RADICALLY OPEN SECURITY

Code Audit Report

Réseaux IP Européens Network Coordination Centre

V 0.6
Diemen, October 1st, 2020
Confidential

Document Properties

Client	Réseaux IP Européens Network Coordination Centre
Title	Code Audit Report
Targets	RPKI-Core Publication Server HSM module
Version	0.6
Pentester	Daniel Attevelt
Authors	Daniel Attevelt, Patricia Piolon
Reviewed by	Patricia Piolon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	September 26th, 2020	Daniel Attevelt	Initial draft
0.2	September 28th, 2020	Patricia Piolon	Restructuring/validation
0.3	September 28th, 2020	Patricia Piolon	Second restructuring pass
0.4	September 29th, 2020	Patricia Piolon	Preliminary Review
0.5	September 29th, 2020	Daniel Attevelt	Processed preliminary review recommendations
0.6	October 1st, 2020	Daniel Attevelt	Added remaining RFC analyses

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	John Sinteur
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between August 3, 2020 and September 25, 2020, Radically Open Security B.V. carried out a code audit for Réseaux IP Européens Network Coordination Centre.

This report contains our findings as well as detailed explanations of exactly how ROS performed the code audit.

1.2 Scope of work

The scope of the code audit was limited to the following target(s):

- RPKI-Core
- Publication Server
- HSM module

The scoped services are broken down as follows:

- Code audit specified targets: 14 days
- Reporting: 3 days
- Extension: 2 days
- **Total effort: 19 days**

1.3 Project objectives

ROS will perform a code audit with RIPE NCC of the RPKI-core and publication server code bases, as well as the software component that interfaces with the HSM. The audit is intended to gain insight into the compliance of various RFCs comprising the RPKI technology as well as the security of RPKI-core, publication server and HSM interface module.

1.4 Timeline

The code audit took place between August 3, 2020 and September 25, 2020.

1.5 Results in a nutshell

All RFCs for RPKI-core and the publication server have been implemented. Regarding the RPKI-core 19 issues have been identified that constitute either a deviation from an RFC, or otherwise constitute a risk to the confidentiality, availability or integrity (CIA) of the system. Regarding the RPKI publication server, 9 such issues have been found. With respect to the HSM, 4 issues have been found that may constitute a risk to the CIA of the RPKI system.

1.6 Methods

The components have been tested for compliance with the identified RFCs. The implicit assumption here is that the RFCs are soundly designed. In other words, if a design flaw in the RFCs themselves would lead to a security problem when implemented correctly, this would naturally be reflected in the implementation, even with 100% adherence. The RFCs have NOT been reviewed for inherent problems; they have been assumed to be designed correctly.

This project consists of three components:

1. RPKI-core, which implements the core functionality of the RPKI.
2. The publication service that exposes RPKI material to the relying parties
3. The Hardware Security Module (HSM), which is used to generate and store key pairs for the Trust Authority (TA)

In order to determine correct implementation, first it was necessary to identify the RFCs related to RPKI. This was done by using RFC6480 as a vantage point and fan out over its several RFCs that make up the components of this infrastructure. These RFCs were cross-referenced with RIPE's CPS to check for overlap.

Once a general overview of what is involved with RPKI was obtained, we could begin with the analysis of the RPKI-core code base. The analysis started with determining the entry points of the application: APIs, web forms, etc. Once these were identified, we carried out a structural analysis to trace and map out each code path. As a more detailed understanding of the application was attained, several components that play a role in the RPKI implementation were identified.

The next step was to structure the attained knowledge by grouping it by RFC. The would allow an RFC-by-RFC analysis and would make it possible to determine if or to what level the RFC are implemented.

Taking each RFC as a starting point, the appropriate software components were identified and meticulously traced, so that they could be held up against the specification by the RFC. In addition, unit tests were consulted to attain insight into the operation of every component.

HSM-specific

There is relatively little code that interacts with the HSM. The code that is in place copies keys from the HSM into a database. This code has been reviewed. Furthermore, the component that does the copying uses bespoke software from Thales. This library has been decompiled and analysed for anomalies.

External libraries

The RPKI-core component relies heavily on the bouncy castle (BC) cryptographic library. As this library forms such an important part of the RPKI functionality, it has been considered in scope with respect to the functions that are actually used by RPKI-core. Unused functions are considered out of scope. RPKI core uses the native Java x509 implementation to represent x509 certificates. All native Java libraries have been considered out of scope. Other external libraries, where relevant, have been considered out of scope, although some research has been done into where they may form a problem to the confidentiality, integrity or availability of the component. In short, we tried to find out if any of the used components carried known vulnerabilities.

