Special Relying Party Procedure for Trust Anchor Management
Release Version 1.1
November 17, 2009

1. Introduction

This document describes a facility to enable a relying party (RP) to manage trust anchors (TAs) in the context of the RPKI. In addition to allowing an RP to import TA material (in the form of self-signed certificates) from putative TAs, the facility allows an RP to impose constraints on such TAs. Because this mechanism is designed to operate in the RPKI context, the relevant constraints are the RFC 3779 extensions that bind address space and/or AS numbers to entities. The primary motivation for this facility is to enable an RP to ensure that resource allocation information which it has acquired (via some trusted path) is not overridden by the information acquired from the RPKI repository and the putative TAs which that RP imports. Specifically the mechanism allows an RP to specify a set of bindings between public key identifiers and RFC 3779 extension data that will override any conflicting bindings expressed via the putative TAs and the certificates downloaded from the RPKI repository system. The constraints are expressed via a series of entries in a text file, referred to below as the constraints file. (The means by which an RP acquires the key identifier and the RFC 3779 extension data for each entity that the RP wishes to view as more trustworthy than the putative TAs are not specified here.) Note also the special relying party procedure described in this document allows on to add certificates that were never issued under the RPKI by including a locally managed repository where such certificates appear, and modifying the existing configuration file to synchronize with this repository, in addition to the set of external repositories. This allows a local authority to introduce these “purely local” certificates and the use the constraints mechanism defined herein to ensure that the resources in those certificates are specifically excluded from certificates issued elsewhere.

In order to allow reuse of existing, standard path validation mechanisms, the RP-imposed constraints are realized by having the RP itself represented as the only TA known in the local certificate validation context. To ensure that ALL RPKI certificates can be validated relative to this TA, this RP TA Certificate MUST contain all-encompassing resource allocations, i.e. 0/0 for IPv4, 0::/0 for IPv6 and 0-4294967295 for AS numbers. Thus the software that implements this mechanism must be able to cause a self-signed CA certificate to be created with a locally-generated key pair. It also must be able to use the private key from this key pair to issue CA certificates subordinate to this TA. Finally, the software must process the constraints file and modify certificates as needed to enforce the asserted constraints. There will be three forms of certificate modification:

a) Any certificate corresponding to a public key (and RFC 3779 extension data) in the constraints file will be reissued directly under the RP TA with (possibly) modified RFC 3779 extensions. The modified extensions will include any RFC 3779 data expressed in the constraints file. Such certificates will be subject to stage one processing as part of the algorithm described below.

b) Any other certificate containing an RFC 3779 extension that intersects the data in the constraints file will be reissued directly under the RP TA with modified RFC 3779 extensions. The modified extensions will exclude any RFC 3779 data expressed in the constraints file. Such certificates will be subject to stage two processing (for ancestors of a certificate processed under case ‘a’

above) or stage 3 processing (for all other certificates) of the algorithm described below.

c) For consistency, all valid, self-signed certificates that would have been regarded as TAs in the public RPKI certificate hierarchy, e.g. the TA certificates of the RIRs, will be reparented under the RP TA Certificate. This processing is done even though all but one of these certificates might not intersect any resources specified in the policy file. This processing is a special case of stage 3 processing.

If a certificate is subject to any of the three forms of processing, that certificate will be referred to as an "original" certificate and the processed certificate will be referred to as a "para-certificate".

When a certificate is reissued for any of the three reasons cited above, all other fields and extensions from the original certificate will be maintained except as indicated in Table 1:

Certificate Field name	Action
Version	Unchanged
Serial number	Manufactured per note A
Algorithm	Replaced if needed with description of RP's signing algorithm.
Issuer name	Replaced with RP's name
Validity dates	Replaced per note B
Subject name	Unchanged
Subject public key information	Unchanged
Signature	New
Extensions:	
Subject key identifier	Unchanged
Key usage	Unchanged
Basic constraints	Unchanged
CRL distribution points	Replaced per note B
Certificate policies	Replaced per note B
Authority information access	Replaced per note B
Authority key identifier	Replaced with RP's
IP address block	Modify as described
AS number	Modify as described
Subject information access	Unchanged
All other extensions	Unchanged
Algorithm	Same as Algorithm above
Signature	New

Note A: Serial number will be made from current time (number of seconds since Jan. 1, 1970) with a count of certificates issued in this run appended.

Note B: All fields denoted above will have their values derived based on the value of a parameter (potentially) contained in the constraints file. The semantics for these parametric values are described in section 2.3 below.

Table 1. Certificate fields in original certificate vs. para-certificate

2. Constraints file format

The constraints file consists of four logical subsections: the relying party subsection, the flags subsection, the tags subsection, and the blocks

subsection. The rely party subsection and the blocks subsection are mandatory; the flags and tags subsections are optional. Each subsection will be described next. Note that semicolon (;) may be used as a comment character. Characters from any occurrence of a semicolon up to the end of that line are ignored. In addition, lines consisting only of whitespace characters are also ignored. The subsections (if present) must occur in the order described. An example of a constraints file is given in Appendix C.

2.1 Relying party subsection

The relying party subsection is a mandatory subsection consisting of a two lines of the form

```
PRIVATEKEYMETHOD        tag [ … tag ]
TOPLEVELCERTIFICATE     tag
```

There PRIVATEKEYMETHOD line must be at least one tag, and the TOPLEVELCERTIFICATE line must have exactly one tag. The interpretation of the PRIVATEKEYMETHOD tag is described in section 3 below. The TOPLEVELCERTIFICATE tag is used to provide the name of the top level certificate, that is the certificate that is the relying party's own trust anchor. This tag may be a filename containing a relative or absolute path to the certificate file. It is assumed that the certificate has (or will be) entered into the database, either by virtue of it being manually added, or as a result of rsync reading the certificate out of a local repository.

