Network Working Group
Internet-Draft
Obsoletes: 6810 (if approved)
Intended status: Standards Track
Expires: April 8, 2016

R. Bush
Internet Initiative Japan
R. Austein
Dragon Research Labs

October 6, 2015

# The Resource Public Key Infrastructure (RPKI) to Router Protocol
# draft-ietf-sidr-rpki-rtr-rfc6810-bis-06

## Abstract

In order to verifiably validate the origin Autonomous Systems and Autonomous System Paths of BGP announcements, routers need a simple but reliable mechanism to receive Resource Public Key Infrastructure (RFC 6480) prefix origin data and router keys from a trusted cache. This document describes a protocol to deliver validated prefix origin data and router keys to routers.

This document describes version 1 of the rpki-rtr protocol.

## Status of This Memo



## Copyright Notice

Table of Contents

## 1. Introduction

In order to verifiably validate the origin Autonomous Systems (ASes) and AS paths of BGP announcements, routers need a simple but reliable mechanism to receive cryptographically validated Resource Public Key Infrastructure (RPKI) [RFC6480] prefix origin data and router keys from a trusted cache. This document describes a protocol to deliver validated prefix origin data and router keys to routers. The design is intentionally constrained to be usable on much of the current generation of ISP router platforms.

Section 3 describes the deployment structure, and Section 4 then presents an operational overview. The binary payloads of the protocol are formally described in Section 5, and the expected Protocol Data Unit (PDU) sequences are described in Section 8. The transport protocol options are described in Section 9. Section 10 details how routers and caches are configured to connect and authenticate. Section 11 describes likely deployment scenarios. The traditional security and IANA considerations end the document.

The protocol is extensible in order to support new PDUs with new semantics, if deployment experience indicates they are needed. PDUs are versioned should deployment experience call for change.

For an implementation (not interoperability) report on the use of this protocol with prefix origin data, see [RFC7128].

### 1.1. Requirements Language

The 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 RFC 2119 [RFC2119] only when they appear in all upper case. They may also appear in lower or mixed case as English words, without special meaning.

### 1.2. Changes from RFC 6810

The protocol described in this document is largely compatible with [RFC6810]. This section summarizes the significant changes.

- o New Router Key PDU type (Section 5.10) added.
- o Explicit timing parameters (Section 5.8, Section 6) added.
- o Protocol version number incremented from zero to one.
- o Protocol version number negotiation (Section 7) added.

## 2. Glossary

The following terms are used with special meaning.

Global RPKI: The authoritative data of the RPKI are published in a distributed set of servers at the IANA, Regional Internet Registries (RIRs), National Internet Registries (NIRs), and ISPs; see [RFC6481].

Cache: A coalesced copy of the published Global RPKI data, periodically fetched or refreshed, directly or indirectly, using the [RFC5781] protocol or some successor protocol. Relying party software is used to gather and validate the distributed data of the RPKI into a cache. Trusting this cache further is a matter between the provider of the cache and a relying party.

Serial Number: A 32-bit strictly increasing unsigned integer which wraps from 2^32-1 to 0. It denotes the logical version of a cache. A cache increments the value when it successfully updates its data from a parent cache or from primary RPKI data. While a cache is receiving updates, new incoming data and implicit deletes are associated with the new serial but MUST NOT be sent until the fetch is complete. A Serial Number is not commensurate between different caches or different protocol versions, nor need it be maintained across resets of the cache server. See [RFC1982] on DNS Serial Number Arithmetic for too much detail on the topic.

Session ID: When a cache server is started, it generates a Session ID to uniquely identify the instance of the cache and to bind it to the sequence of Serial Numbers that cache instance will generate. This allows the router to restart a failed session knowing that the Serial Number it is using is commensurate with that of the cache.

Payload PDU: A protocol message which contains data for use by the router, as opposed to a PDU which just conveys the semantics of this protocol. Prefixes and Router Keys are examples of payload PDUs.

## 3. Deployment Structure

Deployment of the RPKI to reach routers has a three-level structure as follows:

Global RPKI: The authoritative data of the RPKI are published in a distributed set of servers, RPKI publication repositories, e.g., by the IANA, RIRs, NIRs, and ISPs (see [RFC6481]).

Local Caches: A local set of one or more collected and verified caches. A relying party, e.g., router or other client, MUST have a trust relationship with, and a trusted transport channel to, any cache(s) it uses.

Routers: A router fetches data from a local cache using the protocol described in this document. It is said to be a client of the cache. There MAY be mechanisms for the router to assure itself of the authenticity of the cache and to authenticate itself to the cache.

## 4. Operational Overview

A router establishes and keeps open a connection to one or more caches with which it has client/server relationships. It is configured with a semi-ordered list of caches, and establishes a connection to the most preferred cache, or set of caches, which accept the connections.

The router MUST choose the most preferred, by configuration, cache or set of caches so that the operator may control load on their caches and the Global RPKI.

Periodically, the router sends to the cache the most recent Serial Number for which it has has received data from that cache, i.e., the router's current Serial Number, in the form of a Serial Query. When a router establishes a new session with a cache, or wishes to reset a current relationship, it sends a Reset Query.

The cache responds to the Serial Query with all data changes which took place since the given Serial Number. This may be the null set, in which case the End of Data PDU is still sent. Note that the Serial Number comparison used to determine "since the given Serial Number" MUST take wrap-around into account, see [RFC1982].

When the router has received all data records from the cache, it sets its current Serial Number to that of the Serial Number in the End of Data PDU.

When the cache updates its database, it sends a Notify message to every currently connected router. This is a hint that now would be a good time for the router to poll for an update, but is only a hint. The protocol requires the router to poll for updates periodically in any case.

