MaxLength Considered Harmful to the RPKI

Yossi Gilad yossigi@bu.edu
Omar Sagga osagga@bu.edu
Sharon Goldberg goldbe@cs.bu.edu

Boston University

ABSTRACT

User convenience and strong security are often at odds, and most security applications need to find some sort of balance between these two (often opposing) goals. The Resource Public Key Infrastructure (RPKI) [8], a security infrastructure built on top of interdomain routing, is not exempt from this issue. The RPKI uses the *maxLength* attribute to reduce the amount of information that must be explicitly recorded in its cryptographic objects. MaxLength also allows operators to easily reconfigure their networks without modifying their RPKI objects. However, we argue that the maxLength attribute strikes the wrong balance between security and user convenience. In particular, we argue that maxLength is commonly configured in a manner that either obviates the security benefits provided by the RPKI or causes legitimate routes to appear invalid, without providing performance improvements. Therefore, we argue that the maxLength attribute should be eliminated from the RPKI.

1. ROUTING WITH BGP & THE RPKI

The RPKI is designed to secure interdomain routing with the Border Gateway Protocol (BGP). It has recently been standardized [8] and is slowly being rolled out by the Regional Internet Registries (RIRs) and individual network operators [13]. In this section, we use a running example to review how the RPKI protects routing from prefix and subprefix hijacks. We then review the RPKI's maxLength attribute in Section 2.

ROAs. The RPKI uses Route Origin Authorizations (ROAs) to create a trusted mapping from an IP prefix to a set of autonomous systems (ASes) that are authorized to originate (*i.e.,* claim to be the destination for) this prefix. Each ROA contains a set S of IP prefixes, and an AS authorized originate all the IP prefixes in S; the ROA is cryptographically signed by the party that is authorized to allocate these IP prefixes. ROAs protect against some of the most devastating routing attacks; namely, prefix and subprefix hijacks where an AS originates ("hijacks") routes for IP prefixes that it is not authorized to originate, causing the traffic intended for those prefixes to be intercepted by the hijacker's network.

For our running example, we use IP prefix 168.122.0.0/16 which contains all IPv4 addresses from 168.122.0.0 to 168.122.255.255. This IP prefix is allocated to Boston University (BU) which also has AS 111. Therefore, the RPKI should contain a ROA with the IP prefix 168.122.0.0/16 mapped to AS 111. This ROA would be digitally signed by a key that is certified by ARIN (the American Registry for Internet Numbers), the registry that allocated this IP prefix to BU.

Originating routes. AS 111 would originate its prefix by sending this BGP announcement to its neighboring ASes:

"168.122.0.0/16: AS 111"

Because this BGP announcement *matches* the corresponding ROA, it will be considered *valid*. The neighboring ASes would then select this route, and propagate it on to their neighbors. For example, AS 3356 (Level3) is a neighbor of AS 111. After validating the BGP announcement, AS 3356 selects the route, and then announces its selection to its own neighbors with the BGP announcement

"168.122.0.0/16: AS 111, AS 3356"

This process of neighboring ASes validating the route against ROAs in the RPKI, selecting it, and then propagating it to their neighbors would continue until all ASes connected to the origin (directly or indirectly) have learned a route to 168.122.0.0/16. Traffic destined to IP addresses in 168.122.0.0/16 would then flow to AS 111.

Subprefix hijacks. Now suppose there is a subprefix hijack. The attacking AS m originates a BGP announcement for a subprefix of the target prefix, for example:

"168.122.0.0/24: AS m"

In the absence of the RPKI, the hijacker would intercept traffic for *all* addresses in 168.122.0.0/24—rather than flowing to AS 111, all this traffic would flow to AS m instead. This is because routers perform a longest-prefix match when deciding where to forward IP packets. A packet with destination IP 168.122.0.1 will flow to attacker AS m, rather than to AS 111, because 168.122.0.0/24 is a longer matching prefix than 168.122.0.0/16.

This attack is quite devastating, since the location of the attacker or the path he announces is irrelevant. All that matters is that the hijacker's route for the *sub*prefix propagates out into the global Internet. Longest-prefix-match routing ensures that the hijacked route is always preferred over the legitimate route. Several high-profile subprefix-hijacking incidents have exploited this behavior [14, 15].

