Internet Engineering Task Force (IETF)
Request for Comments: 6810
Category: Standards Track
ISSN: 2070-1721
R. Bush
Internet Initiative Japan
R. Austein
Dragon Research Labs
January 2013

The Resource Public Key Infrastructure (RPKI) to Router Protocol

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
   1.1. Requirements Language 3
2. Glossary 3
3. Deployment Structure 4
4. Operational Overview 4
5. Protocol Data Units (PDUs) 6
   5.1. Fields of a PDU 6
   5.2. Serial Notify 8
   5.3. Serial Query 8
   5.4. Reset Query 9
   5.5. Cache Response 9
   5.6. IPv4 Prefix 10
   5.7. IPv6 Prefix 11
   5.8. End of Data 12
   5.9. Cache Reset 12
   5.10. Error Report 12
6. Protocol Sequences 14
   6.1. Start or Restart 14
   6.2. Typical Exchange 15
   6.3. No Incremental Update Available 15
   6.4. Cache Has No Data Available 16
7. Transport 17
   7.1. SSH Transport 18
   7.2. TLS Transport 18
   7.3. TCP MD5 Transport 19
   7.4. TCP-AO Transport 19
8. Router-Cache Setup 20
9. Deployment Scenarios 21
10. Error Codes 22
11. Security Considerations 23
12. IANA Considerations 24
13. Acknowledgments 25
14. References 25
   14.1. Normative References 25
   14.2. Informative References 26

1. Introduction

In order to verifiably validate the origin Autonomous Systems (ASes) of BGP announcements, routers need a simple but reliable mechanism to receive Resource Public Key Infrastructure (RPKI) [RFC6480] cryptographically validated prefix origin data from a trusted cache. This document describes a protocol to deliver validated prefix origin data 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 PDU sequences are described in Section 6. The transport protocol options are described in Section 7. Section 8 details how routers and caches are configured to connect and authenticate. Section 9 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, see [RTR-IMPL]

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.

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 Registry (NIRs), and ISPs; see [RFC6481].

Cache: A coalesced copy of the RPKI, which is periodically fetched/refreshed directly or indirectly from the Global RPKI using the [RFC5781] protocol/tools. 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 that 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. As a cache is receiving, 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 caches, 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 identifier 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.

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., 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 authoritative 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 Serial Number of the highest numbered data it has received from that cache, i.e., the router's current Serial Number. When a router establishes a new connection to a cache, or wishes to reset a current relationship, it sends a Reset Query.

The Cache responds with all data records that have Serial Numbers greater than that in the router's query. This may be the null set, in which case the End of Data PDU is still sent. Note that 'greater' 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 that 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 6.

Fields with unspecified content MUST be zero on transmission and MAY be ignored on receipt.

5.1. Fields of a PDU

PDUs contain the following data elements:

Protocol Version: An eight-bit unsigned integer, currently 0, 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: 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, either the router or the cache finds the value of the session identifier is not the same as the other's, they MUST completely drop the session and the router MUST flush all data learned from that cache.

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 that the router does not hold, or may tell the router to add a record that 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 identifier or a clock-based scheme for generating session identifiers should avoid the risk of session identifier 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 that has as its value the count of the bytes in the entire PDU, including the eight bytes of header that end with the length field.

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

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: ASN allowed to announce this prefix, a 32-bit unsigned integer.

Zero: Fields shown as zero or reserved MUST be zero. The value of such a field MUST 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.

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

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

5.3. Serial Query

Serial Query: The router sends Serial Query to ask the cache for all payload PDUs that have Serial Numbers higher than the Serial Number 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. If there have been no changes since the router last queried, the cache sends an End Of Data PDU.

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      |
   |    0     |    1     |                     |
   +-------------------------------------------+
   |                                           |
   |                 Length=12                 |
   |                                           |
   +-------------------------------------------+
   |                                           |
   |               Serial Number               |
   |                                           |
   `-------------------------------------------'
```

5.4. Reset Query

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).

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

5.5. Cache Response

Cache Response: The cache responds with zero or more payload PDUs. When replying to a Serial Query request (Section 5.3), the cache sends the set of all data records it has with Serial Numbers greater than that sent by the client router. When replying to a Reset Query, 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      |
|    0     |    3     |                     |
+-------------------------------------------+
|                                           |
|                 Length=8                  |
|                                           |
`-------------------------------------------'
```

5.6. IPv4 Prefix

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |   reserved = zero   |
|    0     |    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   |    reserved = zero  |
|    0     |    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

End of Data: The cache tells the router it has no more data for the request.