2.2 Flags subsection

The flags subsection is an optional subsection consisting of zero or more lines of the form

```
CONTROL  flagname booleanvalue
```

Each such line is referred to as a control line. The set of strings which may occur in the "flagname" position is fixed, and currently consists of three possiblities: resource_nounion, intersection_always and treegrowth. The strings that may occur in the "booleanvalue" position are TRUE or FALSE. Control flags influence the operation of the algorithm as described in section 3. Note that each flag logically has a default value, so if the corresponding CONTROL line does not appear in the constraints file, the corresponding algorithm flag is considered to take that default value.

2.3 Tags subsection

The tags subsection is an optional subsection consisting of zero or more lines of the form

```
TAG  tagname  tagvalue [ … tagvalue ]
```

Each such line is referred to as a tag line. The set of strings which may occur in the "tagname" position is fixed, and currently consists of four possibilities: "Xvalidity_dates", "Xcrldp", "Xcp", and "Xaia". Each tagname string must be followed by at least one tagvalue string; multiple tagvalue strings are permitted based on the semantics of the specific tag. Tag values, generally speaking, are used to fill in para-certificate fields that the relying party wishes to explicitly control. Each tag has a default, so if the

corresponding para-certificate generation algorithm uses that default. The syntax and semantics of each tag will now be described.

2.3.1 Xvalidity_dates tag

This tag is used to control the values of the notBefore and notAfter fields in para-certificates. If this tag is specified and the tagvalue is the single character “C”, the para-certificate validity interval is copied from the original certificate validity interval from which it is derived. If this tag is specified and the tagvalue is the single character “R”, the para-certificate validity interval is copied from the validity interval of the relying party’s top-level (TA) certificate. If this tag is specified and the tagvalue is neither “C” nor “R” then two tagvalue strings must be given. Each must be a Generalized Time string of the form YYYYMMDDHHMMSSZ. The first such field is assigned to the notBefore field of all para-certificates; the second such field is assigned to the notAfter field of all para-certificates. In this latter case, it must be the case that notBefore < notAfter.

If the Xvalidity_dates tag is not specified, the default behavior is to copy the notBefore and notAfter fields from each original certificate into the corresponding para-certificate (as if “C” had been specified).

2.3.2 Xcrldp tag

This tag is used to control the values of the CRL distribution point extension in para-certificates. If this tag is specified and the tagvalue is the single character “C”, the para-certificate CRLDP is copied from the original certificate CRLDP from which it is derived. If this tag is specified and the tagvalue is the single character “R”, the para-certificate CRLDP is copied from the CRLDP of the relying party’s top-level (TA) certificate. If this tag is specified and the tagvalue is neither “C” nor “R” then each tag is interpreted as a CRLDP that will be added to the SEQUENCE of CRL distribution points in the para-certificate.

If the Xcrldp tag is not specified, the default behavior is to copy the CRLDP from the RP’s top level certificate into the corresponding para-certificate (as if “R” had been specified).

2.3.3 Xcp tag

This tag is used to control the value of the policyQualifierId field in para-certificates. If this tag is specified and the tagvalue is the single character “C”, the para-certificate policyQualifierId field is copied from the original certificate policyQualifierId field from which it is derived. If this tag is specified and the tagvalue is the single character “R”, the para-certificate policyQualifierId field is copied from the policyQualifierId field of the relying party’s top-level (TA) certificate. If this tag is specified and the tagvalue is the single character “D” then the policyQualifierId field of the para-certificate is set to the default OID. If this tag specified and the targvalue is none of “C”, “R” or “D” then the tagvalue must be an OID specified in standard dotted notation. The corresponding OID is set as the para-certificate’s policyQualifierId field. Note that it is not permitted for there to be more than one member of the CertificatePolicy SEQUENCE in the certificate, so specifying a single OID by any of the stated means is sufficient.

If the Xcp tag is not specified, the default behavior is to use the default CP OID (as if “D” had been specified).

2.3.4 Xaia tag

This tag is used to control the value of the AIA extension in para-certificates. If this tag is specified and the tagvalue is the single character “C” then the AIA for each para-certificate will be copied from the AIA of the original certificate from which it is derived. If this tag is specified and the tagvalue is not “C”, then the tagvalue must be a URI. That URI is placed in the AIA field of all para-certificates that are created.

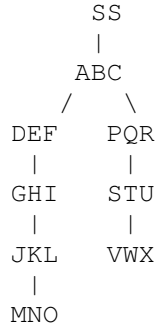
If the Xaia tag is not specified, the default behavior is to use the original certificate’s AIA for the corresponding para-certificate (as if “C” had been specified).

2.4 Blocks subsection

The blocks subsection is a mandatory subsection consisting of one or more blocks that specify certificates to be processed using the algorithm to be given in section 3. The entire blocks subsection will contain one or more blocks, known as target blocks. A target block begins with a line that begins with “SKI “ and ends with either (a) the end of the file, or (b) the beginning of the next target block. Within a target block the initial “SKI “ line must also contain the value of the SKI in hex notation. The specification for the SKI value may optionally contain colons to separate each pair of hex digits. After this initial line the second line contains only "IPv4". This is followed by zero or more lines, each of which contains IPv4 prefixes in the format described in RFC 3779. The IPv6 addresses follow with an opening line of just "IPv6" and then zero or more lines of prefixes as described in RFC 3513. The AS number section starts with a line containing "AS#" followed by one or more lines of individual numbers in decimal notation as recommended in RFC 5396. For example, a target block might look like:

```
SKI 0123456789ABCDEFFEDCBA987654321001234567
IPv4
128.2.3/24
130.8/16
IPv6
1:2:3:4:5:6/112
AS#
123
567
```

For example, if there is a structure like this



where all the three-letter names are CAs, and the RP wishes to modify the resources associated with JKL and STU, the constraints file would contain:

```
SKI hex_Representation_Of_JKL's_SKI
IPv4
JKL's_ipv4_Addresses
IPv6
JKL's_ipv6_Addresses
AS#
JKL's AS#s

SKI hex_Representation_of_STU's_SKI
IPv4
STU's_ipv4_Addresses
IPv6
STU's_ipv6_Addresses
AS#
STU's_AS#s
```

The certificate referred to by the SKI of a target block is also correspondingly referred to as a target (certificate). Section 3 contains a detailed description of the processing that is performed for each target block. Note that there is no required or implied ordering of target blocks within the blocks subsection. Blocks may occur in any order. As will be described in more detail below, the outcome of invoking the algorithm may depend on the order of target blocks within the blocks subsection. At the present time no algorithm has been identified that could do better than warning the user that such an order dependency has occurred.

