Secure Inter-Domain Routing
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2009

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

October 25, 2008

Manifests for the Resource Public Key Infrastructure
draft-ietf-sidr-rpki-manifests-04.txt

Status of this Memo

Abstract

This document defines a "manifest" for use in the Resource Public Key Infrastructure. A manifest is a signed object that contains a listing of all the signed objects in the repository publication point associated with an authority responsible for publishing in the repository. For each certificate, or other forms 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 expose potential attacks against relying parties of the Resource Public Key Infrastructure, such as a man-in-the middle attack of withholding repository data from relying party access, or replaying stale repository data to a relying party's access request.

Table of Contents

1. Introduction

The Resource Public Key Infrastructure (RPKI) [ID.sidr-arch] makes use of a distributed repository system [ID.sidr-repos-struct] to make available a variety of objects needed by relying parties (RPs) such as Internet service providers (ISPs). Because all of the objects stored in the repository system are digitally signed by the entities that created them, attacks that modify these 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substitute "stale" versions of signed objects (i.e., objects that were valid and have not expired, but have since been superseded) or attacks that remove an object that should be present in the repository.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 will make use of a new signed object called a "manifest".

A manifest is a signed object that contains a listing of all the signed objects in the repository publication point associated with an authority responsible for publishing in the repository. For each certificate, Certificate Revocation List (CRL), or other signed object, such as a Route Origination Authorization (ROA), issued by an authority, the authority's manifest contains both the name of the file containing the object, and a hash of the file content. Manifests allow a RP to obtain sufficient information to detect whether the retrieval of objects from an RPKI repository has been compromised by unauthorized object removal, or by the substitution of "stale" versions of objects. Manifests are designed to be used both for Certification Authority (CA) publication points in repositories, that contain subordinate certificates, CRLs and other signed objects, and End Entity (EE) publication points in repositories that contain signed objects.

Manifests are modeled on CRLs, as the issues involved in detecting stale manifests, and detection of potential attacks using manifest replays, etc are similar to those for CRLs. The syntax of the manifest payload differs from CRLs, since RPKI repositories can

contain objects not covered by CRLs, such as digitally signed objects, such as ROAs.

1.1. Terminology

It is assumed that the reader is familiar with the terms and concepts described in "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" [RFC5280]and "X.509 Extensions for IP Addresses and AS Identifiers" [RFC3779].

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 RFC 2119.

2. Manifest Scope

In the case of a CA's manifest for its associated publication repository, the manifest contains the current published certificates issued by this CA, the most recent CRL issued by this CA, and all objects that are signed using a "single-use" EE certificate (i.e., the SIA extension of the EE certificate has an accessMethod OID of id-ad-signedObject), where the EE certificate was issued by this CA.

In the case where multiple CAs share a common publication point, as may be the case when an entity performs a staged key-rollover operation, the repository publication will contain multiple manifests. In such a scenario, each manifest describes only the collection of published products of its associated CA.

In the case of a "multi-use" EE certificate, where an EE has a defined publication repository (i.e., the SIA extension of the EE certificate has an accessMethod OID of id-ad-signedObjectRepository), the EE's manifest contains all published objects that have been signed by the EE's key, and the accessMethod id-as-rpkiManifest points to the publication point of the EE's manifest.

3. Manifest Signing

A CA's manifest is verified using an EE certificate that is

designated in [ID.sidr-res-certs] as a "single-use" EE certificate. The SIA field of the "single-use" EE certificate contains the access method OID of id-ad-signedObject.

The CA MAY chose to sign only one manifest with the private key of the EE certificate, and generate a new EE certificate for each new version of the manifest. This form of use of a "single-use" EE certificate is termed a "one-time-use" EE certificate.

Alternatively the CA MAY chose to use the same EE certificate's private key to sign a sequence of manifests. Because only a single manifest is current at any point in time, the EE certificate is used only to verify a single object at a time. As long as the sequence of objects signed by this EE certificate's private key are published as the same named object, so that the SIA accessMethod id-ad-signedObject value can refer to the current instance of the sequence of such objects, then this sequential multiple use of this "single-use" EE certificate is also valid. This form of use of a "single-use" EE certificate is termed a "sequential-use" EE certificate.

A "multi-use" EE's manifest of it's publication repository is signed with the private key of the EE certificate.