Strictly speaking, a router could track a cache simply by asking for a complete data set every time it updates, but this would be very inefficient. The Serial Number based incremental update mechanism

allows an efficient transfer of just the data records which have changed since last update. As with any update protocol based on incremental transfers, the router must be prepared to fall back to a full transfer if for any reason the cache is unable to provide the necessary incremental data. Unlike some incremental transfer protocols, this protocol requires the router to make an explicit request to start the fallback process; this is deliberate, as the cache has no way of knowing whether the router has also established sessions with other caches that may be able to provide better service.

As a cache server must evaluate certificates and ROAs (Route Origin Attestations; see [RFC6480]), which are time dependent, servers' clocks MUST be correct to a tolerance of approximately an hour.

## 5. Protocol Data Units (PDUs)

The exchanges between the cache and the router are sequences of exchanges of the following PDUs according to the rules described in Section 8.

Reserved fields (marked "zero" in PDU diagrams) MUST be zero on transmission, and SHOULD be ignored on receipt.

### 5.1. Fields of a PDU

PDUs contain the following data elements:

Protocol Version: An eight-bit unsigned integer, currently 1, denoting the version of this protocol.

PDU Type: An eight-bit unsigned integer, denoting the type of the PDU, e.g., IPv4 Prefix, etc.

Serial Number: The Serial Number of the RPKI Cache when this set of PDUs was received from an upstream cache server or gathered from the Global RPKI. A cache increments its Serial Number when completing a rigorously validated update from a parent cache or the Global RPKI.

Session ID: A 16-bit unsigned integer. When a cache server is started, it generates a Session ID to identify the instance of the cache and to bind it to the sequence of Serial Numbers that cache instance will generate. This allows the router to restart a failed session knowing that the Serial Number it is using is commensurate with that of the cache. If, at any time after the protocol version has been negotiated (Section 7), either the router or the cache finds the value of the Session ID is not the

same as the other's, the party which detects the mismatch MUST immediately terminate the session with an Error Report PDU with code 0 ("Corrupt Data"), and the router MUST flush all data learned from that cache.

Note that sessions are specific to a particular protocol version. That is: if a cache server supports multiple versions of this protocol, happens to use the same Session ID value for multiple protocol versions, and further happens to use the same Serial Number values for two or more sessions using the same Session ID but different Protocol Version values, the serial numbers are not commensurate. The full test for whether serial numbers are commensurate requires comparing Protocol Version, Session ID, and Serial Number. To reduce the risk of confusion, cache servers SHOULD NOT use the same Session ID across multiple protocol versions, but even if they do, routers MUST treat sessions with different Protocol Version fields as separate sessions even if they do happen to have the same Session ID.

Should a cache erroneously reuse a Session ID so that a router does not realize that the session has changed (old Session ID and new Session ID have same numeric value), the router may become confused as to the content of the cache. The time it takes the router to discover it is confused will depend on whether the Serial Numbers are also reused. If the Serial Numbers in the old and new sessions are different enough, the cache will respond to the router's Serial Query with a Cache Reset, which will solve the problem. If, however, the Serial Numbers are close, the cache may respond with a Cache Response, which may not be enough to bring the router into sync. In such cases, it's likely but not certain that the router will detect some discrepancy between the state that the cache expects and its own state. For example, the Cache Response may tell the router to drop a record which the router does not hold, or may tell the router to add a record which the router already has. In such cases, a router will detect the error and reset the session. The one case in which the router may stay out of sync is when nothing in the Cache Response contradicts any data currently held by the router.

Using persistent storage for the Session ID or a clock-based scheme for generating Session IDs should avoid the risk of Session ID collisions.

The Session ID might be a pseudo-random value, a strictly increasing value if the cache has reliable storage, etc.

Length: A 32-bit unsigned integer which has as its value the count of the bytes in the entire PDU, including the eight bytes of header which end with the length field.

Flags: The lowest order bit of the Flags field is 1 for an announcement and 0 for a withdrawal. For a Prefix PDU (IPv4 or IPv6), the flag indicates whether this PDU announces a new right to announce the prefix or withdraws a previously announced right; a withdraw effectively deletes one previously announced Prefix PDU with the exact same Prefix, Length, Max-Len, and Autonomous System Number (ASN). Similarly, for a Router Key PDU, the flag indicates whether this PDU announces a new Router Key or deletes one previously announced Router Key PDU with the exact same AS Number, subjectKeyIdentifier, and subjectPublicKeyInfo.

The remaining bits in the flags field are reserved for future use. In protocol version 1, they MUST be 0 on transmission and SHOULD be ignored on receipt.

Prefix Length: An 8-bit unsigned integer denoting the shortest prefix allowed for the prefix.

Max Length: An 8-bit unsigned integer denoting the longest prefix allowed by the prefix. This MUST NOT be less than the Prefix Length element.

Prefix: The IPv4 or IPv6 prefix of the ROA.

Autonomous System Number: A 32-bit unsigned integer representing an ASN allowed to announce a prefix or associated with a router key.

Subject Key Identifier: 20-octet Subject Key Identifier (SKI) value of a router key, as described in [RFC6487].

Subject Public Key Info: a router key's subjectPublicKeyInfo value, as described in [I-D.ietf-sidr-bgpsec-algs]. This is the full ASN.1 DER encoding of the subjectPublicKeyInfo, including the ASN.1 tag and length values of the subjectPublicKeyInfo SEQUENCE.

Zero: Fields shown as zero MUST be zero on transmission. The value of such a field SHOULD be ignored on receipt.

## 5.2. Serial Notify

The cache notifies the router that the cache has new data.

The Session ID reassures the router that the Serial Numbers are commensurate, i.e., the cache session has not been changed.

Upon receipt of a Serial Notify PDU, the router MAY issue an immediate Serial Query (Section 5.3) or Reset Query (Section 5.4) without waiting for the Refresh Interval timer (see Section 6) to expire.

Serial Notify is the only message that the cache can send that is not in response to a message from the router.