ROAs should stop subprefix hijacks. How does the ROA for IP prefix 168.122.0.0/16 and AS 111 stop the subprefix hijack?

Any RPKI-validating router receiving the attacker's BGP announcement should notice that the announcement is *invalid* because (1) the ROA for 168.122.0.0/16 and AS 111

covers the attacker's announcement (since 168.122.0.0/24 is a subprefix of 168.122.0.0/16), and (2) there is no ROA *matching* the attacker's announcement (*i.e.,* ROA for AS m and IP prefix 168.122.0.0/24). If routers ignore invalid BGP announcements, the subprefix hijack will fail, and the attacker fails to intercept traffic destined to BU at AS 111.

2. ENTER MAXLENGTH

We continue with our example to understand maxLength.

De-aggregation. In addition to originating the BGP announcement for 168.122.0.0/16, AS 111 also originates an additional BGP announcement for its subprefix, specifically:

"168.122.225.0/24: AS 111"

Announcing subprefixes of a larger prefix from the same AS is known as *de-aggregation*, and sometimes used for traffic engineering. By announcing the /24 prefix to some neighbors and not others, AS 111 has some modicum of control over the routes other ASes use to reach 168.122.225.0/24.

(Interestingly, routing-security is sometimes also cited as a reason for de-aggregation. By announcing a /24 route, AS 111 ensures that no hijacker can launch a *sub*prefix hijack against the route, since BGP routes for prefixes longer than /24 are commonly discarded by routers [17].)

However, de-aggregation tends to bloat the size of routing tables across all routers on the Internet. As such, network operators often frown upon excessive de-aggregation, *e.g.,* announcing all 2^8 of the /24s constituting a /16 [17].

Now suppose the RPKI only had the ROA (168.122.0.0/16, AS 111), and AS 111 originated the BGP announcements

"168.122.225.0/24: AS 111"

The above BGP announcement is *invalid*, since it has a covering ROA (the ROA for 168.122.0.0/16, AS 111) but no matching ROA (i.e., a ROA for 168.122.225.0/24, AS 111). Routers would therefore drop the route as invalid, stymying AS 111's attempts to de-aggregate its prefix.

Using maxLength to deal with de-aggregation. The RPKI's maxLength attribute provides one solution to this problem. According to RFC 6482 [9]: "When present, the maxLength specifies the maximum length of the IP address prefix that the AS is authorized to advertise."

Thus, suppose we modified the ROA by adding a maxLength of 24. The resulting ROA would be

ROA:(168.122.0.0/16-24, AS 111)

where the notation "-24" means up to prefix-length 24. This ROA authorizes AS 111 to originate any subprefix of 168.122.0.0/16, up to length /24. For example, AS 111 could originate 168.122.225.0/24 as well as all of

168.122.0.0/17,
168.122.128.0/17,
168.122.0.0/18,
⋮
168.122.255.0/24

but not 168.122.0.0/25.

MaxLength thus gives AS 111 some additional flexibility; even if AS 111 does not know how it plans to de-aggregate its prefix 168.122.0.0/16 at the time it requested the ROA, by adding a maxLength of /24 AS 111 is certain that any de-aggregation it performs will not result in invalid routes.

The alternate solution: minimal ROAs. The other way around this problem is to update the ROA to include both prefixes. Since ROAs can support *sets* of IP prefixes, the ROA becomes

ROA:({168.122.0.0/16, 168.122.225.0/24}, AS 111)

This ROA is *minimal* because it includes exactly those two prefixes that are actually announced by AS 111. A ROA is *minimal* when it includes only those prefixes that the AS announces in BGP, and no other prefixes. RFC 6907 [11, Sec 3.2] recommends the use of minimal ROAs in some situations. If AS 111 wanted to issue a minimal ROA, AS 111 would need to know exactly how it plans to de-aggregate its prefixes at the time the ROA was issued.

3. FORGED ORIGIN SUBPREFIX HIJACK

The convenience of maxLength comes with a serious downside—namely, it has the potential to completely eliminate the security provided by the ROA.