4. Manifest Syntax

A manifest is a Cryptographic Message Syntax (CMS) [RFC3852] signed-data object. The general format of a CMS object is:

```
ContentInfo ::= SEQUENCE {
     contentType ContentType,
     content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER
```

A Manifest is a signed-data object. The ContentType used is the signed-data type of id-data, namely the id-signedData OID, 1.2.840.113549.1.7.2. [RFC3852]

4.1. Signed-Data Content Type

According to the CMS specification, signed-data content types shall

have the ASN.1 type SignedData:

```
   SignedData ::= SEQUENCE {
        version CMSVersion,
        digestAlgorithms DigestAlgorithmIdentifiers,
        encapContentInfo EncapsulatedContentInfo,
        certificates [0] IMPLICIT CertificateSet OPTIONAL,
        crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
        signerInfos SignerInfos }

     DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

     SignerInfos ::= SET OF SignerInfo
```

4.1.1. version

The version is the syntax version number. It MUST be 3, corresponding to the signerInfo structure having version number 3.

4.1.2. digestAlgorithms

The digestAlgorithms set MUST include only SHA-256, the OID for which is 2.16.840.1.101.3.4.2.1 [RFC4055]. It MUST NOT contain any other algorithms.

4.1.3. encapContentInfo

encapContentInfo is the signed content, consisting of a content type identifier and the content itself.

```
     EncapsulatedContentInfo ::= SEQUENCE {
       eContentType ContentType,
       eContent [0] EXPLICIT OCTET STRING OPTIONAL }

     ContentType ::= OBJECT IDENTIFIER
```

4.1.3.1. 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 }
```

4.1.3.2. eContent

The content of a Manifest is defined as follows:

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

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

4.1.3.2.1. 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 valid until the time specified in nextUpdate or until a manifest is issued with a greater manifest number, whichever comes first. The revoked EE certificate

for the previous manifest's signature will be removed from the CRL when it expires.

If a "one-time-use" EE certificate is employed to verify a manifest, the EE certificate MUST have an 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 8.4.

4.1.3.2.1.1. version

The version number of the rpkiManifest MUST be 0.

4.1.3.2.1.2. manifestNumber

The manifestNumber field is a sequence number that is incremented each time a new manifest is issued for a given publication point. This field is used to allow a RP to detect gaps in a sequence of published manifest.

4.1.3.2.1.3. thisUpdate

The thisUpdate field contains the time when the manifest was created.

4.1.3.2.1.4. nextUpdate

The nextUpdate 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. If the authority alters any of the items 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, as the manifest will be reissued when a new CRL is published. When a "one-time-use" EE certificate is being 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.

4.1.3.2.1.5. fileHashAlg

The fileHashAlg field contains the OID of the hash algorithm used to hash the files that the authority has placed into the repository. The mandatory to implement hash algorithm is SHA-256 and its OID is

2.16.840.1.101.3.4.2.1. [RFC4055].

4.1.3.2.1.6. fileList

The fileList field contains a sequence of FileAndHash pairs, one for each currently valid signed object that has been issued by the authority. Each FileAndHash pair contains the name of the file in the repository that contains the object in question, and a hash of the file's contents.

4.1.4. certificates

The certificates field MUST be included, and MUST contain the RPKI EE certificate needed to validate this manifest in the context of the RPKI.

4.1.5. crls

This field MUST be omitted.

4.1.6. signerInfos

Signer Infos is defined as a SignerInfo, which is defined under CMS as:

```
SignerInfo ::= SEQUENCE {
  version CMSVersion,
  sid SignerIdentifier,
  digestAlgorithm DigestAlgorithmIdentifier,
  signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }
```

4.1.6.1. version

The version number MUST be 3, corresponding with the choice of SubjectKeyIdentifier for the sid.

4.1.6.2. sid

The sid is defined as:

```
SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier }
```

For a Manifest, the sid MUST be a SubjectKeyIdentifier.

4.1.6.3. digestAlgorithm

The digestAlgorithm MUST be SHA-256, the OID for which is 2.16.840.1.101.3.4.2.1. [RFC4055]

4.1.6.4. signedAttrs

The signedAttrs is defined as signedAttributes:

```
SignedAttributes ::= SET SIZE (1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
  attrType OBJECT IDENTIFIER,
  attrValues SET OF AttributeValue }