If the router receives a Serial Notify PDU during the initial start-up period where the router and cache are still negotiating to agree on a protocol version, the router SHOULD simply ignore the Serial Notify PDU, even if the Serial Notify PDU is for an unexpected protocol version. See Section 7 for details.

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |     Session ID      |
|    1     |    0     |                     |
+-------------------------------------------+
|                                           |
|                Length=12                  |
|                                           |
+-------------------------------------------+
|                                           |
|               Serial Number               |
|                                           |
`-------------------------------------------'
```

## 5.3. Serial Query

The router sends Serial Query to ask the cache for all announcements and withdrawals which have occurred since the Serial Number specified in the Serial Query.

The cache replies to this query with a Cache Response PDU (Section 5.5) if the cache has a, possibly null, record of the changes since the Serial Number specified by the router, followed by zero or more payload PDUs and an End Of Data PDU (Section 5.8).

When replying to a Serial Query, the cache MUST return the minimum set of changes needed to bring the router into sync with the cache. That is, if a particular prefix or router key underwent multiple changes between the Serial Number specified by the router and the cache's current Serial Number, the cache MUST merge those changes to present the simplest possible view of those changes to the router. In general, this means that, for any particular prefix or router key, the data stream will include at most one withdrawal followed by at

most one announcement, and if all of the changes cancel out, the data stream will not mention the prefix or router key at all.

The rationale for this approach is that the entire purpose of the rpki-rtr protocol is to offload work from the router to the cache, and it should therefore be the cache's job to simplify the change set, thus reducing work for the router.

If the cache does not have the data needed to update the router, perhaps because its records do not go back to the Serial Number in the Serial Query, then it responds with a Cache Reset PDU (Section 5.9).

The Session ID tells the cache what instance the router expects to ensure that the Serial Numbers are commensurate, i.e., the cache session has not been changed.

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |     Session ID      |
|    1     |    1     |                     |
+-------------------------------------------+
|                                           |
|                 Length=12                 |
|                                           |
+-------------------------------------------+
|                                           |
|               Serial Number               |
|                                           |
`-------------------------------------------'
```

## 5.4. Reset Query

The router tells the cache that it wants to receive the total active, current, non-withdrawn database. The cache responds with a Cache Response PDU (Section 5.5), followed by zero or more payload PDUs and an End of Data PDU (Section 5.8).

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |         zero        |
|    1     |    2     |                     |
+-------------------------------------------+
|                                           |
|                 Length=8                  |
|                                           |
`-------------------------------------------'
```

## 5.5. Cache Response

The cache responds to queries with zero or more payload PDUs. When replying to a Serial Query (Section 5.3), the cache sends the set of announcements and withdrawals that have occurred since the Serial Number sent by the client router. When replying to a Reset Query (Section 5.4), the cache sends the set of all data records it has; in this case, the withdraw/announce field in the payload PDUs MUST have the value 1 (announce).

In response to a Reset Query, the new value of the Session ID tells the router the instance of the cache session for future confirmation. In response to a Serial Query, the Session ID being the same reassures the router that the Serial Numbers are commensurate, i.e., the cache session has not changed.

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |     Session ID      |
|    1     |    3     |                     |
+-------------------------------------------+
|                                           |
|                 Length=8                  |
|                                           |
`-------------------------------------------'
```

## 5.6. IPv4 Prefix

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |                     |
 | Version  |   Type   |         zero        |
 |    1     |    4     |                     |
 +-------------------------------------------+
 |                                           |
 |                 Length=20                 |
 |                                           |
 +-------------------------------------------+
 |          |  Prefix  |   Max    |          |
 |  Flags   |  Length  |  Length  |   zero   |
 |          |   0..32  |   0..32  |          |
 +-------------------------------------------+
 |                                           |
 |                IPv4 Prefix                |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |         Autonomous System Number          |
 |                                           |
 `-------------------------------------------'
```

The lowest order bit of the Flags field is 1 for an announcement and 0 for a withdrawal.

In the RPKI, nothing prevents a signing certificate from issuing two identical ROAs. In this case, there would be no semantic difference between the objects, merely a process redundancy.

In the RPKI, there is also an actual need for what might appear to a router as identical IPvX PDUs. This can occur when an upstream certificate is being reissued or there is an address ownership transfer up the validation chain. The ROA would be identical in the router sense, i.e., have the same {Prefix, Len, Max-Len, ASN}, but a different validation path in the RPKI. This is important to the RPKI, but not to the router.

The cache server MUST ensure that it has told the router client to have one and only one IPvX PDU for a unique {Prefix, Len, Max-Len, ASN} at any one point in time. Should the router client receive an IPvX PDU with a {Prefix, Len, Max-Len, ASN} identical to one it already has active, it SHOULD raise a Duplicate Announcement Received error.

## 5.7. IPv6 Prefix

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |                     |
 | Version  |   Type   |         zero        |
 |    1     |    6     |                     |
 +-------------------------------------------+
 |                                           |
 |                Length=32                  |
 |                                           |
 +-------------------------------------------+
 |          |  Prefix  |   Max    |          |
 |  Flags   |  Length  |  Length  |   zero   |
 |          |  0..128  |  0..128  |          |
 +-------------------------------------------+
 |                                           |
 +---                                     ---+
 |                                           |
 +---            IPv6 Prefix              ---+
 |                                           |
 +---                                     ---+
 |                                           |
 +-------------------------------------------+
 |                                           |
 |         Autonomous System Number          |
 |                                           |
 `-------------------------------------------'
```

Analogous to the IPv4 Prefix PDU, it has 96 more bits and no magic.

## 5.8. End of Data

The cache tells the router it has no more data for the request.

The Session ID and Protocol Version MUST be the same as that of the corresponding Cache Response which began the, possibly null, sequence of payload PDUs.