2 Results

2.1 Overview

All RFCs have been analysed in a section-by-section fashion in order to determine compliance to the RFC and to identify security problems. The vast majority of all sections covered show no issues and therefore do not show up in the results.

The sections that *have* been found to have issues are listed in the 'Findings' section for that RFC. These should be given priority when following up on the report. Where there are remarks associated with a specific section - e.g. when a section is deemed to be out of scope, or when an implementation complies with the RFC, but implicitly rather than explicitly - they are noted in the "Section-specific Remarks" section.

General remarks regarding the RFC are listed in the "General Remarks" sections.

Pentesting tips - such as what to focus on, or approaches to attack part of the application - are either noted in the "General Remarks" sections or in a section of their own.

Finally, if there is an issue or comment about a specific paragraph of an RFC section, this portion of the text is included as a blockquote, with any remarks listed below:

> Example text to shown how paragraphs of sections of the RFCs are represented in the report.

- This is an example remark about the above block

It is recommended to read the results carefully, and not only focus on the issues. Each section contains comments and recommendations that provide additional information.

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	# of issues
RPKI core	RFC 3779 (page 10)	0
RPKI core	RFC 5280 (page 11)	5
RPKI core	RFC 5652 (page 14)	2
RPKI core	RFC 6480 (page 16)	0
RPKI core	RFC 6481 (page 17)	-
RPKI core	RFC 6482 (page 17)	2
RPKI core	RFC 6484 (page 18)	2
RPKI core	RFC 6485 (page 20)	1
RPKI core	RFC 6486 (page 20)	3
RPKI core	RFC 6487 (page 22)	0
RPKI core	RFC 6488 (page 25)	4
RPKI core	RFC 6489 (page 26)	0

RPKI core	RFC 6490 (page 27)	3
Publication server	RFC 8181 (page 29)	7
Publication server	RFC 8182 (page 31)	2
Hardware Security Module (HSM)	Not bound to RFC (page 33)	4

2.2 RFC 3779 - X.509 Extensions for IP Addresses and AS Identifiers

2.2.1 Findings

- No findings

2.2.2 General Remarks

- The unit test for the validator (`X509ResourceCertificateParentChildValidatorTest`) contains only one test case that checks for an invalid resource set. There should be more, containing more elaborate test cases.
- The unit tests with regard to validation should be expanded. For instance, vary the root certificate. Ideally, each permutation should have a test case.

2.2.3 Pentesting Tips

- [redacted]
- [redacted]

2.3 RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

2.3.1 Findings

4.1.2.2. Serial Number

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

4.1.2.4. Issuer

- This field is validated correctly upon generation (not null)
- The BC implementation seems to be correct, but does not seem to explicitly have support for the specified required set of standard attributes. The RFC specifies that the implementation should be prepared to receive a certain set of standard 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.

5.2.3. CRL Number

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.

- This isn't guaranteed by validation logic

5.3. CRL Entry Extensions

5.3.1. Reason Code

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.

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

7.1. Internationalized Names in Distinguished Names

Implementations may encounter certificates and CRLs with names encoded using `TeletexString`, `BMPString`, or `UniversalString`, but support for these is OPTIONAL.

- It appears BC does not implement support for these types

2.3.2 Section-specific Remarks

4.1.2.4. Issuer

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

- Not implemented

4.1.2.7. Subject Public Key Info

- BC uses an ASN1Vector instead of a Sequence, as specified by the RFC to implement this feature

4.1.2.8. Unique Identifiers

- Could not verify. It seems the code base does not use these identifiers.

4.2.1.6. Subject Alternative Name

- Appears not to be implemented

4.2.1.7. Issuer Alternative Name

- Appears not to be implemented

4.2.1.8. Subject Directory Attributes

- Appears not to be implemented

4.2.1.10. Name Constraints

- Not implemented

4.2.1.11. Policy Constraints

- Not implemented

5.2.2. Issuer Alternative Name

- Not used, out of scope

5.2.4. Delta CRL Indicator

- Not used, out of scope

5.2.5. Issuing Distribution Point

- Not used, out of scope

5.2.6. Freshest CRL (a.k.a. Delta CRL Distribution Point)

- Not used, out of scope

5.2.7. Authority Information Access

- Not used, out of scope

5.3.3. Certificate Issuer

- Not used, out of scope

6. Certification Path Validation

- It appears X509 certificates are validated by an algorithm that is implemented by `java.security.cert.X509Certificate`. As such, it is considered out of scope