AttributeValue ::= ANY
```

The signedAttr element MUST be present and MUST include the content-type and message-digest attributes. The signer MAY also include the signing-time signed attribute, the binary-signing-time signed attribute, or both signing-time attributes. Other signed attributes that are deemed appropriate MAY also be included. The intent is to allow additional signed attributes to be included if a future need is identified. This does not cause an interoperability concern because unrecognized signed attributes are ignored by the relying party.

The signedAttr MUST include only a single instance of any particular attribute. Additionally, even though the syntax allows for a SET OF AttributeValue, in a Manifest the attrValues MUST consist of only a single AttributeValue.

4.1.6.4.1. Content-Type Attribute

The ContentType attribute MUST be present. The attrType OID for the ContentType attribute is 1.2.840.113549.1.9.3.

The attrValues for the ContentType attribute in a Manifest MUST be 1.2.840.113549.1.9.16.1.26, matching the eContentType in the EncapsulatedContentInfo.

4.1.6.4.2. Message-Digest Attribute

The MessageDigest Attribute MUST be present. The attrType OID for the MessageDigest Attribute is 1.2.840.113549.1.9.4.

The attrValues for the MessageDigest attribute contains the output of

the digest algorithm applied to the content being signed, as specified in Section 11.1 of [RFC3852].

4.1.6.4.3. SigningTime Attribute

The SigningTime attribute MAY be present. The presence of absence of the SigningTime attribute in no way affects the validation of the Manifest (as specified in Section Section 7).

The attrType OID for the SigningTime attribute is 1.2.840.113549.1.9.5.

The attrValues for the SigningTime attribute is defined as:

```
id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

SigningTime ::= Time

Time ::= CHOICE {
  utcTime UTCTime,
  generalizedTime GeneralizedTime }
```

The Time element specifies the time, based on the local system clock, at which the digital signature was applied to the content.

4.1.6.4.4. BinarySigningTime Attribute

The signer MAY include a BinarySigningTime attribute, specifying the time at which the digital signature was applied to the content. If both the BinarySigningTime and SigningTime attributes are present, the time that is represented by the binary-signing-time attribute MUST represent the same time value as the signing-time attribute. The presence or absence of the Binary-SigningTime attribute in no way affects the validation of the Manifest (as specified in Section Section 7).

The binary-signing-time attribute is defined in [RFC4049] as:

```
id-aa-binarySigningTime OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) aa(2) 46 }

BinarySigningTime ::= BinaryTime

BinaryTime ::= INTEGER (0..MAX)
```


4.1.6.5. signatureAlgorithm

The signatureAlgorithm MUST be RSA (rsaEncryption), the OID for which is 1.2.840.113549.1.1.1.

4.1.6.6. signature

The signature value is defined as:

```
SignatureValue ::= OCTET STRING
```

The signature characteristics are defined by the digest and signature algorithms.

4.1.6.7. unsignedAttrs

unsignedAttrs MUST be omitted.

4.2. ASN.1

The following is the ASN.1 specification of the CMS-signed Manifest.

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER

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

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

id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
```



```
                        us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }

SignedData ::= SEQUENCE {
  version CMSVersion,
  digestAlgorithms DigestAlgorithmIdentifiers,
  encapContentInfo EncapsulatedContentInfo,
  certificates [0] IMPLICIT CertificateSet OPTIONAL,
  crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
  signerInfos SignerInfos }

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo

SignerInfo ::= SEQUENCE {
  version CMSVersion,
  sid SignerIdentifier,
  digestAlgorithm DigestAlgorithmIdentifier,
  signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
```

```
  subjectKeyIdentifier [0] SubjectKeyIdentifier }

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
  attrType OBJECT IDENTIFIER,
  attrValues SET OF AttributeValue }

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING

id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }

ContentType ::= OBJECT IDENTIFIER

id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }

MessageDigest ::= OCTET STRING
```



```
id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }

SigningTime ::= Time

Time ::= CHOICE {
  utcTime UTCTime,
  generalizedTime GeneralizedTime }

id-aa-binarySigningTime OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) aa(2) 46 }

BinarySigningTime ::= BinaryTime

BinaryTime ::= INTEGER (0..MAX)
```

5. Manifest Generation

5.1. CA Manifest Generation

Each CA in the RPKI publishes the certificates and CRLs it issues at a publication point in the RPKI repository system. To create a manifest, each CA MUST perform the following steps:

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

2. If using a "one-time-use" EE certificate, or if a key pair was generated in step 1, issue a "single-use" EE certificate for this key pair to enable relying parties to verify the signature on the manifest.

