SIDR
Internet-Draft
Intended status: Standards Track
Expires: May 30, 2015

R. Kisteleki
RIPE NCC
B. Haberman
JHU APL

November 26, 2014

Securing RPSL Objects with RPKI Signatures
draft-ietf-sidr-rpsl-sig-06.txt

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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

Objects stored in 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, and therefore has to go through an authentication procedure (provide a password, certificate, e-mail signature, etc.) However, for objects transferred between resource databases, the authentication is not guaranteed. This means when downloading an object stored in this database, one can reasonably safely claim that the object is authentic, but for an imported object one cannot. Also, once such an object is downloaded from the database, it becomes a simple (but still structured) text file with no integrity protection. More importantly, the authentication and integrity guarantees associated with these objects do not always ensure that the entity that generated them is authorized to make the assertions implied by the data contained in the objects.

A potential use for resource certificates [RFC6487] is to use them to secure such (both imported and downloaded) database objects, by applying a form of digital signature over the object contents. A

maintainer of such signed database objects MUST possess a relevant resource certificate, which shows him/her as the legitimate holder of an Internet number resource. This mechanism allows the users of such database objects to verify that the contents are in fact produced by the legitimate holder(s) of the Internet resources mentioned in those objects. It also allows the signatures to cover whole RPSL objects, or just selected attributes of them. In other words, a digital signature created using the private key associated with a resource certificate can offer object security in addition to the channel security already present in most of such databases. Object security in turn allows such objects to be hosted in different databases and still be independently verifiable.

The capitalized 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. Signature Syntax and Semantics

When signing an RPSL object, the input for the signature process is transformed into a sequence of strings of (ASCII) data. The approach is similar to the one used in DKIM (Domain Key Identified Mail) [RFC4871]. In the case of RPSL, the object-to-be-signed closely resembles an SMTP header, so it seems reasonable to adapt DKIM's relevant features.

2.1. General Attributes, Meta Information

The digital signature associated with an RPSL object is itself a new attribute named "signature". It consists of mandatory and optional fields. These fields are structured in a sequence of name and value pairs, separated by a semicolon ";" and a white space. 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 that serves as an identifier; the value is an ASCII string the contents of which 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 MUST be set to "1".

2. Reference to the certificate corresponding to the private key used 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. The value of this field MUST be an "rsync://..." or an "http[s]://..." URL. Any non URL-safe characters (including semicolon ";" and plus "+") must be URL encoded.

3. Signature method (field "m"): what hash and signature algorithms were used to create the signature. The allowed algorithms which can be used for the signature are specified in [RFC6485].

4. Time of signing (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 more than one attribute is enumerated). The list must 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 using the appropriate private key and the calculated hash value of the object as inputs. The value of this field is the digital signature in base64 encoding [RFC4648].

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. 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 applications for such a mechanism include 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.

2.2. Signed Attributes

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 digital signatures.

Some of these attributes are automatically added by the database, some are database-dependent, yet others do not carry operationally important information. This specification allows the maintainer of such an object to define which attributes are signed and which are not, from among all the attributes of the object; in other words, we define a way of including important attributes while excluding irrelevant ones. Allowing the maintainer an object to select the attributes that are covered by the digital signature achieves the goals established in Section 1.

The type of the object determines the minimum set of attributes that MUST be signed. The signer MAY choose to sign additional attributes, in order to provide integrity protection for those attributes too.

When verifying the signature of an object, the verifier has to check whether the signature itself is valid, and whether all the specified attributes are referenced in the signature. If not, the verifier MUST reject the signature and threat the object as a regular, non-signed RPSL object.

2.3. Storage of the Signature Data

The result of applying the signature mechanism once is exactly one new attribute for the object. As an illustration, the structure of a signed RPSL object is as follows:

```
attribute1:  value1
attribute2:  value2
attribute3:  value3
...
signature:   v=1; c=rsync://.....; m=sha256WithRSAEncryption;
             t=9999999999;
             a=attribute1+attribute2+attribute3+...;
             b=<base64 data>
```

2.4. Number Resource Coverage

Even if the signature(s) over the object are valid according to the signature validation rules, they may not be relevant to the object; they also need to cover the relevant Internet number resources mentioned in the object.

Therefore the Internet number resources present in [RFC3779] extensions of the certificate referred to in the "c" field of the signature (or in the union of such extensions in the "c" fields of the certificates, in case multiple signatures are present) MUST cover the resources in the primary key of the object (e.g., value of the "aut-num:" attribute of an aut-num object, value of the "inetnum:" attribute of an inetnum object, values of "route:" and "origin:" attributes of a route object, etc.).

2.5. Validity Time of the Signature

The validity time interval of a signature is the intersection of the validity time of the certificate used to verify the signature, the "not before" time specified by the "t" field of the signature, and the optional "not after" time specified by the "x" field of the signature.

When checking multiple signatures, these checks are applied to each signature, individually.

3. Signature Creation and Validation Steps

3.1. Canonicalization

The notion of canonicalization is essential to digital signature generation and validation whenever data representations may change between a signer and one or more signature verifiers. Canonicalization defines how one transforms an a representation of data into a series of bits for signature generation and verification. The task of canonicalization is to make irrelevant differences in representations of the same object, which would otherwise cause signature verification to fail. Examples of this could be:

1. data transformations applied by the databases that host these objects (such as notational changes for IPv4/IPv6 prefixes, automatic addition/modification of "changed" attributes, etc.)

2. the difference of line terminators across different systems.

This means that the destination database might change parts of the submitted data after it was signed, which would cause signature verification to fail. This document specifies strict canonicalization rules to overcome this problem.

