RPSTIR Database Requirements

Last update: February 8, 2012

The RPSTIR software has functioned well with the MySQL database schema for several years. However, the addition of denial-of-service countermeasures, as well as the advent of new requirements for local trust anchor management, have shown areas where the database schema can be improved. This document defines the requirements (i.e. based on operations needed by the relying party) for the new database schema. Items marked with (\*) indicate requirements that were not present when the original database schema was designed.

# Top-Level Requirements (“T”)

1. DB must support retrieval of RPKI objects from remote repositories in arbitrary order.
2. DB must gracefully handle time-related state changes. This includes both the normal passage of time, as well as relying party (RP) decisions to deviate from nominal behavior such as instituting grace periods for all or part of the global RPKI.
3. DB must satisfy speed requirements. In the current operational concept, relying parties are expected to initiate a sweep of the global RPKI every 4 hours. The processing time for a global sweep must not exceed the 4 hour interval.
4. (\*) DB must support denial-of-service countermeasures against the following attack vectors:
   1. Potentially long lists in certs, CRLs, manifests, and ROAs
   2. Unbounded publication in repositories (by breadth or depth)
   3. Certificate validation loops or other attempts to sabotage certification path discovery
   4. Confinement violations: objects published by one CA (or in one repository) may be designed to affect objects published by other CAs (or in other repositories).
   5. Elevation of privilege: DB should be designed such that write permissions are given only to the processes that need it.
5. (\*) In case of unauthorized deletions in remote repositories, DB must support fallback to a local cache of previously downloaded files.
6. (\*) DB must support Local Trust Anchor/monitor operations
   1. DB must provide a mechanism to distinguish between the following classes of CA certificates: IANA trust anchor (1), IANA-issued RIR certs (5), RIR trust anchor (5), and ISP (all others). In addition, there must be a mechanism to associate each IANA-issued RIR cert with its corresponding RIR trust anchor cert.
   2. DB must provide a mechanism to detect IP address overlap between pairs of certificates that do not have an ancestor relationship (e.g. between RIR trust anchor certificates). ROAs issued over contested address space can affect routing, so the IP address overlap detection mechanism must also detect any ROAs issued over the contested address space. However, note that overlap may occur in normal “make-before-break” transfers, so this detection mechanism must be sensitive enough to distinguish transfers from contested resources.
   3. DB must support selective rollback of affected certificates and ROAs.
7. (\*) DB must support tracking and disabling a rogue CA from doing further damage.
8. (\*) RPSTIR must have the ability to generate forensic information. This requires DB support.
   1. RPSTIR must support analysis of retrieval failures over time. A persistently unreachable SIA could indicate a repository misconfiguration or a targeted DoS of a portion of the RPKI. DB must support persistent storage of each SIA’s history of reachability.
   2. DB schema must provide a mechanism for collecting global statistics (e.g. running total of manifest errors), and adapting to future statistics with relatively few schema modifications.
9. (\*) DB must support the notion of an “active set”: the set of RPKI objects that can affect routing. For example, under normal circumstances an expired certificate is removed from the active set. However, if a grace period is configured for the certificate, it returns to the active set.
10. DB must support the operation of an rpki-rtr cache server. In particular, at any point in time (e.g. even during a sweep of the global RPKI), it must be capable of delivering a consistent set of rpki-rtr PDUs to a router requesting an update.

