SIDR
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2009

R. Kisteleki
RIPE NCC
J. Boumans
RIPENCC

October 27, 2008

Securing RPSL Objects with RPKI Signatures
draft-kisteleki-sidr-rpsl-sig-00.txt

Status of this Memo

Abstract

This document describes a method to allow parties to electronically sign RPSL-like objects and validate such electronic signatures. This allows relying parties to detect accidental or malicious modifications on such objects. It also allows parties who run Internet Routing Registries or similar databases, but do not yet have RPSS-like authentication of the maintainers of certain objects, to verify that the additions or modifications of such database objects are done by the legitimate holder(s) of the Internet resources mentioned in those objects.

Table of Contents

1. Introduction

Objects issued by resource databases, like the RIPE DB, are generally protected by an authentication mechanism: anyone creating or modifying an object in the database has to have proper authorization to do so, therefore has to go through an authentication procedure (provide a password, certificate, e-mail signature, etc.) However, for objects transferred between resource databases, like for example AS Numbers, the authentication is not guaranteed. This means when one downloads an object issued from this database, one can reasonably safely claim that the object is valid, but for an imported object one can not. Also, once such an object is downloaded from the database, it becomes a simple (but still structured) text file with no integrity protection.

A potential usage for resource certificates could be to use them to secure such (both imported and downloaded) database objects, by applying a form of electronic signature over the object contents. Maintainers of such signed database objects should have their relevant resource certificate, which shows them as the legitimate holder of an Internet number resource. This would allow for the users of such database objects to verify that the contents are in fact produced by the legitimate holder(s) of the legitimate holder(s) of the Internet resources mentioned in those objects.

In other words, electronic signatures created using resource certificates can introduce object security in addition to the channel security already present in most of such databases.

2. Meaning of a signature

By signing an RPSL object, the signer of the object expresses that:

- they have the right to use the resource that the object refers to (ie. found as the primary key or in some other field of the object);
- they are responsible for the contents of the object; and
- they understand and agree with the contents of the object, up to the extend of the signed parts.

3. Actual Implementation, Syntax of a Signature

When signing an RPSL object, the input for the signature process is treated as a well-structured piece of information. The approach is similar to the one used in DKIM (Domain Key Identified Mail) [RFC4871]. In RPSL's case, the object-to-be-signed closely resembles an SMTP header, so it seems reasonable to adapt DKIM's relevant

features.

3.1. General Attributes, Meta Information of a Signature

The actual signature over such an object is itself a new attribute. It has a proposed attribute name "signature" and its contents consist of mandatory and optional fields. These fields are structured in a sequence of name=value pairs, separated by a semicolon ";" and a white space except for the last field, which is not followed by this. Collectively these fields make up the value for the new "signature" attribute. The "name" part of such a component is always a single ASCII character identifier, whereas value is an ASCII string whose contents depend on the field type. Mandatory fields must appear exactly once, whereas optional fields MUST appear at most once.

Mandatory fields of the "signature" attribute:

1. Version number of the signature (field "v"). This field currently MUST be set to "1".

2. Reference to the certificate corresponding to the private key used by the signer to sign this object (field "c"). This is a URL of type "rsync" or "http(s)" that points to a specific resource certificate in an RPKI repository. Inclusion of the certificate itself would have several drawbacks; the reference gives much more flexibility. The value of this field MUST be an "rsync://..." or an "http[s]://..." URL.

3. Signature method: what hash and signature and what crypto algorithm is/was used to create the signature (field "m"). The value of this field MUST be set to "rsa-sha1" or "rsa-sha256". Software for signature creation and/or validation must understand both algorithms.

4. Time of signing according to the signer's clock (field "t"). The format of the value of this field is the number of seconds since Unix EPOCH (00:00:00 on January 1, 1970 in the UTC time zone). The value is expressed as the decimal representation of an unsigned integer.

5. The signed attributes (field "a"). This is a list of attribute names, separated by an ASCII "+" character if there are more than one attributes mentioned. The list must only include any attribute at most once.

6. The signature itself (field "b"). This MUST be the last field in the list. The signature is the output of the signature algorithm used over the PKCS#1 version 1.5 (RFC3447) padded hash value over

the input. The value of this field is the base64 encoded value of the signature.