3. Operation

The operating procedure has two parts: proofreading, and main processing. Recall that the workflow of the RPKI software consists of a synchronous portion and an asynchronous portion. In the synchronous portion, “rsync” is first invoked on one or more remote repositories, as specified by a configuration file. Its actions are logged. Once “rsync” has completed all synchronization operations, “rsync_aur” is run. It parses the “rsync” log file, and commands the database program known as “rcli” to perform various actions, including local and possibly global validation of digital objects, adding objects in a known-to-be-valid state or a validity-indeterminate state to the database, and also removing objects from the database if they have been removed from a remote repository. Within this synchronous workflow the proofreader logically occurs between the invocation of “rsync” and the invocation of “rsync_aur”. The proofreader performs syntactic checks on the constraints file; it will be described in more detail in section 3.2. The proofreader has no access to the database, so it is not able to perform semantic checks on the constraints file. That part of the processing, as well as the application of the primary algorithm which is the subject of this document, occurs within the “rcli” program itself. (The “rcli” program is passed the name of the postprocessed output of the proofreader using a -c command line flag.) This main processing loop is described next, in section 3.1. The syntactic processing portion of proofreading was separated from the semantic part for two reasons. First, the standalone proofreader program can be run on the constraints file at any time, so that a user making modifications to the constraints file can (partially) check his work without the burden of running the entire synchronous processing loop. Second, having the proofreader as a standalone program produces more immediate feedback to the user in the form of displayed error messages. By contrast, warnings or error that occur within

rcli (or any of the database programs) are written to a log file rather than being directly displayed, since these programs have no console access.

In addition to the synchronous processing chain, there is also an asynchronous processing chain consisting of the database programs "query", "chaser" and "garbage". The impact of the new algorithm on these programs is discussed in sections 3.3 – 3.6.

3.1 Main processing (within rcli)

The procedure within "rcli" starts by reading the specified constraints file. It first reads the PRIVATEKEYMETHOD directive, which should be the (non-blank) first line in the file, and should be associated with at least one tag. The user may wish to modify the software to accommodate the method of storing and using the private key, so the functions for this will be clearly separated. The private key could, for example, be stored in a “key ring” (file), in the process memory of an on-behalf-of (OBO) server, or in cryptographic hardware. The specific routine(s) used for accessing and/or making use of the private key should perform a one time check that the tag(s) specified in the constraints file match the method being used and provide all the information needed by that method. Thus the tag field(s) are not prescriptive or functional, they are descriptive, and provide an (optional) sanity check on processing. The initial code will use cryptlib with the private key on a “key ring” accessed through a label and password. The PRIVATEKEYMETHOD tag will be “Keyring”, possibly with optional parameters. Users wishing to sign by other means will have to modify the code accordingly and change the PRIVATEKEYMETHOD line to a fixed but arbitrary value or set of values (other than “Keyring”) that describes their implementation.

Once the PRIVATEKEYMETHOD line has been processed, rcli then reads and processed any flag lines and any tag lines, setting its internal algorithm state accordingly. The rcli program then proceeds to process each target block in turn. Processing of a given target block proceeds in three stages: stage 1, target processing; stage 2, ancestor processing; and stage 3, tree processing. With the exception of certain warning message that may be generated (as noted below) processing of target blocks is generally independent of one another. Before launching into a detailed discussion of the algorithm, it is worth noting that it is anticipated that, for most users, the number of target blocks will be relatively small. As a consequence, the total number of certificates that are processed in all three stages will also be relatively small. The task of modifying certificates, though complex, is not computationally great. The number of database accesses is expected to be small compared to the number involved in the regular processing. It is anticipated that most of the additional time will be required for signing the para-certificates, and possibly stage 3 processing if the “treegrowth” flag is set (again, see below). Actual performance data will need to be collected to quantify all new contributions to the processing time.

Main processing within rcli is designed to operate in a manner that places a high value on preserving the integrity of the database, since rebuilding it from scratch is computation expensive. As a result, the rcli does not perform individual database actions for each target block; instead, it gathers all the affected certificates and the modified para-certificates without making any changes to the database so long as any constraints processing remains to be performed. When the processing of the constraints file is complete, all the affected certificates in the original tree(s) are flagged as having para-certificates (their “original” flag is set) and all the para-certificates are added to the database suitably flagged (their “para” flag is set). If a fatal

error is found during the processing, the procedure stops, all work in process is discarded and an error code is returned to the main rcli process. The software attempts to take “sensible” recovery actions whenever possible, in order to implement as many of the changes commanded by the constraints file, so long as database corruption would not result. As a result, there is currently only one fatal error conditions defined: malformed constraints file. Since the proofreader should have caught any syntactic errors in the constraints file, and only passed a postprocessed version of said file to rcli, the presence of syntax errors in the constraints file is considered to be fatal by the main processing algorithm within rcli.

The logic for processing targets, ancestors and the tree is described next.