The attack. To understand why, we return to our example. Suppose the RPKI had the ROA

ROA:(168.122.0.0/16-24, AS 111)

with maxLength 24, and that AS 111 originates the following two BGP announcements:

"168.122.0.0/16: AS 111"
"168.122.225.0/24: AS 111"

We now point out that a hijacker can intercept 100% of the traffic destined to *any* subprefix of 168.122.0.0/16 (up to length /24) except for those addresses in 168.122.225.0/24. For instance, to intercept all traffic for IP prefix 168.122.0.0/24, the hijacker just needs to send this BGP announcement:

"168.122.0.0/24: AS 111, AS m"

This is a *forged-origin subprefix hijack* [3, 6]. Why does it work?

1. The hijacker's BGP announcement falsely claims that m is a neighbor of AS 111. But the RPKI does not provide a means to validate this claim, so other ASes and routers have no way to know that this is false.

2. The hijacker's BGP announcement is valid according the RPKI, since the ROA (168.122.0.0/16-24, AS 111) authorizes BGP routes for 168.122.0.0/24 where AS 111 is the first hop (*i.e.,* origin AS).

3. AS 111 does not actually originate a route for 168.122.0.0/24. This means that the hijacker's route is the *only* route to 168.122.0.0/24.

4. Longest-prefix-match routing ensures that the hijacker's route to the subprefix 168.122.0.0/24 is always preferred over the legitimate route 168.122.0.0/16.

Thus, if the hijacker's route propagates through the Internet, the hijacker will intercept *all* traffic destined for IP addresses in 168.122.0.0/24.

Impact. This forged-origin subprefix hijack has exactly the same impact as a regular subprefix hijack. Any ROA

with maxLength m longer than the prefix p is vulnerable to a forged-origin subprefix hijack, unless *every* subprefix of p of length m is legitimately announced in BGP. This is unfortunate, since one might argue that the whole point of the RPKI is to stop subprefix hijacks [8, 10].

4. WHAT DO THE RFCS SAY?

RFC 7115 [3] mentions the forged-origin subprefix hijack:

> One advantage of minimal ROA length is that the forged origin attack does not work for subprefixes that are not covered by overly long max length. For example, if, instead of 10.0.0.0/16-24, one issues 10.0.0.0/16 and 10.0.42.0/24, a forged origin attack cannot succeed against 10.0.666.0/24. They must attack the whole /16, which is more likely to be noticed because of its size.

We agree with this view. We also point out that "attacking the whole /16" is not only "more likely to be noticed", but also much *less effective*. We return to our running example to explain why.

Suppose BU had issued the minimal ROA

> ROA:({168.122.0.0/16, 168.122.225.0/24}, AS 111)

Then, the forged origin subprefix hijack we described above would fail, since this minimal ROA ensures that the hijacker's route for 168.122.0.0/24 is invalid. Thus, to use the forged-origin trick, the hijacker m would need to "attack the whole /16", by sending the BGP announcement

> "168.122.0.0/16: AS 111, AS m"

Why is this different from the forged-origin subprefix hijack described above? The difference is that here, AS 111 actually does originate a BGP announcement for 168.122.0.0/16. Therefore, other ASes have two ways they can reach 168.122.0.0/16: either the announcement from the hijacker m *or* from the legitimate origin AS 111.

In [10], this attack was called the *one-hop hijack*, and it was found to have significantly less impact than a subprefix hijack. This is because the hijacker m can no longer exploit longest-prefix-match routing to intercept all of the traffic to hijacked prefix. Instead, the attacker is forced to advertise a longer and much less-appealing route to the prefix. As a result, the traffic splits between the hijacker m and the legitimate AS 111, where most ASes choose to route by the legitimate path (see [10]).

RFC 7115 [3] goes on to offer the following solid advice:

> Operators should be conservative in use of max length in ROAs. For example, if a prefix will have only a few sub-prefixes announced, multiple ROAs for the specific announcements should be used as opposed to one ROA with a long max length.

But we also should note, that "multiple ROAs" are not actually needed here, since ROAs support *sets* of IP prefixes.

RFC6907 similarly recommends issuing minimal ROAs [11, Sec. 3.2], but also seems to contradict itself by suggesting that a non-minimal ROA be used in the following situation:

> An organization (Org A with ASN 64496) has been allocated the prefix 10.1.0.0/16; it wishes to announce the aggregate and any or all more specific prefixes up to and including a maximum length of /20, but never any more specific than a /20

RFC6907 goes on to recommend that Org A issues

> ROA:(10.1.0.0/16-20, AS 64469)

with maxLength of 20. However, if AS 64496 does not announce *all* subprefixes of 10.1.0.0/16, it is vulnerable to forged-origin subprefix hijack.

