

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 11, 2012

R. Bush
Internet Initiative Japan
R. Austein
ISC
July 10, 2011

The RPKI/Router Protocol
draft-ietf-sidr-rpki-rtr-14

Abstract

In order to formally validate the origin ASs of BGP announcements, routers need a simple but reliable mechanism to receive RPKI [[I-D.ietf-sidr-arch](#)] prefix origin data from a trusted cache. This document describes a protocol to deliver validated prefix origin data to routers.

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 [[RFC2119](#)].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 11, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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

1. Introduction

In order to formally validate the origin ASs of BGP announcements, routers need a simple but reliable mechanism to receive RPKI [[I-D.ietf-sidr-arch](#)] formally validated prefix origin data from a trusted cache. This document describes a protocol to deliver validated prefix origin data to routers.

[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 to support new PDUs with new semantics when and as needed, as indicated by deployment experience. PDUs are versioned should deployment experience call for change.

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, RIRs, NIRs, and ISPs, see [[I-D.ietf-sidr-repos-struct](#)].

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 update software such as rcynic 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 monotonically increasing ordinal which wraps from $2^{32}-1$ to 0. It denotes the logical version of a cache. A cache increments the value by one 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 marked 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 serial number arithmetic.

Nonce: When a cache server is started, it generates a nonce 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.

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 [[I-D.ietf-sidr-repos-struct](#)].

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 which 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 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 which are time dependent, servers' clocks MUST be correct to a tolerance of approximately an hour.

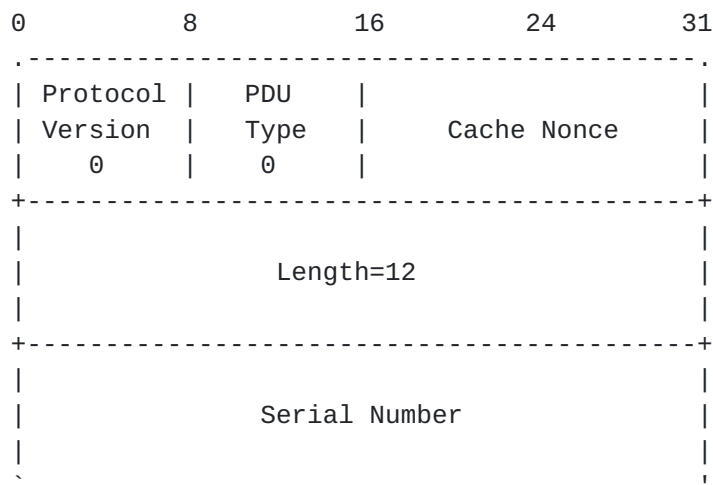
5. Protocol Data Units (PDUs)

The exchanges between the cache and the router are sequences of exchanges of the following PDUs according to the rules described in [Section 6](#).

5.1. Serial Notify

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

The Cache Nonce reassures the router that the serial numbers are commensurate, i.e. the cache session has not been changed.

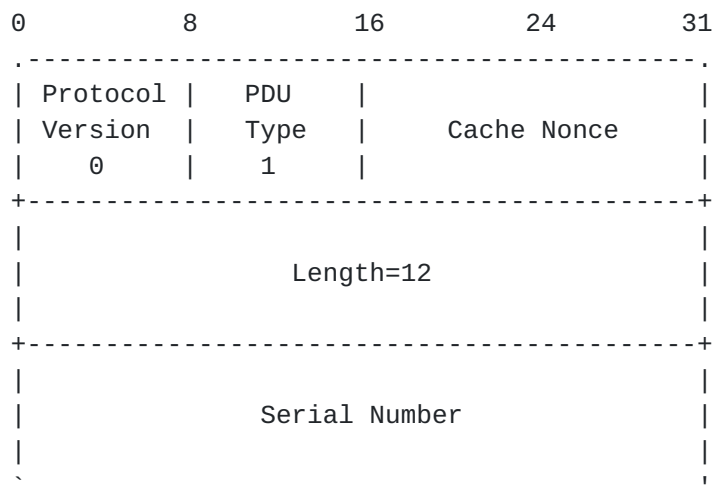


5.2. Serial Query

Serial Query: The router sends Serial Query to ask the cache for all payload PDUs which have serial numbers higher than the serial number in the Serial Query.