7.2. Internationalized Domain Names in GeneralName

- This presumably takes place in the validation of a certificate, which is considered out of scope, as it takes place in the Java security library

7.3. Internationalized Domain Names in Distinguished Names

- This presumably takes place in the validation of a certificate, which is considered out of scope, as it takes place in the Java security library

7.4. Internationalized Resource Identifiers

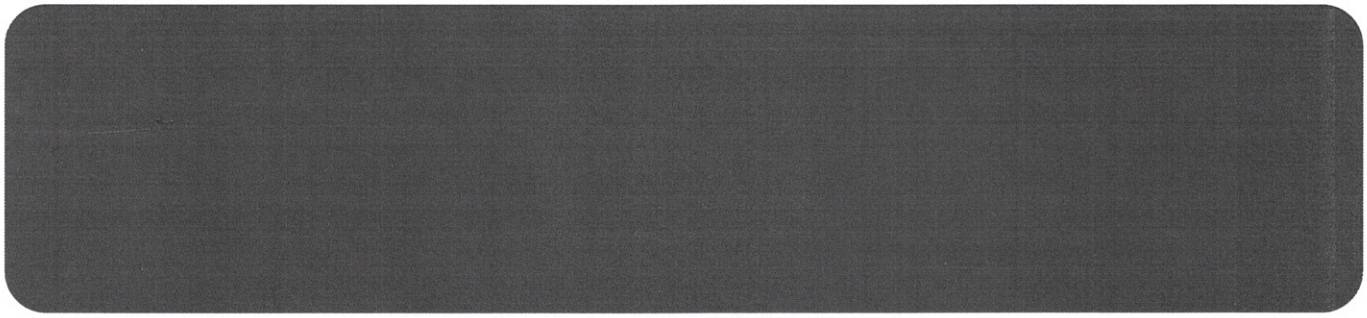
- This presumably takes place in the validation of a certificate, which is considered out of scope, as it takes place in the Java security library

8. Security Considerations

CAs MUST encode the distinguished name in the subject field of a CA certificate identically to the distinguished name in the issuer field in certificates issued by that CA. If CAs use different encodings, implementations might fail to recognize name chains for paths that include this certificate. As a consequence, valid paths could be rejected.

- This seems to be the case, because BC seems to lack support for different types. See 7.1

2.3.3 Pentesting Tips



2.4 RFC 5652 - Cryptographic Message Syntax (CMS)

2.4.1 Findings

5.2 EncapsulatedContentInfo Type

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.

- `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()`

- Remarks on signature verification: The bulk of the signature verification is done in the private `doVerify()` function of the `SignerInformation` class. This function is of a very low quality. Lines 352-423 contain many controls paths that make it hard to evaluate what is going on. Further more, it's basically one blurb of code that could easily be split up in dedicated functions that do one thing. The function is unit tested in the `NewDataSingedTest` class. Even though there is a fair amount of testing taking place, it nowhere near covers all permutations. For example, there are no tests in place that particularly test the different attributes a singer-info type may have, such as content-type or signing-time. The unit tests seem to minimally test the verification process.

2.4.2 Section-specific Remarks

6. Enveloped-data Content Type

- This functionality is not used by RPKI-core. It is therefore considered out of scope.

8. Encrypted-data Content Type

- This functionality is not used by RPKI-core. It is therefore considered out of scope.

10.1.3. KeyEncryptionAlgorithmIdentifier

- Out of scope

10.1.4. ContentEncryptionAlgorithmIdentifier

- Out of scope

10.1.5. MessageAuthenticationCodeAlgorithm

- Out of scope

10.1.6. KeyDerivationAlgorithmIdentifier

- Out of scope

2.4.3 General Remarks

- The classes of the root package `net.ripe.rpki.commons.crypto.cms` (RPKI*) are not unit-tested. The procided tests that should cover this logic could be considered integration tests.
- It is recommended to write extensive tests for these classes as these form the core of CMS generation / verification. Furthermore, it is recommended to write these tests exhaustively.
- The tests that are already in place that test the different types of signed objects (ROA, Manifest) should be extended, as they currently are lacking in test cases.
- There seems to be a different implementation of CMS lurking in the BC 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.
- Signature verification is a method in the BC library. BC does check that all fields are conforming to the RFC, however the way it does this is a bit obscure and could be organized better. This may be a risk when development continues and regression errors are introduced.

2.4.4 Recommendations

- It is recommended to review the relevant bouncy castle CMS code and perhaps find a way to refactor that code, or use the different implementation mentioned.

