Sizing Estimates for a Fully Deployed RPKI

[Verisign Labs Technical Report #1120005 version 1]

Eric Osterweil
Verisign Labs
eosterweil@verisign.com

Danny McPherson
Verisign Labs
dmcpherson@verisign.com

1 Introduction

As the Resource Public Key Infrastructure (RPKI) [4] has begun its early deployment trials, an estimate of how large a global RPKI deployment would have to be in order to certify today's Internet has become increasingly relevant. Communities of engineers, operators, and researchers have begun looking at this sizing question, and other questions as well. In this document, we present a "back-of-the-envelope" style evaluation of what a global RPKI might look like, and how it might scale. This work is intended to be a first stab at addressing this, and the authors welcome feedback, corrections, comments, etc. With that caveat, we begin from the perspective of a thought exercise, and we pose the following questions:

- What would be the size of a fully deployed RPKI, in today's Internet?
- What is the complexity of finding/learning/gathering a fully deployed RPKI?
- How long might such a gathering process actually take?

From these questions, we consider the size to be the sum of the RPKI's various objects contained across the global set of SIA repositories. These object types (described in [4]) are: CA certificates, End Entity (EE) certificates, Route Origin Authorizations (ROAs), Manifests, Certificate Revocation Lists (CRLs), and ghostbuster records. We use this characterization to claim that since each certificate is maintained in an SIA repository, the caching system must download all objects from all SIAs before validation computation can begin. So, today we have 5 RIRs running repositories. It's not clear if that will always be the case (the so-called "hosted" approach), or if one day each AS will run its own SIA. So our analysis considers a range of cases where there are: 5 (the current set of RIRs), 10, 100, 1000, 10000, 42000 (the number of ASNs today) repositories.

What we find below is that with a lower-bound estimate of about 1.4 million objects, a globally deployed RPKI would likely take about a month to ensure RPs get changes into caches. While the methodology and measurements in this document are meant to be clear and concise, they are simply submitted as candidates of what a globally deployed RPKI *might* look like. Corrections are welcome, and alternate methodologies and measurements are very much encouraged. This work is intended to be a living document, which should evolve as operational experience is gained with the RPKI.

2 Methodology

In our candidate methodology, we enumerate the different types of objects according to the naming scheme in Table 1. Next, we consider the size of what's deployed in today's Internet: 42,000 ASNs, 431,000 Prefixes, and we estimate an average of 10 routers in every AS (so 420,000 routers). We consider the router estimate to be a lower bound because some ASes (such as Verisign) have hundreds of BGP speaking routers, and some larger ISPs have thousands of BGP speaking routers. With the above, Table 2 maps our object types to size estimates based on the size of the Internet today.

We note, in the above, that the number of objects in E^P and O^{ROA} relates to the allocation hierarchy. In many ways, this is a superset of the prefixes in the RIB because:

- Allocations can be suballocated multiple times, each needing a separate E^P and O^{ROA}, but only represented once in the RIB.

The set of all CA certs for ASNs is	C^{AS}
The set of all EE certs for ASes is	E^{AS}
The set of all Ghostbusters records	O^{G}
The set of all EE certs for prefixes is	E^{P}
The set of all ROAs is	O^{ROA}
The set of all Manifests is	O^{M}
The set of all CRLs is	O^{CRL}
The set of all Router EE certs is	E^{Rtr}

Table 1: Object sets

C^{AS} = number of ASes: #AS
E^{AS} = number of ASes: #AS
O^{G} = number of SIAs
E^{P} = number of prefixes (we explain below) : #Prefixes
O^{ROA} = number of prefixes : #Prefixes
O^{M} = number of SIAs
O^{CRL} = number of SIAs
E^{Rtr} = 10×number of ASes : $10 \times$ #Prefixes

Table 2: Approximation of object set sizes

- In addition, in regards to multihoming, we need one ROA and one EE certificate for each feasible origin for a prefix, yet we will only see on entry in a RIB.
- The sets E^{P} and O^{ROA} do not account for multiple publication points (redundant repositories).

As a result, our guess is that estimating the size of the allocation set by the RIB is a large underestimate. Without a tool such as `http://eyeP.cs.ucla.edu/`, it's not easy to measure a more precise number.