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |                     |
 | Version  |   Type   |     Session ID      |
 |    1     |    7     |                     |
 +-------------------------------------------+
 |                                           |
 |                 Length=24                 |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |               Serial Number               |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |              Refresh Interval             |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |               Retry Interval              |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |              Expire Interval              |
 |                                           |
 `-------------------------------------------'
```

The Refresh Interval, Retry Interval, and Expire Interval are all 32-bit elapsed times measured in seconds, and express the timing parameters that the cache expects the router to use to decide when next to send the cache another Serial Query or Reset Query PDU. See Section 6 for an explanation of the use and the range of allowed values for these parameters.

### 5.9. Cache Reset

The cache may respond to a Serial Query informing the router that the cache cannot provide an incremental update starting from the Serial Number specified by the router. The router must decide whether to issue a Reset Query or switch to a different cache.

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |                     |
 | Version  |   Type   |         zero        |
 |    1     |    8     |                     |
 +-------------------------------------------+
 |                                           |
 |                 Length=8                  |
 |                                           |
 `-------------------------------------------'
```

### 5.10. Router Key

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |          |          |
 | Version  |   Type   |   Flags  |   zero   |
 |     1    |    9     |          |          |
 +-------------------------------------------+
 |                                           |
 |                  Length                   |
 |                                           |
 +-------------------------------------------+
 |                                           |
 +---                                     ---+
 |          Subject Key Identifier           |
 +---                                     ---+
 |                                           |
 +---                                     ---+
 |                (20 octets)                |
 +---                                     ---+
 |                                           |
 +-------------------------------------------+
 |                                           |
 |                 AS Number                 |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |          Subject Public Key Info          |
 |                                           |
 `-------------------------------------------'
```

The lowest order bit of the Flags field is 1 for an announcement and 0 for a withdrawal.

The cache server MUST ensure that it has told the router client to have one and only one Router Key PDU for a unique {SKI, ASN, Subject Public Key} at any one point in time.  Should the router client

receive a Router Key PDU with a {SKI, ASN, Subject Public Key} identical to one it already has active, it SHOULD raise a Duplicate Announcement Received error.

Note that a particular ASN may appear in multiple Router Key PDUs with different Subject Public Key values, while a particular Subject Public Key value may appear in multiple Router Key PDUs with different ASNs. In the interest of keeping the announcement and withdrawal semantics as simple as possible for the router, this protocol makes no attempt to compress either of these cases.

Also note that it is possible, albeit very unlikely, for multiple distinct Subject Public Key values to hash to the same SKI. For this reason, implementations MUST compare Subject Public Key values as well as SKIs when detecting duplicate PDUs.

### 5.11. Error Report

This PDU is used by either party to report an error to the other.

Error reports are only sent as responses to other PDUs.

The Error Code is described in Section 12.

If the error is generic (e.g., "Internal Error") and not associated with the PDU to which it is responding, the Erroneous PDU field MUST be empty and the Length of Encapsulated PDU field MUST be zero.

An Error Report PDU MUST NOT be sent for an Error Report PDU. If an erroneous Error Report PDU is received, the session SHOULD be dropped.

If the error is associated with a PDU of excessive length, i.e., too long to be any legal PDU other than another Error Report, or a possibly corrupt length, the Erroneous PDU field MAY be truncated.

The diagnostic text is optional; if not present, the Length of Error Text field MUST be zero. If error text is present, it MUST be a string in UTF-8 encoding (see [RFC3269]).

```
 0          8          16         24        31
 .-------------------------------------------.
 | Protocol |   PDU    |                     |
 | Version  |   Type   |     Error Code      |
 |    1     |    10    |                     |
 +-------------------------------------------+
 |                                           |
 |                  Length                   |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |       Length of Encapsulated PDU          |
 |                                           |
 +-------------------------------------------+
 |                                           |
 ~           Copy of Erroneous PDU           ~
 |                                           |
 +-------------------------------------------+
 |                                           |
 |           Length of Error Text            |
 |                                           |
 +-------------------------------------------+
 |                                           |
 |              Arbitrary Text               |
 |                    of                     |
 ~          Error Diagnostic Message         ~
 |                                           |
 `-------------------------------------------'
```

## 6. Protocol Timing Parameters

Since the data the cache distributes via the rpki-rtr protocol are retrieved from the Global RPKI system at intervals which are only known to the cache, only the cache can really know how frequently it makes sense for the router to poll the cache, or how long the data are likely to remain valid (or, at least, unchanged). For this reason, as well as to allow the cache some control over the load placed on it by its client routers, the End Of Data PDU includes three values that allow the cache to communicate timing parameters to the router.

Refresh Interval: This parameter tells the router how long to wait before next attempting to poll the cache, using a Serial Query or Reset Query PDU. The router SHOULD NOT poll the cache sooner than indicated by this parameter. Note that receipt of a Serial Notify PDU overrides this interval and allows the router to issue an immediate query without waiting for the Refresh Interval to

expire. Countdown for this timer starts upon receipt of the containing End Of Data PDU.

Minimum allowed value: 1 second.

Maximum allowed value: 86400 seconds (one day).

Recommended default: 3600 seconds (one hour).

Retry Interval: This parameter tells the router how long to wait before retrying a failed Serial Query or Reset Query. The router SHOULD NOT retry sooner than indicated by this parameter. Note that a protocol version mismatch overrides this interval: if the router needs to downgrade to a lower protocol version number, it MAY send the first Serial Query or Reset Query immediately. Countdown for this timer starts upon failure of the query, and restarts after each subsequent failure until a query succeeds.

Minimum allowed value: 1 second.

Maximum allowed value: 7200 seconds (two hours).

Recommended default: 600 seconds (ten minutes).

Expire Interval: This parameter tells the router how long it can continue to use the current version of the data while unable to perform a successful query. The router MUST NOT retain the data past the time indicated by this parameter. Countdown for this timer starts upon receipt of the containing End Of Data PDU.

Minimum allowed value: 600 seconds (ten minutes).

Maximum allowed value: 172800 seconds (two days).

Recommended default: 7200 seconds (two hours).

If the router has never issued a successful query against a particular cache, it SHOULD retry periodically using the default Retry Interval, above.

## 7. Protocol Version Negotiation

A router MUST start each transport connection by issuing either a Reset Query or a Serial Query. This query will tell the cache which version of this protocol the router implements.

If a cache which supports version 1 receives a query from a router which specifies version 0, the cache MUST downgrade to protocol

version 0 [RFC6810] or send a version 1 Error Report PDU with Error Code 4 ("Unsupported Protocol Version") and terminate the connection.

If a router which supports version 1 sends a query to a cache which only supports version 0, one of two things will happen.

1. The cache may terminate the connection, perhaps with a version 0 Error Report PDU. In this case the router MAY retry the connection using protocol version 0.

2. The cache may reply with a version 0 response. In this case the router MUST either downgrade to version 0 or terminate the connection.

In any of the downgraded combinations above, the new features of version 1 will not be available.

If either party receives a PDU containing an unrecognized Protocol Version (neither 0 nor 1) during this negotiation, it MUST either downgrade to a known version or terminate the connection, with an Error Report PDU unless the received PDU is itself an Error Report PDU.

The router MUST ignore any Serial Notify PDUs it might receive from the cache during this initial start-up period, regardless of the Protocol Version field in the Serial Notify PDU. Since Session ID and Serial Number values are specific to a particular protocol version, the values in the notification are not useful to the router. Even if these values were meaningful, the only effect that processing the notification would have would be to trigger exactly the same Reset Query or Serial Query that the router has already sent as part of the not-yet-complete version negotiation process, so there is nothing to be gained by processing notifications until version negotiation completes.

Caches SHOULD NOT send Serial Notify PDUs before version negotiation completes. Note, however, that routers MUST handle such notifications (by ignoring them) for backwards compatibility with caches serving protocol version 0.

Once the cache and router have agreed upon a Protocol Version via the negotiation process above, that version is stable for the life of the session. See Section 5.1 for a discussion of the interaction between Protocol Version and Session ID.

If either party receives a PDU for a different Protocol Version once the above negotiation completes, that party MUST drop the session; unless the PDU containing the unexpected Protocol Version was itself

an Error Report PDU, the party dropping the session SHOULD send an Error Report with an error code of 8 ("Unexpected Protocol Version").

## 8. Protocol Sequences

The sequences of PDU transmissions fall into three conversations as follows:

### 8.1. Start or Restart