# Derived Requirements (“D”)

1. (Derived from T1) DB must support “event-driven” operation where the addition and removal of RPKI files is input as a stream of events. This must include an “UNKNOWN” validation state, in order to support out-of-order retrieval from repositories.
2. (Derived from T2) DB must support “aging” operations which may bring objects in or out of their validity intervals.
3. (Derived from T3) In general, assume that RPKI objects will be read more than they will be written to (added/updated).
4. (Derived from T2 and T5) DB must support a “grace-period” for RPKI objects, in case of short-term loss of repository access.
5. (Derived from T5) A malicious person with access to a repository but not the CA private key should not be able to cause removal of an RPKI object or revert an object to a previous version. To address this, the database may need to reference multiple files with the same URI.
   1. DB must support a permanent store of files, not affected by active rsync (which will remove files in the repository mirroring process). For efficient retrieval, we propose a simplified content-addressable filesystem such as that used in the git version control system, where each file is stored and referenced based on its SHA-256 hash.
6. (Derived from T6) DB must support efficient “overlap” detection. This may not be possible within a SQL database, and may require an external “interval tree” data structure. Alternatively, if detection does not need to be immediate, a batch process can be run periodically.
7. (Derived from T6) DB must support identification of RIRs or other classes of CAs, whether manually or automatically (Q: what is needed?).
8. (Derived from T7) DB must support per-CA history and the ability to disable a specific CA from further processing (e.g. if it goes rogue publishes a million certs, we don’t want to download from its repo anymore).
9. (Derived from T3 and T4) DB must efficiently support the following operations (categorized by file type). We define “efficient” to mean that the amortized cost of all operations needed to process a global RPKI of size n, must not exceed O(n log n). Open question: Is this actually possible? There are some operations that are inherently O(n log n) on a small subset of the RPKI hierarchy. It may be that the total cost of processing a generic RPKI cannot be guaranteed to be better than O(n^2 log n), but that in practice, the global RPKI places acceptable limits on the constants.
   1. Adding any file other than a manifest
      1. Check manifest if hashes match.
      2. If all files on the manifest are present and there are no files not on the manifest, mark all relevant files as manifest-clean.
      3. If the new file is not on the manifest, mark all relevant files as not manifest-clean.
   2. Adding a certificate
      1. Determine parent cert's validity.
      2. Check CRL(s) to determine if the cert has been revoked.
      3. If applicable, update the validity of all children.
   3. Adding or Updating a CRL
      1. Determine parent cert's validity.
      2. Check whether CRL is newer or older than existing CRL, and retire old CRL.
      3. Check all covered certs to see if they are revoked.
   4. Adding a manifest
      1. Check whether MFT is newer or older than existing MFT, and retire old MFT.
      2. Check all covered files’ hashes.
      3. If the files all match and there are no extra or missing files, mark all the files as manifest-clean.
   5. Adding a ROA:
      1. Recursively find the closest parent cert that doesn't have the inherit bit set for IP addresses.
      2. If there is one, check that the ROAs prefixes are a subset of the above cert's IP resources.
      3. If there is no parent with inherit not set, mark the ROA as needing checking.
   6. Expiring or revoking a cert:
      1. Recurse over the children, reverting back to “unknown” validation state.
      2. Remove cert.
      3. Q: Do we need a knob for treating expiration and revocation differently?

# Other Open Questions

1. What fields should go into the database? E.g. Is it true that in general, any data item that can only be used by operations that require the entire file should not go in the database?

# Current Database Schema (for reference only, NOT under review)

/\*

The sceme for adding in SQL statements to scmtabbuilder is the following:

column names should start with a lowercase letter

all SQL keywords should be uppercase

\*/

static scmtab scmtabbuilder[] =

{

{ /\* RPKI\_CERT \*/

/\*

Usage notes: valfrom and valto are stored in GMT. local\_id is a unique

identifier with the new one obtained via max(local\_id) + 1

\*/

"rpki\_cert",

"CERTIFICATE",

"filename VARCHAR(256) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL DEFAULT 1,"

"subject VARCHAR(512),"

"issuer VARCHAR(512) NOT NULL,"

"sn BIGINT NOT NULL,"

"flags INT UNSIGNED DEFAULT 0,"

"ski VARCHAR(128) NOT NULL,"

"aki VARCHAR(128),"

"sia VARCHAR(1024),"

"aia VARCHAR(1024),"

"crldp VARCHAR(1024),"

"sig VARCHAR(520) NOT NULL,"

"hash VARCHAR(256),"

"valfrom DATETIME NOT NULL,"

"valto DATETIME NOT NULL,"

"sigval INT UNSIGNED DEFAULT 0,"