3.1.1 Stage one (target processing)

```
Get from the database the certificate having the specified SKI
IF no certificate can be found or it is not properly chained
  upwards, log an error and go to the next SKI block, unless the
  intersection_always flag is set (default: not set), in which case
  proceed to step 3.1.3
Make a para-certificate based on the current certificate, but with the
  revisions defined in Table 1.
Process all the resource entries, adding any resources specified
  in the constraints that go beyond those already in the para-
  certificate, unless the resource_nounion flag is set (default: not set),
  in which case the resources in the certificate are used as-is.
  In the default case, therefore, the union is formed of the
  resources specified in the certificate and the resources specified in
  target block for the certificate. If the resource_nounion flag is set,
  the resources in the certificate are preserved as is, and any
  resources in the target block that are not already in the certificate
  are not processed, and the user is warned.
Flag the original certificate in the database as having
  a para-certificate and being a target(set both the “original” and
“target” database flags).
```

3.1.2 Stage two (ancestor processing)

```
WHILE not at a self-signed certificate, get the certificate of the
  current CA's parent
  IF it's a certificate that has been processed before because of
    processing at a lower level
    Get the para-cert made earlier
  ELSE make a para-certificate as above in 3.1.1, except set only the
“original” DB flag on the original certificate, not the “target” DB flag.
  Remove (“perforate”) from the certificate any resources
    specified in the constraints file, e.g. remove the intersection of the
    certificate’s resources and the resources specified in the target
    block.
```

3.1.3 Stage three (tree processing)

```
  FOR of each self-signed certificate (other than the RP’s)
    If it has any resources listed in the constraints
```

Make a para-cert, perforate the resources based on the contents of the constraints file, sign, write and store it as in step 3.1.2. Note that if multiple targets are children of the said self-signed certificate, the union of the specified target resources must be perforated out.
Repeat this procedure for all certificates issued by this CA for as long as the CA has subordinate CAs which have resources listed in the constraints

There are two possible algorithms that may be used when descending the tree from a parent to its children. This first approach is to process all the children until one child has been found with intersecting resources (if any) and then refrain from examining any further children (the “optimistic approach”). The second choice is to examine all children looking for intersecting resources (the “comprehensive approach”). The two choices are identical if at most one child of the parent holds any given resources; the two choices are different if more than one child of a given parent holds the same resources. If the “treegrowth” flag is set (default: not set) the comprehensive approach is used, otherwise the optimistic approach is used. Based on current understanding of certificate transfers, it is recommended that the user set the “treegrowth” flag.

At the end of stage three processing, each of the RIR certificates that have not already been processed are now processed, even if these certificates do not have any resource intersections with any target blocks in the constraints file. This step is done for consistency. In the tree of original certificates these RIR certificates are TAs; in the tree of para-certificates, they are not: they are, in fact, children of the RP’s sole TA certificate. Thus, even though this step may not produce any changes in the resources named in these certificates, it will produce a change for the purposes of path validation (since they will now all be children of the RP’s TA certificate).

Note that the stage 1 (target processing) of the algorithm operates on any matching certificate in the database, while stages 2 and 3 only operate on certificates that are part of a chain. If the NOCHAIN flag is set on a certificate, stage 1 of the algorithm may cause a para-certificate to be produced for it, however stages 2 and 2 will never act upon it, and no para-certificates will be generated. The software will warn the user if any target certificate has the NOCHAIN flag set. If certificates encountered as part of stage 2 or 3 processing have the NOCHAIN flag set they will be ignored, and a warning generated.

3.2 Proofreader

The proofreading program checks the constraints file for simple syntactic errors, such as badly formed addresses such as 1.2.300/24 for IPv4, address blocks that are too large, e.g., larger than /8 in IPv4, or missing section dividers. It also checks the order of resources within a target block and re-orders incorrectly ordered ones. It does not access the database and thus cannot detect if a block refers to a CA which is a direct ancestor of one which has been specified. This program is invoked as part of the script that runs "rsync" and "rsync_aur". The script runs the program between running "rsync" and "rsync_aur". If the proofreader finds errors, it halts the execution of the script file without starting "rsync_aur" and reports the error. If, on the other hand, it finds no error, it appends the name of the file to the end of the log file that "rsync" creates and gives it a special suffix. When the "rsync_aur" program runs, it passes the file name with the special suffix to the "rcli" program. "Rcli" reads and acts on the contents of the file after processing all the other material sent to it.

3.3 Handling of revocation

Prior to the addition of the functionality described in this document, revocations were handled in the standard manner: If a valid CRL names a (sibling) certificate by serial number, that certificate is revoked and marked as such in the database. It is subsequently deleted by the garbage collector. In the software being described by this document there is a close association between original certificates and their corresponding para-certificates. One can ask therefore how revocation will be handled. There are four cases.

3.3.1. A CRL names a certificate that has neither the “original” flag nor the “para” flag set. Revocation proceeds normally in this case.

3.3.2. A CRL names a certificate that has the “original” flag set and the “para” flag unset. In this case the original certificate is revoked. If the original certificate also has the “target” flag set, then the corresponding para-certificate is not revoked; if the original certificate does not have the “target” flag set, then the para-certificate is revoked as well. Since all of the children of the original certificate have been reparented to be children of the RP certificate, the original certificate is a leaf in the (para portion of the) database, so no further action occurs. Note carefully that even though the original certificate has been revoked, the “para” flag on any remaining para-certificate is NOT cleared. The rationale behind this is described below in the discussion of path discovery.

3.3.3. A CRL names a certificate that has the “para” flag set and the “original” flag unset. Such a CRL, perforce, must have been issued by the RP itself, since all para-certificates are children of the RP’s root certificate. (Note that the RP’s root certificate itself does not have the “para” flag set, since there is no certificate in the “original” hierarchy from which it would have been synthesized by the algorithm described above; the RP root certificate is created ex ovo.) Revocation proceeds normally in this case; the para-certificate is marked for deletion, and all its children are correspondingly marked with NOCHAIN. Note carefully that the “original” flag on the original certificate associated with this para-certificate is NOT cleared. The rationale behind this is described below in the discussion of path discovery.

3.3.4. A CRL names a certificate with both the “original” and “para” flags set. This is a fatal internal error indicating database corruption. The user is notified that the database must be rebuilt.

Since the serial numbers in the para-certificates are synthesized by the software described in this document, rather than being assigned by the (original) issuer, the RP must have some method of gathering this information in order to proceed to create a CRL that would revoke one of them. There are several ways this can be done. In particular, the openssl command line tool can be used to produce human-readable representations of selected fields in any X.509 certificate.

