Securing RPSL Objects with RPKI Signatures

draft-kisteleki-sidr-rpsl-sig-00.txt

Robert Kisteleki, IETF73, Minneapolis

RPSLSIG: Why?

Problems we're looking at:

- Not all IRRs / IR databases have good enough authentication for maintaining objects
  - Some databases also function as mirrors
- Once an object is fetched from a database, one cannot verify its authenticity
- Difficult to spot malicious modifications (or typos) to these objects

RPSLSIG: signatures on RPSL objects

One potential use of RPKI certificates is to allow signatures on such objects:

- Create electronic signatures over the contents of such objects
- Prove that the legitimate holder of the contained resources created/maintains the object
- Provide integrity protection for the object even if it leaves its original database
- Provide “object security” in addition to existing channel security

RPSLSIG: signatures on RPSL objects

Some properties of the RPSLSIG approach:

- It’s not specific to any RPSL object type
  - General enough to cover route[6], inet[6]num, aut-num, as-block, …
- Allows multiple signatures
  - Useful for route[6] objects, but not restricted to those
- Possible to incrementally roll out

RPSLSIG: 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 extent of the signed parts.

RPSLSIG: what to sign

Simple “blob” signing does not work:

- Generally, the database can change some of the contents => signature fails
  - CR/LF changes
  - adding changed:, source: attributes
  - Other “minor” changes can happen
- Signature has to fit in an RPSL-like structure
- The **content** needs to be signed, not the format
- Solution: selectively sign part of the content that carries real operational content, does not change and/or define rules to overcome minor changes.

RPSLSIG: RPSL-like objects

Look at the structure of an RPSL-like objects:

```
attribute1: value1
attribute2: value2
attribute3: value3
```

- Looks like an SMTP header, null body.
- We were inspired by DKIM

RPSLSIG: Attribute selection

The signer is allowed to pick which attributes he actually signs.

- We defined a minimum set for the main object types
  - In order to avoid disagreements over what should have been signed
- The signer can still choose to sign more attributes
- The list of signed attributes becomes part of the signature

RPSLSIG: Normalization

Be aware of the database-inflicted changes, like:

- 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.

RPSLSIG: C14n

Basic steps:

- Uppercase/lowercase conversion
- Drop comments (#blah)
- White space conversion
- Multi-line attribute conversion (to one line format)
- Keep attribute names in the lines.
- Standardize line endings

RPSLSIG: The signature itself

The signature itself could be:

- DKIM style
  - fits the contents and structure very well
  - user-readable for the most part
  - simple
- CMS
  - well defined ASN1 structure
  - more difficult to do multiple signatures
  - output have to be tweaked to RPSL-like structure anyway

We chose the DKIM style approach.

RPSLSIG: The signature itself

Where to put the signature?

- Existing “remarks:” attribute
  - Backwards compatible
  - Makes it difficult to sign other “remarks:” lines
  - Still needs a special label to identify signature
    - Clients need to be modified to understand/make use it
- New “signature:” attribute
  - This is an extension
  - The signature is a new attribute, should be expressed as such
  - Compatibility with existing clients can still be guaranteed
    - With switches and conscious default behavior of servers
  - Clients need to be modified to understand/make use of it

RPSLSIG: an example

```
inetnum:         193.0.0.0 - 193.0.7.255
netname:         RIPE-NCC
descr:           RIPE Network Coordination Centre
descr:           Amsterdam, Netherlands
remarks:         Used for RIPE NCC infrastructure.
country:         NL
admin-c:         AMR68-RIPE
admin-c:         BRD-RIPE
tech-c:          OPS4-RIPE
status:          ASSIGNED PI
mnt-by:          RIPE-NCC-MNT
mnt-lower:       RIPE-NCC-MNT
signature:       v=1; c=rsync://rpki.ripe.net/….cer; m=rsa-sha1;
   t=1234567890; a=inetnum+netname+country+status; b=<base64-data>
source:          RIPE # Filtered
```

RPSLSIG: Signature fields

Defined fields:

- Version (v)
- Reference to signer’s certificate (c)
- Signature method (m)
- Signing time (t)
- Signed attributes (a)
- The signature itself (b)
- Optional: expiration time (x)
- Optional: reference to other signatures (o)

RPSLSIG: Signature creation 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 attribute (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.

RPSLSIG: Signature verification steps

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.

RPSLSIG: Open questions

Further work is needed still:

- Multiple signatures referring to each other - is it useful enough?
- Character encoding issues? Unicode?
- Sync with others who are thinking along similar lines.

Questions?

Robert Kisteleki, Jos Boumans

robert@ripe.net

jib@ripe.net