MaxLength Considered Harmful to the RPKI

Yossi Gilad, Omar Sagga, and Sharon Goldberg
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) [Lepinski and Kent 2012], a security infrastructure built on top of interdomain routing, is not immune to 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. Our network measurements, however, suggest that the maxLength attribute strikes the wrong balance between security and user convenience. We therefore believe that operators should stop using maxLength. We give operational recommendations and develop software that allow operators to reap many of the benefits of maxLength without its significant security costs.

1. INTRODUCTION

Efforts to secure interdomain routing with the Border Gateway Protocol (BGP) have been ongoing for decades. To date, however, the RPKI [Lepinski and Kent 2012] is the only approach that has seen widespread deployment [NIST 2016]. The IETF is diligently working towards standardizing BGPsec [Lepinski 2012], a more robust security enhancement that should be deployed on top of the RPKI. BGPsec calls for a wholesale replacement of the BGP protocol and heavyweight online cryptography, while providing meagre benefits in partial deployment [Lychev et al. 2013]. Thus, it seems likely that it will take years before BGPsec deployment becomes a reality.

We therefore consider a setting where the RPKI is deployed, but BGPsec is not. Even in this setting, network operators can reap valuable security benefits [Lychev et al. 2013, Goldberg 2014] by dropping routes that the RPKI deems invalid. By doing this, operators can prevent some of the most devastating attacks on BGP: *prefix hijacks* and *subprefix hijacks*.

However, RPKI objects have an attribute called *maxLength*. We argue that when maxLength is misconfigured, it obviates most of the RPKI's security benefits. Specifically, an RPKI object with a misconfigured maxLength is vulnerable to a *forged-origin subprefix hijack*, which is as devastating as the traditional subprefix hijack that the RPKI is designed to prevent. These misconfigurations are common: our network measurements confirm that the vast majority of RPKI objects that use maxLength in the wild are vulnerable to this attack. We argue that the RPKI RFCs are not sufficiently consistent on the limitations of maxLength. We also show how most of the benefits attained through maxLength can be achieved without exposing users to attacks, and present open-source software that achieves this goal. We conclude with operational recommendations for the use of maxLength.

2. HOW THE RPKI SECURES BGP

We use a running example to review how the RPKI secures routing. Section 3 reviews maxLength.

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 the identifying number of an AS authorized to 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 a hijacking AS originates ("hijacks") routes for IP prefixes that it is not authorized to originate.

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 originates its prefix by sending a 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 could select this route, and propagate it 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 following BGP announcement (i.e., prepending its own AS number):

> "168.122.0.0/16: AS 3356, AS 111"

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. This behavior has been exploited in several high-profile incidents [Peterson 2013, McCullagh 2008].

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.

3. 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:

"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.[1] However, de-aggregation tends to bloat the size of routing tables across all routers on the Internet. As such, network operators frown upon excessive de-aggregation, *e.g.,* announcing all 2^8 of the /24s constituting a /16 [Smith et al. 2006].

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 [Lepinski et al. 2012]: "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.

Alternate solution: ROAs with sets of prefixes. Alternatively, because ROAs support *sets* of prefixes, one could just update the ROA to include both prefixes:

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

Minimal ROAs. A ROA is *minimal* [Manderson et al. 2013, § 3.2] when it includes only those prefixes that the AS announces in BGP, and no other prefixes. The ROA above is *minimal*, because it includes exactly the two prefixes announced by AS 111. Minimal ROAs come with less flexibility, because the AS must know exactly what prefixes it plans to announce at the time the ROA is issued.

4. FORGED-ORIGIN SUBPREFIX HIJACK

The convenience of maxLength comes with a serious downside. Specifically, when maxLength is used to issue ROAs that are not minimal, these ROAs are subject to a *forged-origin subprefix hijack*. This attack is as devastating as the traditional subprefix hijack that ROAs are designed to prevent.

A non-minimal ROA. Continuing with our example, suppose that AS 111 originates the two BGP announcements:

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

and that the RPKI had the ROA

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

[1] 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 [Smith et al. 2006].

with maxLength 24. This ROA is not minimal because it authorizes routes that are not announced in BGP.