We'll say that the set of objects in the global RPKI is:

$$O^{Total} = \{C^{AS}, E^{AS}, O^{G}, E^{P}, O^{ROA}, O^{M}, O^{CRL}, E^{Rtr}\}$$

In order to just accommodate the number of active routes in the BGP routing system today, the overall size of this would be:

$$|O^{Total}| = |C^{AS}| + |E^{AS}| + |O^{G}| + |E^{P}| + |O^{ROA}| + |O^{M}| + |O^{CRL}| + |E^{Rtr}|$$

$$= \#AS + \#AS + \#SIAs + \#Prefixes + \#Prefixes + \#SIAs + \#SIAs + 10 \times \#AS$$

$$= 12 \times \#AS + 2 \times \#Prefixes + 3 \times \#SIAs$$

$$= 1,366,000 + 3 \times \#SIAs$$

3 Evaluation

While 1.4 million objects could be considered to be a large corpus (especially as a lower bound estimate), it will certainly grow whenever any keys are being rolled over and changed, it will necessarily almost double during cryptographic algorithm rollovers, and it doesn't include those objects needed for the actual allocation hierarchy.So, again, we submit that this is a lower bound.

Next, to estimate the amount of *time* it takes for caches to actually gather a fully deployed RPKI (at today's Internet's scale), let's examine the algorithm. We consider the set of interconnected SIA repositories to be a directed graph, in which the set of vertices V are discovered by gathering (i.e. an SIA repository contains certificates who have SIA and AIA URI pointers to potentially external repositories), and to traverse all of it, we use a depth-first search approach. This approach's complexity is well known to be: $O(|V| + |E|)$, where V is the set of vertices, and E is the set of edges.

Repository	Avg sync time	Avg num objects	Avg seconds / object
"An ISP"	3.14	3	1.05
"RIR"	945.96	234	4.04
ARIN Pilot	83.42	131.46	0.635
APNIC	1006.95	1405	0.717
RIPE	2269.75	3800	0.597
RIPE	2065.41	3785	0.546
RIPE	2004.8	4066.32	0.493
"rpki101.fra2.de.euro-transit.net"	4.61	5	0.922
APNIC	944.56	1469.14	0.643
AfriNIC	75.45	82	0.920
Aggregate	**9404.05**	**14980.92**	**0.628**

Table 3: Repo sync stats: Average: 0.628 seconds/object

Number of Repositories	Number of Objects	Time to Gather	Days to Gather
5	1366015	857857.42	9.92890532407407
10	1366030	857866.84	9.92901435185185
100	1366300	858036.4	9.93097685185185
1000	1369000	859732	9.95060185185185
10000	1396000	876688	10.1468518518519
42000	1492000	936976	10.8446296296296

Table 4: Gathering times as the number of repositories grows.

In the RPKI, V is the set of SIA repositories, and E is the set of unique pairs of vertices in V that come from certificates in each SIA repository and which point to external SIA or AIA URIs (for example, certificates chaining to their CAs). From this understanding, we need to estimate how long it would actually take to follow this algorithm (with the above set of objects) in order for a cache to discover all of the SIA repositories in the Internet (on each running). To map this complexity to actual deployment numbers, we consider the presentation [1]. In this presentation, we see 10 different performance graphs for RPKI repositories. These graphs each explicitly show the average number of objects and the average sync time. Table 3 lists these values and the computed aggregate statistics across all of them.

With these numbers, Table 4 lists estimates for how long it would take to download from a varying number of SIA repositories containing objects for the Internet, if RPKI were *fully* deployed, today. We note that in our evaluation, we should have modeled a latency factor for traversing edges in E (for example, DNS takes time, timeouts, etc.). However, to continue to focus on a lower bound, we ignored the E cost in our calculations.

Figure 1 illustrates that as the number of SIA repositories grows, so too does the sync time for the system. Due to the nature of caching systems, one cannot be sure caches have fetched new data until waiting roughly two times the the cycle time (in some systems this is a TTL). Thus, it could be on the order of a month ($2 \times 10.84 = 21.68$ days) before it becomes a reliable assumption that changes to a repository are fully fetched (note: we still claim that this is an extreme lower bound). In addition to these numbers, [1] appears to analyze the caching behavior of a large ISP with an internal hierarchical caching system. This would mean that after a master cache (for a single ISP) has gathered all of the objects from across the global RPKI, it must begin replicating this to other internal "tiers" of caches (inside the ISP). Therefore, for such ISPs, we must also add the numbers from [1] to this (as internal caches must replicate). In [1], the replication time was between 70 and 80 *minutes*. However, while the authors describe their internal caching testbed as being, "fairly large scale," their simulated RPKI deployment only had 14,000 objects (vs. our estimated 1.5 million, in Table 4). As a result, one might worry that with strictly linear scaling, this replication time for internal caches could become $70 \times \frac{1,492,000}{14,000} = 7460$ more minutes (or 5.2 more days). This could make estmates inflate to $2 \times (10.84 + 5.2) = \textit{32.08 days}$. Admittedly, each ISP may choose different caching strategies, but this hierarchical model provides a lot of operational robustness, though it complicates convergence time significantly. One final note, the above analysis does not add in time for potential system failures, latency as load increases in SIA systems, etc.