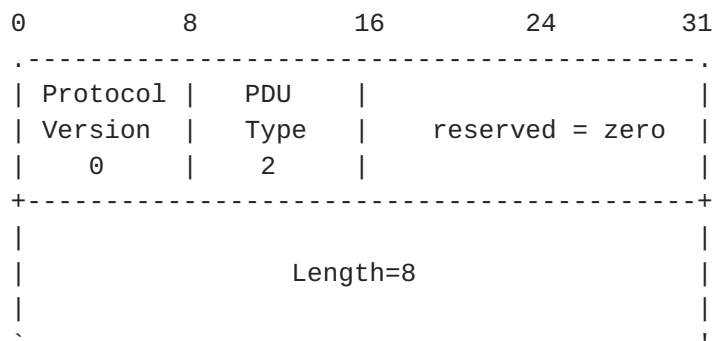
The cache replies to this query with a Cache Response PDU ([Section 5.4](#)) if the cache has a record of the changes since the serial number specified by the router. If there have been no changes since the router last queried, the cache responds with an End Of Data PDU. If the cache does not have the 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 5.8](#)).

The Cache Nonce 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.



5.3. Reset Query

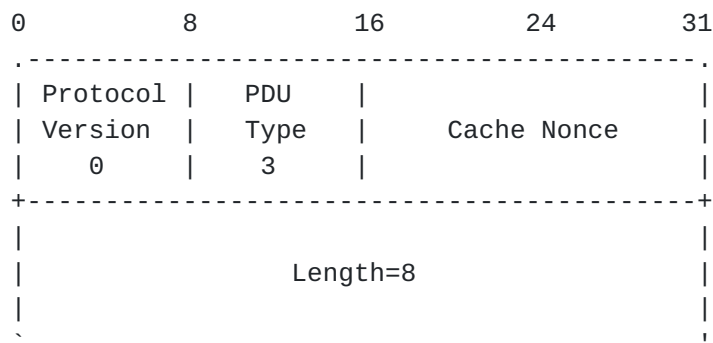
Reset Query: The router tells the cache that it wants to receive the total active, current, non-withdrawn, database. The cache responds with a Cache Response PDU ([Section 5.4](#)).



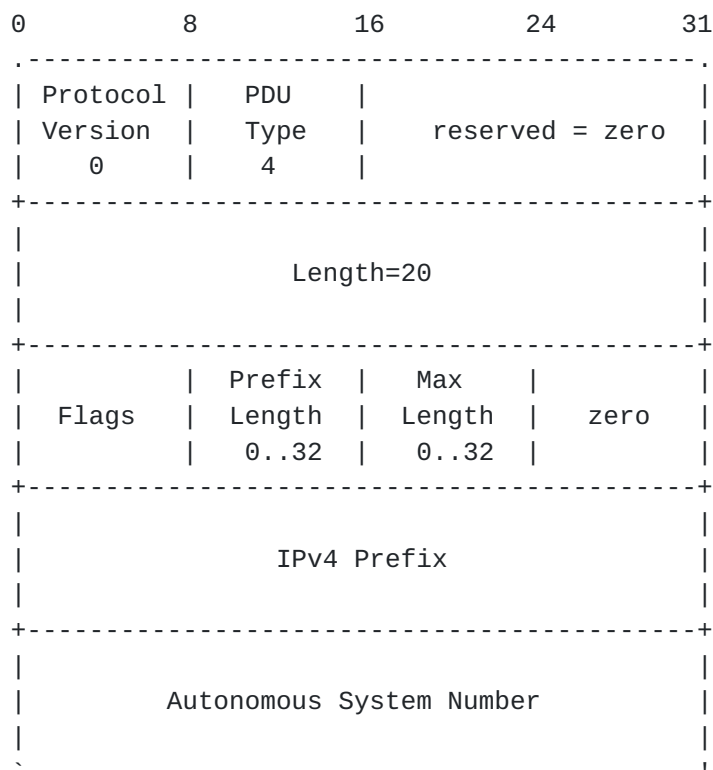
5.4. Cache Response

Cache Response: The cache responds with zero or more payload PDUs. When replying to a Serial Query request ([Section 5.2](#)), 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 Cache Nonce tells the router the instance of the cache session for future confirmation. In response to a Serial Query, the Cache Nonce being the same reassures the router that the serial numbers are commensurate, i.e. the cache session has not changed.