5. BENEFITS OF MAXLENGTH?

In this section we quantify the benefits and problems rooted in the maxLength attribute. We argue that maxLength can provide only meager benefits. We also argue that the problems associated with maxLength are significant.

Our analysis is based on network measurements. Specifically, we downloaded all ROAs from the RPKI publication points and compared them against the routing entries in the BGP tables of all Route Views collectors [2]; our datasets aggregate ROAs and BGP advertisements from September 10-13, 2016.

Using maxLength almost always creates vulnerabilities. First, we observe that only 4129 (about 16.5%) of the prefixes in ROAs authorize a maxLength longer than the prefix length.[1] However, *almost all* of these prefixes (89%) are vulnerable to a forged-origin-subprefix hijack. To measure this phenomenon, we counted how many prefixes p with maxLength m longer than p do *not* have BGP announcements for every subprefix of p up to length m. Thus, we see that almost all users 'taking advantage' of the maxLength feature are unwittingly opening themselves up to forged-origin subprefix hijacks.

Benefit? Fewer prefixes included in ROAs. One might argue that an important benefit of the maxLength attribute is that it reduces the number of IP prefixes that need to be authorized by ROAs. We now argue that this is *not* a significant benefit.

To do this, we measure the number of additional prefixes that would need to be included in ROAs if (1) maxLength is eliminated and (2) only minimal ROAs were used (see Section 2). Specifically, we count the number of prefixes that are (a) announced in BGP and (b) are also a subprefix of a prefix that is authorized by a ROA in the RPKI.

We find 13K such prefixes. (Note that a total of 41K prefixes that are announced in BGP are covered by ROAs.) Minimal ROAs would need to cover exactly these 13K additional prefixes. We stress, however, that a single ROA supports authorizing a single AS to announce a *set* of prefixes. Thus, we could deal with these 13K additional prefixes *without* adding any additional ROAs (and associated cryptographic overhead) to the RPKI. Instead, we just convert each individual maxLength-using ROA to a minimal ROA (that does not use maxLength) which includes more prefixes to cover exactly those prefixes announced through BGP (see Section 2). Minimal ROAs also come with security benefits, since they prevent forged-origin subprefix hijacks.

Benefit? Reducing load on routers. One might instead argue that maxLength reduces the load on routers.

[1]Length refers to the length of an IP prefix, *e.g.,* 87.254.32.0/19 has length 19.

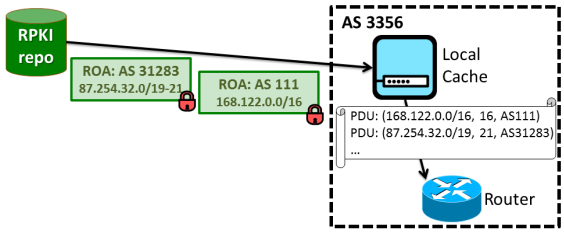

Figure 1: How routers obtain information from the RPKI.

Figure 1 shows how routers obtain information from the RPKI. Typically, each AS will have a trusted *local cache* (typically a general-purpose machine) that downloads the complete set of ROAs from all publicly-available RPKI repositories. The local cache then cryptographically validates these ROAs and creates a list of "Protocol Data Units" (PDUs). Each PDU is essentially a tuple of (IP prefix, maxLength, origin AS) [4, 12]. The local cache then sends these PDU lists to routers in its AS via the *rpki-rtr* protocol [1]. Finally, routers use the list of PDUs to determine if a given BGP announcement is valid [1, 4, 12].

The list of PDUs could become longer if the RPKI used minimal ROAs without the maxLength attribute. For instance, instead of having a single maxLength-using PDU for 87.254.32.0/19-20, the list would instead have three non-maxLength-using PDUs for 87.254.32.0/19, 87.254.32.0/20, 87.254.48.0/20. This could have a performance impact on routers. Our measurements above indicate that if (1) maxLength was eliminated from the RPKI and (2) minimal ROAs were used instead, today's routers would need to process 13K additional PDUs (about 42% more records).