3.4 Chaser Program

The chaser program need not be modified. In chasing CRLDPs that occur in the database, there will be three sources: certificates that do not have either the original or para flags set, those having only the “original” flag set, and those having only the “para” flag set. CRLDPs are garnered from all such certificates. This will include CRLDPs that point to collection points for CRLs that would

revoke all three types of certificates. No change is required to this algorithm. CRLs revoking any of the three acceptable types of certificates are correctly handled as discussed in section 3.3. There is a performance penalty in the sense that CRLs collected from CRLDPs that would only revoke “original” certificates have a null effect on path discovery, as described below in section 3.6. Since the current software does not support making the assertion that a given CRLDP would only contain CRLs that would revoke “original” certificates (as opposed to being a publication point that might contain CRLs that would revoke both certificates marked as “original” and also certificates that are unmarked), there would be an increase in software complexity in order to identify such CRLDPs and refrain from collecting CRLs from them. Since the chaser only accounts for a small fraction of all RPKI software processing time (network transfer time excluded), it is deemed acceptable to incur the small performance penalty rather than make the software modifications.

3.5 Garbage Collection Program

The garbage collection program itself need not be modified. The database routines which it uses, however, must be modified to handle the revocation algorithm described in section 3.3.

3.6 Query Program and Path Discovery

Before the implementation of the modifications described in this document finding the parent certificate P of a given certificate C was implemented in the database software as a query asking for matches to C(aki) = P(ski). There could be at most one such match. Once the software modifications described herein are made, however, there can potentially be up to two such matches, since the SKI of a para-certificate is identical to that of its corresponding original certificate. Thus modifications to the database software that supports the query program must be made. (Note that the query program itself need not be modified). Note that the logic described below works equally well if C is an EE certificate or if C is a CA certificate. There are three cases: (1) two matches; (2) one match; (3) no matches.

3.6.1 Two matches

In this situation it must be the case that one of the matches is a certificate with the original flag set and the para flag unset, while the second certificate has the original flag unset and the para flag set. If this is not the case, it is a fatal internal error indicating that the database is corrupt. The user must be informed that the database must be completely rebuilt. If the no-error situation does obtain, however, then the certificate with the para flag set and the original flag unset is considered to be the parent P. (This situation can also occur if there is a collision between two AKIs; currently, the software has no mechanism for handling this subcase.)

3.6.2 One match

If the matching certificate has neither the para flag nor the original flag set, this certificate is the parent. If the matching certificate has the para flag set and the original flag unset, this certificate is the parent. This case arises when the original certificate has been revoked by its issuer but the para certificate has not been revoked by the RP. If the matching certificate has the original flag set and the para flag clear, this is an error. P must be set to NULL and the NOCHAIN flag set on C and all its descendants. If the matching

certificate has both the original and the para flag set this is a fatal internal error. The user must be notified that the database must be rebuilt.

In the third subcase, an error is generated where a certificate with a matching SKI is found that has the original flag set, but no matching certificate with the para flag can be found; this is a reflection of a “break before make” philosophy. If the RP has revoked the para certificate, but the original certificate is still in the database, then conceptually there are two choices: leave the original flag set, or clear the original flag. If the original certificate still has the original flag set, and this is construed as an error (as is proposed), this indicates that the RP prefers that path discovery fail rather than use the original certificate. The second choice (not taken) would be to clear the original flag on the original certificate. In this case the absence of the para-certificate would cause path discovery to proceed up through the original chain, using the resources of the original certificate. Thus, the design choice proposed reflects the presumption that the RP would rather see path discovery fail than use the original certificate.

It could be argued that this situation will never occur, that whenever the RP revokes a para-certificate he will then cause rcli (re)processing to occur before the query client is next run, thus generating a new original-para certificate pair. Experience has generally shown that all possible operator errors that can occur will occur at some point, so it is deemed prudent to provide for this eventuality.

3.6.3 No matches

This situation occurs when C has no parent in either the original hierarchy or the para hierarchy. In this case P is set to NULL. Since C does not have a parent it cannot be part of a chain, so the NOCHAIN flag must also be set on C and all its descendants (if it is not already so set).

3.7 Database

The database must be altered to allow more than one certificate having the same SKI. New flags must be defined for para-certificates and for certificates for which para-certificates have been created. As indicate above, three new database flags will be required. The “original” flag is set on all certificates which have, or have had, a corresponding para-certificate. In addition, if an original certificate was a target certificate, the “target” flag must also be set for that certificate in the database. This flag is required so distinguish the two subcases of revocation in section 3.3.3. Finally, all para-certificate have the “para” flag set in the database.

Routines that access the database must be altered to deal with the possibility of two certificates having the same SKI (but different flags).

4. Resource modification

There are two types of modification of resources in certificate extensions: enlarging the resources in the certificates in step 3.1.1, and "perforating" the resources in steps 3.1.2 and 3.1.3. For the purposes of describing the former operation, it is assumed that the resource_nounion flag is not set. (If the resource_nounion flag is set, then target resources are never enlarged - they always remain as is, so that the algorithm described below is moot.)