The attack. 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 performs a *forged-origin subprefix hijack* [Gilad et al. 2017, Bush 2014] by sending this BGP announcement:

"168.122.0.0/24: AS m, AS 111"

Why does it work? (1) The hijacker's BGP announcement falsely claims that m is a neighbor of AS 111. But other ASes and routers have no way to know that this is false, because the RPKI does not provide a means to validate this claim (and BGPsec is not deployed in our setting). (2) The hijacker's BGP announcement is valid according to the RPKI, since the non-minimal ROA authorizes BGP routes for 168.122.0.0/24 with AS 111 as the origin AS. (3) AS 111 never originates a route for 168.122.0.0/24. Crucially, 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.

It's as bad as a subprefix hijack! 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. Thus, the attack has exactly the same impact as a regular subprefix hijack. And a regular subprefix hijack is more damaging than a prefix hijack. Indeed, one might argue that the whole point of the RPKI is to stop subprefix hijacks [Lychev et al. 2013, Lepinski and Kent 2012].

What's new here? An expert reader might wonder about the difference between the *forged-origin subprefix hijack* and the traditional *forged-origin hijack* [Lychev et al. 2013, Goldberg 2014]. In a traditional forged-origin hijack, the hijacker also (1) falsely claims to be a neighbor of the legitimate origin AS, and (2) announces a route that is valid according to the RPKI. However, because it is traditionally assumed that the RPKI only authorizes routes that are announced in BGP, the hijacker's route is for the *exact same prefix* that is legitimately announced in BGP. Thus, the forged-origin hijacker m would announce:

"168.122.0.0/16: AS m, AS 111"

Now, the hijacker's route is *not* the *only* route to the hijacked prefix. Rather than attracting *all* of the victim's traffic, the traffic must *split* between the hijacker's route and the legitimate route. This subtlety makes a huge difference in the effectiveness of the attack. [Lychev et al. 2013] shows that, during a traditional forged-origin hijack, the majority of traffic (on average) is still forwarded on the legitimate route. Meanwhile, when non-minimal ROAs authorize routes that are not announced in BGP, a forged-origin subprefix hijack on these routes causes *all* of the traffic to be intercepted by the hijacker.

Who is vulnerable? Any prefix in a non-minimal ROA is vulnerable. In particular, any prefix p in a ROA with maxLength m longer than p is vulnerable, unless *every* subprefix of p of length m is legitimately announced in BGP.

5. WHAT DO THE RFCS SAY?

RFC 7115 [Bush 2014] mentions minimal ROAs:

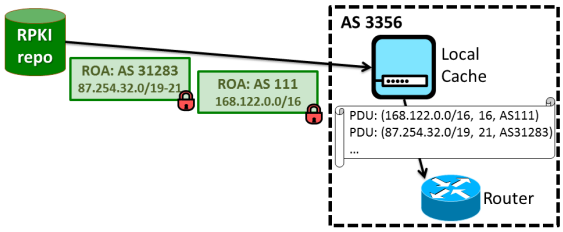


Figure 1: How routers get RPKI information.

> One advantage of minimal ROA length is that this attack does not work for sub-prefixes 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, an 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.

Note that "attacking the whole /16" is not only "more likely to be noticed", but also much *less effective* than a forged-origin subprefix-hijack against 10.0.666.0/24. To see why, return to our running example and suppose that BU had the minimal ROA described at the end of Section 3. Then, a *forged-origin subprefix hijack* on 168.122.0.0/24 (as described in Section 4) would fail, since this minimal ROA ensures that the hijacker's route is invalid. Thus, the hijacker m would need to "attack the whole /16", using a traditional *forged-origin hijack* announcement

"168.122.0.0/16: AS m, AS 111"

As remarked in Section 4, this causes traffic to split between the hijacker m and the legitimate AS 111, with the majority of ASes choosing to route by the legitimate path (see [Lychev et al. 2013]). Unlike the forged-origin *subprefix* hijack, this attack does not allow the hijacker to attract *all* of the traffic, and is thus significantly less effective [Lychev et al. 2013]. RFC 7115 [Bush 2014] also offers this 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 note that "multiple ROAs" are not required since ROAs support *sets* of IP prefixes.

RFC6907 similarly recommends issuing minimal ROAs [Manderson et al. 2013, § 3.2], but also suggests [Manderson et al. 2013, § 3.8] using a non-minimal ROA (that is vulnerable to a forged-origin subprefix hijacks) when ASes wish to deaggregate prefixes per our description in § 3. Specifically, RFC6907 describes 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* the /20 subprefixes of 10.1.0.0/16, it is vulnerable to a forged-origin subprefix hijacks.