It seems that maxLength does reduce the number of PDUs that must be processed by routers. But how does this would scale if RPKI was ubiquitously adopted? We now obtain a lower bound on the compression (in terms of reducing the number of PDUs processed by routers) provided by the maxLength attribute. To do this, we suppose that *every* IP prefix currently announced in BGP was covered by a *maximally-permissive* ROA. A maximally-permissive ROA authorizes each of its prefixes to its longest possible maxLength: namely, every IPv4 prefix has maxLength /32, while every IPv6 prefix has maxLength /128.[2]

Our BGP dataset has 683K advertised (IP prefixes, AS) pairs. If all of these IP prefixes were covered by maximally permissive ROAs, we were surprised to find that we still need these ROAs to include 641K prefixes! Thus, the maximum compression provided by the maxLength (in terms of reducing the number of PDUs processed by routers) is just 6.2%. Thus, even if we completely ignore the risk of forged-origin subprefix hijacks, the compression achieved by the maxLength in a full-deployment scenario is fairly small. This follows because most ASes do not send BGP announcements for subprefixes of their prefixes, and therefore cannot benefit from the compression provided by the RPKI's maxLength attribute.

[2]Maximally-permissive ROAs are terrible at preventing forged-origin subprefix hijacks (Section 3). A maximally-permissive ROA is almost always vulnerable to forged-origin subprefix hijacks, since we rarely see BGP announcements for IPv4 prefixes longer than /24, and IPv6 prefixes longer than /48. We consider maximally-permissive ROAs only because they obtain the maximum achievable compression from the maxLength attribute.

In Section 6, we present software that we built to preprocess the PDU list to reduce the number additional PDUs without introducing forged-origin subprefix hijacks. Our software achieves a compression rate of 6.1% in the same full-deployment scenario, without using maxLength and without allowing for forged-origin subprefix hijacks.

Problem: maxLength encourages misconfigurations. We speculate that the misuse of maxLength also causes many legitimate prefixes advertised through BGP, to appear *invalid*. Specifically, network operators often specify a maxLength that is too restrictive (*i.e.,* too short) in their ROAs. Specifying a maxLength shorter than the prefix announced by an AS in BGP causes the legitimate advertisement to appear as a subprefix hijack. Any AS that filters packets based on ROAs in the RPKI would discard the legitimate BGP advertisement as invalid, and thus could potentially cause a disconnection between the two ASes.

Using over-restrictive maxlength is a common error in current RPKI deployments. RPKI monitors and studies [7, 13] show that this is the cause for almost 60% of invalid prefixes. This corresponds to about 6% of all prefixes that are announced in BGP and covered by ROAs. Some of these many invalid BGP announcements might result from actual BGP hijacking, but most are likely due to errors.

Summary: maxLength provides few benefits. The motivation behind the maxLength attribute is simple: To reduce the number of prefixes specified in ROAs and thus also the number of PDUs that a router uses to filter invalid BGP announcements. However, we have just shown that maxLength can only provide very limited benefits, and in practice, it is often not used. Moreover, when maxLength is used, network operators often set it to the wrong value: either too restrictive, which may cause loss of legitimate traffic, or too permissive, which allows for forged-origin subprefix hijacks. We conclude that the problems caused by the maxLength field outweigh the benefits it can provide today and in the future, when the RPKI be ubiquitously adopted.

6. COMPRESSING MINIMAL ROAS

We now present a software application for improving the RPKI in absence of maxLength. Our software may also be deployed with today's RPKI. Our source code is public, it is available here [16].

Build your own maxLength. Recall that a router validating routes against the RPKI is given a list of PDUs, where each is a tuple of (IP prefix, length, maxLength, origin AS) [4, 12]. The list of PDUs is used to determine if a given BGP announcement is valid [1, 4, 12]. Thus, if we use minimal ROAs that do not use the maxLength attribute, then this list of PDUs could become longer, potentially harming performance.

Importantly, routers obtain this list of PDUs from a trusted *local cache*, as shown in Figure 1. Thus, to avoid pushing a longer list of PDUs to routers, we can instead have the local cache transform a list of PDUs that do not use the maxLength attribute, into a list of PDUs that do. We built software to do this. Our software runs on the local cache, thus avoiding any changes to BGP routers and conforming with RPKI's deployment architecture.