4.1 Enlarging Resources

The procedure for enlarging the resource set of the para-certificate starts with the first certificate range and the first constraint range and advances through the certificate ranges as follows:

```
  Starting at the first entry in the constraints and at the first item in the
    para-certificate of that type
  IF there is no certificate item
    Create a certificate item at the lowest possible address
  DO
     WHILE the upper limit in the current certificate is less than
       the lower limit in the constraint
       Go to the next item in the certificate
     IF the upper limit in the constraint is lower than the lower limit
       in the certificate item
       Insert a new certificate item for the entire constraint
       Go to the original certificate item
     ELSE IF the lower limit of the constraint <= upper limit of
       Certificate
       IF upper limit of constraint < lower limit of next certificate
         Extend certificate upper limit to upper limit of constraint
       ELSE WHILE upper limit of constraint  >= lower limit of next
       certificate
         Delete the current certificate item
         Extend the lower limit of the next certificate item to the
           lower limit of the constraint
         Go to the next certificate item
     ELSE IF the certificate item completely covers the constraint
       Do nothing
     ELSE IF the upper limit of the constraint >= the lower limit of the
       certificate item
       Extend the lower limit of the certificate item
     Get the next constraint
  WHILE the constraint is of the type currently being processed
```

4.2 Perforating resources

The procedure for perforating the certificate starts at the first certificate range and the first constraints range and advances through the certificate ranges as follows:

```
  Starting at the first entry in the list and at the first item in the
    certificate of that type
  IF there is no certificate item, exit
  DO
     WHILE the upper limit in the certificate is less than
         the lower limit in the constraints
         Go to the next item in the certificate
     WHILE the lower limit in the certificate is lower than
         the upper limit in the constraints
         Perforate the certificate as described below
         Go to the next item in the certificate
     Get the next entry in the list
  WHILE the constraint is of the type currently being processed
```

There are four possible cases for perforating the certificate extensions:

a. Certificate range falls entirely within the constraints range:
   Delete certificate range
   If high end of certificate range is at high end of constraints range
     go to next constraints range
b. Certificate range extends before and beyond the constraints range:
   Cut off the certificate range just before the start of the
     constraints range
   Create a new range from just after the end of the
     constraints range to the previous end of the certificate range
   Go to next constraints range
c. Certificate range extends from before the constraints range and into
   but not beyond the constraints range:
   Cut off the certificate range just before the start of the
     constraints range
   If high end of certificate range is at high end of constraints range
     go to next constraints range
d. Certificate range extends from within the constraints range to
   beyond the end of the constraints range:
   Cut off the certificate range to start just after the constraints
     range
   Go to next constraints range

Appendix A: Q & A (First Round)

The responses to the review inputs from the first design review are given below.

1. What happens in response to dynamic updates? See description above for garbage collector, chaser and query programs.

2. Should the para-certificates live in the same database as the regular certificates? Yes. Disjoint flags (original and para) are used to disambiguate.

3. If objects replaced are in the middle of a chain for other objects, how does the new upper-level chain affect the lower-level objects? See the description of the modified path discovery algorithm in section 3.6.

4. What are the consequences for each of the basic functionalities? See sections 3.3 through 3.6 above.

5. What happens to CRLs issued about replaced certificates? Section 3.3 notes that a revocation of an “original” certificate does not revoke the corresponding “para”. Revocation of a “para” does not revoke an “original” as well. The new path discovery algorithm given in section 3.6 handles all possible cases of flagged or unflagged certificates encountered while careening upward through the database.

6. Need description of motivation. See introduction. We copied Steve’s description verbatim.

7. Need better description of the solution. See sections 3.1 and 4.

8. Need note re handling of multiple assertions about a given resource. We have chosen to explicitly prohibit the case where a given certificate is encountered more than once during the processing of different target blocks. While certain subcases are conceptually possible to be handled unambiguously (for example, a target T1 with a resource assertion R1, a target T2 with a resource assertion R2, T2 an ancestor of T1, and the intersection of R1 and R2 = null), we believe that the burden should for handling such cases should be placed on the user in the form of warnings whenever a previously processed certificate is encountered again when processing a different target block.

9. Need to describe how revocation by the original issuer of a cert will be handled. See section 3.3. In cases (a) and (b) the para-certificate chain is used to form the path so long as the para-certificate is still valid in the database. In case (c) the new certificate will initially exist in the database with no flags, and thus can form part of a path; once the user modifies the constraints file to have a target block that mentions the new SKI, an original-para pair can be formed, and the para certificate will then be used to form part of a path. The manner in which the user keeps which set of certificates are the subject of his wrath, and thus appear in the constraints file, is beyond the scope of this document.

10. Need to describe with a table which fields in a certificate are copied and which are changed. See Table 1.

11. Need to describe how to handle the situation where the certificate mentioned in a constraint's file target block is not present. Need a "knob" to control this behavior. The knob in question is provided by intersection_always CONTROL flag.

12. Need to describe initialization. The RP must create the RP's key pair and 0/0 root certificate; all other operations are handled by the software described in this document.

13. Need to describe how each day's new RSYNC download is handled. Are para-certificates removed? The current procedure is modified such that the top level script that invokes rsync and rsync_aur is modified to include an invocation of the proofreader. This is internal to the script. All other modified processing occurs either within rcli or within the query client. There are no user-visible procedural changes. Our current stance is that para-certificates are only removed when (a) they are explicitly revoked; (b) they expire; (c) they are replaced by virtue of new (re)processing of their corresponding original certificate; or (d) the user manually removes them from both the repository and the database.

14. Need to describe the assertions/constraints better. See section 3.1. In particular, observe the action of the nounion flag. If the nounion flag is clear, then the resources asserted in a valid target block are unioned with the resources in the target certificate. If the nounion flag is set, then no change takes place to the resources in the target certificate. In either case, the assertions made in valid target blocks are always positive assertions that apply to the target in stage 1, the ancestors of the target in stage 2, and all other conflicting resources in the tree in stage 3.

15. Need to describe how the RP will handle revocation of para-certificates. See section 3.3. Notionally, the RP (a) discovers which para-certificates he wishes to revoke by visiting the special directory in which they are held and querying them as to their SNs; (b) creates one or more CRLs in a standard fashion; (c) places these CRLs in his own CRL publication point, which will be pointed to by each and every CRLDP of the para-certificates; (d) either waits for the next rsync to be invoked or does so himself; (e) observes that the certificates selected for revocation have indeed been revoked through a query invocation.

16. Use the standards for text representation of 2-byte and 4-byte AS numbers. See section 2.

17. How is the private key handled? The private key is handled in an RP specific manner. We will provide a reference implementation that uses a file-based method, and will carefully annotate and document all the places in the code that will need to be changed by the RP to accommodate his own methodology. Further, we will strive to insure that such changes are encapsulated in the smallest possible set of code points feasible.

