Internet Engineering Task Force (IETF)
Request for Comments: 6486
Category: Standards Track
ISSN: 2070-1721

R. Austein
ISC
G. Huston
APNIC
S. Kent
M. Lepinski
BBN

February 2012

Manifests for the Resource Public Key Infrastructure (RPKI)

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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6486.

Table of Contents

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substitute "stale" versions of signed objects (i.e., objects that were valid and have not expired, but have since been superseded) or attacks that remove an object that should be present in the repository.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. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Manifest Scope

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

* the set of (non-expired, non-revoked) certificates issued and published by this CA,

* the most recent CRL issued by this CA, and

* 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 an entity 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 MAY choose to sign only one manifest with each generated private key, and 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.

Alternatively, the CA MAY elect to use the same private key to sign a sequence of manifests. Because only a single manifest (issued under a single CA instance) is current at any point in time, the associated EE certificate is used to verify only a single object at a time. As long as the sequence of objects verified by this EE certificate are published using the same file name, then this sequential, multiple use of the EE certificate is also valid. This form of use of an EE certificate is termed a "sequential-use" EE certificate.

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.

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.

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.

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. If a manifest encompasses a CRL, 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. If a "one-time-use" EE certificate is used to verify the manifest, then 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 includes 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.

4.3. 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.

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. If no key pair exists, or if using a "one-time-use" EE certificate with a new key pair, generate a key pair.

2. If using a "one-time-use" EE certificate, or if a key pair was generated in step 1, or if using a "sequential-use" EE certificate that will expire before the intended nextUpdate time of this manifest, issue an EE certificate for this key pair.

   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.

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

   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.

   In the case of a "sequential-use" EE certificate, the validity times of the EE certificate MUST encompass the time interval from thisUpdate to nextUpdate.

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.

   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.

6. In the case of a key pair that is to be used only once, in conjunction with a "one-time-use" EE certificate, 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 on or 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 SHOULD 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 Use of Manifests

The goal of an RP is to determine which signed objects to use for validating assertions about INRs and their use (e.g., which ROAs to use in the construction of route filters). Ultimately, this selection is a matter of local policy. However, in the following sections, we describe a sequence of tests that the RP SHOULD perform to determine the manifest state of the given publication point. We then discuss the risks associated with using signed objects in the publication point, given the manifest state; we also provide suitable warning text that SHOULD be placed in a user-accessible log file. It is the responsibility of the RP to weigh these risks against the risk of routing failure that could occur if valid data is rejected, and to implement a suitable local policy. Note that if a certificate is deemed unfit for use due to local policy, then any signed object that is validated using this certificate also SHOULD be deemed unfit for use (regardless of the status of the manifest at its own publication point).

6.1. Tests for Determining Manifest State

For a given publication point, the RP SHOULD perform the following tests to determine the manifest state of the publication point:

1. For each CA using this publication point, select the CA's current manifest (the "current" manifest is the manifest issued by this CA having the highest manifestNumber among all valid manifests, and where manifest validity is defined in Section 4.4).

   If the publication point does not contain a valid manifest, see Section 6.2. Lacking a valid manifest, the following tests cannot be performed.

2. To verify completeness, an RP MAY check that every file at each publication point appears in one and only one current manifest, and that every file listed in a current manifest is published at the same publication point as the manifest.

   If there exist files at the publication point that do not appear on any manifest, or files listed in a manifest that do not appear at the publication point, then see Section 6.5, but still continue with the following test.

3. Check that the current time (translated to UTC) is between thisUpdate and nextUpdate.

   If the current time does not lie within this interval, then see Section 6.4, but still continue with the following tests.

4. Verify that the listed hash value of every file listed in each manifest matches the value obtained by hashing the file at 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 see Section 6.6.

5. An RP MAY check that the contents of each current manifest conforms to the manifest's scope constraints, as specified in Section 2.

6. If a current manifest contains entries for objects that are not within the scope of the manifest, then the out-of-scope entries

SHOULD be disregarded in the context of this manifest. If there is no other current manifest that describes these objects within that other manifest's scope, then see Section 6.2.

For each signed object, if all of the following conditions hold:

* the manifest for its publication and the associated publication point pass all of the above checks;

* the signed object is valid; and

* the manifests for every certificate on the certification path used to validate the signed object and the associated publication points pass all of the above checks;

then the RP can conclude that no attack against the repository system has compromised the given signed object, and the signed object MUST be treated as valid (relative to manifest checking).