```
Cache                           Router
  ~                               ~
  | <----- Reset Query -------- | R requests data (or Serial Query)
  |                               |
  | ----- Cache Response -----> | C confirms request
  | ------- Payload PDU ------> | C sends zero or more
  | ------- Payload PDU ------> |   IPv4 Prefix, IPv6 Prefix,
  | ------- Payload PDU ------> |   or Router Key PDUs
  | ------- End of Data ------> | C sends End of Data
  |                               |   and sends new serial
  ~                               ~
```

When a transport connection is first established, the router MAY send a Reset Query and the cache responds with a data sequence of all data it contains.

Alternatively, if the router has significant unexpired data from a broken session with the same cache, it MAY start with a Serial Query containing the Session ID from the previous session to ensure the Serial Numbers are commensurate.

This Reset Query sequence is also used when the router receives a Cache Reset, chooses a new cache, or fears that it has otherwise lost its way.

The router MUST send either a Reset Query or a Serial Query when starting a transport connection, in order to confirm that router and cache are speaking compatible versions of the protocol. See Section 7 for details on version negotiation.

To limit the length of time a cache must keep the data necessary to generate incremental updates, a router MUST send either a Serial Query or a Reset Query periodically. This also acts as a keep-alive at the application layer. See Section 6 for details on the required polling frequency.

## 8.2. Typical Exchange

```
Cache                           Router
  ~                               ~
  | -------- Notify ----------> |  (optional)
  |                             |
  | <----- Serial Query ------- | R requests data
  |                             |
  | ----- Cache Response -----> | C confirms request
  | ------- Payload PDU ------> | C sends zero or more
  | ------- Payload PDU ------> |   IPv4 Prefix, IPv6 Prefix,
  | ------- Payload PDU ------> |   or Router Key PDUs
  | ------- End of Data ------> | C sends End of Data
  |                             |   and sends new serial
  ~                               ~
```

The cache server SHOULD send a notify PDU with its current Serial Number when the cache's serial changes, with the expectation that the router MAY then issue a Serial Query earlier than it otherwise might. This is analogous to DNS NOTIFY in [RFC1996]. The cache MUST rate limit Serial Notifies to no more frequently than one per minute.

When the transport layer is up and either a timer has gone off in the router, or the cache has sent a Notify, the router queries for new data by sending a Serial Query, and the cache sends all data newer than the serial in the Serial Query.

To limit the length of time a cache must keep old withdraws, a router MUST send either a Serial Query or a Reset Query periodically. See Section 6 for details on the required polling frequency.

## 8.3. No Incremental Update Available

```
Cache                           Router
  ~                               ~
  | <-----  Serial Query ------ | R requests data
  | ------- Cache Reset ------> | C cannot supply update
  |                             |   from specified serial
  | <------ Reset Query ------- | R requests new data
  | ----- Cache Response -----> | C confirms request
  | ------- Payload PDU ------> | C sends zero or more
  | ------- Payload PDU ------> |   IPv4 Prefix, IPv6 Prefix,
  | ------- Payload PDU ------> |   or Router Key PDUs
  | ------- End of Data ------> | C sends End of Data
  |                             |   and sends new serial
  ~                               ~
```

The cache may respond to a Serial Query with a Cache Reset, informing the router that the cache cannot supply an incremental update from the Serial Number specified by the router. This might be because the cache has lost state, or because the router has waited too long between polls and the cache has cleaned up old data that it no longer believes it needs, or because the cache has run out of storage space and had to expire some old data early. Regardless of how this state arose, the cache replies with a Cache Reset to tell the router that it cannot honor the request. When a router receives this, the router SHOULD attempt to connect to any more preferred caches in its cache list. If there are no more preferred caches, it MUST issue a Reset Query and get an entire new load from the cache.

## 8.4. Cache Has No Data Available

