmsParser::parseRouteOriginAttestation`()). On line 128, an ASN is parsed as an id, which upon closer inspection means that it is an integer. There is a problem with this. Since an ASN is an unsigned short (16 bit, 65536 possible values), this could cause problems on the RP's end. If for some reason, the ASN fed to this logic is -66 for example, this would validate OK, but could be cast as positive on the RP's end, e.g. if this were a C implementation on a

router. The same goes for a ASN > 65535, which would just wrap around mod 65536.
Recommendation: Validate if the ASN falls between 0 and 65535.

Response: The value for the ASN was partially checked (for overflow when casting from `BigInteger` to long) in the location the reviewer reported. Further checks were done in another location. We updated the documentation to clarify this. See https://github.com/RIPE-NCC/rpki-commons/pull/45.
Please note that the ROS states that ASN is a 16-bit integer, however this is currently a 32-bit integer.

3.3 `ipAddrBlocks`

RFC: Within a `ROAIPAddress` structure, the address's field represents prefixes as a sequence of type `IPAddress`. (See [RFC3779] for more details). If present, the `MaxLength` MUST be an integer greater than or equal to the length of the accompanying prefix, and less than or equal to the length (in bits) of an IP address in the address family (32 for IPv4 and 128 for IPv6).

ROS: This is incorrectly validated. `RoaCmsParser:`:112 compares the max prefix length to `Integer.MAX_VALUE`, which is 2147483647, regardless of IP family. This could lead to problems on the RP's end when interpreting the max value.

Response: We have updated RPKI Commons to clarify that the validity of the `maxLength` is performed later, see: https://github.com/RIPE-NCC/rpki-commons/commit/729ad976e9a0168c704d2d66d473324b9ab71a47

2.8 RFC 6484 - Certificate Policy for RPKI

2.8.1 Findings

3.4. Identification and Authentication for Revocation Request

RFC: The specific procedures employed for these purposes MUST be described by the CPS for the CA.

ROS: The CPS does not adequately describe its identification and authentication for revocation requests. The identification and authentication of a user is described in section 3.2.3. What is missing is a clear description of what a user is authorized to do. This section alludes to the "can make revocation request" authorization, but rather than mentioning "The user has to authenticate, see 3.2.3", it is implied, which leads to confusion. Perhaps make a list of all authorizations, stick it in 3.2.3 and make a reference to 3.2.3 in all headings that describe such authorization.

Response: Agreed, will be incorporated in the next revision of the CPS.

6.1.6. Public Key Parameters Generation and Quality Checking

ROS: A RIPE NCC employee has mentioned that the HSM holds up to RIPE NCC's own test set. Apart from this, the device is regarded as trusted. Trusting the equipment that arguably holds one of the most important private keys should not come lightly. It is therefore recommended having the HSM audited and pentested to the utmost detail. Any security certifications the device has should be validated by an independent party.

Response: Agreed, this work has already been scheduled.

2.9 RFC 6485 - The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)

2.9.1 Findings

3.1. Public Key Format

RFC: `subjectPublicKey`: `RSAPublicKey` MUST be used to encode the certificate's `subjectPublicKey` field, as specified in [RFC4055].

ROS: This BC implementation uses a `DERBitString` to encode the `subjectPublicKey` field instead of `RSAPublicKey`. It is not clear what the implications are. Apparently BC treats this field generically, to allow for different key types and may not cause any problems. However, it is not in accordance with the RFC.

Response: Bouncycastle, being a general-purpose crypto library does not need this restriction. In RPKI commons we validate the public key signing algorithm to be RSA, so only RSA public keys can be used from any RPKI code.

2.9.2 Comments per section

3.1 Public Key Format

ROS: The function `net.ripe.rpki.commons.crypto.x509cert.x509CertificateBuilderHelper::validateCertificateField` could be expanded by an explicit check for the right algorithm for the public key.

Response: We added validation logic that checks whether the signing algorithm for the public key is RSA.