5.5. IPv4 Prefix



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

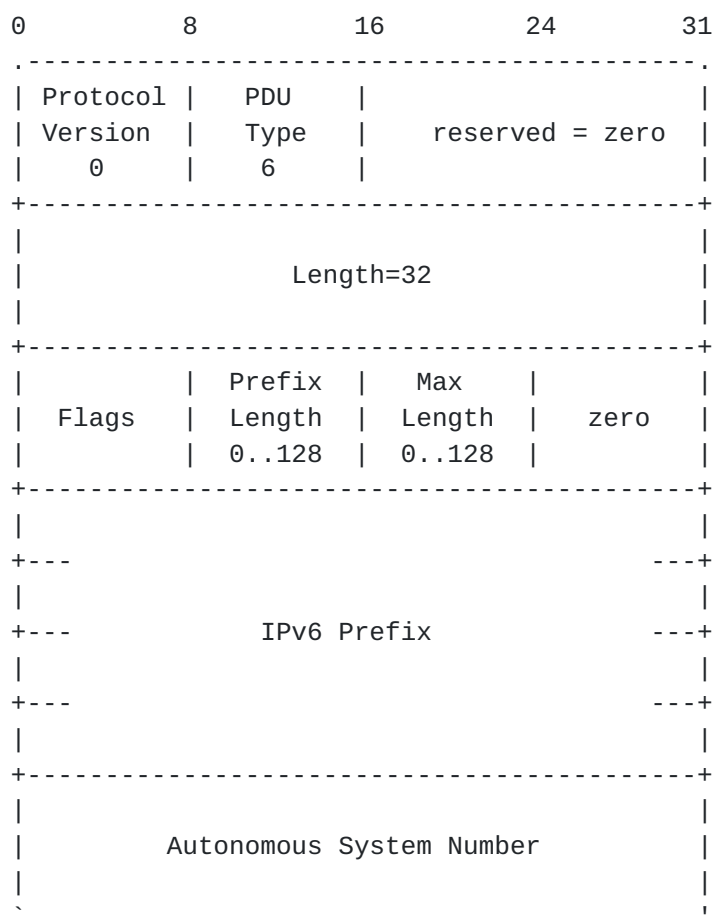
In the RPKI, nothing prevents a signing certificate from issuing two identical ROAs, and nothing prohibits the existence of two identical route: or route6: objects in the IRR. In this case there would be no semantic difference between the objects, merely a process redundancy.

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.

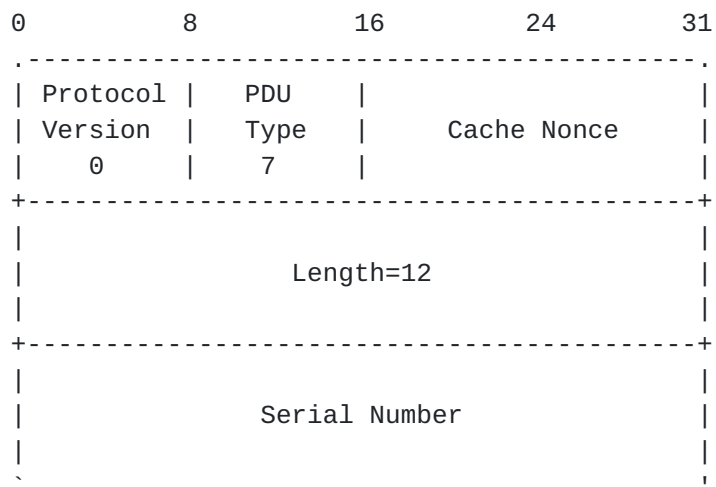
5.6. IPv6 Prefix



5.7. End of Data

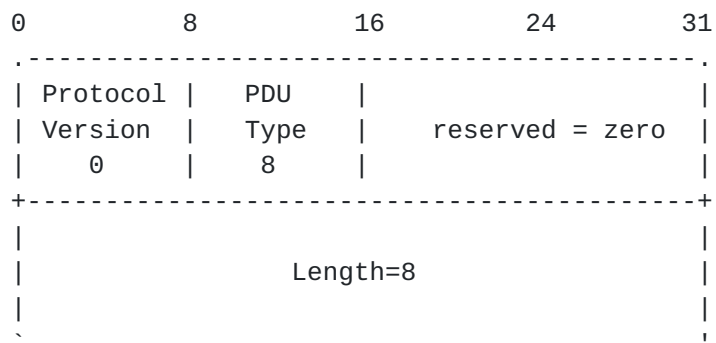
End of Data: Cache tells router it has no more data for the request.

