

SIDROPS
Internet-Draft
Updates: [6486](#) (if approved)
Intended status: Standards Track
Expires: January 1, 2022

R. Austein
Arrcus, Inc.
G. Huston
APNIC
S. Kent
Independent
M. Lepinski
New College Florida
July 8, 2021

**Manifests for the Resource Public Key Infrastructure (RPKI)
draft-ietf-sidrops-6486bis-05**

Abstract

This document defines a "manifest" for use in the Resource Public Key Infrastructure (RPKI). A manifest is a signed object (file) that contains a listing of all the signed objects (files) in the repository publication point (directory) associated with an authority responsible for publishing in the repository. For each certificate, Certificate Revocation List (CRL), or other type of signed objects issued by the authority that are published at this repository publication point, the manifest contains both the name of the file containing the object and a hash of the file content. Manifests are intended to enable a relying party (RP) to detect certain forms of attacks against a repository. Specifically, if an RP checks a manifest's contents against the signed objects retrieved from a repository publication point, then the RP can detect "stale" (valid) data and deletion of signed objects.

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 January 1, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1.	Introduction	3
1.1.	Requirements Language	3
2.	Manifest Scope	4
3.	Manifest Signing	4
4.	Manifest Definition	4
4.1.	eContentType	4
4.2.	eContent	5
4.2.1.	Manifest	5
4.2.2.	Names in FileAndHash objects	7
4.3.	Content-Type Attribute	7
4.4.	Manifest Validation	7
5.	Manifest Generation	7
5.1.	Manifest Generation Procedure	7
5.2.	Considerations for Manifest Generation	9
6.	Relying Party Processing of Manifests	9
6.1.	Manifest Processing Overview	10
6.2.	Acquiring a Manifest for a CA	11
6.3.	Detecting Stale and or Prematurely-issued Manifests	11
6.4.	Acquiring Files Referenced by a Manifest	11
6.5.	Matching File Names and Hashes	11
6.6.	Failed Fetches	12
7.	Publication Repositories	12
8.	Security Considerations	12
9.	IANA Considerations	13
10.	Acknowledgements	13
11.	References	13
11.1.	Normative References	13
11.2.	Informative References	15
Appendix A.	ASN.1 Module	15
	Authors' Addresses	16

1. Introduction

The Resource Public Key Infrastructure (RPKI) [[RFC6480](#)] makes use of a distributed repository system [[RFC6481](#)] to make available a variety of objects needed by relying parties (RPs). Because all of the objects stored in the repository system are digitally signed by the entities that created them, attacks that modify these published objects are detectable by RPs. However, digital signatures provide no protection against attacks that substitute "stale" versions of signed objects (i.e., objects that were valid and have not expired, but have since been superseded) or attacks that remove an object that should be present in the repository. To assist in the detection of such attacks, the RPKI repository system can make use of a signed object called a "manifest".

A manifest is a signed object that enumerates all the signed objects (files) in the repository publication point (directory) that are associated with an authority responsible for publishing at that publication point. Each manifest contains both the name of the file containing the object and a hash of the file content, for every signed object issued by an authority that is published at the authority's repository publication point. A manifest is intended to allow an RP to detect unauthorized object removal or the substitution of stale versions of objects at a publication point. A manifest also is intended to allow an RP to detect similar outcomes that may result from a man-in-the-middle attack on the retrieval of objects from the repository. Manifests are intended to be used in Certification Authority (CA) publication points in repositories (directories containing files that are subordinate certificates and Certificate Revocation Lists (CRLs) issued by this CA and other signed objects that are verified by end-entity (EE) certificates issued by this CA).

Manifests are modeled on CRLs, as the issues involved in detecting stale manifests and potential attacks using manifest replays, etc., are similar to those for CRLs. The syntax of the manifest payload differs from CRLs, since RPKI repositories contain objects not covered by CRLs, e.g., digitally signed objects, such as Route Origination Authorizations (ROAs).

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. Manifest Scope

A manifest associated with a CA's repository publication point contains a list of:

- o the set of (non-expired, non-revoked) certificates issued and published by this CA,
- o the most recent CRL issued by this CA, and
- o all published signed objects that are verifiable using EE certificates [[RFC6487](#)] issued by this CA.