2.10 RFC 6486 - Manifests for the Resource Public Key Infrastructure (RPKI)

2.10.1 Findings

4.2.1. Manifest

RFC: If a "one-time-use" EE certificate is employed to verify a manifest, the EE certificate MUST have a validity period that coincides with the interval from `thisUpdate` to `nextUpdate`, to prevent needless growth of the CA's CRL.

ROS: Not implemented correctly. The validity interval of the EE certificate exceeds the `nextUpdate` time of the manifest. See under 5.1

5.1. Manifest Generation Procedure

RFC: In the case of a "one-time-use" EE certificate, the validity times of the EE certificate MUST exactly match the `thisUpdate` and times of the manifest.

ROS: The certificate being used is one-time-use, but the validity times of the certificate (now - 5 mins) until (now + 7 days) are not the same as the `thisUpdate` time and `nextUpdate` time of the manifest (now), (now + 24h) respectively. It appears the certificate is being treated as a sequential-use rather than as a one-time-use certificate with regard to validity time and update times.

Response: The code has been updated and we explained the flow a bit better in the code.

2.12 RFC 6488 - Signed Object Template for the Resource Public Key Infrastructure (RPKI)

2.12.1 Findings

2.1.5. CRLs

ROS: CRLs are not used by the application, but there also seems to be no explicit check for the absence of this field. A signed object created by this application will comply with this RFC on this point, but it fails to reject an invalid object created with a different tool.

Response: We updated the code to add validation checks, see: https://github.com/RIPE-NCC/rpki-commons/pull/66

2.1.6.1. version

ROS: The version is determined by the BC lib based on the structure of the message. If the id contains another ASN1 notated field (which it should) then the version is 3. If it is not, the version is 1. There are no checks to explicitly check for the version to be 3 though.

Response: We updated the code to add validation checks, see https://github.com/RIPE-NCC/rpki-commons/pull/70

2.1.6.4. `signedAttrs`

RFC: The signer MAY also include the `signing-time` attribute [RFC5652], the `binary-signing-time` attribute [RFC6019]

ROS: Only `signing-time` is used. `binary-signing-time` is not supported. This may cause problems when parsing CMS data from a different tool.

Response: We added `binary-signing-time` support, following the RFC. So, both `signing-time`, or `binary-signing-time` attributes are supported. If both are present, they should be equal.

2.1.6.7. `unsignedAttrs`

ROS: Unsigned attributes like counter counter signature are not explicitly forbidden by the application, as specified in the RFC. A CMS that conforms to RFC 5652, but not to this RFC will still be handled by the system. It might trigger a code path that it is not supposed to be triggered because of the presence of the unsigned attribute and cause a failure. Especially since counter signatures seem to be poorly implemented in BC. It is recommended to place a check early in the code path to reject a CMS message when it contains unsigned attributes.

Response: We updated the code to add validation checks, see https://github.com/RIPE-NCC/rpki-commons/pull/70

2.14 RFC6490 - Resource Public Key Infrastructure (RPKI) Trust Anchor Locator

2.14.1 Findings

ROS: Possible RCE attack (low probability)
In order to generate the TAL file, the TA file is parsed. The TA file is an XML file that is parsed using an XML serializer. This takes place in the `net.ripe.rpki.ta.persistence.TrustAnchorSerializer.deserialize()` method. This XML serializer uses Xtream 1.4.11.1 to perform the heavy lifting. A common attack vector in serializers is insecure deserialization. This version of Xstream is in itself a patch against insecure deserialization, which was of itself a regression issue.

Response: We investigated our usage of XStream. We were already using XStream security settings to allow-list classes that XStream could deserialise, mitigating the recent vulnerabilities. Furthermore we replaced XStream with a simple DOM-parser for all generation and parsing of externally supplied XML in the code.

ROS: KeyStore Passphrase is hard-coded in source file package `net.ripe.rpki.ta.keystore.TrustAnchorKeyStore` on line 23 contains a hard-coded passphrase:`private final char[] keyStorePassphrase = "X ".toCharArray();` It is bad practice to put sensitive information hard coded in the code base, for multiple reasons.

Response: The hardcoded passphrase was not used in any environment. We used HSM smart cards instead of the passphrase.