2.5 RFC 6480 - An Infrastructure to Support Secure Internet Routing

This is the umbrella RFC for the RPKI technology. It ties together different RFCs that together make up this technology. As such, it is too abstract to be considered for audit. The general rule here is that the level of compliance to the RFCs of the different components determines the level of compliance to this RFC.

2.6 RFC 6481 - A Profile for Resource Certificate Repository Structure

2.6.1 Findings

- Not evaluated

2.6.2 General Remarks

- 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.

2.7 RFC6482 - A Profile for Route Origin Authorizations (ROAs)

2.7.1 Findings

3.2. asID

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

- 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.

3.3. ipAddrBlocks

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).

- 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.

2.7.2 Section-specific Remarks

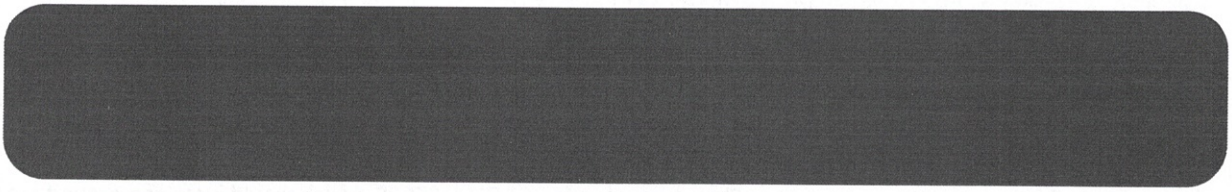
4. ROA Validation

- Out of scope, RP specific

2.7.3 General Remarks

- The unit tests for the ROA CMS parser only contain tests for the happy path of the tested methods. This will guarantee the implementation works for sound data, but does not say anything for cases in which the provided data is unsound, and as such gives a false sense of security.

2.7.4 Pentesting Tips



2.8 RFC 6484 - Certificate Policy for RPKI

2.8.1 Findings

3.4. Identification and Authentication for Revocation Request

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

- 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.

6.1.6. Public Key Parameters Generation and Quality Checking

- A RIPE employee has mentioned that the HSM holds up to RIPE'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.

2.8.2 Section-specific Remarks

2.2. Publication of Certification Information

Each CA MUST publish the certificates (intended for public consumption) that it issues via the repository system.

Each CA MUST publish the CRLs (intended for public consumption) that it issues via the repository system.

Each CA MUST publish its RPKI signed objects (intended for public consumption) via the repository system.

- It is pretty clear that this is in place, but is better verified by means of a pentest

2.4 Access Controls on Repositories

Each CA or repository operator MUST implement access controls to prevent unauthorized persons from adding, modifying, or deleting repository entries.

- The RPKI-core has a login screen to authenticate, but there are also command line tools that can be run. It will take a pentest to properly assess the quality of these controls.

3.1.2. Need for Names to Be Meaningful

- This is best verified through a pentest

6.1.3. Public Key Delivery to Certificate Issuer

- Confirmed that the online CA and hosted CAs share the same system and no pubkey transport takes place

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

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

- 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.

2.9.2 Comments per section

3.1. Public Key Format

- The function `net.ripe.rpki.commons.crypto.x509cert.X509CertificateBuilderHelper::validateCertificateFi` could be expanded by an explicit check for the right algorithm for the public key.

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

2.10.1 Findings

4.2.1. Manifest

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.

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

If a "sequential-use" EE certificate is employed to verify a manifest, the EE certificate's validity period needs to be no shorter than the `nextUpdate` time of the current manifest. The extended validity time raises the possibility of a substitution attack using a stale manifest, as described in Section 6.4.

- This attack seems also feasible in the one-time-use case of the EE certificate. An attacker could suppress a newly created manifest with for instance CRLs, with a stale manifest which has been previously generated. The RP would generate a warning that the manifest is stale, but the signature is still valid per the validity interval of the certificate.

5.1. Manifest Generation Procedure

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

- 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.

2.10.2 Section-specific Remarks

5.2. Considerations for Manifest Generation

A new manifest MUST be issued and published on or before the `nextUpdate` time.

- Could not verify, needs to be checked with a pentest

6. Relying Party Use of Manifests

- Out of scope

7. Publication Repositories

- Needs to be checked with a pentest

2.11 RFC 6487 - A Profile for X.509 PKIX Resource Certificates

2.11.1 Findings

- None

2.11.2 Comments per section

4.6. Validity

While a CA is typically advised against issuing a certificate with a validity period that spans a greater period of time than the validity period of the CA's certificate that will be used to validate the issued certificate, in the context of this profile, a CA MAY have valid grounds to issue a subordinate certificate with a validity period that exceeds the validity period of the CA's certificate.