Every RPKI signed object includes, in the Cryptographic Message Syntax (CMS) [[RFC3370](#)] wrapper of the object, the EE certificate used to verify it [[RFC6488](#)]. Thus, there is no requirement to separately publish that EE certificate at the CA's repository publication point.

Where multiple CA instances share a common publication point, as can occur when a CA performs a key-rollover operation [[RFC6489](#)], the repository publication point will contain multiple manifests. In this case, each manifest describes only the collection of published products of its associated CA instance.

3. Manifest Signing

A CA's manifest is verified using an EE certificate. The SubjectInfoAccess (SIA) field of this EE certificate contains the access method OID of id-ad-signedObject.

The CA MUST sign only one manifest with each generated private key, and MUST generate a new key pair for each new version of the manifest. This form of use of the associated EE certificate is termed a "one-time-use" EE certificate.[[RFC6487](#)]

4. Manifest Definition

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

4.1. eContentType

The eContentType for a manifest is defined as id-ct-rpkiManifest and has the numerical value of 1.2.840.113549.1.9.16.1.26.


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

```
id-ct OBJECT IDENTIFIER ::= { id-smime 1 }
```

```
id-ct-rpkiManifest OBJECT IDENTIFIER ::= { id-ct 26 }
```

[4.2.](#) eContent

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

```
Manifest ::= SEQUENCE {
    version      [0] INTEGER DEFAULT 0,
    manifestNumber INTEGER (0..MAX),
    thisUpdate    GeneralizedTime,
    nextUpdate    GeneralizedTime,
    fileHashAlg   OBJECT IDENTIFIER,
    fileList      SEQUENCE SIZE (0..MAX) OF FileAndHash
}
```

```
FileAndHash ::= SEQUENCE {
    file      IA5String,
    hash      BIT STRING
}
```

[4.2.1.](#) Manifest

The manifestNumber, thisUpdate, and nextUpdate fields are modeled after the corresponding fields in X.509 CRLs (see [[RFC5280](#)]). Analogous to CRLs, a manifest is nominally current until the time specified in nextUpdate or until a manifest is issued with a greater manifest number, whichever comes first.

Because 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 in the manifest, to prevent needless growth of the CA's CRL.

The data elements of the manifest structure are defined as follows:

version:

The version number of this version of the manifest specification MUST be 0.

manifestNumber:

This field is an integer that is incremented each time a new manifest is issued for a given publication point. This field allows an RP to detect gaps in a sequence of published manifests.

As the manifest is modeled on the CRL specification, the ManifestNumber is analogous to the CRLNumber, and the guidance in [\[RFC5280\]](#) for CRLNumber values is appropriate as to the range of number values that can be used for the manifestNumber. Manifest numbers can be expected to contain long integers. Manifest verifiers MUST be able to handle number values up to 20 octets. Conforming manifest issuers MUST NOT use number values longer than 20 octets.

thisUpdate:

This field contains the time when the manifest was created. This field has the same format constraints as specified in [\[RFC5280\]](#) for the CRL field of the same name.

nextUpdate:

This field contains the time at which the next scheduled manifest will be issued. The value of nextUpdate MUST be later than the value of thisUpdate. The specification of the GeneralizedTime value is the same as required for the thisUpdate field.

If the authority alters any of the items that it has published in the repository publication point, then the authority MUST issue a new manifest before the nextUpdate time. Each manifest encompasses a CRL, and the nextUpdate field of the manifest MUST match that of the CRL's nextUpdate field, as the manifest will be re-issued when a new CRL is published. When a new manifest is issued before the time specified in nextUpdate of the current manifest, the CA MUST also issue a new CRL that revokes the EE certificate corresponding to the old manifest.

fileHashAlg:

This field contains the OID of the hash algorithm used to hash the files that the authority has placed into the repository. The hash algorithm used MUST conform to the RPKI Algorithms and Key Size Profile specification [\[RFC6485\]](#).

fileList:

This field is a sequence of FileAndHash objects. There is one FileAndHash entry for each currently valid signed object that has been published by the authority (at this publication point). Each FileAndHash is an ordered pair consisting of the name of the file in the repository publication point (directory) that contains the object in question and a hash of the file's contents.