2.2 TAL and Trust Anchor Certificate Considerations

RFC: If an entity wishes to withdraw a self-signed CA certificate as a putative trust anchor for any reason, including key rollover, the entity MUST remove the object from the location referenced in the TAL.

ROS: This behaviour is not explicitly implemented. Rather, the object referenced in the TAL is overwritten when a new certificate is generated. It is recommended to perform a hard delete on the old object before generating a new object.

Response: We will investigate this.

2.15 RFC 8181 - A Publication Protocol for the Resource Public Key Infrastructure (RPKI)

2.15.1 Findings

2. Protocol Specification

RFC: The publication protocol uses XML [XML] messages wrapped in signed Cryptographic Message Syntax (CMS) messages, carried over HTTP transport [RFC7230]. The CMS encapsulation is identical to that used in Section 3.1 (and subsections) of RFC 6492 [RFC6492].

ROS: It has been determined that even though XML is used to convey messages, they are not wrapped in CMS, as specified by the RFC. This implies that an attacker who has succeeded in bypassing the layer 4 security and can either MITM or send messages could alter which certificates get published or withdrawn, as well as control the location where the certificates are retrieved from. The CMS requirement safeguards that this does not happen.

RFC: The content of the POST and the server's response will be a well- formed CMS [RFC5652] object with OID = 1.2.840.113549.1.7.2 as described in Section 3.1 of [RFC6492].

ROS: No evidence of this being implemented was found

ROS: The XML is validated against a schema located in `.//resources/rpki-publication-schema.rng.` The RFC specifies that version 4 should be used, however the schema expects the version to be 3.

Response: This publication service is a privately used service, following https://tools.ietf.org/html/draft-ietf-sidr-publication-07. If this changes, we will follow RFC8181. Please see: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server

2.4. Error Handling

RFC: In all other cases, errors result in an XML `<report_error/>` PDU. Like the rest of this protocol, `<report_error/>` PDUs are CMS-signed XML messages and thus can be archived to provide an audit trail.

RFC: The "tag" attribute of the `<report_error/>` PDU associated with a `<publish/>` or `<withdraw/>` PDU MUST be set to the same value as the "tag" attribute in the PDU which generated the error. A client can use the "tag" attribute to determine which PDU caused processing of an update to fail.

ROS: The `<report_error>` tag does not have a `"tag"` attribute

RFC: The body of the `<report_error/>` element contains two sub-elements

ROS: There is no support for these optional elements

Response: This publication service is a privately used service, following https://tools.ietf.org/html/draft-ietf-sidr-publication-07. If this were to change, we will follow RFC8181. Please see: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server

2.5. Error Codes

ROS: Not implemented

Response: This publication service is a privately used service, following https://tools.ietf.org/html/draft-ietf-sidr-publication-07. If this were to change, we will follow RFC8181. Please see: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server

2.6. XML Schema

ROS: The used schema (`rpki-publication-schema.rng`) does not seem to match the RFC.

Response: This publication service is a privately used service, following https://tools.ietf.org/html/draft-ietf-sidr-publication-07. If this were to change, we will follow RFC8181. Please see: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server

2.15.2 General Remarks

ROS: This implementation seems to be a light-weight version of what is specified in the RFC. The use of CMS is omitted, perhaps because of the added complexity, and cryptographic protection has been offloaded to layer
4 (TLS). The risk here is that once there is a change in configuration, for instance TLS is causing problems, someone reading the RFC might think that HTTP is equally good and use HTTP instead of HTTPS, by which the protection that CMS is supposed to provide would go out of this window, potentially result in very bad things happening (DOS of the internet... *dramatic pause*).

Furthermore, error handling seems not to conform to the RFC, but should be verified more thoroughly with a pentest

ROS: Pentest the publication server

Response: This work is scheduled for June 2021.

ROS: Put tools in place that monitor the ports used, that make sure HTTP can NEVER be used to POST messages from CA > server.

Response: All ports used are monitored accordingly.