```
Cache                          Router
  ~                              ~
  | <-----  Serial Query ------ | R requests data
  | ---- Error Report PDU ----> | C No Data Available
  ~                              ~

Cache                          Router
  ~                              ~
  | <-----  Reset Query ------- | R requests data
  | ---- Error Report PDU ----> | C No Data Available
  ~                              ~
```

The cache may respond to either a Serial Query or a Reset Query informing the router that the cache cannot supply any update at all. The most likely cause is that the cache has lost state, perhaps due to a restart, and has not yet recovered. While it is possible that a cache might go into such a state without dropping any of its active sessions, a router is more likely to see this behavior when it initially connects and issues a Reset Query while the cache is still rebuilding its database.

When a router receives this kind of error, the router SHOULD attempt to connect to any other caches in its cache list, in preference order. If no other caches are available, the router MUST issue periodic Reset Queries until it gets a new usable load from the cache.

# 9. Transport

The transport-layer session between a router and a cache carries the binary PDUs in a persistent session.

To prevent cache spoofing and DoS attacks by illegitimate routers, it is highly desirable that the router and the cache be authenticated to each other. Integrity protection for payloads is also desirable to protect against monkey-in-the-middle (MITM) attacks. Unfortunately, there is no protocol to do so on all currently used platforms. Therefore, as of the writing of this document, there is no mandatory-to-implement transport which provides authentication and integrity protection.

To reduce exposure to dropped but non-terminated sessions, both caches and routers SHOULD enable keep-alives when available in the chosen transport protocol.

It is expected that, when the TCP Authentication Option (TCP-AO) [RFC5925] is available on all platforms deployed by operators, it will become the mandatory-to-implement transport.

Caches and routers MUST implement unprotected transport over TCP using a port, rpki-rtr (323); see Section 14. Operators SHOULD use procedural means, e.g., access control lists (ACLs), to reduce the exposure to authentication issues.

Caches and routers SHOULD use TCP-AO, SSHv2, TCP MD5, or IPsec transport.

If unprotected TCP is the transport, the cache and routers MUST be on the same trusted and controlled network.

If available to the operator, caches and routers MUST use one of the following more protected protocols.

Caches and routers SHOULD use TCP-AO transport [RFC5925] over the rpki-rtr port.

Caches and routers MAY use SSHv2 transport [RFC4252] using the normal SSH port. For an example, see Section 9.1.

Caches and routers MAY use TCP MD5 transport [RFC2385] using the rpki-rtr port. Note that TCP MD5 has been obsoleted by TCP-AO [RFC5925].

Caches and routers MAY use TCP over IPsec transport [RFC4301] using the rpki-rtr port.

Caches and routers MAY use TLS transport [RFC5246] using a port, rpki-rtr-tls (324); see Section 14.

### 9.1. SSH Transport

To run over SSH, the client router first establishes an SSH transport connection using the SSHv2 transport protocol, and the client and server exchange keys for message integrity and encryption. The client then invokes the "ssh-userauth" service to authenticate the application, as described in the SSH authentication protocol [RFC4252]. Once the application has been successfully authenticated, the client invokes the "ssh-connection" service, also known as the SSH connection protocol.

After the ssh-connection service is established, the client opens a channel of type "session", which results in an SSH session.

Once the SSH session has been established, the application invokes the application transport as an SSH subsystem called "rpki-rtr". Subsystem support is a feature of SSH version 2 (SSHv2) and is not included in SSHv1. Running this protocol as an SSH subsystem avoids the need for the application to recognize shell prompts or skip over extraneous information, such as a system message that is sent at shell start-up.

It is assumed that the router and cache have exchanged keys out of band by some reasonably secured means.

Cache servers supporting SSH transport MUST accept RSA and Digital Signature Algorithm (DSA) authentication and SHOULD accept Elliptic Curve Digital Signature Algorithm (ECDSA) authentication. User authentication MUST be supported; host authentication MAY be supported. Implementations MAY support password authentication. Client routers SHOULD verify the public key of the cache to avoid monkey-in-the-middle attacks.

### 9.2. TLS Transport

Client routers using TLS transport MUST present client-side certificates to authenticate themselves to the cache in order to allow the cache to manage the load by rejecting connections from unauthorized routers. In principle, any type of certificate and certificate authority (CA) may be used; however, in general, cache operators will wish to create their own small-scale CA and issue certificates to each authorized router. This simplifies credential rollover; any unrevoked, unexpired certificate from the proper CA may be used.

Certificates used to authenticate client routers in this protocol MUST include a subjectAltName extension [RFC5280] containing one or more iPAddress identities; when authenticating the router's

certificate, the cache MUST check the IP address of the TLS connection against these iPAddress identities and SHOULD reject the connection if none of the iPAddress identities match the connection.

Routers MUST also verify the cache's TLS server certificate, using subjectAltName dNSName identities as described in [RFC6125], to avoid monkey-in-the-middle attacks. The rules and guidelines defined in [RFC6125] apply here, with the following considerations:

Support for DNS-ID identifier type (that is, the dNSName identity in the subjectAltName extension) is REQUIRED in rpki-rtr server and client implementations which use TLS. Certification authorities which issue rpki-rtr server certificates MUST support the DNS-ID identifier type, and the DNS-ID identifier type MUST be present in rpki-rtr server certificates.

DNS names in rpki-rtr server certificates SHOULD NOT contain the wildcard character "*".

rpki-rtr implementations which use TLS MUST NOT use CN-ID identifiers; a CN field may be present in the server certificate's subject name, but MUST NOT be used for authentication within the rules described in [RFC6125].

The client router MUST set its "reference identifier" to the DNS name of the rpki-rtr cache.