4.2.2. Names in FileAndHash objects

Names that appear in the `fileList` MUST consist of one or more characters chosen from the set `a-z`, `A-Z`, `0-9`, `-` (HYPHEN), or `_` (UNDERSCORE), followed by a single `.` (DOT), followed by a three-letter extension. The extension MUST be one of those enumerated in the "RPKI Repository Naming Scheme" registry maintained by IANA [[IANA-NAMING](#)].

As an example, `'vixxBTS_TVXQ-2pmGOT7.cer'` is a valid filename.

4.3. Content-Type Attribute

The mandatory content-type attribute MUST have its `attrValues` field set to the same OID as `eContentType`. This OID is `id-ct-rpkiManifest` and has the numerical value of `1.2.840.113549.1.9.16.1.26`.

4.4. Manifest Validation

To determine whether a manifest is valid, the RP MUST perform the following checks in addition to those specified in [[RFC6488](#)]:

1. The `eContentType` in the `EncapsulatedContentInfo` is `id-ad-rpkiManifest` (OID `1.2.840.113549.1.9.16.1.26`).
2. The version of the `rpkiManifest` is `0`.
3. In the `rpkiManifest`, `thisUpdate` precedes `nextUpdate`.
4. The `thisUpdate` and `nextUpdate` fields in the manifest MUST match the corresponding fields in the CRL associated with the manifest.

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

5. Manifest Generation

5.1. Manifest Generation Procedure

For a CA publication point in the RPKI repository system, a CA MUST perform the following steps to generate a manifest:

1. Generate a new key pair for use in a "one-time-use" EE certificate.
2. Issue an EE certificate for this key pair. The CA MUST revoke the EE certificate used for the manifest being replaced.

This EE certificate MUST have an SIA extension access description field with an accessMethod OID value of id-ad-signedobject, where the associated accessLocation references the publication point of the manifest as an object URL.(RPs are required to verify this.)

This EE certificate MUST describe its Internet Number Resources (INRs) using the "inherit" attribute, rather than explicit description of a resource set (see [[RFC3779](#)]).(RPs are required to verify this.)

The validity interval of the EE certificate MUST exactly match the thisUpdate and nextUpdate times of the manifest.

Note: An RP MUST verify all of these syntactic constraints.

3. The EE certificate MUST NOT be published in the authority's repository publication point.
4. Construct the manifest content.

The manifest content is described in [Section 4.2.1](#). The manifest's fileList includes the file name and hash pair for each object issued by this CA that has been published at this repository publication point (directory). The collection of objects to be included in the manifest includes all certificates issued by this CA that are published at the CA's repository publication point, the most recent CRL issued by the CA, and all objects verified by EE certificates that were issued by this CA that are published at this repository publication point. (Sections [6.1-5](#) describes the checks that an RP MUST perform in support of the manifest content noted here.)

Note that the manifest does not include a self reference (i.e., its own file name and hash), since it would be impossible to compute the hash of the manifest itself prior to it being signed.

5. Encapsulate the manifest content using the CMS SignedData content type (as specified [Section 4](#)), sign the manifest using the private key corresponding to the subject key contained in the EE certificate, and publish the manifest in the repository system publication point that is described by the manifest. (RPs are required to verify the CMS signature.)
6. Because the key pair is to be used only once, the private key associated with this key pair MUST now be destroyed.

5.2. Considerations for Manifest Generation

A new manifest **MUST** be issued and published before the nextUpdate time.

An authority **MUST** issue a new manifest in conjunction with the finalization of changes made to objects in the publication point. An authority **MAY** perform a number of object operations on a publication repository within the scope of a repository change before issuing a single manifest that covers all the operations within the scope of this change. Repository operators **MUST** implement some form of repository update procedure that mitigates, to the extent possible, the risk that RPs that are performing retrieval operations on the repository are exposed to inconsistent, transient, intermediate states during updates to the repository publication point (directory) and the associated manifest.