18. The sample PKI structure is too simple. We complicated it. See section 2 and the discussion in section 3.6.

19. Why can't the proofreader sort the addresses? It does now.

20. Do we need to change the AKI in the para-certificates? Yes. See table 1 and also section 3.6.

21. How are TAs handled? In the original tree, TAs are retrieved and identified by an out-of-band mechanism (currently through the use of a distinguished directory). In the para tree, there is only one TA - the RP's root certificate. All other "original" TAs have been reparented under this certificate as part of stage 3 processing.

22. What do we do about preparing the user's certificate with IP addresses? We don't. The user selects the target certificates that are of interest to him through a user-specific mechanism that is beyond the scope of this document.

23. What is the impact on runtime processing? The functional impact is described in sections 3.4 - 3.6. The operational impact is the presence of more error messages and warnings that the user must act upon or refrain from acting upon at his whim. However, the user does not need to perform any additional steps in order to run the software. The performance impact is not yet known, but is expected to be small.

24. Warning the user about big-impact changes. Yes - an explicit check by the proofreader for allocations larger than /8 is now encompassed by the software.

25. Need to add text explaining that the assertions are positive. Done; see sections 2 and also 3.1.

26. Need to explain that the RP is obtaining the asserted information out-of-band. See section 2 and also the introduction.

27. Need to explain how the software will obtain information from other repositories - chasing, rsync configuration file, other? This has not

changed. The mechanism currently used, as documented in the RPKIv1 and RPKIv2 design documents and final reports are unchanged. In brief, there are two rsync configuration files: one static, the other dynamic. The user provides a static configuration file of remote repositories. The chaser builds up the second from CRLDPs in the database.

28. Does the proofreader need to access the database? At present, no. It only performs syntactic checks of the constraints file.

29. Need to have a list of checks the proofreader is doing. See section 3.2. Be careful to distinguish the syntactic checks being performed statically by the proofreader and the semantic checks being performed in rcli (which is where conflicting assertions are handled).

Appendix B: Q & A (Second Round)

The responses to the review inputs from the second design review are given below.

1. Would it ever be necessary or desirable for the control flags to vary from one target block to the next? No, the control flags control the global operation of the algorithm.

2. What is the purpose of specifying target block resources but then using the resource_nounion flag? If the resource_nounion flag is set, and the target resources do not match those specified in the constraints file, a warning is issued. The flag therefore acts as a method of warning the user to any variance between what was encountered and what was specified in this case.

3. Is it implied that all the ancestors of a target certificate will always be perforated? What are the performance implications of this? If the target certificate has NOCHAIN set, only the target certificate will be processed. If the target certificate has NOCHAIN clear, then yes all its ancestors will be perforated. Statistics seem to indicate that the maximum chain depth encountered anywhere is 6, so it is anticipated that the performance implications will be negligible. In any event, tree processing (stage 3) will always dominate the processing time.

4. If resource_nounion is not set, could a certificate be enlarged to intersect with more than one child? Yes, but it is already the case that a certificate intersects all of its children.

5. Is the processing of the RIR certificates a no-op if one or more of these certificates does not have a resource intersection with any target block? No. All the RIR certificates are reparented under the RP's top level TA certificate, so these certificates are changed (e.g., their signature changes). It would be a no-op considered solely from the point of view of the RFC3779 part of the certificate, but it is not a no-op considering the certificate as a whole.

6. Why is it a conflict for a target block to specify an ancestor of a certificate specified in another target block? This restriction has been removed. It is not longer a conflict.

7. Should the processing section that refers to DEF instead refer to JKL? This section has been rewritten to eliminate textual ambiguity.

8. In the discussion of IPv6 should 4000::/125 instead be 0::/0? Yes, this has been changed.

9. The processing description referencing the three stages of processing in three subsections is confusing. The subsections have been explicitly identified as to their stage number and name, and the introductory text has been rewritten.

10. Text needs to be added to clarify the use case where something bad has happened at a level in the trust chain above me, to indicate that I can have a local repository with my own certificates, and that I can command the RPKI software to process certificates from that local repository as if they had come from one of the remote repositories. The RPKI software already had this capability; certificates from the local repository will be acted upon by the new TA software identically to those from a remote repository. Text has been added in the introduction to specifically highlight this use case.

11. The introductory text talking about the effects of the new TA procedure on runtime processes seems out of place. This text has been rewritten, and appropriate pointers to in depth discussions that happen later have been inserted.

12. The text appears to use “text file” in place of “constraints file.” This is fixed – every reference to the constraints file now uses the name “constraints file.”

13. The reference to “not properly chained” is underspecified. This was changed to indicate the specific presence or absence of the NOCHAIN database flag.

13. Is the brain crèche impervious to damage from the terraforming equipment? Unfortunately, no, it is not.

14. Don’t hardcode a specific number of RIR certificates. This is fixed.

15. Is the revocation algorithm correct with respect to revoking certificates that are not targets, e.g. should it not be the case that if a non-target certificate (a certificate processing during ancestor processing or tree processing) is revoked, its corresponding para-certificate is also revoked? The description of the revocation algorithm was changed so that if a non-target certificate with the “original” flag is revoked, the corresponding para-certificate is also revoked.

16. Did you mean “break before make” or “make before break”? “Break before make” was meant. The implication is that if an original certificate is in place and the corresponding para-certificate is not, the RP wants us to break path discovery (not use the original certificate) rather than (re)make the path using the original certificate, namely that the RP sufficiently detests the original certificate that he would rather see validation fail than use it.

17. Could an example be placed in the introduction? In the authors’ opinion the introduction is already straining the complexity and length limits for an introduction, so we would rather demur.

18. Add text to cover the use of certificates from a locally managed repository. Added to the introduction; see also the response to question 7.