- For Manifests, the validity time of the manifest EE certificate is specified to be now -5 minutes to now + 7 days. `(net.ripe.rpki.domain.manifest.ManifestEntity:149)`
- For ROA this is from now to infinity `(net.ripe.rpki.domain.roa.RoaSpecification:125)`
- Upon generation of the signed EE certificate, there is no explicit checking taking place if the validity of the resource certificate does not extend the validity of the signing CA. `(See CertificateManagementServiceImpl::issueSingleUseEeResourceCertificate())`
- The RFC is a bit ambivalent regarding this topic, see below. The current behaviour is allowed and as such it is not a finding. However, it deserves some thought if EE end times should not be open-ended.
- It is recommended to build validation logic to check if the validity of the EE certificates does not extend that of the signing CA certificate.

4.6.2. notAfter

It is valid for a certificate to have a value for this field that post-dates the same field value in any superior certificate. The same caveats apply to RP's assumptions relating to the certificate's validity at any time other than the current time.

This is ambivalent information with respect to 4.6.

4.8.2. Subject Key Identifier

> This extension MUST appear in all resource certificates. This extension is non-critical.

- This is not guaranteed by the current implementation. The `subjectkeyidentifier` field is set with a setter, and upon building, it is checked if the field is set. This allows the possibility for this extension not to be present in the generated certificate. It is recommended to remove the setter + check and to add the extension upon generation by default. If this is not possible, the X509CertificateBuilderHelper should be subclassed and used instead of the current class to enforce compliance of this rule. The default value is set to true, but it makes no sense to generate non-compliant certificates. Furthermore, there are no checks in place that check for the presence of this extension. This includes a unit-test. See `*.crypto.x509cert.X509CertificateBuilderHelper::createCertificateGenerator()`.

4.8.3. Authority Key Identifier

- The comments about section 4.8.2. apply here as well.

4.8.4. Key Usage

- The current implementation does not guarantee this extension is present. Even though all requirements are met in the current implementation, a future change might brake this. It is recommended to have the blockquote always try to add this extension, but with a validity check before the line is triggered.

4.8.5. Extended Key Usage

- The current implementation does not guarantee this extension is NOT present. See above points. Furthermore, a unit test should be written to cover this test-case.

4.8.6. CRL Distribution Points

- This logic is better tested with a pen test.
- It is recommended to put unit-tests in place for the different RFC requirements.

4.8.7. Authority Information Access

- This logic is better placed in yet another sub-class of the (below) proposed `X509ResourceCertificateBuilderHelper`, but in this case for self-signed certificates.
- It's recommended to put attention to this in a pen-test.

4.8.8. Subject Information Access

- This seems to be implemented correctly, but is quite complex. Verify with pen-test

4.8.9. Certificate Policies

- The current implementation is a composition which adds one policy to the list of policies that causes the extension to be loaded. However, a future programmer could still add another policy and break compliance.

4.8.10. IP Resources

> Either the IP Resources extension, or the AS Resources extension, or both, MUST be present in all RPKI certificates, and if present, MUST be marked critical.

- There is no explicit check or test for this case in `X509CertificateBuilderHelper::addResources()`

4.8.11. AS Resources

- See 4.8.10.

6. Resource Certificate Requests

- This functionality seems to be implemented in `CertificateIssuanceRequest`.
- This appears to be a roll-your-own CIR, as no specific format is actually generated, as processing the CIR takes place within the system.
- It is not clear if the implemented logic should conform to the RFC.

7.1. Resource Extension Validation

- Validation takes place in `X509ResourceCertificateParentChildValidator`
- This deserves exceptional attention during a pen-test. Recommended smart fuzz testing.

2.11.3 Recommendations

- The current X509CertificateBuilderHelper is a free-form builder helper that works for all types of x509 certificates. The blockquote that uses this class is tasked with setting bits that determine if extensions are to be included. It is recommended to build a `X509ResourceCertificateBuilderHelper` that takes care of this. As such, it is guaranteed that any resource certificate is compliant. If for certain reasons this is not feasible, it is recommended to expand the existing unit tests for this helper class, to reflect the requirements of the RFC.

- It is recommended to build yet a different sub-class especially for self-signed certificates.
- An attempt to tackle the mentioned problems regarding the above recommendations is already in place in `net.ripe.rpki.commons.crypto.x509cert.RpkiCaCertificateBuilder`, but does not seem to be used. Perhaps due to an unfinished refactoring cycle?
- For a pen-test, allocate time and resources to try to break the validation of the certificate chain, this to yield bugs in the validation/parsing logic.