'Compressing' minimal ROAs. Conceptually, our software compresses a set of minimal ROAs that *do not use maxLength* to a set of minimal ROAs that *do use maxLength.*

As an example, consider the following minimal ROA which does not use maxLength:

> ROA: ({87.254.32.0/19, 87.254.32.0/20, 87.254.48.0/20, 87.254.32.0/21 }, AS 31283)

Our software would compress it to a minimal ROA that does use maxLength:

> ROA: ({87.254.32.0/19-20, 87.254.32.0/21 }, AS 31283)

Notice that this 'compressed' ROA is *minimal*, *i.e.,* contains exactly the same set of prefixes as its uncompressed version. Importantly, we do *not* compress the ROA to

> ROA: (87.254.32.0/19-21, AS 31283)

since this is not a minimal ROA. Indeed, this ROA is vulnerable to a forged-origin subprefix hijack since it validates prefixes that are *not* part of the original minimal ROA (*e.g.,* 87.254.40.0/21); see Section 3.

6.1 Software architecture

Today's RPKI Tools contain a utility program called `scan_roas` that the local cache uses to transform a set of ROAs that have been downloaded from the RPKI and cryptographically validated, into a list of PDUs, *aka*, (IP prefix, maxLength, origin AS) tuples [1]. Our utility is called `compress_roas` and acts as a drop-in alternative to `scan_roas`. `compress_roas` first calls `scan_roas` on a set of cryptographically-validated ROAs, and obtains a list of valid (IP prefix, maxLength, origin AS)-tuples. Then, we compress this set of tuples to another set of tuples that *do* use the maxLength attribute.

We envision using `compress_roas` in a future RPKI where the maxLength attribute is not used, so that each of the (IP prefix, maxLength, origin AS)-tuples output by `scan_roas` would just have maxLength equal to length of the IP prefix. This, however, is not a requirement for `compress_roas`. In fact, as we show below, `compress_roas` can reduce the number of (IP prefix, maxLength, origin AS)-tuples generated by *today's RPKI*.

Deployment. Our software is easy to integrate into the RPKI Relying Party Tools [1]. We just execute the following command on the local cache:

```
rpki-rtr cronjob --scan-roas compress-roas [roa-dir] [pdu-dir]
```

This sets up a cronjob that takes the cryptographically-validated ROAs in `[roa-dir]` and converts them into a list of (IP prefix, length, maxLength, origin AS)-tuples stored in `[pdu-dir]` that will be communicated to routers using the rpki-rtr protocol. The usual `scan_roas` utility is replaced with our `compress_roas` utility (which keeps the same interface).

Compression algorithm. Our algorithm takes in a list of (IP prefix, maxLength, AS)-tuples and compresses it using tries (*i.e.,* prefix trees). For each AS number in the list, we generate a trie for IPv4 and a trie for IPv6. The key to each trie is the string $\$prefix$ where \$ is a delimiter, and *prefix* is a binary representation of an IP prefix and its length. For instance, the ROA containing the IP prefix 8.0.0.0/8 for AS 3356 would be in AS 3365's IPv4 trie under the key \$00001000, while the IP prefix 8.0.0.0/9 for AS 3356 would be in AS 3365's IPv4 trie under the key \$000010000.

As shown in Figure 2, each node in a trie corresponds to some (AS, prefix, maxLength)-tuple that exists in a valid

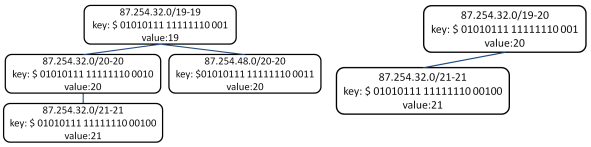

Figure 2: The IPv4 prefix trie for the minimal ROAs for AS 31283 without maxlength (left) and after compression with `compress_roas` (right). `compress_roas` reduces the number of output PDUs from four to two.

ROA. The value of the trie node corresponds to the maxlength specified the tuple. If the tuple came from a ROA that does not use the maxLength attribute (as we envision for all ROAs in the future RPKI) then this trie node has maxlength value identical to the length of the prefix.