"ipblen INT UNSIGNED DEFAULT 0,"

"ipb BLOB,"

"ts\_mod TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,"

"local\_id INT UNSIGNED NOT NULL UNIQUE,"

" PRIMARY KEY (filename, dir\_id),"

" KEY ski (ski, subject),"

" KEY aki (aki, issuer),"

" KEY lid (local\_id),"

" KEY sig (sig),"

" KEY isn (issuer, sn)",

NULL,

0

},

{ /\* RPKI\_CRL \*/

/\*

Usage notes: this\_upd and next\_upd are stored in GMT. local\_id is a

unique identifier obtained as max(local\_id) + 1

issuer is the actual CRL issuer, obtained from the issuer field of

the CRL (direct CRL). snlist is the list of serial numbers for this issuer.

It is an array of bignums. The number of bignums in the list is snlen. Some

of these revocations may already have happened and the corresponding sn set

to 0 in the list. sninuse keeps track of the number of serial numbers that

are not zero in the list. When this number drops to 0, the entire CRL may be

deleted from the DB.

Note that snlist is of type MEDIUMBLOB, indicating that it can hold at most

16M/8 = 2M entries.

\*/

"rpki\_crl",

"CRL",

"filename VARCHAR(256) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL DEFAULT 1,"

"issuer VARCHAR(512) NOT NULL,"

"last\_upd DATETIME NOT NULL,"

"next\_upd DATETIME NOT NULL,"

"crlno BIGINT DEFAULT 0,"

"aki VARCHAR(128),"

"sig VARCHAR(520) NOT NULL,"

"hash VARCHAR(256),"

"snlen INT UNSIGNED DEFAULT 0,"

"sninuse INT UNSIGNED DEFAULT 0,"

"snlist MEDIUMBLOB,"

"flags INT UNSIGNED DEFAULT 0,"

"local\_id INT UNSIGNED NOT NULL UNIQUE,"

" PRIMARY KEY (filename, dir\_id),"

" KEY issuer (issuer),"

" KEY aki (aki),"

" KEY sig (sig),"

" KEY lid (local\_id)",

NULL,

0

},

{ /\* RPKI\_ROA \*/

/\*

Usage notes: the ski is the ski of the signing cert, and is thus

effectively the parent of this ROA. The asn is the AS number from

the ROA (there is only one now, not a list). The IP address information

is not stored here; it must be fetched from the file itself using

the ROA read code. local\_id is as with certs and crls.

\*/

"rpki\_roa",

"ROA",

"filename VARCHAR(256) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL DEFAULT 1,"

"ski VARCHAR(128) NOT NULL,"

"sig VARCHAR(520) NOT NULL,"

"sigval INT UNSIGNED DEFAULT 0,"

"hash VARCHAR(256),"

"ip\_addrs VARCHAR(32768) NOT NULL,"

"asn INT UNSIGNED NOT NULL,"

"flags INT UNSIGNED DEFAULT 0,"

"local\_id INT UNSIGNED NOT NULL UNIQUE,"

" PRIMARY KEY (filename, dir\_id),"

" KEY asn (asn),"

" KEY sig (sig),"

" KEY lid (local\_id),"

" KEY ski (ski)",

NULL,

0

},

{ /\* RPKI\_MANIFEST \*/

"rpki\_manifest",

"MANIFEST",

"filename VARCHAR(256) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL DEFAULT 1,"

"ski VARCHAR(128) NOT NULL,"

"hash VARCHAR(256),"

"this\_upd DATETIME NOT NULL,"

"next\_upd DATETIME NOT NULL,"

"cert\_id INT UNSIGNED NOT NULL,"

"files MEDIUMBLOB,"

"fileslen INT UNSIGNED DEFAULT 0,"

"flags INT UNSIGNED DEFAULT 0,"

"local\_id INT UNSIGNED NOT NULL UNIQUE,"

" PRIMARY KEY (filename, dir\_id),"