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

2.12.1 Findings

2.1.5. CRLs

- 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.

2.1.6.1. version

- 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.

2.1.6.4. signedAttrs

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

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

2.1.6.7. unsignedAttrs

- 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.

2.13 RFC 6489 - Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)

2.13.1 Findings

- None that would indicate a function al security issue with the implementation of this RFC.

2.13.2 Section-specific Remarks

4.2. RPKI Signed Objects

The RFC specifies two approaches of dealing with signed objects. The following is chosen:

> A CA MAY choose to treat this EE certificate the same way that it deals with CA certificates, i.e., to copy over all fields and extensions, and MAY change only the `notBefore` date and the serial number. If the CA adopts this approach, then the new EE certificate is inserted into the CMS wrapper, but the signed context remains the same. (If the signing time or binary signing time values in the CMS wrapper are non-null, they MAY be updated to reflect the current time.)

2.13.3 General Remarks

- It is advised to refactor the key rollover process. After consultation with RIPE staff, it was mentioned that the implemented `CAResourceClasses` are obsolete. These classes make the logic needlessly complex and hard to maintain. From a security perspective, simple == secure. Furthermore, it is recommended to separate the preparation (rekeying, signing, revoking) and publication steps, i.e. to not automatically publish when properties change. Usually, this is desired, but in case of key rollover, apparently it is not. One could for instance issue multiple events: one does the preparatory work, the other publishes. In case of key rollover, one could simply not emit the publication event, and only do so when it is necessary.

- Additionally, it is *highly* recommended to apply block comments to meticulously describe what is going on. Not only does this make code audits faster (and cheaper!), but in case of doing work on the code, it is way easier to grasp what is going on.
- Furthermore, it is not clear if this is in place, but it is advised to write integration tests for the rollover process. There are many different moving parts that are dependent on each other with respect to time, and as such transcend the unit level.
- Finally, a pentest should cover this part explicitly if an integration test is not yet in place. The output of that test could be used to set up the appropriate test parameters.

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

2.14.1 Findings

- 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.
- 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 = "XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXX".toCharArray();` It is bad practice to put sensitive information hard coded in the code base, for multiple reasons:
  1. From a programming perspective, it lacks flexibility
  2. From a security perspective, this passphrase is now immortalized in git. When the source code leaks, an attacker could get their hands on this information.

2.2.

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.

- 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.

2.14.2 Section-specific Remarks

2.2.

The referenced object MUST be a self-signed CA certificate that conforms to the RPKI certificate profile [RFC6487].

- Check during pentest

The trust anchor MUST be published at a stable URI. When the trust anchor is reissued for any reason, the replacement CA certificate MUST be accessible using the same URI.

- This is true as long as the subject name stays the same, because the URI is based on the subject name. If this changes, the URI changes with it.

It MUST NOT use the "inherit" form of the INR extension(s). The INR set described in this certificate is the set of number resources for which the issuing entity is offering itself as a putative trust anchor in the RPKI

- Could not validate if the "inherit" form is not used. This may be correctly implemented, but it could not be verified.

2.14.3 Recommendations

- It is recommended to look for an alternative to the XML serializer, or build a parser rather than using a serializer.
- Remove the hard-coded passphrase, and build logic that sets this variable as necessary.

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

2.15.1 Findings

2. Protocol Specification

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].

- 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.

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].

- No evidence of this being implemented was found

2.1. Common XML Message Format

- 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.

2.4. Error Handling

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.

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.

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

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

- There is no support for these optional elements

2.5. Error Codes

- Not implemented

2.6. XML Schema

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

5. Security Considerations

2.15.2 General Remarks

- 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

Each RPKI publication protocol message is wrapped in a signed CMS message, which provides message integrity protection and an auditable form of message authentication. Because of these protections at the application layer, and because all the data being published are intended to be public information in any case, this protocol does not, strictly speaking, require the use of HTTPS or other transport security mechanisms. There may, however, be circumstances in which a particular publication operator may prefer HTTPS over HTTP anyway, as a matter of (BPKI) CA policy. Use of HTTP versus HTTPS here is, essentially, a private matter between the repository operator and its clients. Note, however, that even if a client/server pair uses HTTPS for this protocol, message authentication for this protocol is still based on the CMS signatures, not HTTPS.

- Emphasizing this block, due to its importance.

2.15.3 Recommendations

- Implement CMS, as specified by the RFC
- Pentest the publication server
- Put tools in place that monitor the ports used, that make sure HTTP can NEVER be used to POST messages from CA > server.

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

2.16.1 Findings

Abstract

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.

- 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.

4.3. HTTPS Considerations

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.