Since the manifest object URL is included in the SIA of issued certificates, a new manifest **MUST NOT** invalidate the manifest object URL of previously issued certificates. This implies that the manifest's publication name in the repository, in the form of an object URL, is unchanged across manifest generation cycles.

When a CA entity is performing a key rollover, the entity **MAY** choose to have two CA instances simultaneously publishing into the same repository publication point. In this case, there will be one manifest associated with each active CA instance that is publishing into the common repository publication point (directory).

6. Relying Party Processing of Manifests

Each RP **MUST** use the current manifest of a CA to control addition of listed files to the set of signed objects the RP employs for validating basic RPKI objects: certificates, ROAs, and CRLs. Any files not listed on the manifest **MUST NOT** be used for validation of these objects. However, files not listed on a manifest **MAY** be employed to validate other signed objects, if the profile of the object type explicitly states that such behavior is allowed (or required). Note that relying on files not listed in a manifest may allow an attacker to effect substitution attacks against such objects.

As noted earlier, manifests are designed to allow an RP to detect manipulation of repository data, errors by a CA or repository manager, and/or active attacks on the communication channel between an RP and a repository. Unless all of the files enumerated in a manifest can be obtained by an RP during a fetch operation, the fetch is considered to have failed and the RP **MUST** retry the fetch later.

[RFC6480] suggests (but does not mandate) that the RPKI model employ fetches that are incremental, e.g., an RP transfers files from a publication point only if they are new/changed since the previous, successful, fetch represented in the RP's local cache. This document avoids language that relies on details of the underlying file transfer mechanism employed by an RP and a publication point to effect this operation. Thus the term "fetch" refers to an operation that attempts to acquire the full set of files at a publication point, consistent with the id-ad-rpkiManifest URI extracted from a CA certificate's SIA (see below).

If a fetch fails, it is assumed that a subsequent fetch will resolve problems encountered during the fetch. Until such time as a successful fetch is executed, an RP SHOULD use cached data from a previous, successful fetch. This response is intended to prevent an RP from misinterpreting data associated with a publication point, and thus possibly treating invalid routes as valid, or vice versa.

The processing described below is designed to cause all RPs with access to the same local cache and RPKI repository data to acquire the same set of validated repository files. It does not ensure that the RPs will achieve the same results with regard to validation of RPKI data, since that depends on how each RP resolves any conflicts that may arise in processing the retrieved files. Moreover, in operation, different RPs will access repositories at different times, and some RPs may experience local cache failures, so there is no guarantee that all RPs will achieve the same results with regard to acquisition or validation of RPKI data.

Note also that there is a "chicken and egg" relationship between the manifest and the CRL for a given CA instance. If the EE certificate for the current manifest is revoked, i.e., it appears in the current CRL, then the CA or publication point manager has made a serious error. In this case the fetch has failed; proceed to [Section 6.6](#). Similarly, if the CRL is not listed on a valid, current manifest, acquired during a fetch, the fetch has failed; proceed to [Section 6.6](#), because the CRL is considered missing.

[6.1](#). Manifest Processing Overview

For a given publication point, an RP MUST perform a series of tests to determine which signed object files at the publication point are acceptable. The tests described below ([Section 6.2](#) to [Section 6.5](#)) are to be performed using the manifest identified by the id-ad-rpkiManifest URI extracted from a CA certificate's SIA. All of the files referenced by the manifest MUST be located at the publication point specified by the id-ad-caRepository URI from the (same) CA certificate's SIA. The manifest and the files it references MUST

reside at the same publication point. If an RP encounters any files that appear on a manifest but do not reside at the same publication point as the manifest the RP MUST treat the fetch as failed, and a warning MUST be issued (see [Section 6.6](#) below).

Note that, during CA key rollover [[RFC6489](#)], signed objects for two or more different CA instances will appear at the same publication point. Manifest processing is to be performed separately for each CA instance, guided by the SIA id-ad-rpkiManifest URI in each CA certificate.

[6.2.](#) Acquiring a Manifest for a CA

The RP MUST fetch the manifest identified by the SIA id-ad-rpkiManifest URI in the CA certificate. If an RP cannot retrieve a manifest using this URI, or if the manifest is not valid ([Section 4.4](#)), an RP MUST treat this as a failed fetch and, proceed to [Section 6.6](#); otherwise proceed to [Section 6.3](#).