2.16 RFC 8182 - The RPKI Repository Delta Protocol (RRDP)

2.16.1 Findings

Abstract

RFC: In the Resource Public Key Infrastructure (RPKI), Certificate Authorities (CAs) publish certificates, including end-entity certificates, Certificate Revocation Lists (CRLs), and RPKI signed objects to repositories. Relying Parties retrieve the published information from those repositories. This document specifies a new RPKI Repository Delta Protocol (RRDP) for this purpose. RRDP was specifically designed for scaling. It relies on an Update Notification File which lists the current Snapshot and Delta Files that can be retrieved using HTTPS (HTTP over Transport Layer Security

(TLS)), and it enables the use of Content Distribution Networks (CDNs) or other caching infrastructures for the retrieval of these files.

ROS: The current implementation of the publication server has no support for TLS, instead it uses plain HTTP to connect RPs to the server. This opens the link Repository <> RP up to delay and replay attacks by a MITM. See comments pertaining to 4.3 below.

Response: We added additional text to the readme on: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server
"The publication server relies on a reverse proxy for TLS on the port used for RRDP. Since all Relying Party implementations require HTTPS URLs for the RRDP repository, for production usage the publication server **MUST** be deployed behind a reverse proxy that performs TLS termination, with a trusted network link between reverse proxy and publication server."

4.3. HTTPS Considerations

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

ROS: This is a valid concern. An attacker owning a network node between endpoint could therefore control communications between these endpoints. In this case the communications between the RRDP server and the relying party. An attacker could thus withhold an update to the CRL of the CA that validates the legitimate use
of an IP resource by the ASN. This in the case an attacker has taken illicit control over a resource and the CA is trying to rectify this. Currently, an attacker could prevent this from happening and retain control by withholding the update to the CRL.

Response: We added additional text to the readme on: https://github.com/RIPE-NCC/rpki-publication-server/blob/master/README.md#rpki-publication-server
"The publication server relies on a reverse proxy for TLS on the port used for RRDP. Since all Relying Party implementations require HTTPS URLs for the RRDP repository, for production usage the publication server **MUST** be deployed behind a reverse proxy that performs TLS termination, with a trusted network link between reverse proxy and publication server."

2.17 Hardware Security Module (HSM)

2.17.1 Overview

ROS: It appears the communication between the HSM and RPKI-core is facilitated by a package that has been build for RIPE by Thales As far as is known, RPKI-core uses just one function of this library; the KeyImport tool, which allows a user or process to dump to keys from the HSM into a DB of their choosing.

Response: This was a one-time migration. The key import tool no longer exists.

2.17.2 Findings

ROS: After analysis it seems that this tool may be used to exfil the encrypted keys to a different database. The proposed approach would be for an attacker to program a database adapter that points to a database of his choosing and put this new class in the .m2 path. (if possible). Then call the import tool with the newly created adapter as a parameter.

Response: Only internal staff has the authorisation to perform these actions. Also, please note that the key import tool no longer exists.

ROS: There is no transport security in place on the connection from the migration tool to the database endpoint. An attacker who has interface listening capabilities on either the sender or receiving host could sniff the traffic and obtain the encrypted keys.

ROS: There *seems* to be no transport security in place on the connection from the HSM to the migration tool.

ROS: There seems to be no authentication taking place between the HSM and the migration tool

ROS: The last two findings above are not conclusive, as the component that facilitates this connection has proven difficult to analyse and it might be that these protections are implemented in a different way, perhaps in a dependency of the tool itself.

Response: The application (through Java cryptography extensions for nCipher) communicates with a 'hardserver' process local on the machine. This hardserver process communicates with the HSM in a specific VLAN. The hardserver is configured to communicate with specific HSMs using the 'impath' protocol that authenticates the HSM and provides a secure connection.

2.17.3 General remarks

Redacted: remarks about proprietary code regarding the HSM

2.17.4 Recommendations

Redacted: remarks about proprietary code regarding the HSM

3. Future Work

Redacted: remarks about proprietary code regarding the HSM

Appendix 2 Audit notes

Redacted because it mentions the platforms used during the audit.