## 9.3. TCP MD5 Transport

If TCP MD5 is used, implementations MUST support key lengths of at least 80 printable ASCII bytes, per Section 4.5 of [RFC2385]. Implementations MUST also support hexadecimal sequences of at least 32 characters, i.e., 128 bits.

Key rollover with TCP MD5 is problematic. Cache servers SHOULD support [RFC4808].

## 9.4. TCP-AO Transport

Implementations MUST support key lengths of at least 80 printable ASCII bytes. Implementations MUST also support hexadecimal sequences of at least 32 characters, i.e., 128 bits. Message Authentication Code (MAC) lengths of at least 96 bits MUST be supported, per Section 5.1 of [RFC5925].

The cryptographic algorithms and associated parameters described in [RFC5926] MUST be supported.

## 10. Router-Cache Setup

A cache has the public authentication data for each router it is configured to support.

A router may be configured to peer with a selection of caches, and a cache may be configured to support a selection of routers. Each must have the name of, and authentication data for, each peer. In addition, in a router, this list has a non-unique preference value for each server. This preference merely denotes proximity, not trust, preferred belief, etc. The client router attempts to establish a session with each potential serving cache in preference order, and then starts to load data from the most preferred cache to which it can connect and authenticate. The router's list of caches has the following elements:

Preference: An unsigned integer denoting the router's preference to connect to that cache; the lower the value, the more preferred.

Name: The IP address or fully qualified domain name of the cache.

Key: Any needed public key of the cache.

MyKey: Any needed private key or certificate of this client.

Due to the distributed nature of the RPKI, caches simply cannot be rigorously synchronous. A client may hold data from multiple caches but MUST keep the data marked as to source, as later updates MUST affect the correct data.

Just as there may be more than one covering ROA from a single cache, there may be multiple covering ROAs from multiple caches. The results are as described in [RFC6811].

If data from multiple caches are held, implementations MUST NOT distinguish between data sources when performing validation.

When a more preferred cache becomes available, if resources allow, it would be prudent for the client to start fetching from that cache.

The client SHOULD attempt to maintain at least one set of data, regardless of whether it has chosen a different cache or established a new connection to the previous cache.

A client MAY drop the data from a particular cache when it is fully in sync with one or more other caches.

See Section 6 for details on what to do when the client is not able to refresh from a particular cache.

If a client loses connectivity to a cache it is using, or otherwise decides to switch to a new cache, it SHOULD retain the data from the previous cache until it has a full set of data from one or more other caches. Note that this may already be true at the point of connection loss if the client has connections to more than one cache.

## 11. Deployment Scenarios

For illustration, we present three likely deployment scenarios.

Small End Site: The small multihomed end site may wish to outsource the RPKI cache to one or more of their upstream ISPs. They would exchange authentication material with the ISP using some out-of-band mechanism, and their router(s) would connect to the cache(s) of one or more upstream ISPs. The ISPs would likely deploy caches intended for customer use separately from the caches with which their own BGP speakers peer.

Large End Site: A larger multihomed end site might run one or more caches, arranging them in a hierarchy of client caches, each fetching from a serving cache which is closer to the Global RPKI. They might configure fall-back peerings to upstream ISP caches.

ISP Backbone: A large ISP would likely have one or more redundant caches in each major point of presence (PoP), and these caches would fetch from each other in an ISP-dependent topology so as not to place undue load on the Global RPKI.

Experience with large DNS cache deployments has shown that complex topologies are ill-advised as it is easy to make errors in the graph, e.g., not maintain a loop-free condition.

Of course, these are illustrations and there are other possible deployment strategies. It is expected that minimizing load on the Global RPKI servers will be a major consideration.

To keep load on Global RPKI services from unnecessary peaks, it is recommended that primary caches which load from the distributed Global RPKI not do so all at the same times, e.g., on the hour. Choose a random time, perhaps the ISP's AS number modulo 60 and jitter the inter-fetch timing.

## 12. Error Codes

This section contains a preliminary list of error codes. The authors expect additions to the list during development of the initial implementations. There is an IANA registry where valid error codes are listed; see Section 14. Errors which are considered fatal SHOULD cause the session to be dropped.

0: Corrupt Data (fatal): The receiver believes the received PDU to be corrupt in a manner not specified by another error code.

1: Internal Error (fatal): The party reporting the error experienced some kind of internal error unrelated to protocol operation (ran out of memory, a coding assertion failed, et cetera).

2: No Data Available: The cache believes itself to be in good working order, but is unable to answer either a Serial Query or a Reset Query because it has no useful data available at this time. This is likely to be a temporary error, and most likely indicates that the cache has not yet completed pulling down an initial current data set from the Global RPKI system after some kind of event that invalidated whatever data it might have previously held (reboot, network partition, et cetera).

3: Invalid Request (fatal): The cache server believes the client's request to be invalid.

4: Unsupported Protocol Version (fatal): The Protocol Version is not known by the receiver of the PDU.

5: Unsupported PDU Type (fatal): The PDU Type is not known by the receiver of the PDU.

6: Withdrawal of Unknown Record (fatal): The received PDU has Flag=0 but a matching record ({Prefix, Len, Max-Len, ASN} tuple for an IPvX PDU, {SKI, ASN, Subject Public Key} tuple for a Router Key PDU) does not exist in the receiver's database.

7: Duplicate Announcement Received (fatal): The received PDU has Flag=1 but a matching record ({Prefix, Len, Max-Len, ASN} tuple for an IPvX PDU, {SKI, ASN, Subject Public Key} tuple for a Router Key PDU) is already active in the router.