- 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.

2.16.2 Section-specific Remarks

3.2. Certificate Authority Use

The accessLocation MUST be an HTTPS URI as defined in [RFC7230] that will point to the Update Notification File for the Repository Server that publishes the products of this Certificate Authority certificate.

* Could not verify, check with pentest

3.3. Repository Server Use

3.3.2. Publishing Updates

* This is really better tested on a running system, rather than a code audit.

3.4. Relying Party Use

* Out of scope

2.16.3 Recommendations

- Include this RFC in the scope of the pentest.
- Implement HTTPS for the RRDP side of the publication server, specifically the suggestions mentioned in section 4.3
- Instruct the pentesters to read section 5, as it mentions interesting ways this system could be abused.

2.17 Harware Security Module (HSM)

2.17.1 Overview

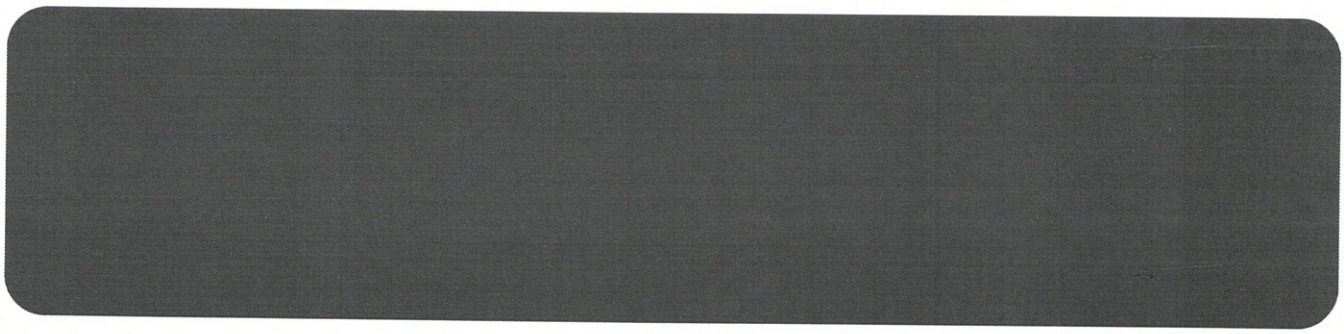
It appears the communication between the HSM and RPKI-core is facilitated by a package that has been build for RIPE by Thales As fas as is known, RPKI-core uses just one function of this library; the `KeyImport` tool, which allows a user or process to dump to keys from the HSM into a DB of their choosing.

2.17.2 Findings

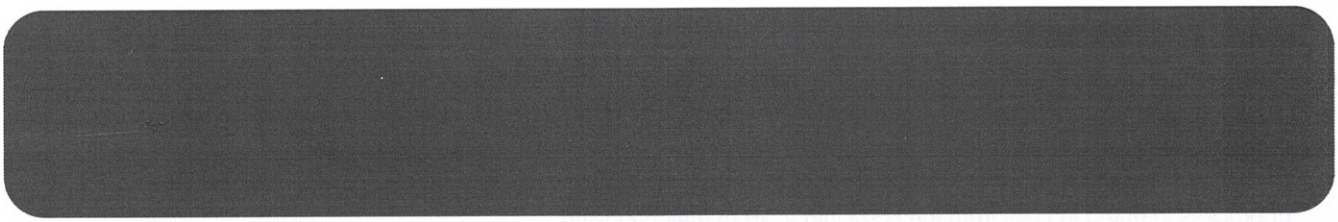
- After analysis it seems that this tool may be used to exfil the encrypted keys to a different database. The proposed approach would be for an attacker to program a database adapter that points to a database of his choosing and put this new class in the `.m2` path. (if possible). Then call the import tool with the newly created adapter as a parameter.
- 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.
- There *seems* to be no transport security in place on the connection from the HSM to the migration tool.
- There seems to be no authentication taking place between the HSM and the migration tool

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.

2.17.3 General Remarks



2.17.4 Recommendations



- Review the network architecture and firewall rules to not allow outbound traffic from hosts that contain sensitive information.

3 Future Work

Penetration test

It is *highly* recommended to have a crystal box penetration test performed on a system that is as close to the live system as possible. An application is but a part of a system, and as such, a code audit only gives a partial impression of the security of a system. An attacker is required to break through multiple layers of security, many of which lie outside the context of the application itself. It is therefore necessary to test an application in its natural habitat in order to *reliably* say something about the security of a system as a whole.

As not all RFCs could by audited, it is recommended to focus on these RFCs. Furthermore, it is recommended to read the comments for each RFC, take note of any pentesting tips there and incorporate them within the scope.