" KEY lid (local\_id)",

NULL,

0

},

{ /\* RPKI\_CTA \*/

"rpki\_cta",

"COMPOUNDTRUSTANCHOR",

"filename VARCHAR(256) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL DEFAULT 1,"

"ski\_rta VARCHAR(128) NOT NULL,"

"ski\_ee VARCHAR(128) NOT NULL,"

"hash VARCHAR(256),"

"flags INT UNSIGNED DEFAULT 0,"

"local\_id INT UNSIGNED NOT NULL UNIQUE,"

" PRIMARY KEY (filename, dir\_id),"

" KEY lid (local\_id)",

NULL,

0

},

{ /\* RPKI\_DIR \*/

"rpki\_dir",

"DIRECTORY",

"dirname VARCHAR(4096) NOT NULL,"

"dir\_id INT UNSIGNED NOT NULL,"

" PRIMARY KEY (dir\_id),"

" KEY dirname (dirname)",

NULL,

0

},

{ /\* RPKI\_METADATA \*/

"rpki\_metadata",

"METADATA",

"rootdir VARCHAR(4096) NOT NULL,"

"inited TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,"

"rs\_last TIMESTAMP DEFAULT 0,"

"qu\_last TIMESTAMP DEFAULT 0,"

"gc\_last TIMESTAMP DEFAULT 0,"

"ch\_last TIMESTAMP DEFAULT 0,"

"flags INT UNSIGNED DEFAULT 0,"

"local\_id INT UNSIGNED DEFAULT 1,"

" PRIMARY KEY (local\_id)",

NULL,

0

},

// these tables really should be specified in the server

// directory, but there was no good way to do that and not

// risk them not being created at initialization

{ /\* RTR\_SESSION \*/

"rtr\_session",

"RTR\_SESSION",

"session\_id SMALLINT UNSIGNED NOT NULL,"

" PRIMARY KEY (session\_id)",

NULL,

0

},

{ /\* RTR\_UPDATE \*/

"rtr\_update",

"RTR\_UPDATE",

"serial\_num INT UNSIGNED NOT NULL,"

"prev\_serial\_num INT UNSIGNED," // NULL indicates no previous serial number currently exists

"create\_time DATETIME NOT NULL,"

"has\_full BOOLEAN NOT NULL,"

" PRIMARY KEY (serial\_num),"

" UNIQUE KEY (prev\_serial\_num),"

" KEY create\_time (create\_time)",

NULL,

0

},

{ /\* RTR\_FULL \*/

"rtr\_full",

"RTR\_FULL",

"serial\_num INT UNSIGNED NOT NULL,"

"asn INT UNSIGNED NOT NULL,"

"ip\_addr VARCHAR(50) NOT NULL,"

" PRIMARY KEY (serial\_num, asn, ip\_addr)",

NULL,

0

},

{ /\* RTR\_INCREMENTAL \*/

"rtr\_incremental",

"RTR\_INCREMENTAL",

"serial\_num INT UNSIGNED NOT NULL," /\* serial number immediately after

the incremental changes, i.e.

after reading all of rtr\_incremental

where serial\_num = x, the client

is at serial number x \*/

"is\_announce BOOLEAN NOT NULL," /\* announcement or withdrawal \*/

"asn INT UNSIGNED NOT NULL,"

"ip\_addr VARCHAR(50) NOT NULL,"

" PRIMARY KEY (serial\_num, asn, ip\_addr)",

NULL,

0

},

} ;

# Proposed Database Schema 1st Pass (for reference, NOT under review)

-- NOTE: all the tables begin with 'rpstir\_'. This prefix may be configurable.