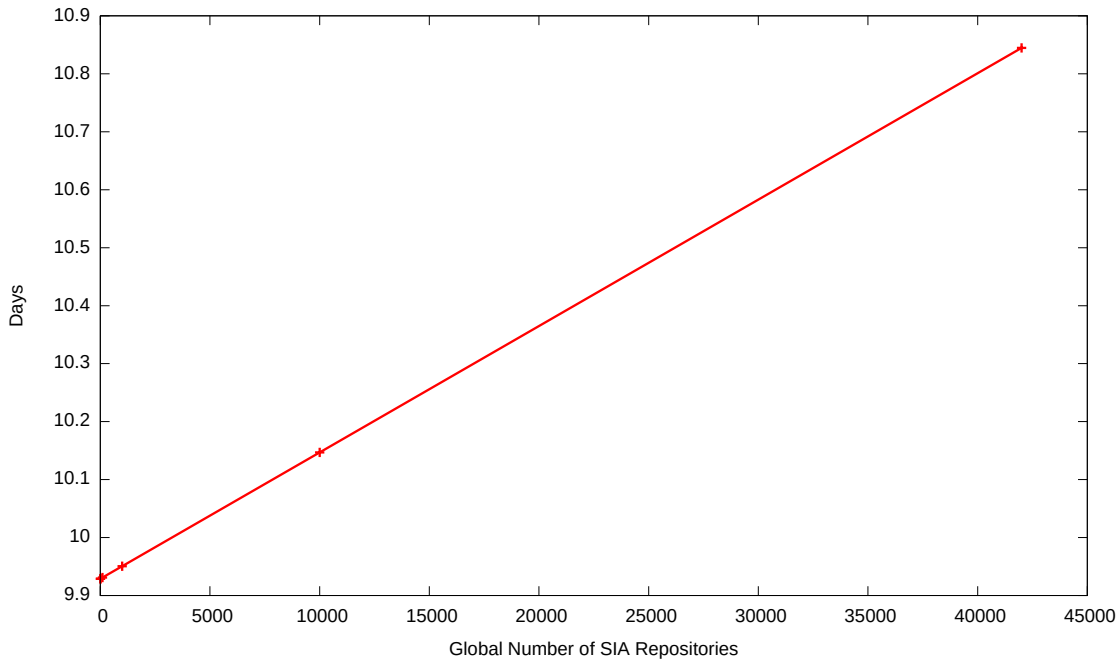

Figure 1: Between 10 and 11 *days* to sync. That means that operators have to count on 20 to 22 *days* after making a change that they want RPs to pick up (twice the polling period).

4 Conclusion

The calculations in this document, while just candidate representations of RPKI, project a very long lower-bound on the potential time needed for RP caches to gather all objects in the RPKI. Furthermore, when repositories in the RPKI undergo key rollovers, repositories *and caches* must store old and new objects for about a month (twice the polling period) before they can safely expunge them. This, in turn, will directly inflate the above numbers, and cause longer delays. For example, if an algorithm is rolled over [3], the system will have roughly twice as many objects as it normally does. This will double the sync time, and cause even *less* responsiveness to key rollovers. Also, if rsync scales sub-linearly, then it is likely that these numbers will grow. Again, if rsync periodically has to restart because objects being transferred have changed [2], this underestimate will get worse. In addition to this, active CRLs will *not* (by design) include expired data. Therefore, as soon as operators (following advice in existing drafts) begin to use expired data, Relying Parties (RPs) will necessarily be forced to keep track of old data everywhere (thus increasing the object count, again). Prior to this document, only publications such as [1] reported scaling measurement. While this presentation (and its relative incarnations) purport to be "fairly large scale" they appear to have estimated object counts roughly two orders of magnitude too small. We mention this simply to motivate the importance of having a living document such as this one. Finally, this analysis does not even take into account churn or systemic dependencies. Such issues should be discussed in similar (but separate) technical notes.

References

[1] RPKI Propagation. In *NANOG 56*, October 2012. `http://www.nanog.org/meetings/nanog56/abstracts.php?pt=MjAwMCZuYW5vZzU2&nm=nanog56`.

[2] Validation of RPKI objects using a local cache. In *IETF 85, SIDR WG*, November 2012. `http://www.ietf.org/proceedings/85/slides/slides-85-sidr-7.pdf`.

[3] R. Gagliano, S. Kent, and S. Turner. Algorithm agility procedure for rpki, draft-ietf-sidr-algorithm-agility.

[4] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480, Feb. 2012.