Evaluation of the HSM

[redacted] evaluate all security controls that surround the HSM. This includes evaluation of administrative controls (access policies, backup procedures, emergency procedures, etc.), detective controls (which processes are in place to detect unauthorized access or other security problems) and preventive controls (which controls are in place to prevent unauthorized access to the HSM). What should also be included is a security review of partners of RIPE. Keys are backed up, off-site. Which security controls do RIPE's partners have in place?

The penultimate way to test RIPE's ability to withstand an attack, is to actually attack it. A red-team test to this end could be considered. This is a type of attack where a security company is tasked with trying to breach the security of a system. Unlike a pentest, this test also tries to bypass physical defenses, gain entry to the facility, use social engineering to obtain key information, etc.

Improvement of unit tests

Some effort should be allocated to improve the quality of unit tests. The focus should be to identify unit tests with no test coverage in key logic, or with only a happy path test. While writing unit tests, it is not uncommon to unearth problems that would otherwise have gone unnoticed.

Implementation of integration tests

Perhaps this point is superfluous as they might already be in place, however, it is recommended to create integration tests for the various functions of the RPKI-core and publication server. Whereas unit tests test the application at the unit level, an integration test tests how well the units work together. A prime candidate for such a test would be the key rollover functionality. The current implementation is legacy, but what is special about it is its non-atomic nature - several operations spaced out in time are needed to perform a key rollover. This is a property that makes it impossible to test at the unit level and makes manual testing cumbersome.

4 Conclusion

General impression

The code bases of RPKI-core and the publication server are of high quality. The code bases have been nicely compartmentalized in discrete logic units, tied together in established design patterns. RPKI-core adheres to interface-based programming patterns, which allows for this compartmentalization. There is always a risk in over-engineering a program using the object oriented paradigm, but this hardly seems the case regarding this project. Methods generally do only one thing, and do it well.

The same can not be said for the BC library. Even though the code is compartmentalized well, the implementation of certain methods is rather sloppy. To name just one example: one method was encountered with as many as 4 nested if/elseif/else statements. This style of coding immediately draws an attacker's attention, as sloppy code is usually vulnerable code.

The BC library is severely lacking in unit tests. There are none in place for the methods that are used by RPKI-core. This means the user of the library has to rely on faith rather than proof of the correct operation of the library.

Quality of unit tests

Even though the stated 75% test coverage seems to be met, the quality of the unit tests is, in certain parts, lacking in both number and quality. Generally speaking, using test coverage as an indicator of code quality is a bit misleading. There is no linear relationship with test coverage and quality of the tested code. If for example only the happy paths are tested, you get test coverage, but the only statement that can be reliably made is that the unit functions correctly if the units it depends upon supply it with the expected parameters. Or, in other words, that the unit works correctly if the units it depends upon also work correctly.

In this light, there is room for improvement for existing unit tests. It seems that unit test have been put in place, in part to guarantee the quality of the code base, but also in part to meet the coverage criterion.

Unit tests tended to lack coverage of cases where the input is NOT as expected. This is especially important with respect to the security of an application, as it is the root of exploitation. Hacking takes place in the differential between what a program is designed to do, and what a program actually does.

RPKI-core

The implementation of RPKI-core complies with the RPKI RFCs to a high degree, although there some issues are present. The implementation of the TAL relies on a package with a known track record of insecure deserialization, and contains a hard-coded passphrase. The Key Rollover implementation (RFC 6489) contains legacy code, and a rather obscure implementation in general. The implementation of manifests (RFC 6486) contains some issues regarding the validity of the EE certificate, in conjunction with the next update time. This could possibly be exploited by an attacker, using a replay attack. Lastly, there are some problems with in-logic validation of data types that are performed during parsing.

RPKI publication server

This component lacks the implementation of a CMS wrapper around the XML messages that it takes as input from a CA. This provides an attacker with a greater attack surface while attacking this particular part of the RPKI system. Furthermore, data egress to Relaying Parties is not protected by any form of transport security. This opens the system

up to a MITM attack and data withholding or replaying attacks. As such, this component seems to be the most at risk component within the RPKI system.

Hardware security module

The component of RPKI core that copies data from the HSM to the core database may be vulnerable to unauthorized exfiltration of sensitive cryptographic material.

Appendix 1 Testing team

Daniel Attevelt	Daniel is a passionate security engineer, with more than 30 years of technical experience. He has completed a software developer career ending in a lead developer / Chief Security Officer role before working as a cyber security contractor.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.

Appendix 2 Audit notes