Optional fields of the "signature" attribute:

1. Signature expiration time (field "x"). The format of the value of this field is the number of seconds since Unix EPOCH (00:00:00 on January 1, 1970 in the UTC time zone). The value is expressed as the decimal representation of an unsigned integer.

2. [Yet to be decided] Reference(s) to other party's certificate(s) (field "o"). If such certificates are mentioned (referred to) in any signature, then this signature should be considered valid only in case when there are other signatures over this current object, and these other signatures refer to and can be verified with the certificates mentioned in this field. This mechanism allows having multiple signatures over an object in such a way that all of these signatures have to be present and valid for the whole signature to be considered valid. This would allow interdependent multi-party signatures over an object. One application for such a mechanism can be the case of a route[6] object, where both the prefix owner's and the AS owner's signature is expected (if they are different parties). The value of this field MUST be a list of "rsync://..." or "http[s]://..." URLs. If there are more such reference URLs, then they must be separated with a plus "+" sign. Any non URL-safe characters (including semicolon ";" and plus "+") must be URL encoded in all such URLs.

3.2. Selecting attributes-to-be-signed

One can look at an RPSL object as an (ordered) set of attributes, each having a "key: value" syntax. Understanding this structure can help in developing more flexible methods for applying electronic signatures.

Some of these attributes are automatically added by the database, some are database-dependent, yet others do not carry operationally important information. Therefore it seems reasonable to define which attributes are actually signed and which are not; in other words, we define a way of including important attributes while excluding some irrelevant ones. Selecting such attributes and creating an electronic signature exclusively over these attributes provides a reasonable approach for this.

The signer can pick which attributes are signed and in which order. The selection of the attributes carries operational value, while the order is an important detail needed for consistent signature

verification. This approach can accommodate local policies (e.g. some maintainers would want to sign 'remark' attributes too if they contain contact information, while others would not want this). Also, if there are new attributes added to an object type in the future, or even completely new object types are introduced, then the signature software components can easily be configured to deal with them.

A drawback of this approach is that the verifier not only has to check whether the signature itself is valid, but also has to check if the signed attributes contain everything that the verifier deems important. For example, the signer might have decided that the "origin" attribute is not signed, while the verifier's policy states that these attributes must be signed. In this case the verifier would reject the signature, which would render object as such less trusted in the verifier's eyes.

3.3. Normalization

3.3.1. Internal Normalizations in Databases

Normalization defines how one transforms an object-to-be-signed into a series of bits that can be signed (fed into a hash algorithm, the result into a signature algorithm, etc). The task of normalization is to hide away differences over various representations of the same object, which would otherwise result in invalid signatures, even though the important bits do not differ in two different representations. An example of this could be the difference of line terminators across different systems.

Because of database consistency rules and database operational reasons several database use internal normalization techniques that can change the format and/or actual content of some of the signed fields. Examples include:

- Representation of IPv6 addresses: always use the long form over the short form.
- Representation of IPv4 prefixes: use x.x.x.x-y.y.y.y notation, or x.x.x/y
- Key-cert objects have their fingerprint, method and owner lines auto-corrected if supplied incorrectly.
- "Changed" attribute is automatically corrected / filled in.

This means that the destination database in fact can change parts of the submitted data after it was signed. Then results in an invalid signature. As a potential remedy, if the signer of an object is not fully aware of the transformations the database will do to the object upon submission, then:

- the object should be first submitted to the destination database
- the database will apply the internal normalization rules
- the signer then downloads the object from the database and applies the signature to the resulting object.

The drawback here is that if there happen to be two different databases with different such rules, then signed objects cannot 'travel' between these without being re-signed in the appropriate format.

3.3.2. Normalization in Terms of an Electronic Signature

The following steps must be applied in order to achieve a normalized form of an object, before the actual signature process can begin:

1. Uppercase/lowercase conversion, except for the value of the to-be-created "signature" attribute. We believe that this causes no risks, although some parts of the input can potentially be case sensitive.