[6.3.](#) Detecting Stale and or Prematurely-issued Manifests

The RP MUST check that the current time (translated to UTC) is between thisUpdate and nextUpdate. If the current time lies within this interval, proceed to [Section 6.4](#). If the current time is earlier than thisUpdate, the CA has made an error; this is a failed fetch and the RP MUST proceed to [Section 6.6](#). If the current time is later than nextUpdate, then the manifest is stale; this is a failed fetch and RP MUST proceed to [Section 6.6](#); otherwise proceed to [Section 6.4](#).

[6.4.](#) Acquiring Files Referenced by a Manifest

The RP MUST acquire all of the files enumerated in the manifest (fileList) from the publication point. If there are files listed in the manifest that cannot be retrieved from the publication point, the fetch has failed and the RP MUST proceed to [Section 6.6](#); otherwise, proceed to [Section 6.5](#).

[6.5.](#) Matching File Names and Hashes

The RP MUST verify that the hash value of each file listed in the manifest matches the value obtained by hashing the file acquired from the publication point. If the computed hash value of a file listed on the manifest does not match the hash value contained in the manifest, then the fetch has failed and the RP MUST proceed to [Section 6.6](#); otherwise proceed to [Section 6.6](#).

6.6. Failed Fetches

If a fetch fails for any of the reasons cited in 6.2-6.5, the RP MUST issue a warning indicating the reason(s) for termination of processing with regard to this CA instance. It is RECOMMENDED that a human operator be notified of this warning.

Termination of processing means that the RP SHOULD continue to use cached versions of the objects associated with this CA instance, until such time as they become stale or they can be replaced by objects from a successful fetch. This implies that the RP MUST not try to acquire and validate subordinate signed objects, e.g., subordinate CA certificates, until the next interval when the RP is scheduled to fetch and process data for this CA instance.

7. Publication Repositories

The RPKI publication system model requires that every publication point be associated with one or more CAs, and be non-empty. Upon creation of the publication point associated with a CA, the CA MUST create and publish a manifest as well as a CRL. A CA's manifest will always contain at least one entry, i.e., a CRL issued by the CA [[RFC6481](#)], corresponding to the scope of this manifest.

Every published signed object in the RPKI [[RFC6488](#)] is published in the repository publication point of the CA that issued the EE certificate, and is listed in the manifest associated with that CA certificate.

8. Security Considerations

Manifests provide an additional level of protection for RPKI RPs. Manifests can assist an RP to determine if a repository object has been deleted, occluded, or otherwise removed from view, or if a publication of a newer version of an object has been suppressed (and an older version of the object has been substituted).

Manifests cannot repair the effects of such forms of corruption of repository retrieval operations. However, a manifest enables an RP to determine if a locally maintained copy of a repository is a complete and up-to-date copy, even when the repository retrieval operation is conducted over an insecure channel. In cases where the manifest and the retrieved repository contents differ, the manifest can assist in determining which repository objects form the difference set in terms of missing, extraneous, or superseded objects.

The signing structure of a manifest and the use of the nextUpdate value allows an RP to determine if the manifest itself is the subject of attempted alteration. The requirement for every repository publication point to contain at least one manifest allows an RP to determine if the manifest itself has been occluded from view. Such attacks against the manifest are detectable within the time frame of the regular schedule of manifest updates. Forms of replay attack within finer-grained time frames are not necessarily detectable by the manifest structure.

9. IANA Considerations

As [RFC6488] created and populated the registries "RPKI Signed Object" and three-letter filename extensions for "RPKI Repository Name Schemes," no new action is requested of the IANA.

10. Acknowledgements

The authors would like to acknowledge the contributions from George Michelson and Randy Bush in the preparation of the manifest specification. Additionally, the authors would like to thank Mark Reynolds and Christopher Small for assistance in clarifying manifest validation and RP behavior. The authors also wish to thank Job Snijders, Oleg Muravskiy, and Sean Turner for their helpful review of this document.