The following steps MUST be applied in order to achieve canonicalized representation of an object, before the actual signature (verification) process can begin:

1. Comments (anything beginning with a "#") MUST be omitted.

2. Any trailing white space MUST be omitted.

3. A multi-line attribute MUST be converted into its single-line equivalent. This is accomplished by:

   * Converting all line endings to a single blank space.

   * Concatenating all lines into a single line.

   * Replacing the trailing blank space with a single new line ("\n").

4. Numerical fields must be converted to canonical representations. These include:

   * Date and time fields MUST be converted to 64-bit NTP Timestamp Format [RFC5905].

   * AS numbers MUST be converted to ASPLAIN syntax [RFC5396].

   * IPv6 addresses must be canonicalized as defined in [RFC5952].

   * IPv4 addresses MUST be converted to a 32-bit representation (e.g., Unix's inet_aton()).

   * All IP prefixes (IPv4 and IPv6) MUST be represented in CIDR notaion [RFC4632].

5. The name of each attribute MUST be converted into lower case, and MUST be kept as part of the attribute line.

6. Tab characters ("\t") MUST be converted to spaces.

7. Multiple whitespaces MUST be collapsed into a single space (" ") character.

8. All line endings MUST be converted to a singe new line ("\n") character (thus avoiding CR vs. CRLF differences).

3.2. Signature Creation

Given an RPSL object, in order to create the digital signature, the following steps MUST be performed:

1. For each signature, a new key pair and certificate SHOULD be used. Therefore the signer SHOULD create a single-use key pair and end-entity resource certificate (see [RFC6487]) to be used for signing this object this time.

2. Create a list of attribute names referring to the attributes that will be signed (contents of the "a" field). The minimum set of these attributes is determined by the object type; the signer MAY select additional attributes.

3. Arrange the selected attributes according to the selection sequence specified in the "a" field as above, omiting all attributes that will not be signed.

4. Construct the new "signature" attribute, with all its fields, leaving the value of the "b" field empty.

5. Apply canonicalization rules to the result (including the "signature" attribute).

6. Create the signature over the results of the canonicalization process (according to the signature and hash algorithms specified in the "m" field of the signature attribute).

7. Insert the base64 encoded value of the signature as the value of the "b" field.

8. Append the resulting "signature" attribute to the original object.

3.3. Signature Validation

In order to validate a signature over such an object, the following steps MUST be performed:

1. Verify the syntax of the "signature" attribute (ie. whether it contains the mandatory and optional components and the syntax of these fields mathces the specification as described in section 2.1.)

2. Fetch the certificate referred to in the "c" field of the "signature" attribute, and check its validity using the steps described in [RFC6487].

3. Extract the list of attributes that were signed using the signer from the "a" field of the "signature" attribute.

4. Verify that the list of signed attributes matches the miminum set of attributes for that object type.

5. Arrange the selected attributes according to the selection sequence provided in the value of the "a" field, omitting all non-signed attributes.

6. Replace the value of the signature field "b" of the "signature" attribute with an empty string.

7. Apply the canonicalization procedure to the selected attributes (including the "signature" attribute).

8. Check the validity of the signature using the signature algorithm specified in the "m" field of the signature attribute, the public key contained in the certificate mentioned in the "c" field of the signature, the signature value specified in the "b" field of the signature attribute, and the output of the canonicalization process.

4. Signed Object Types, Set of Signed Attributes

This section describes a list of object types that MAY signed using this approach, and the set of attributes that MUST be signed for these object types.

This list generally excludes attributes that are used to maintain referential integrity in the databases that carry these objects, since these usually make sense only 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; therefore should only be done in order to provide full integrity protection of the object itself.

The newly constructed "signature" attribute is always included in the list.

as-block:

* as-block
* org
* signature

aut-num:

* aut-num
* as-name

* member-of
* import
* mp-import
* export
* mp-export
* default
* mp-default
* signature

inet[6]num:

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

route[6]:

* route[6]
* origin
* holes
* org
* member-of
* signature

For each signature, the RFC3779 extension appearing in the certificate used to verify the signature SHOULD include a resource entry that is equivalent to, or covers ("less specific" than) the

following resources mentioned in the object the signatrure is attached to:

- For the as-block object type: the resource in the "as-block" attribute.

- For the aut-num object type: the resource in the "aut-num" attribute.

- For the inet[6]num object type: the resource in the "inet[6]num" attribute.

- For the route[6] object type: the resource in the "route[6]" or "origin" (or both) attributes.

5. 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 [RFC6487]

- MUST have an [RFC3779] extension that contains or covers at least one Internet number resource included 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 [RFC6487]).

6. Security Considerations

RPSL objects stored in the IRR databases are public, and as such there is no need for confidentiality. Each signed RPSL object can have its integrity and authenticity verified using the supplied digital signature and the referenced certificate.

Since the RPSL signature approach leverages X.509 extensions, the security considerations in [RFC3779] apply here as well.

7. IANA Considerations

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

8. Acknowledgements

The authors would like to acknowledge the valued contributions from Jos Boumans, Steve Kent, and Sean Turner in preparation of this document.

9. Normative References

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

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

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, August 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4871] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, May 2007.

[RFC5396] Huston, G. and G. Michaelson, "Textual Representation of Autonomous System (AS) Numbers", RFC 5396, December 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.

[RFC6485] Huston, G., "The 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.

Authors' Addresses

Robert Kisteleki

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

Brian Haberman
Johns Hopkins University Applied Physics Lab

Email: brian@innovationslab.net