2. Comments (anything beginning with a "#") must be dropped.

3. Any trailing white space must be dropped.

4. All multi-line attributes are converted into their single-line equivalent.

5. The attribute names must be kept as part of one the attribute lines.

6. Multiple whitespaces must be collapsed into a single space (" ") character.

7. Add line endings must be converted to a singe new line ("\n") character (thus avoiding CR vs. CRLF differences).

3.4. Storage of the Signature Data

The result of the signature mechanism is exactly one new attribute for the object. As a summary of the method described above, the structure of this is as follows:

```
attribute1:  value1
attribute2:  value2
attribute3:  value3
...
signature:   v=1; c=rsync://.....; m=rsa-sha1; t=9999999999;
```

```
a=attribute1+attribute2+attribute3+...;
b=&lt;base64 data&gt;
```

4. Signature creation and validation steps

Given an RPSL object, in order to create the actual signature, the following steps are needed:

- Potentially submit the object-to-be-signed to the destination database, and download the resulting database-normalized object.
- Potentially create a one-off key pair and certificate to be used for signing this object this time. Alternatively, one can reuse the same key pair / certificate for multiple signatures.
- Based on the object type, the minimum set and the local policies, create a list of attribute names referring to the attributes that will be signed (contents of the "a" field).
- Arrange the selected attributes according to the selection sequence provided above, while filtering out the non-signed attributes.
- Construct the would-be "signature" attribute, with all its fields, leaving the "b" field empty (NULL value).
- Apply normalization procedure to the selected attributes (including the "signature" attribute).
- Create the signature over the results of the previous step (hash and sign).
- Attach the base64 encoded value of the signature to the "b" field.
- Append the resulting final "signature" attribute to the original object.

In order to validate a signature over such an object, the following steps are necessary:

- Check proper syntax of the "signature" attribute.
- Fetch the certificate referred to in the "c" field of the "signature" attribute, and check its validity using the steps described in [ID.sidr-res-certs].

- Check whether the signature (base64 decoded value of the "b" field) is correct when verified with the public key found in the certificate.
- Extract the list of attributes that were signed by the signer from the "a" field of the "signature" attribute".
- Verify that the list of signed attributes contains the minimum set of attributes for that object type.
- Potentially check local policy whether the list of the signed attributes conforms to it.
- Arrange the selected attributes according to the selection sequence provided above, while filtering out the non-signed attributes.
- Replace the value of the signature filed of the "signature" attribute with an empty string (NULL value).
- Apply normalization procedure to the selected attributes (including the "signature" attribute).
- Check whether the hash value of the so constructed input matches the one in the signature.

5. Signed Object Types, Minimum Set of Signed Attributes

This section describes a list of object types that could be signed using this approach, and a minimum set of attributes which MUST be signed for those object types.

The signer MAY chose to sign other attributes in addition to the required minimum set. In this case the additional attributes MUST be listed in the "a" field besides the minimum list.

This list generally excludes attributes that are used to maintain referential integrity in the databases that carry these objects, since these usually only make sense within the context of such a database, whereas the scope of the signatures is only one specific object. Since the attributes in the referred object (such as mnt-by, admin-c, tech-c, ...) can change without any modifications to the signed object, signing such attributes could lead to false sense of security in terms of the contents of the signed data.

```
as-block:
*  as-block
*  org

aut-num:
*  aut-num
*  as-name
*  member-of
*  import
*  mp-import
*  export
*  mp-export
*  default
*  mp-default

inet[6]num:
*  inet[6]num
*  netname
*  country
*  org
*  status

route[6]:
*  route[6]
*  origin
*  holes
*  org
*  member-of
```

6. Keys and Certificates used for Signature and Verification

The certificate that is referred to in the signature (in the "c" field):

- MUST be an end-entity (ie. non-CA) certificate
- MUST conform to the X.509 PKIX Resource Certificate profile [ID.sidr-res-certs]
- MUST have an [RFC3779] extension that contains or covers at least one Internet resource mentioned in a signed attribute
- SHOULD NOT be used to verify more than one signed object (ie. should be a "single-use" EE certificate, as defined in [ID.sidr-res-certs]).

7. Open Issues

Work still needs to be done for the following questions:

- Does character encoding pose a real problem?
- Is it feasible and does it provide value, if, while creating multiple signatures, those signatures refer to each other?

8. Security Considerations

[To be Completed.]

9. IANA Considerations

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

10. Normative References

[ID.sidr-res-certs] Huston, G., Michaleson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", Internet Draft draft-ietf-sidr-res-certs, October 2008.

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

Authors' Addresses

Robert Kisteleki

Email: robert@ripe.net
URI: http://www.ripe.net

Jos Boumans

Email: jib@ripe.net
URI: http://www.ripe.net

Full Copyright Statement

Intellectual Property