For a node with key $\$k$, we refer to the nodes $\$k||0$ and $\$k||1$ as its left and right *direct children*. A node can therefore have at most two direct children.

To compress tuples, we iterate through the trie using a depth-first search (DFS). Just before we backtrack out of each trie `node`, we process it as follows:

```
if (node has both direct children):
  minChildVal = min(lChild.value, rChild.value)
  if (minChildVal > node.value):
      # Adjust father's maxlength to cover children
      node.value = minChildVal
  if (lChild.value <= node.value):
      # left child now covered by father
      delete trie[lChild]
  if (rChild.value <=  node.value):
      # right child now covered by father
      delete trie[rChild]
```

Thus, as the DFS backtracks through the trie, each trie node is assigned a new value (*i.e.,* a maxlength) if both its direct children exist. The assigned maxlength of the node is determined from the maxlength of its children. The child is then deleted if the child's maxlength does not exceed the parent's maxlength. Once the DFS completes, we output a set of (IP prefix, maxlength, AS)-tuples that correspond to the remaining trie nodes.

6.2 Performance

We evaluate our software in several ways, both in today's RPKI partial deployment status, and in a future scenario where RPKI is fully deployed. Table 1 summarizes our results and compares with our findings on the current RPKI from Section 5. Our implementation's website instructs how to reproduce our results [16].

Today's RPKI. First, we consider its impact on today's RPKI. In our RPKI dataset, there are 6K ROAs that comprise 28,828 distinct tuples of (IP prefix, maxlength, AS). Each tuple is inserted into one of the tries and compressed with `compress_roas`. The result was **27,125** tuples, for a compression rate of 5.90%. Thus, our software provides some benefits even on today's (maxLength-using) RPKI.

Today's RPKI, however, is vulnerable to forged-origin subprefix hijacks. What if today's RPKI was hardened against these hijacks, by converting every existing ROA into a minimal ROA that does not use maxLength? To convert each ROA in our RPKI dataset into a minimal ROA with no maxLength, we identify the IP prefixes that are made valid

scenario	# tuples	secure?
Today	28,828	X
Today (compressed)	**27,125**	X
Today, minimal ROAs, no maxLength	41,517	✓
Today, minimal ROAs, with maxLenth (compressed)	**38,839**	✓
Full deployment, minimal ROAs, no maxLength	683,671	✓
Full deployment, minimal ROAs, with maxLenth	**641,915**	✓
Full deployment, maximally-permissive ROAs	641,464	X

Table 1: Number of PDUs that processed by routers in the scenarios of Section 6. The third column has ✓ if scenario is robust to forged-origin subprefix hijack and X otherwise.

by that ROA and are announced in our BGP dataset. We then modify the ROA so that it contains only those IP prefixes. We still have the same number of ROAs, but now (instead of the status quo 28,828 tuples) we have a total of 41,517 (IP prefix, AS)-pairs. We use `compress_roas` to compress these 41,517 pairs to obtain 38,839 (IP prefix, maxLength, origin AS)-tuples, for a compression rate of 6.45%.

While this represents 35% more tuples than the status quo, the status quo (with non-minimal ROAs) is vulnerable to forged-origin subprefix hijacks, and the scenario we have just evaluated is not.

RPKI in full deployment. We consider a future full-deployment RPKI scenario, where every IP prefix announced in our BGP dataset is validated by a ROA. As discussed in Section 5, our BGP dataset has 683,671 (IP prefixes, AS) pairs. This corresponds exactly to the number of (IP prefix, AS) pairs that would be contained in ROAs if the RPKI used only minimal ROAs and no maxLength.

Compressing these with `compress_roas` results in 641,915 (IP prefix, length, maxLength)-tuples that are minimal, *i.e.,* are not vulnerable to a forged-origin subprefix hijacks. This is close to the 641,464 lower-bound of (prefix, maxLength, AS)-tuples needed to validate all BGP announcements, when prefixes are covered by maximally-permissive ROAs (see Seciton 5) and vulnerable to forged-origin subprefix hijacks.