8: Unexpected Protocol Version (fatal): The received PDU has a Protocol Version field that differs from the protocol version negotiated in Section 7.

## 13. Security Considerations

As this document describes a security protocol, many aspects of security interest are described in the relevant sections. This section points out issues which may not be obvious in other sections.

Cache Validation: In order for a collection of caches as described in Section 11 to guarantee a consistent view, they need to be given consistent trust anchors to use in their internal validation process. Distribution of a consistent trust anchor is assumed to be out of band.

Cache Peer Identification: The router initiates a transport connection to a cache, which it identifies by either IP address or fully qualified domain name. Be aware that a DNS or address spoofing attack could make the correct cache unreachable. No session would be established, as the authorization keys would not match.

Transport Security: The RPKI relies on object, not server or transport, trust. That is, the IANA root trust anchor is distributed to all caches through some out-of-band means, and can then be used by each cache to validate certificates and ROAs all the way down the tree. The inter-cache relationships are based on this object security model; hence, the inter-cache transport can be lightly protected.

However, this protocol document assumes that the routers cannot do the validation cryptography. Hence, the last link, from cache to router, is secured by server authentication and transport-level security. This is dangerous, as server authentication and transport have very different threat models than object security.

So the strength of the trust relationship and the transport between the router(s) and the cache(s) are critical. You're betting your routing on this.

While we cannot say the cache must be on the same LAN, if only due to the issue of an enterprise wanting to off-load the cache task to their upstream ISP(s), locality, trust, and control are very critical issues here. The cache(s) really SHOULD be as close, in the sense of controlled and protected (against DDoS, MITM) transport, to the router(s) as possible. It also SHOULD be topologically close so that a minimum of validated routing data are needed to bootstrap a router's access to a cache.

The identity of the cache server SHOULD be verified and authenticated by the router client, and vice versa, before any data are exchanged.

Transports which cannot provide the necessary authentication and integrity (see Section 9) must rely on network design and operational controls to provide protection against spoofing/corruption attacks. As pointed out in Section 9, TCP-AO is the long-term plan. Protocols which provide integrity and authenticity SHOULD be used, and if they cannot, i.e., TCP is used as the transport, the router and cache MUST be on the same trusted, controlled network.

## 14. IANA Considerations

This section only discusses updates required in the existing IANA protocol registries to accommodate version 1 of this protocol. See [RFC6810] for IANA Considerations from the original (version 0) protocol.

All existing entries in the IANA "rpki-rtr-pdu" registry remain valid for protocol version 0. All of the PDU types allowed in protocol version 0 are also allowed in protocol version 1, with the addition of the new Router Key PDU. To reduce the likelihood of confusion, the PDU number used by the Router Key PDU in protocol version 1 is hereby registered as reserved (and unused) in protocol version 0.

The policy for adding to the registry is RFC Required per [RFC5226], either Standards Track or Experimental.

Assuming that the registry allows range notation in the Protocol Version field, the updated "rpki-rtr-pdu" registry will be:

| Protocol Version | PDU Type | Description |
|---|---|---|
| 0-1 | 0 | Serial Notify |
| 0-1 | 1 | Serial Query |
| 0-1 | 2 | Reset Query |
| 0-1 | 3 | Cache Response |
| 0-1 | 4 | IPv4 Prefix |
| 0-1 | 6 | IPv6 Prefix |
| 0-1 | 7 | End of Data |
| 0-1 | 8 | Cache Reset |
| 0 | 9 | Reserved |
| 1 | 9 | Router Key |
| 0-1 | 10 | Error Report |
| 0-1 | 255 | Reserved |

All existing entries in the IANA "rpki-rtr-error" registry remain valid for all protocol versions. Protocol version 1 adds one new error code:

| Error Code | Description |
|---|---|
| 8 | Unexpected Protocol Version |

## 15. Acknowledgments

The authors wish to thank Nils Bars, Steve Bellovin, Tim Bruijnzeels, Rex Fernando, Richard Hansen, Paul Hoffman, Fabian Holler, Russ Housley, Pradosh Mohapatra, Keyur Patel, David Mandelberg, Sandy Murphy, Robert Raszuk, Andreas Reuter, Thomas C. Schmidt, John Scudder, Ruediger Volk, Matthias Waehlisch, and David Ward. Particular thanks go to Hannes Gredler for showing us the dangers of unnecessary fields.

No doubt this list is incomplete. We apologize to any contributor whose name we missed.

## 16. References

### 16.1. Normative References

[I-D.ietf-sidr-bgpsec-algs]
Turner, S., "BGPsec Algorithms, Key Formats, & Signature Formats", draft-ietf-sidr-bgpsec-algs-11 (work in progress), August 2015.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

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

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC3269] Kermode, R. and L. Vicisano, "Author Guidelines for Reliable Multicast Transport (RMT) Building Blocks and Protocol Instantiation documents", RFC 3269, April 2002.

[RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, BCP 26, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", RFC 6487, February 2012.

[RFC6810] Bush, R. and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol", RFC 6810, January 2013.

[RFC6811] Mohapatra, P., Scudder, J., Ward, D., Bush, R., and R. Austein, "BGP Prefix Origin Validation", RFC 6811, January 2013.

## 16.2. Informative References

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, August 1996.

[RFC4808] Bellovin, S., "Key Change Strategies for TCP-MD5", RFC 4808, March 2007.

[RFC5781] Weiler, S., Ward, D., and R. Housley, "The rsync URI Scheme", RFC 5781, February 2010.

[RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, February 2012.

[RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", RFC 6481, February 2012.

[RFC7128] Bush, R., Austein, R., Patel, K., Gredler, H., and M. Waehlisch, "Resource Public Key Infrastructure (RPKI) Router Implementation Report", RFC 7128, February 2014.

## Authors' Addresses

Randy Bush
Internet Initiative Japan
5147 Crystal Springs
Bainbridge Island, Washington 98110
US

Email: randy@psg.com

Rob Austein
Dragon Research Labs

Email: sra@hactrn.net