-- NOTE: unless otherwise specified, all hash columns of type binary(32) are the sha256 of the entire file

-- NOTE: CMS objects are stored with the same hash in both rpstir\_rpki\_cert and their respective type's table (e.g. rpstir\_rpki\_manifest)

-- TODO: check KEYs

-- database-level metadata

-- current version: SELECT schema\_version FROM rpstir\_metadata ORDER BY installed DESC LIMIT 1;

-- on initializing or upgrading the schema to version foo: INSERT INTO rpstir\_metadata (schema\_version) VALUES (foo);

CREATE TABLE rpstir\_metadata (

schema\_version int unsigned DEFAULT NULL, -- NULL indicates a development version with no version number

installed timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (installed)

);

-- (bi)map URIs to file hashes

-- hashes are used as unique IDs for all types of rpki objects in this schema

CREATE TABLE rpstir\_rpki\_file (

uri varchar(1023) NOT NULL, -- where the file was downloaded from

hash binary(32) NOT NULL, -- sha256 maybe?, filename could be e.g. /path/to/rpki/CACHE/01/23456789abcdef...

-- hash maybe should be the same as the alg used by manifests?

-- length would be different for different choice of hash function

downloaded datetime NOT NULL,

file\_type ENUM('cer', 'crl', 'roa', 'mft') DEFAULT NULL, -- NULL indicates unrecognized

parses boolean NOT NULL DEFAULT FALSE, -- avoid trying to reparse the same file if it fails the first time

flags bigint unsigned NOT NULL DEFAULT 0,

PRIMARY KEY (hash, uri), -- more useful as a constraint than for SELECT

KEY uri (uri, downloaded), -- find latest file for a specified uri

KEY uri\_good (uri, file\_type, parses, downloaded) -- find latest file of the same type that actually can be parsed for a specified uri

);

-- map internal hash to any other hash algs used in e.g. manifests

-- This should make algorithm agility somewhat more feasible in the future.

CREATE TABLE rpstir\_rpki\_hash (

hash binary(32) NOT NULL, -- hash used throughout the schema

alg ENUM('sha256') NOT NULL, -- alternate hash algorithm

data varbinary(512) NOT NULL, -- alternate hash

-- NOTE: these keys assume there are no hash collisions for any algorithm

PRIMARY KEY (hash, alg), -- lookup an alternate hash based on local hash

UNIQUE KEY (alg, data) -- lookup a local hash based on alternate hash

);

CREATE TABLE rpstir\_rpki\_cert\_asn (

hash binary(32) NOT NULL,

first\_asn int unsigned NOT NULL,

last\_asn int unsigned NOT NULL,

PRIMARY KEY (hash, first\_asn),

CHECK (first\_asn <= last\_asn)

);

CREATE TABLE rpstir\_rpki\_cert\_ip (

hash binary(32) NOT NULL,

first\_ip varbinary(16) NOT NULL, -- binary encoding, network byte order

last\_ip varbinary(16) NOT NULL, -- ditto

PRIMARY KEY (hash, first\_ip),

CHECK (length(first\_ip) = 4 OR length(first\_ip) = 16),

CHECK (length(first\_ip) = length(last\_ip)),

CHECK (first\_ip <= last\_ip)

);

CREATE TABLE rpstir\_rpki\_cert\_aia (

hash binary(32) NOT NULL,

preference int unsigned NOT NULL, -- lower number is more preferred

uri varchar(1023) NOT NULL,

PRIMARY KEY (hash, preference)

);

CREATE TABLE rpstir\_rpki\_cert\_sia (

hash binary(32) NOT NULL,

method ENUM('id-ad-caRepository', 'id-ad-rpkiManifest', 'id-ad-signedObject') NOT NULL,

preference int unsigned NOT NULL, -- lower number is more preferred

uri varchar(1023) NOT NULL,

PRIMARY KEY (hash, method, preference)

);

CREATE TABLE rpstir\_rpki\_cert\_crldp (

hash binary(32) NOT NULL,

uri varchar(1023) NOT NULL,

PRIMARY KEY (hash, uri)

);

CREATE TABLE rpstir\_rpki\_cert (

hash binary(32) NOT NULL,

subject varchar(511) NOT NULL,

issuer varchar(511) NOT NULL,

sn bigint unsigned NOT NULL,

ski binary(20) NOT NULL,

aki binary(20) DEFAULT NULL,

valfrom datetime NOT NULL,

valto datetime NOT NULL,

inherit\_asn boolean NOT NULL DEFAULT FALSE,

inherit\_ip boolean NOT NULL DEFAULT FALSE,

PRIMARY KEY (hash),

KEY ski (ski, subject),

KEY aki (aki, issuer),

KEY sig (sig),

KEY isn (issuer, sn)