Thus, if today's RPKI eliminated maxLength and started using minimal ROAs, we would see some increase (35%) in the number of tuples that must be processed by routers. In a future RPKI deployment, however, this increase becomes insignificant; in fact, using minimal ROAs along with our `compress_roas` software gives us almost the optimal number of PDUs. Thus provides more evidence in favor of removing the maxLength from the RPKI.

Computational overhead. We tested the `compress_roas` utility on an Intel i7-6700 machine. Compressing today's (partially-deployed) RPKI took 1.2 seconds and required 29MB of memory, while compressing the full-deployment scenario took 21 seconds and 280MB memory. Performance could also be improved by parallelizing across tries.

7. ELIMINATE MAXLENGTH

We see little value in continuing to support the maxLength attribute in ROAs. While only about 16% of prefixes in ROAs use the maxLength attribute, almost all of them are using it wrong. In fact, by using maxLength, many operators are unwittingly vulnerable to forged-origin subprefix hijacks that obviate the RPKI's security guarantees.

We therefore suggest eliminating the maxLength attribute. Instead, maxLength-using ROAs should be replaced with *minimal ROAs* that explicitly enumerate the set of IP prefixes that an AS is authorized to originate.

We presented our `compress_roas` software to limit the impact of this change on router performance. Also, today's ROAs already support having sets of IP prefixes. So, switching to minimal ROAs will not increase the number of cryptographic computations that must be performed for the RPKI.

To sum up, system complexity is a key challenge for the RPKI, especially because it can lead to security vulnerabilities (Section 3) and errors in deployment [5, 18]. Thus, by getting rid of the maxLength attribute, we can make the RPKI simpler, less error prone, and more secure.

Acknowledgements

We thank earlier collaborators on RPKI research for useful discussions: Avichai Cohen, Danny Cooper, Ethan Heilman, Amir Herzberg, Michael Schapira, Leonid Reyzin. This research was supported, in part, by NSF awards 1414119, 1350733 1012910, and a gift from Cisco.

8. REFERENCES

[1] RPKI Relying Party Tools. `https://github.com/dragonresearch/rpki.net`.

[2] University of Oregon Route Views Project. `http://www.routeviews.org/`.

[3] R. Bush. Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI). RFC 7115 (Best Current Practice), Jan. 2014.

[4] R. Bush and R. Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810 (Proposed Standard), Jan. 2013.

[5] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg. On the Risk of Misbehaving RPKI Authorities. *HotNets XII*, 2013.

[6] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman. Are We There Yet? On RPKI's Deployment and Security. In *NDSS*, 2017. To appear, available online.

[7] D. Iamartino, C. Pelsser, and R. Bush. Measuring BGP Route Origin Registration and Validation. In *PAM*, LNCS, pages 28–40, 2015.

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

[9] M. Lepinski, S. Kent, and D. Kong. A Profile for Route Origin Authorizations (ROAs). RFC 6482 (Proposed Standard), Feb. 2012.

[10] R. Lychev, S. Goldberg, and M. Schapira. BGP Security in Partial Deployment: Is the Juice worth the Squeeze? In *SIGCOMM*, pages 171–182. ACM, 2013.

[11] T. Manderson, K. Sriram, and R. White. Use Cases and Interpretations of Resource Public Key Infrastructure (RPKI) Objects for Issuers and Relying Parties. RFC 6907 (Informational), Mar. 2013.

[12] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. BGP Prefix Origin Validation. RFC 6811 (Proposed Standard), Jan. 2013.

[13] NIST. RPKI Monitor. `http://rpki-monitor.antd.nist.gov/`, 2016.

[14] A. Peterson. Researchers say U.S. Internet traffic was re-routed through Belarus. That's a problem. *Washington Post*, November 20 2013.

[15] Rensys Blog. Pakistan hijacks YouTube. `http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml`.

[16] O. Sagga, Y. Gilad, and S. Goldberg. compress_roas. online at `https://github.com/yossigi/compress_roas`.

[17] P. Smith, R. Evans, and M. Hughes. *RIPE Routing Working Group Recommendations on Route Aggregation.* RIPE, 2006. `https://www.ripe.net/publications/docs/ripe-399`.

[18] M. Wählisch, O. Maennel, and T. C. Schmidt. Towards Detecting BGP Route Hijacking using the RPKI. In *SIGCOMM*, pages 103–104, 2012.