The Session ID MUST be the same as that of the corresponding Cache Response that began the, possibly null, sequence of data PDUs.

```
0          8          16         24        31
.-------------------------------------------.
| Protocol |   PDU    |                     |
| Version  |   Type   |     Session ID      |
|    0     |    7     |                     |
+-------------------------------------------+
|                                           |
|                 Length=12                 |
|                                           |
+-------------------------------------------+
|                                           |
|               Serial Number               |
|                                           |
`-------------------------------------------'
```

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   |    reserved = zero  |
|    0     |    8     |                     |
+-------------------------------------------+
|                                           |
|                 Length=8                  |
|                                           |
`-------------------------------------------'
```

5.10. 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 10.

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      |
|    0     |    10     |                     |
+-------------------------------------------+
|                                           |
|                  Length                   |
|                                           |
+-------------------------------------------+
|                                           |
|       Length of Encapsulated PDU          |
|                                           |
+-------------------------------------------+
|                                           |
~           Copy of Erroneous PDU           ~
|                                           |
+-------------------------------------------+
|                                           |
|           Length of Error Text            |
|                                           |
+-------------------------------------------+
|                                           |
|              Arbitrary Text               |
|                    of                     |
~          Error Diagnostic Message         ~
|                                           |
`-------------------------------------------'
```

6. Protocol Sequences

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

6.1. Start or Restart

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

When a transport session 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.

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 no less frequently than once an hour. This also acts as a keep-alive at the application layer.

As the cache MAY not keep updates for little more than one hour, the router MUST have a polling interval of no greater than once an hour.

6.2. Typical Exchange

```
   Cache                         Router
     ~                             ~
     | -------- Notify ----------> |  (optional)
     |                             |
     | <----- Serial Query ------- | R requests data
     |                             |
     | ----- Cache Response -----> | C confirms request
     | ------- IPvX Prefix ------> | C sends zero or more
     | ------- IPvX Prefix ------> |   IPv4 and IPv6 Prefix
     | ------- IPvX Prefix ------> |   Payload 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 no less frequently than once an hour.

6.3. 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
     | ------- IPvX Prefix ------> | C sends zero or more
     | ------- IPvX Prefix ------> |   IPv4 and IPv6 Prefix
     | ------- IPvX Prefix ------> |   Payload 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.

6.4. 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.

7. 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 that 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 12. 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 a the normal SSH port. For an example, see Section 7.1.

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 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 12.

7.1. 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.

7.2. 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 that use TLS. Certification authorities that 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 that 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.

7.3. 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].

7.4. 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. MAC (Message Authentication Code) 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.

8. 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.

A client SHOULD delete the data from a cache when it has been unable to refresh from that cache for a configurable timer value. The default for that value is twice the polling period for that 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.

9. 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 that 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 that 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.

10. Error Codes

This section contains a preliminary list of error codes. The authors expect additions to the list this section during development of the initial implementations. There is an IANA registry where valid error codes are listed; see Section 12. Errors that 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 other error codes.

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 record for the {Prefix, Len, Max-Len, ASN} tuple does not exist in the receiver's database.

7: Duplicate Announcement Received (fatal): The received PDU has an identical {Prefix, Len, Max-Len, ASN} tuple as a PDU that is still active in the router.

11. 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 that may not be obvious in other sections.

Cache Validation: In order for a collection of caches as described in Section 9 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 session 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.

But, 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 that cannot provide the necessary authentication and integrity (see Section 7) must rely on network design and operational controls to provide protection against spoofing/corruption attacks. As pointed out in Section 7, TCP-AO is the long-term plan. Protocols that 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.

12. IANA Considerations

IANA has assigned 'well-known' TCP Port Numbers to the RPKI-Router Protocol for the following, see Section 7:

```
rpki-rtr
rpki-rtr-tls
```

IANA has created a registry for tuples of Protocol Version / PDU Type, each of which may range from 0 to 255. The name of the registry is "rpki-rtr-pdu". The policy for adding to the registry is RFC Required per [RFC5226], either Standards Track or Experimental. The initial entries are as follows:

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

IANA has created a registry for Error Codes 0 to 255. The name of the registry is "rpki-rtr-error". The policy for adding to the registry is Expert Review per [RFC5226], where the responsible IESG Area Director should appoint the Expert Reviewer. The initial entries should be as follows:

Error Code	Description
0	Corrupt Data
1	Internal Error
2	No Data Available
3	Invalid Request
4	Unsupported Protocol Version
5	Unsupported PDU Type
6	Withdrawal of Unknown Record
7	Duplicate Announcement Received
255	Reserved

IANA has added an SSH Connection Protocol Subsystem Name, as defined in [RFC4250], of 'rpki-rtr'.

13. Acknowledgments

The authors wish to thank Steve Bellovin, Rex Fernando, Paul Hoffman, Russ Housley, Pradosh Mohapatra, Keyur Patel, Sandy Murphy, Robert Raszuk, John Scudder, Ruediger Volk, and David Ward. Particular thanks go to Hannes Gredler for showing us the dangers of unnecessary fields.

14. References

14.1. Normative References

[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", BCP 14, RFC 2119, 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.

[RFC4250] Lehtinen, S. and C. Lonvick, "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.

[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", BCP 26, RFC 5226, 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.

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

14.2. 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.

[RTR-IMPL] Bush, R., Austein, R., Patel, K., Gredler, H., and M. Waehlisch, "RPKI Router Implementation Report", Work in Progress, January 2012.

Authors' Addresses

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

EMail: randy@psg.com

Rob Austein
Dragon Research Labs

EMail: sra@hactrn.net