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



5.8. 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.



5.9. Error Report

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

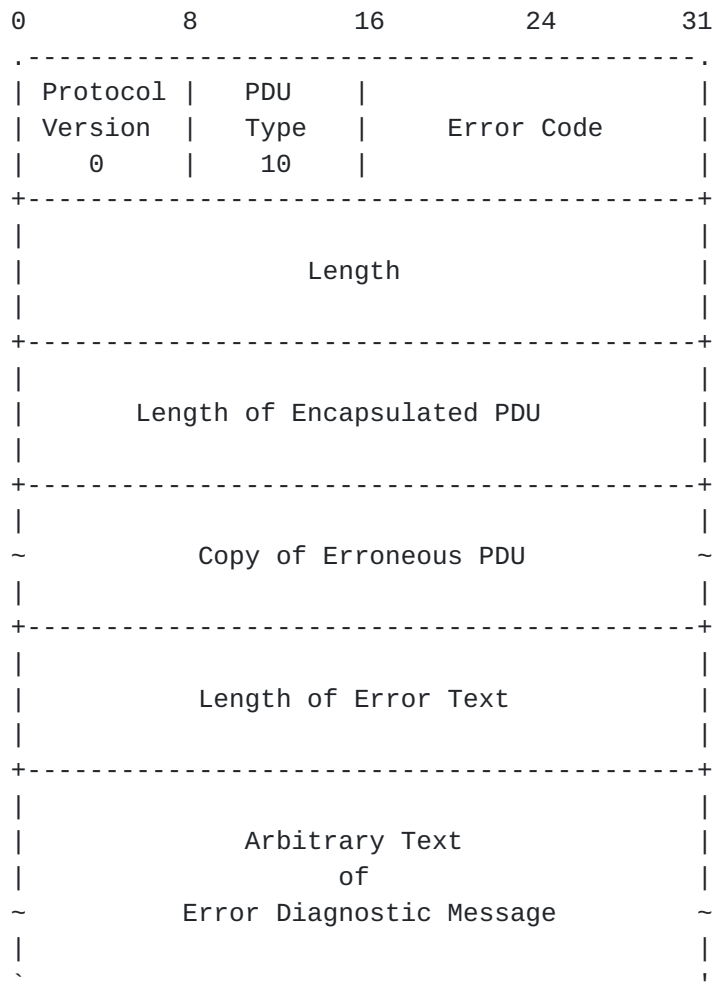
The Error Code is described in [Section 10](#).

If the error is not associated with any particular PDU, 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 the error is associated with a PDU of excessive, or possibly corrupt, length, the Erroneous PDU field MAY be truncated.

The diagnostic text is optional, if not present the Length of Error Text field SHOULD be zero. If error text is present, it SHOULD be a string in US-ASCII, for maximum portability; if non-US-ASCII characters are absolutely required, the error text MUST use UTF-8 encoding.



[5.10.](#) Fields of a PDU

PDUs contain the following data elements:

Protocol Version: An ordinal, currently 0, denoting the version of this protocol.

PDU Type: An ordinal, denoting the type of the PDU, e.g. IPv4 Prefix, etc.

Serial Number: The serial number of the RPKI Cache when this ROA was received from the cache's up-stream cache server or gathered from the global RPKI. A cache increments its serial number when completing an rigorously validated update from a parent cache, for example via rcynic. See [[RFC1982](#)] on DNS Serial Number Arithmetic for too much detail on serial number arithmetic.

Cache Nonce: When a cache server is started, it generates a nonce 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 nonces they hold disagree, they **MUST** completely drop the session and the router **MUST** flush all data learned from that cache.

The nonce might be a pseudo-random, a monotonically increasing value if the cache has reliable storage, etc. An implementation which uses a fine granularity of time for the Serial Number might never change the Cache Nonce.

Length: A 32 bit ordinal 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, 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 Prefix PDU with the exact same Prefix, Length, Max-Len, and ASN.

Prefix Length: An ordinal denoting the shortest prefix allowed for the prefix.