6.2. Missing Manifests

The absence of a current manifest at a publication point could occur due to an error by the publisher or due to (malicious or accidental) deletion or corruption of all valid manifests.

When no valid manifest is available, there is no protection against attacks that delete signed objects or replay old versions of signed objects. All signed objects at the publication point, and all descendant objects that are validated using a certificate at this publication point, SHOULD be viewed as suspect, but MAY be used by the RP, as per local policy.

The primary risk in using signed objects at this publication point is that a superseded (but not stale) CRL would cause an RP to improperly accept a revoked certificate as valid (and thus rely upon signed objects that are validated using that certificate). This risk is somewhat mitigated if the CRL for this publication point has a short time between thisUpdate and nextUpdate (and the current time is within this interval). The risk in discarding signed objects at this publication point is that an RP may incorrectly discard a large number of valid objects. This gives significant power to an adversary that is able to delete a manifest at the publication point.

Regardless of whether signed objects from this publication are deemed fit for use by an RP, this situation SHOULD result in a warning to the effect that: "No manifest is available for <pub point name>, and thus there may have been undetected deletions or replay substitutions from the publication point."

In the case where an RP has access to a local cache of previously issued manifests that are valid, the RP MAY use the most recently previously issued valid manifests for this RPKI repository publication collection for each entity that publishes at this publication point.

6.3. Invalid Manifests

The presence of an invalid manifest at a publication point could occur due to an error by the publisher or due to (malicious or accidental) corruption of a valid manifest. An invalid manifest MUST never be used, even if the manifestNumber of the invalid manifest is greater than that of other (valid) manifests.

There are no risks associated with using signed objects at a publication point containing an invalid manifest, provided that valid manifests that collectively cover all the signed objects are also present.

If an invalid manifest is present at a publication point that also contains one or more valid manifests, this situation SHOULD result in a warning to the effect that: "An invalid manifest was found at <pub point name>, this indicates an attack against the publication point or an error by the publisher. Processing for this publication point will continue using the most recent valid manifest(s)."

In the case where the RP has access to a local cache of previously issued (valid) manifests, an RP MAY make use of that locally cached data. Specifically, the RP MAY use the locally cached, most recent, previously issued, valid manifest issued by the entity that (appears to have) issued the invalid manifest.

6.4. Stale Manifests

A manifest is considered stale if the current time is after the nextUpdate time for the manifest. This could be due to publisher failure to promptly publish a new manifest, or due to (malicious or accidental) corruption or suppression of a more recent manifest.

All signed objects at the publication point issued by the entity that has published the stale manifest, and all descendant signed objects that are validated using a certificate issued by the entity that has published the stale manifest at this publication point, SHOULD be viewed as somewhat suspect, but MAY be used by the RP as per local policy.

The primary risk in using such signed objects is that a newer manifest exists that, if present, would indicate that certain objects have been removed or replaced. (For example, the new manifest might show the existence of a newer CRL and the removal of one or more revoked certificates). Thus, the use of objects from a stale manifest may cause an RP to incorrectly treat invalid objects as valid. The risk is that the CRL covered by the stale manifest has been superseded, and thus an RP will improperly treat a revoked certificate as valid. This risk is somewhat mitigated if the time between the nextUpdate field of the manifest and the current time is short. The risk in discarding signed objects at this publication point is that the RP may incorrectly discard a large number of valid objects. This gives significant power to an adversary that is able to prevent the publication of a new manifest at a given publication point.

Regardless of whether signed objects from this publication are deemed fit for use by an RP, this situation SHOULD result in a warning to the effect that: "A manifest found at <pub point name> is no longer current. It is possible that undetected deletions have occurred at this publication point."

Note that there is also the potential for the current time to be before the thisUpdate time for the manifest. This case could be due to publisher error or a local clock error; in such a case, this situation SHOULD result in a warning to the effect that: "A manifest found at <pub point name> has an incorrect thisUpdate field. This could be due to publisher error, or a local clock error, and processing for this publication point will continue using this otherwise valid manifest."