11. References

11.1. Normative References

[IANA-NAMING]

"RPKI Repository Name Schemes",
<<https://www.iana.org/assignments/rpki/rpki.xhtml#name-schemes>>.

[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>>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 5280](#), DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.

- [RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", [RFC 6481](#), DOI 10.17487/RFC6481, February 2012, <<https://www.rfc-editor.org/info/rfc6481>>.
- [RFC6482] Lepinski, M., Kent, S., and D. Kong, "A Profile for Route Origin Authorizations (ROAs)", [RFC 6482](#), DOI 10.17487/RFC6482, February 2012, <<https://www.rfc-editor.org/info/rfc6482>>.
- [RFC6485] Huston, G., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)", [RFC 6485](#), DOI 10.17487/RFC6485, February 2012, <<https://www.rfc-editor.org/info/rfc6485>>.
- [RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", [RFC 6487](#), DOI 10.17487/RFC6487, February 2012, <<https://www.rfc-editor.org/info/rfc6487>>.
- [RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", [RFC 6488](#), DOI 10.17487/RFC6488, February 2012, <<https://www.rfc-editor.org/info/rfc6488>>.
- [RFC6493] Bush, R., "The Resource Public Key Infrastructure (RPKI) Ghostbusters Record", [RFC 6493](#), DOI 10.17487/RFC6493, February 2012, <<https://www.rfc-editor.org/info/rfc6493>>.
- [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>>.
- [RFC8209] Reynolds, M., Turner, S., and S. Kent, "A Profile for BGPsec Router Certificates, Certificate Revocation Lists, and Certification Requests", [RFC 8209](#), DOI 10.17487/RFC8209, September 2017, <<https://www.rfc-editor.org/info/rfc8209>>.
- [RFC8488] Muravskiy, O. and T. Bruijnzeels, "RIPE NCC's Implementation of Resource Public Key Infrastructure (RPKI) Certificate Tree Validation", [RFC 8488](#), DOI 10.17487/RFC8488, December 2018, <<https://www.rfc-editor.org/info/rfc8488>>.
- [X.690] "X.690", <<https://www.itu.int/rec/T-REC-X.690-199511-S!Cor1>>.

11.2. Informative References

- [RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", [RFC 3370](#), DOI 10.17487/RFC3370, August 2002, <<https://www.rfc-editor.org/info/rfc3370>>.
- [RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", [RFC 3779](#), DOI 10.17487/RFC3779, July 2004, <<https://www.rfc-editor.org/info/rfc3779>>.
- [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>>.
- [RFC6489] Huston, G., Michaelson, G., and S. Kent, "Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)", [BCP 174](#), [RFC 6489](#), DOI 10.17487/RFC6489, February 2012, <<https://www.rfc-editor.org/info/rfc6489>>.

Appendix A. ASN.1 Module

```
RPKIManifest { iso(1) member-body(2) us(840) rsadsi(113549)
               pkcs(1) pkcs9(9) smime(16) mod(0) 60 }
```



```
DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

-- IMPORTS NOTHING --

-- Manifest Content Type: OID

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

id-ct OBJECT IDENTIFIER ::= { id-smime 1 }

id-ct-rpkiManifest OBJECT IDENTIFIER ::= { id-ct 26 }

-- Manifest Content Type: eContent

Manifest ::= SEQUENCE {
  version          [0] INTEGER DEFAULT 0,
  manifestNumber   INTEGER (0..MAX),
  thisUpdate       GeneralizedTime,
  nextUpdate       GeneralizedTime,
  fileHashAlg      OBJECT IDENTIFIER,
  fileList         SEQUENCE SIZE (0..MAX) OF FileAndHash
}

FileAndHash ::= SEQUENCE {
  file  IA5String,
  hash  BIT STRING
}

END
```

Authors' Addresses

Rob Austein
Arrcus, Inc.

Email: sra@hactrn.net

Geoff Huston
APNIC
6 Cordelia St
South Brisbane QLD 4101
Australia

Email: gih@apnic.net

Stephen Kent
Independent

Email: kent@alum.mit.edu

Matt Lepinski
New College Florida
5800 Bay Shore Rd.
Sarasota, FL 34243
USA

Email: mlepinski@ncf.edu