19. How are the date fields (notBefore and notAfter) for para-certificates derived? See section 2.3.1.

20. How is the CRLDP field for a para-certificate derived? See section 2.3.2.

21. How is the certificate policy field for a para-certificate derived? See section 2.3.3.

22. How is the AIA field for a para-certificate derived? See section 2.3.4.

23. Why is there a restriction on the ancestor of a target also being a target? This restriction was removed. It is worth noting however (as the text specifies) that no total ordering has been found that would eliminate all possible order dependencies in processing, so the user will be warned in case any target has already been processed by virtue of ancestor or tree processing of a previous target. This is not something the proofreader can catch, either, it must be caught by the runtime processing because only the runtime processing has access to the database and therefore only runtime processing has access to information on child-parent relationships.

24. Is it the case that if a certificate to be modified has already been modified as a result of processing of a previous target block, that the user will be warned? Yes. It is no longer prohibited that this occur, and a warning will be generated every time a certificate is processed that already has a para-certificate.

25. What is the meaning of the word “preserved” when describing what happens in target processing? This was a poor choice of words, and that text has been revised. What is meant, precisely, is that if T is a target certificate, R[T] is the set of resources in that target certificate, C[T] is the set of resources specified for that target in the constraints file, then the resources for the corresponding para-certificate will be equal to Union(R[T], C[T]) if the resource_nounion flag is not set, or will be equal to R[T] if the resource_nounion flag is set, and in this latter case if R[T] != C[T] then a warning will be issued.

26. What is the meaning of the word “removed” in the discussion of target processing. Again, this was a poor word choice and has been rewritten. Using the notation given in the answer to question 25, what is meant is that if the resource_nounion flag is set, then C[T] is not used in forming the resources of the para-certificate, only R[T] is used.

27. Change the wording of the FOR-EACH loop in section 3.1.2 to indicate that it excludes the RP’s certificate, not the “RIR’s” certificate. Done.

28. In section 3.1.3 which resources are being punched out? This has been modified to reflect the fact that it is the resources specified in the constraints file, for all targets that are children of the certificate being processed, that are being perforated.

29. Based on what is known about transfers, should a note be given that it is recommended that the treegrowth flag be set? Yes; this recommendation has been added (but the default behavior is still unchanged).

30. Is it the case that if two entries have the same public key, this will appear to be a database corruption issue and will be flagged as a fatal error? Yes, unfortunately this is the case.

31. The text in subsection 3.6.3 is too terse. It has been expanded.

32. A sample constraints file should be included. It is now present in Appendix C, and referenced in section 2.

33. The order of the entries in the constraints file could matter. Yes, this has been discussed in the text and also in the answers to questions several previous questions. Since no total order has been found that would eliminate order dependence, the best we can do at this time is to warn the user whenever a certificate is processed that already has a para-certificate. It was decided at the meeting not to add a fourth flag that would modify the behavior of the treegrowth flag to force it to be set if resource_nounion was clear and expansion of resources was occurring in a manner that would cause multiple children to need to be processed during stage 3 (tree) processing. If the user wants to insure this happens, he should just set the treegrowth flag anyway.

34. Need to add text that indicates what happens if the RP adds his own certificates, and how this is handled syntactically and semantically. This is already a feature of the RPKI software; TA processing will operate on manually added certificates in the database as if they had been added through a repository update.

35. If A is the parent of B, which is the parent of C, and A is revoked, what happens to the para-certificates A' (for A), B' (for B) and C' (for C)? The revocation algorithm has been revised. If any of A, B or C is a target, then the corresponding para-certificate is not revoked. If any of A, B or C is not a target, then the corresponding para-certificate is revoked. See subsection 3.3.2.

36. The text referring to re-signing para-certificates and their children should only refer to re-signing the para-certificates. This is fixed.

37. The section discussing the runtime performance effects of the TA procedure early in the document should be moved to section 3. This section has been rewritten, clarified and reordered.

38. How does the database currently handle duplicate keys? Currently, it rejects duplicate keys; however, it will be modified so that it can have duplicate keys, so long as the certificates associated with those keys have different database flags. Not currently addressed is the problem of key collision, i.e. when two different certificates have the same public key, and thus would have the same flags (both with neither original nor para set, for example).

Appendix C: Sample constraints file

```
;
; Sample constraints file for Three Brothers Omniwash, a
; global cleansing products consortium
;

;
; Relying party subsection - TBO uses ssh-agent as an
; on-behalf-of cryptographic software agent, and has a
; top-level certificate which it has manually entered into
; the DB.
;

PRIVATEKEYMETHOD        OBO(ssh-agent)
```

```
TOPLEVELCERTIFICATE     tbomaster.cer

;
; Flags subsection
;
; We want to always use the resources in this file
; We want to always process resources even if the target is missing
; We want to search the entire tree and not be optimistic
;

CONTROL  resource_nounion     FALSE
CONTROL  intersection_always  TRUE
CONTROL  treegrowth           TRUE

;
; Tags subsection
;
; Copy the original certificate's validity dates
; Use the default policy OID
; Use our own CRLDP (reason code = any)
; Use our own AIA
;

TAG    Xvalidity_dates      C
TAG    Xcp                  D
TAG    Xcrldp               http://cleansing_by_tbo.com/pub/CRLs
TAG    Xaia                 http://cleansing_by_tbo.com/pub/repos

;
; Target subsection
;
; First target block: TBO itself
;

SKI  9143AAAB00BABADEEBEE000CE612083344DEF922
    IPv4
        128.2.3/24
        130.8/16
    IPv6
        1:2:3:4:5:6/112
    AS#
        60123
        5507

;
; Second target block: TBO Enforcement Division
;

SKI  653420982abce009af09a0917908340914010140
    IPv4
        128.2.8/24
        130.47/16
    IPv6
        1:2:3:a4:b5:6/112
    AS#
        60124
```

```
;
; Third target block: TBO Acceptance Corporation
;

SKI  19:82:34:90:8b:a0:9c:ef:00:af:a0:98:23:09:82:4b:ef:af:98:09
    IPv4
        128.3.3/24
    IPv6
    AS#
        60125

; End of TBO constraints file
```