    * This EE certificate has 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 IP number resources using the "inherit" attribute, rather than explicit description of a resource set.

    * In the case of a "one-time-use" EE certificate, the validity times of the EE certificate SHOULD exactly match the thisUpdate and nextUpdate times of the manifest, and MUST encompass the interval from thisUpdate to nextUpdate.

    * 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 SHOULD NOT be published in the authority's repository publication point.

4. Construct the manifest content. 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. The manifest content is described in Section 4.1.3.2.1. The manifest's fileList includes the file names and hash pair for all objects associated with this CA that have been published at the CA's repository publication point. The collection of objects to be included in the manifest includes all subordinate 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 "single-use" EE certificates that were issued by this CA that are published at the CA's repository publication point.

5. Encapsulate the Manifest content using the CMS SignedData content type (as specified in Section Section 4), sign the manifest using the private key corresponding to the EE certificate, and publish the manifest in 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 SHOULD now be destroyed.

5.2. End Entity Manifest Generation

EE repository publication points are only used in conjunction with "multi-use" EE Certificates. In this case the EE Certificate has two accessMethods specified in its SIA field. The id-ad-signedObjectRepository accessMethod has an associated accessLocation that points to the repository publication point of the objects signed by this EE certificate, as specified in [ID.sidr-res-certs]. The id-ad-rpkiManifest accessMethod has an associated access location that points to the manifest object as an object URL, that is associated with this repository publication point. This manifest describes all the signed objects that are to be found in that publication point that have been signed by this EE certificate, and the hash value of each product (excluding the manifest itself).

To create a manifest, each "multi-use" EE MUST perform the following steps:.

o Construct the Manifest content. 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. The manifest content is described in Section 4.1.3.2.1. The manifest's fileList includes the file names and hash pair for all objects verified using that EE certificate that have been published at the EE's repository publication point.

- Encapsulate the Manifest content using the CMS SignedData content type (as specified in Section Section 4), sign the manifest using the EE certificate, and publish the manifest in repository system publication point that is described by the manifest.

"Single Use" EE certificates (EE certificates with an SIA accessMethod OID of id-as-signedObject) do not have repository publication points. The object signed by the "Single Use" EE certificate is published in the repository publication point of the CA certificate that issued the EE certificate, and is listed in the corresponding manifest for this CA certificate.

5.3. Common Considerations for Manifest Generation

- A new manifest MUST be issued 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 synchronization function on the repository to ensure that relying parties who are performing retrieval operations on the repository are not exposed to intermediate states during changes to the repository and the associated manifest.

- Since the manifest object URL is included in the SIA of issued certificates then 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 one that is unchanged across manifest generation cycles.

- In the case of a CA publication point manifest, when the entity is performing a key rollover the entity MAY chose to have multiple CAs publishing at the same publication point. In this case there will be one manifest associated with each active CA that is publishing into the common repository publication point.

- In the case of an EE publication point the manifest lists all published objects verified using that EE certificate. Multiple EEs may share a common repository publication point, in which case there will be one manifest associated with each active EE that is publishing into the common repository publication point.

6. Processing Certificate Requests

When an EE certificate is intended for use in verifying multiple objects, the certificate request for the EE certificate MUST include in the SIA of the request an access method OID of id-ad-signedObjectRepository where the associated access location refers to the publication point for objects signed by this EE certificate, and MUST include in the SIA of the request an access method OID of id-ad-rpkiManifest, where the associated access location refers to the publication point of the manifest that is associated with published objects that are verified using this EE certificate [ID.sidr-res-certs].

When an EE certificate is used to sign a single object, the certificate request for the EE certificate MUST include in the SIA of the request an access method OID of id-ad-signedObject, where the associated access location refers to the publication point of the single object that is verified using this EE certificate. The certificate request MUST NOT include in the SIA of the request the access method OID of id-ad-rpkiManifest.

In accordance with the provisions of [ID.sidr-res-certs], all certificate issuance requests for a CA certificate SHOULD include in the SIA of the request the id-ad-caRepository access method, and also the id-ad-rpkiManifest access method that references the intended publication point of the manifest in the associated access location in the request.

The issuer MUST either honor these values in the issued certificate or reject the request entirely.