6.5. Mismatch between Manifest and Publication Point

If there exist valid signed objects that do not appear in any manifest, then, provided the manifest is not stale (see Section 6.4), it is likely that their omission is an error by the publisher. It is also possible that this state could be the result of a (malicious or accidental) replacement of a current manifest with an older, but still valid, manifest. However, regarding the appropriate interpretation of such objects, it remains the case that if the objects were intended to be invalid, then they should have been revoked using whatever revocation mechanism is appropriate for the signed object in question. Therefore, there is little risk in using such signed objects. If the publication point contains a stale manifest, then there is a greater risk that the objects in question were revoked, along with a missing Certificate Revocation List (CRL), the absence of which is undetectable since the manifest is stale. In any case, the use of signed objects not present on a manifest, or descendant objects that are validated using such signed objects, is a matter of local policy.

Regardless of whether objects not appearing on a manifest are deemed fit for use by the RP, this situation SHOULD result in a warning to the effect that: "The following files are present in the repository at <pub point name>, but are not listed on any manifest <file list> for <pub point name>."

If there exists files listed on the manifest that do not appear in the repository, then these objects are likely to have been improperly (via malice or accident) deleted from the repository. A primary purpose of manifests is to detect such deletions. Therefore, in such a case, this situation SHOULD result in a warning to the effect that: "The following files that should have been present in the repository at <pub point name> are missing <file list>. This indicates an attack against this publication point, or the repository, or an error by the publisher."

6.6. Hash Values Not Matching Manifests

A file appearing on a manifest with an incorrect hash value could occur because of publisher error, but it also may indicate that an attack has occurred.

If an object appeared on a previous valid manifest with a correct hash value, and it now appears with an invalid hash value, then it is likely that the object has been superseded by a new (unavailable) version of the object. If the object is used, there is a risk that the RP will be treating a stale object as valid. This risk is more significant if the object in question is a CRL. If the object can be validated using the RPKI, the use of these objects is a matter of local policy.

If an object appears on a manifest with an invalid hash and has never previously appeared on a manifest, then it is unclear whether the available version of the object is more or less recent than the version indicated by the manifest. If the manifest is stale (see Section 6.4), then it becomes more likely that the available version is more recent than the version indicated on the manifest, but this is never certain. Whether to use such objects is a matter of local policy. However, in general, it is better to use a possibly outdated version of the object than to discard the object completely.

While it is a matter of local policy, in the case of CRLs, an RP SHOULD endeavor to use the most recently issued valid CRL, even where the hash value in the manifest matches an older CRL or does not match any available CRL for a CA instance. The thisUpdate field of the CRL can be used to establish the most recent CRL in the case where an RP has more than one valid CRL for a CA instance.

Regardless of whether objects with incorrect hashes are deemed fit for use by the RP, this situation SHOULD result in a warning to the effect that: "The following files at the repository <pub point name> appear on a manifest with incorrect hash values <file list>. It is possible that these objects have been superseded by a more recent version. It is very likely that this problem is due to an attack on the publication point, although it also could be due to a publisher error."

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, namely, the CRL issued by the CA upon repository creation [RFC6481].

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

This document registers the following in the "RPKI Signed Object" registry created by [RFC6488]:

Name: Manifest
OID: 1.2.840.113549.1.9.16.1.26
Reference: [RFC6486] (this document)

This document registers the following three-letter filename extension for "RPKI Repository Name Schemes" registry created by [RFC6481]:

Filename extension: mft
RPKI Object: Manifest
Reference: [RFC6481]

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 Sean Turner for his helpful review of this document.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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, May 2008.

[RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", RFC 6481, February 2012.

[RFC6485] Huston, G., "A Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)", RFC 6485, February 2012.

[RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", RFC 6487, February 2012.

[RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", RFC 6488, February 2012.

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

11.2. Informative References

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC3779] Lynn, C., Kent, S., and K. Seo, "X.509 Extensions for IP Addresses and AS Identifiers", RFC 3779, June 2004.

[RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.

[RFC6489] Huston, G., Michaelson, G., and S. Kent, "Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)", BCP 174, RFC 6489, February 2012.

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
Internet Systems Consortium

EMail: sra@hactrn.net

Geoff Huston
APNIC
6 Cordelia St
South Brisbane, QLD 4101
Australia

EMail: gih@apnic.net
URI: http://www.apnic.net

Stephen Kent
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
USA

EMail: kent@bbn.com

Matt Lepinski
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
USA

EMail: mlepinski@bbn.com