6. BENEFITS OF MAXLENGTH?

We use network measurements to analyze the impact of the maxLength attribute. 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 [rou 2017]; we report results from the dataset of ROAs and BGP advertisements on June 1, 2017.

Problem: Using maxLength almost always creates vulnerabilities. First, we observe that only 4630 (about 12%) of the prefixes in ROAs have a maxLength longer than the prefix length. The vast majority of these ROAs are not minimal. Specifically, *almost all* of these prefixes (84%) are vulnerable to forged-origin subprefix hijacks. To measure this, we counted the number of prefixes p with maxLength m longer than p where some subprefix of p up to length m is *not* announced in BGP. Thus, we see that almost all users 'taking advantage' of the maxLength feature are unwittingly opening themselves up to attacks.

Benefit? Fewer prefixes included in ROAs. One might argue that a benefit of maxLength is that it reduces the number of ROAs. We now argue otherwise.

In our dataset, 40K prefixes are authorized by ROAs. What if (1) maxLength is eliminated and (2) only minimal ROAs were used? Then, we find that 13K *additional* prefixes would need to be added to ROAs. (To obtain this number, we count the number of prefixes that are (a) announced in BGP, and (b) are also a covered by a prefix that is authorized by a ROA in the RPKI.) We stress, however, that we could deal with these 13K additional prefixes *without* adding any additional ROAs (and associated cryptographic material) to the RPKI: we just convert each original non-minimal ROA to a minimal ROA that has the *set* of prefixes announced in BGP.

Benefit? Reducing load on routers. One might argue that maxLength reduces load on routers. We find that holds in today's scenario (where the RPKI is partially deployed), but not if the RPKI is fully deployed.

Figure 1 shows how routers get information from the RPKI. Each AS has a trusted *local cache* (typically a general-purpose machine) that downloads the complete set of ROAs from all default RPKI repositories. The local cache cryptographically validates the ROAs and creates a list of "Protocol Data Units" (PDUs). Each PDU is essentially a tuple of (IP prefix, maxLength, origin AS) [Mohapatra et al. 2013, Bush and Austein 2013]. The local cache sends the PDU list to the routers in its AS using the (*RPKI-to-Router*) protocol [rpk 2017]. Finally, routers use the PDU list to determine validity of BGP announcements [Bush and Austein 2013, Mohapatra et al. 2013, rpk 2017].

The PDU list could become longer if we replace all non-minimal maxLength-using ROAs with minimal ROAs that do not use maxLength. For instance, a single maxLength-using PDU for prefix 87.254.32.0/19-21 (shown in Figure 1), would be replaced with four non-maxLength-using PDUs (*i.e.,* one PDU for each subprefix of 87.254.32.0/19 up to length /21, that is announced in BGP). This could impact performance at routers.

The measurements described above indicate that today's routers would need to process 13K additional PDUs (a 33% increase). Thus, in today's scenario (where the RPKI is partially deployed), we find that maxLength does reduce the number of PDUs processed by routers. Yet, the number of PDUs is not very high.

Does this finding hold if the RPKI was fully deployed? To find out, we suppose that *every* IP prefix in our BGP dataset was covered by a *maximally-permissive ROA*. A maximally-permissive ROA authorizes each of its prefixes to the longest possible maxLength: namely, every IPv4 prefix has maxLength /32, while every IPv6 prefix has maxLength /128. (Maximally-permissive ROAs are vulnerable to forged-origin subprefix hijacks; we use them only to bound the maximum compression provided by maxLength.) Our BGP dataset has 777K advertised (IP prefix, AS) pairs (IPv4 and IPv6 prefixes). If all of these pairs were covered by maximally-permissive ROAs, these ROAs would still need to include 729K prefixes! Thus, in the full-deployment scenario, the maximum compression provided by the maxLength (in terms of reducing the number of PDUs processed by routers) is just 6.2%. This follows because most ASes do not send BGP announcements for subprefixes of their prefixes.

In Section 7, we present software that we built to preprocess the PDU list to reduce the number additional PDUs without introducing vulnerabilities to forged-origin subprefix hijacks. Our software achieves a compression rate of 6.1% in the same full-deployment scenario, very close to the above 6.2% bound.

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 [NIST 2016, Iamartino et al. 2015] 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 one-hop 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.