7. Manifest Validation

To determine whether a manifest is valid, the relying party must

perform the following checks:

1. Verify that the Manifest complies with this specification. In particular, verify the following:

    a. The contentType of the CMS object is SignedData (OID 1.2.840.113549.1.7.2)

    b. The version of the SignedData object is 3.

    c. The digestAlgorithm in the SignedData object is SHA-256 (OID 2.16.840.1.101.3.4.2.1).

    d. The certificates field in the SignedData object is present and contains an EE certificate whose Subject Key Identifier (SKI) matches the sid field of the SignerInfo object.

    e. The crls field in the SignedData object is omitted.

    f. The eContentType in the EncapsulatedContentInfo is id-ad-rpkiManifest (OID 1.2.840.113549.1.9.16.1.26).

    g. The version of the rpkiManifest is 0.

    h. In the rpkiManifest, thisUpdate precedes nextUpdate.

    i. The version of the SignerInfo is 3.

    j. The digestAlgorithm in the SignerInfo object is SHA-256 (OID 2.16.840.1.101.3.4.2.1).

    k. The signatureAlgorithm in the SignerInfo object is RSA (OID 1.2.840.113549.1.1.1).

    l. The signedAttrs field in the SignerInfo object is present and contains both the ContentType attribute (OID 1.2.840.113549.1.9.3) and the MessageDigest attribute (OID 1.2.840.113549.1.9.4).

    m. The unsignedAttrs field in the SignerInfo object is omitted.

2. Use the public key in the EE certificate to verify the signature on the Manifest.

3. Verify that the EE certificate is a valid end-entity certificate in the resource PKI by constructing a valid certificate path to a trust anchor. (See [ID.sidr-res-certs] for more details.)

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

8. Relying Party Use of Manifests

The goal of the relying party is to determine which signed objects to use for routing-related tasks, (e.g., which ROAs to use in the construction of route filters). Ultimately, this is a matter of local policy. However, in the following sections, we describe a sequence of tests that the relying party 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; and provide suitable warning text that should placed in a user-accessible log file. It is the responsibility of the relying party to weigh these risks against the risk of routing failure that could occur if valid data is rejected, and construct a suitable local policy. Note that if a certificate is deemed unfit for use do to local policy, then any descendant object that is validated using this certificate should also be deemed unfit for use (regardless of the status of the manifest at its own publication point).

8.1. Tests for Determining Manifest State

For a given publication point, the relying party should perform the following tests to determine the manifest state of the publication point:

1. For each entity using this publication point, select the entity's manifest having highest manifestNumber among all valid manifests (where manifest validity is defined in Section Section 7).

* If the publication point does not contain a valid manifest, see Section Section 8.2. Lacking a valid manifest, the following tests cannot be performed.

2. Check that the current time is between thisUpdate and nextUpdate.

   * If the current time does not lie within this interval then see Section 8.4, but still continue with the following tests.

3. Check that every file at the publication point appears in one and only one manifest, and that every file listed in each manifest appears at the publication point.

   * If there exists 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 8.5 but still continue with the following test.

4. Check that listed hash value of every file listed in each manifest matches the value obtained by hashing the file at the publication point.

   * If there exist files at the publication point whose hash does not match the hash value listed in the manifest, then see Section 8.6.

For a particular 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 certificate path used to validate the signed object, and the associated publication points, pass all of the above checks;

then the relying party can conclude that no attack against the repository system has compromised the given signed object, and the signed object MUST be treated as valid.

8.2. Missing Manifests

The absence of a valid 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 somewhat suspect, but may be used by the relying party, as per local policy.

The primary risk in using signed objects at this publication point is that a deleted CRL causes the relying party to improperly treat 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 the relying party may incorrectly discard a

large number of valid objects. This gives significant power to an adversary that is able to delete all manifests at the publication point.

Regardless of whether signed objects from this publication are deemed fit for use by the relying party, 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 the relying party has access to a local cache of previously issued manifests that are valid, the relying party MAY use the most recently previously issued valid manifests for this RPKI repository publication collection in this case for each entity that publishes at his publication point.

8.3. Invalid Manifests

The presence of invalid manifests 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 is greater than that on valid manifests.

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

In the case where the relying party has access to a local cache of previously issued manifests that are valid, the relying party MAY use the locally cached most recently previously issued valid manifest issued by the entity that issued the invalid manifest in this case.

8.4. 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 of a more recent manifest.

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 somewhat suspect, but may be used by the relying party as per local policy.