);

CREATE TABLE rpstir\_rpki\_signs (

signed\_hash binary(32) NOT NULL,

signer\_hash binary(32) NOT NULL,

correct boolean NOT NULL DEFAULT FALSE, -- correctness of the signature only

PRIMARY KEY (signed\_hash, signer\_hash)

);

CREATE TABLE rpstir\_rpki\_crl\_sn (

hash binary(32) NOT NULL,

serial bigint unsigned NOT NULL,

revocation\_date datetime NOT NULL,

PRIMARY KEY (hash, serial)

);

CREATE TABLE rpstir\_rpki\_crl (

hash binary(32) NOT NULL,

issuer varchar(511) NOT NULL,

last\_upd datetime NOT NULL,

next\_upd datetime NOT NULL,

crlno int unsigned NOT NULL,

aki binary(20) NOT NULL,

PRIMARY KEY (hash),

KEY issuer (issuer),

KEY aki (aki)

);

CREATE TABLE rpstir\_rpki\_manifest\_files (

hash binary(32) NOT NULL,

filename varchar(255) NOT NULL,

file\_hash varbinary(512) NOT NULL,

PRIMARY KEY (hash, filename)

);

CREATE TABLE rpstir\_rpki\_manifest (

hash binary(32) NOT NULL,

manifest\_number int unsigned NOT NULL,

this\_upd datetime NOT NULL,

next\_upd datetime NOT NULL,

file\_hash\_alg ENUM('sha256') NOT NULL,

PRIMARY KEY (hash),

);

CREATE TABLE rpstir\_prefix (

id bigint unsigned NOT NULL AUTO\_INCREMENT,

prefix varbinary(16) NOT NULL, -- binary encoding, network byte order, filled with 0s to the full length for the address family

prefix\_length tinyint unsigned NOT NULL,

max\_prefix\_length tinyint unsigned NOT NULL,

PRIMARY KEY (id),

UNIQUE KEY (prefix, prefix\_length, max\_prefix\_length),

CHECK (length(prefix) = 4 OR length(prefix) = 16),

CHECK (prefix\_length <= max\_prefix\_length),

CHECK (max\_prefix\_length <= length(prefix) \* 8)

);

CREATE TABLE rpstir\_rpki\_roa\_prefix (

hash binary(32) NOT NULL,

prefix\_id bigint unsigned NOT NULL,

PRIMARY KEY (hash, prefix\_id)

);

CREATE TABLE rpstir\_rpki\_roa (

hash binary(32) NOT NULL,

asn int unsigned NOT NULL,

PRIMARY KEY (hash)

);

CREATE TABLE rpstir\_rtr\_full (

serial\_num int unsigned NOT NULL,

asn unsigned NOT NULL,

prefix\_id bigint unsigned NOT NULL

PRIMARY KEY (serial\_num, asn, prefix\_id)

);

CREATE TABLE rpstir\_rtr\_incremental (

serial\_num int unsigned NOT NULL,

is\_announce boolean NOT NULL,

asn int unsigned NOT NULL,

prefix\_id bigint unsigned NOT NULL,

PRIMARY KEY (serial\_num, asn, prefix\_id)

);

CREATE TABLE rpstir\_rtr\_session (

session\_id smallint unsigned NOT NULL,

PRIMARY KEY (session\_id)

);

CREATE TABLE rpstir\_rtr\_update (

serial\_num int unsigned NOT NULL,

prev\_serial\_num int unsigned DEFAULT NULL,

create\_time datetime NOT NULL,

has\_full boolean NOT NULL,

PRIMARY KEY (serial\_num),

UNIQUE KEY prev\_serial\_num (prev\_serial\_num),

KEY create\_time (create\_time)

);