7. COMPRESSING MINIMAL ROAS

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) from a ROA [Mohapatra et al. 2013, Bush and Austein 2013]. The list of PDUs is

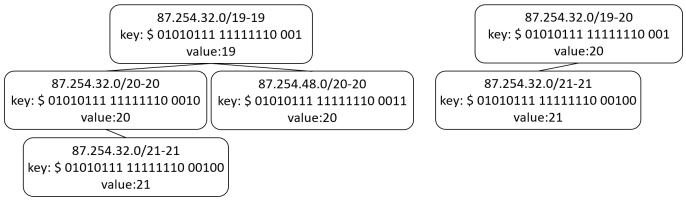


Figure 2: The IPv4 prefix trie for the minimal ROAs for AS 31283 without maxlength (left), and after compression with `compress_roas` (right). `compress_roas` reduces the number of output PDUs from four to two.

created by the trusted local cache, as shown in Figure 1. Thus, to avoid pushing a longer list of PDUs to routers, we now present software that runs on the local cache, and transforms a list of PDUs that do not use the maxLength attribute into a list of PDUs that do. Because it runs on the local cache, our software requires no changes to routers and conforms with today's RPKI architecture. Our software is publicly available [Sagga et al. 2017].

Build your own maxLength. Conceptually, our software compresses a set of ROAs that *do not use maxLength* to a set of ROAs that *do use maxLength*. Consider the following minimal ROA:

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

The above ROA does not use maxLength. Our software would compress it to the maxLength-using ROA:

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

This 'compressed' ROA is still minimal, because it covers 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 and is vulnerable to forged-origin subprefix hijacks (on IP prefix 87.254.40.0/21).

7.1 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 [rpk 2017]. 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 the length of the IP prefix. However, `compress_roas` can also reduce the number of (IP prefix, maxLength, origin AS)-tuples generated by today's (maxLength-using) RPKI.

Deployment. Our software is easy to integrate into the RPKI Relying Party Tools [rpk 2017]. 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) as shown in Figure 2. 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. Each trie node corresponds to some (AS, prefix, maxLength)-tuple in a valid ROA. The value of the trie node is the maxlength specified in the tuple. If the tuple came from a ROA that does not use the maxLength attribute (as we envision for the future RPKI), then the trie node's maxlength value is identical to the prefix length.

For a trie node with key $\$k$, we refer to the nodes $\$k||0$ and $\$k||1$ as its left and right *direct children*. A trie 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:

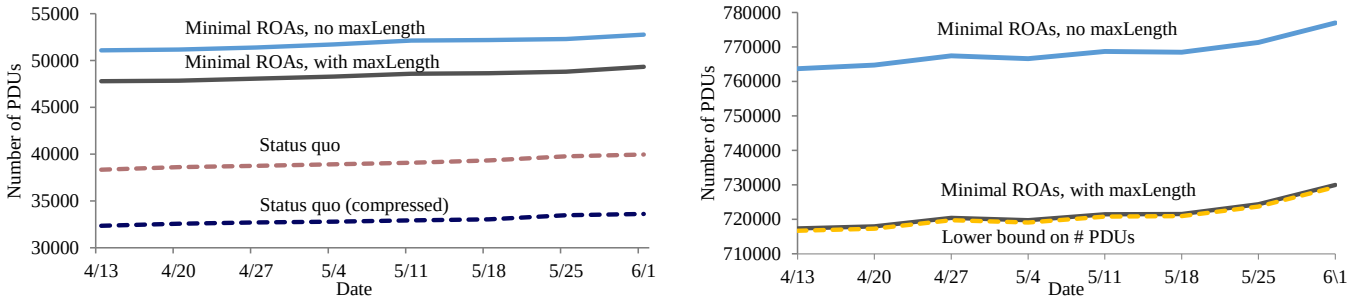
```
if (node has both direct children):
  minChildVal = min(lChild.value, rChild.value)
  if (minChildVal > node.value):
      # Adjust father's maxlength to cover children
      node.value = minChildVal
  if (lChild.value <= node.value):
      # left child now covered by father
      delete trie[lChild]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

As the DFS backtracks through the trie we run the compression function in Algorithm 1. 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 the minimum maxLength of its two children. The child is then deleted if the child's maxLength does not exceed the parent's maxLength. When the DFS completes, we get a set of (IP prefix, maxLength, AS)-tuples that correspond to the remaining trie nodes.

7.2 Performance

We evaluate our software both in today's RPKI partial deployment status, and in a future scenario where RPKI is fully deployed. We use datasets that aggregates ROAs and BGP advertisements on a weekly basis, from 4/13/2017 to 6/1/2017. Figure 3 presents results across all datasets. Table 1 and the discussion below is for the 6/1/2017 dataset. Our implementation's website shows how to reproduce our results [Sagga et al. 2017].

Today's RPKI. Our dataset has 7499 ROAs, comprising 39,949 distinct tuples of (IP prefix, maxLength, AS). Each tuple is inserted into one of the tries and compressed with



(a) Today's RPKI deployment

(b) RPKI in full deployment

Figure 3: Number of PDUs processed by routers under different scenarios along a timeline. A solid line marks scenarios that are safe against forge-origin subprefix hijacks. Scenarios with dashed lines are vulnerable.

procedure compress(*node*, *trie*):

if *node has both direct children* **then**
  minChildVal ← min{*node.lChild.value*, *node.rChild.value*}
  if *minChildVal* ¿ *node.value* **then**
    // Adjust father's maxlength to cover children
    node.value ← *minChildVal*
  if *node.lChild.value* ≤ *node.value* **then**
    // left child now covered by father
    delete *trie*[*node.lChild*]
  if *node.rChild.value* ≤ *node.value* **then**
    // right child now covered by father
    delete *trie*[*node.rChild*]
end

Algorithm 1: ROAs compression function. Called for every *node* as we backtrack from iterating over *trie* using DFS.

scenario	# PDUs	secure?
Today	39,949	*X*
Today (compressed)	**33,615**	*X*
Today, minimal ROAs, no maxLength	52,745	✓
Today, minimal ROAs, with maxLength (compressed)	**49,308**	✓
Full deployment, minimal ROAs, no maxLength	776,945	✓
Full deployment, minimal ROAs, with maxLength	**730,008**	✓
Full deployment, lower bound # PDUs (max permissive ROAs)	729,371	*X*

Table 1: Number of PDUs that processed by routers in the scenarios of Section 7 from the dataset of 6/1/2017.

`compress_roas`. The result is **33,615** tuples, for a compression of 15.90%. Thus, our software is beneficial even for 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 (1) identify the IP prefixes that are made valid by that ROA and are announced in our BGP dataset, and (2) modify the ROA so that it contains only those IP prefixes. We have the same number of ROAs, but now (instead of the status quo 39,949 tuples) we have a total of 52,745 (IP prefix, AS)-pairs. We use `compress_roas` to compress these 52,745 pairs to obtain **49,308** (IP prefix, maxLength, origin AS)-tuples, for a compression of 6.5%. Even with `compress_roas`, we still have 23% more tuples than the status quo; however, the status quo is vulnerable to forged-origin subprefix hijacks, and the scenario we just evaluated is not.

RPKI in full deployment. We consider a future full deployment scenario where the RPKI is hardened against forged-origin subprefix hijacks. That is, we assume every IP prefix announced in our BGP dataset is validated by a minimal ROA that does not use maxLength. Our BGP dataset has 776,945 (IP prefix, AS) pairs. This is exactly the number of (IP prefix, AS) pairs contained in ROAs if the RPKI used only minimal ROAs and no maxLength. Applying `compress_roas` gives **730,008** (IP prefix, length, maxLength)-tuples. This is very close to the lower bound of 729,371 tuples for the full-deployment scenario with maximally-permissive ROAs (which is vulnerable forged-origin prefix hijacks; see Section 6). Figure 3b shows that this result is consistent across all our measurements.

Thus, if today's RPKI eliminated maxLength and started using minimal ROAs (to immune against forged-origin subprefix hijacks), we would see some increase (23%) in the number of PDUs 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.

Computational overhead. We tested the `compress_roas` utility on an Intel i7-6700 machine. Compressing today's (partially-deployed) RPKI took 2.4 seconds and required 19MB of memory, while the full-deployment scenario took 36 seconds and 290MB memory. Performance could be improved by parallelizing across tries.