The primary risk in using signed objects at this publication point is that a newer manifest exists that, if present, would indicate that certain objects are have been removed or replaced. (e.g., the new manifest if present might show the existence of a newer CRL and the removal of several revoked certificates). Thus use of objects on a stale manifest may cause the relying party to incorrectly treat several invalid objects as valid. The risk is that a stale CRL

causes the relying party to 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 relying party 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 the relying party, this situation should result in a warning to the effect that: "The manifest for <pub point name> is no longer current. It is possible that undetected deletions have occurred at this publication point."

Note that there is also a less common case where the current time is before the thisUpdate time for the manifest. This case could be due to publisher error, or a local clock error, and in such a case this situation should result in a warning to the effect that: "The 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."

8.5. Mismatch between Manifest and Publication Point

If there exist otherwise valid signed objects that do not appear in any manifest, then provided the manifest is not stale (see Section Section 8.4) 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 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 manifest in question is stale, then there is a greater risk that the objects in question were revoked with a missing CRL, whose absence 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 relying party, 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 on the manifest <file list>."

If there exist 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 manifest. 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."

8.6. Hash Values Not Matching Manifests

A file appearing on a manifest with an incorrect hash value could occur because of publisher error, but it is likely to indicate that a serious error has occurred.

If an object appeared on a previous valid manifest with a correct hash value and 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 relying party will be treating a stale object as valid. This risk is more significant if the object in question is a CRL. Assuming that the object is validated in 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 whose hash appears in the manifest. If the manifest is stale (see Section 8.4) then it becomes more likely that the available version is more recent that 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 a relying party 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 CRL hand. The thisUpdate field of the CRL can be used to establish the most recent CRL in the case where a relying party has more than one valid CRL at hand.

Regardless of whether objects with incorrect hashes are deemed fit for use by the relying party, 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 could also be due to a publisher error."

9. Publication Repositories

The RPKI publication system model requires that every publication point be associated with one or more CAs or one or more EEs, 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. The manifest will contain at least one entry, the CRL issued by the CA upon repository creation. Upon the creation of the publication point associated with an EE, the EE MUST create and publish a manifest. The manifest in an otherwise empty repository publication point associated with an EE will contain no entries in the manifest's fileList sequence (i.e., the ASN.1 SEQUENCE will have a length of zero). [ID.sidr-repos-struct]

For each signed object, the EE certificate used to verify the object is either a single-use certificate, used to verify a single signed object, or a multiple-use certificate. In the case of a single-use EE certificate, the signed object is published in the repository publication point of the CA that issued the single use EE certificate, and is listed in the manifest associated with that CA certificate. In the case where an EE certificate is used to verify multiple objects, each signed object is published in the EE certificate's repository publication point and listed in the manifest associated with the EE certificate.

10. Security Considerations

Manifests provide an additional level of protection for relying parties of the repository system. Manifests can assist a relying party to determine if repository objects have been occluded or other

removed from view, and to determine if an older version of an object has been substituted for the current object .

Manifests cannot repair the effects of such forms of attempted corruption of repository retrieval operations, but are capable of allowing a relying party 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 conduction over an insecure channel. In those 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 older objects .

The signing structure of a manifest and the use of the nextUpdate value allows the relying party 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 a relying party to determine is 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 .

11. IANA Considerations

[Note to IANA, to be removed prior to publication: there are no IANA considerations stated in this version of the document.]

12. 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 relying party behavior.

13. Normative References

[ID.sidr-arch]

Lepinski, M., Kent, S., and R. Barnes, "An Infrastructure to Support Secure Internet Routing", Work in progress: Internet Drafts draft-ietf-sidr-arch-03.txt, February 2008.

[ID.sidr-repos-struct]

Huston, G., Loomans, R., and G. Michaleson, "A Profile for Resource Certificate Repository Structure", Work in progress: Internet Drafts draft-huston-sidr-repos-struct-02.txt, June 2008.

[ID.sidr-res-certs]
Huston, G., Michaleson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", Work in progress: Internet Drafts draft-ietf-sidr-res-certs-10.txt, June 2008.

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

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC4049] Housley, R., "BinaryTime: An Alternate Format for Representing Date and Time in ASN.1", RFC 4049, April 2005.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[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.

Authors' Addresses

Rob Austein
Internet Systems Consortium
950 Charter St.
Redwood City, CA 94063
USA

Email: sra@isc.org

Geoff Huston
Asia Pacific Network Information Centre
33 Park Rd.
Milton, QLD 4064
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

Full Copyright Statement

Intellectual Property

pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.