Max Length: An ordinal 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 ordinal.

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

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 Cache Nonce 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 cannot supply update
~	~

Cache	Router
~	~
<----- Reset Query -----	R requests data
---- Error Report PDU ---->	C cannot supply update
~	~

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 Protocol Data Units (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 are authenticated to each other. Integrity protection for payloads is also desirable to protect against monkey in the middle attacks. Unfortunately, there is no protocol to do so on all currently used platforms. Therefore, as of this document, there is no mandatory to implement transport which provides authentication and integrity protection.

It is expected that, when 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, to be assigned, see [Section 12](#). Operators SHOULD use procedural means, ACLs, ... to reduce the exposure to authentication issues.

If available to the operator, caches and routers SHOULD 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 SSH transport [[RFC4252](#)] using 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.

Caches and routers MAY use IPsec transport [[RFC4301](#)] using the RPKI-Rtr port.

Caches and routers MAY use TLS transport [[RFC5246](#)] using using a port, RPKI-Rtr TLS, to be assigned, see [Section 12](#).

[7.1](#). SSH Transport

To run over SSH, the client router first establishes an SSH transport connection using the SSH 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 [RFC 4252](#) [[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 DSA authentication, and SHOULD accept 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 use client-side certificates for authentication. While in principle any type of certificate and certificate authority may be used, in general cache operators will generally wish to create their own small-scale CA and issue certificates to each authorized router. This simplifies credential roll-over; any unrevoked, unexpired certificate from the proper CA may be used. If such certificates are used, the CN field [[RFC5280](#)] MUST be used to denote the router's identity.

Clients SHOULD verify the cache's certificate as well, to avoid monkey-in-the-middle attacks.

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 lengths of at least 96 bits MUST be supported, per [section 5.3 of \[RFC5925\]](#).

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

8. Router-Cache Set-Up

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 in order of preference. 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 ordinal 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 can not 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 [[I-D.ietf-sidr-pfx-validate](#)].

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 synch 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 multi-homed 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 one or more up-streams' caches. 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 multi-homed 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 up-stream ISP caches.

ISP Backbone: A large ISP would likely have one or more redundant caches in each major 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 publication infrastructure.

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 maintaining 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.

10. Error Codes

This section contains a preliminary list of error codes. The authors expect additions to this section during development of the initial implementations. 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 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/PrefixLength/MaxLength triple 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 which 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 which 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. I.e. 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 can not 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 can not 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 MUST be verified and authenticated by the router client, and vice versa, before any data are exchanged.

12. IANA Considerations

This document requests the IANA to assign TCP Port Numbers to the RPKI-Router Protocol for the following, see [Section 7](#):

RPKI-Rtr
RPKI-Rtr TLS

This document requests the IANA to create a registry for tuples of Protocol Version / PDU Type, each of which may range from 0 to 255. The name of the registry should be rpki-rtr-pdu. The policy for adding to the registry is RFC Required per [\[RFC5226\]](#), either standards track or experimental. The initial entries should be as follows:

Protocol Version	PDU Type
-----	-----
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

This document requests the IANA to create a registry for Error Codes 0 to 255. The name of the registry should be 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:

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

This document requests the IANA to add an SSH Connection Protocol Subsystem Name, as defined in [\[RFC4250\]](#), of 'rpki-rtr'.

13. Acknowledgments

The authors wish to thank Steve Bellovin, Rex Fernando, 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

- [I-D.ietf-sidr-pfx-validate] Mohapatra, P., Scudder, J., Ward, D., Bush, R., and R. Austein, "BGP Prefix Origin Validation", [draft-ietf-sidr-pfx-validate-01](#) (work in progress), February 2011.
- [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.
- [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.
- [RFC4808] Bellovin, S., "Key Change Strategies for TCP-MD5", [RFC 4808](#), March 2007.
- [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.

[14.2.](#) Informative References

[I-D.ietf-sidr-arch]
Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", [draft-ietf-sidr-arch-13](#) (work in progress), May 2011.

[I-D.ietf-sidr-repos-struct]
Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", [draft-ietf-sidr-repos-struct-08](#) (work in progress), June 2011.

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

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

Authors' Addresses

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

Phone: +1 206 780 0431 x1
Email: randy@psg.com

Rob Austein
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
USA

Email: sra@isc.org