8. CAREFUL WITH MAXLENGTH

Today, only about 12% of prefixes in the RPKI use the maxLength attribute, but almost all of them are using it wrong. Operators that use maxLength are almost always vulnerable to forged-origin subprefix hijacks. These hijacks are just as harmful as the traditional subprefix hijacks that the RPKI is designed to prevent, thus obviating much of the RPKI's security benefits.

We therefore suggest that operators replace their maxLength-using ROAs with ROAs that (1) do not use maxLength and (2) are minimal, *i.e.,* that explicitly enumerate the set of IP prefixes that an AS actually originates in BGP. ROAs already support sets of IP prefixes, so switching to minimal ROAs would not change the number of ROAs or crypto-

graphic computations required. This switch can easily be accomplished without changing the RPKI standard, simply by modifying the popular user interfaces used for ROA configuration [RIPE 2016, ARIN 2017, LACNIC 2017, APNIC 2017, AfriNIC 2015] (*i.e.,* by removing the option to manually input maxLength). To limit impact on router performance, our `compress_roas` software can be used on RPKI local caches. In sum, avoiding the maxLength attribute makes 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.

9. REFERENCES

2017. RPKI Relying Party Tools. `https://github.com/dragonresearch/rpki.net`. (2017).

2017. University of Oregon Route Views Project. (2017). `http://www.routeviews.org/`.

AfriNIC. 2015. RPKI V2.0 announcement to the public. `https://www.afrinic.net/library/news/1407-rpki-v20-announcement-to-the-public`. (2015).

APNIC. 2017. Route Origin Authorizations (ROA): Create your ROA now in MyAPNIC. `https://www.apnic.net/manage-ip/apnic-services/resource-certification/roa/`. (2017).

ARIN. 2017. Submitting a ROA Request. `https://www.arin.ne/resources/rpki/roarequest.html`. (2017).

R. Bush. 2014. Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI). RFC 7115 (Best Current Practice). (Jan. 2014). `http://www.ietf.org/rfc/rfc7115.txt`

R. Bush and R. Austein. 2013. The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810 (Proposed Standard). (Jan. 2013). `http://www.ietf.org/rfc/rfc6810.txt`

Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. 2017. Are We There Yet? On RPKI's Deployment and Security. In *NDSS*.

Sharon Goldberg. 2014. Why is it taking so long to secure internet routing? *Commun. ACM* 57, 10 (2014), 56–63.

Daniele Iamartino, Cristel Pelsser, and Randy Bush. 2015. Measuring BGP Route Origin Registration and Validation. In *PAM (LNCS)*. 28–40. `http://dx.doi.org/10.1007/978-3-319-15509-8`

LACNIC. 2017. Manual de Usuario, Sistema de Certificacion de Recursos - RPKI, Version 1.0. `http://lacnic.net/documentos/rpki/Manual_Usuario_RPKI_Beta_SP_v1.pdf`. (2017).

M. Lepinski (Ed.). 2012. *BGPSEC Protocol Specification.* IETF Network Working Group, Internet-Draft. Available from `http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-04`.

M. Lepinski and S. Kent. 2012. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational). (Feb. 2012). `http://www.ietf.org/rfc/rfc6480.txt`

M. Lepinski, S. Kent, and D. Kong. 2012. A Profile for Route Origin Authorizations (ROAs). RFC 6482 (Proposed Standard). (Feb. 2012). `http://www.ietf.org/rfc/rfc6482.txt`

Robert Lychev, Sharon Goldberg, and Michael Schapira. 2013. BGP Security in Partial Deployment: Is the Juice worth the Squeeze?. In *SIGCOMM*. ACM, 171–182. `http://dl.acm.org/citation.cfm?id=2486001`

T. Manderson, K. Sriram, and R. White. 2013. Use Cases and Interpretations of Resource Public Key Infrastructure (RPKI) Objects for Issuers and Relying Parties. RFC 6907 (Informational). (March 2013). `http://www.ietf.org/rfc/rfc6907.txt`

Declan McCullagh. 2008. How Pakistan knocked YouTube offline (and how to make sure it never happens again). (February 2008).

P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. 2013. BGP Prefix Origin Validation. RFC 6811 (Proposed Standard). (Jan. 2013). `http://www.ietf.org/rfc/rfc6811.txt`

NIST. 2016. RPKI Monitor. `http://rpki-monitor.antd.nist.gov/`. (2016).

Andrea Peterson. 2013. Researchers say U.S. Internet traffic was re-routed through Belarus. That's a problem. *Washington Post* (November 20 2013).

RIPE. 2016. Managing ROAs. `https://www.ripe.net/manage-ips-and-asns/resource-management/certification/resource-certification-roa-management`. (September 2016).

Omar Sagga, Yossi Gilad, and Sharon Goldberg. 2017. compress_roas. Available online at `https://github.com/yossigi/compress_roas`. (2017).

Philip Smith, Rob Evans, and Mike Hughes. 2006. *RIPE Routing Working Group Recommendations on Route Aggregation.* RIPE. `https://www.ripe.net/publications/docs/ripe-399`.