Experiment 5 - BGP and RPKI

DE-CIX Academy

Version 0.3 - Students Copy

1 Network Setup

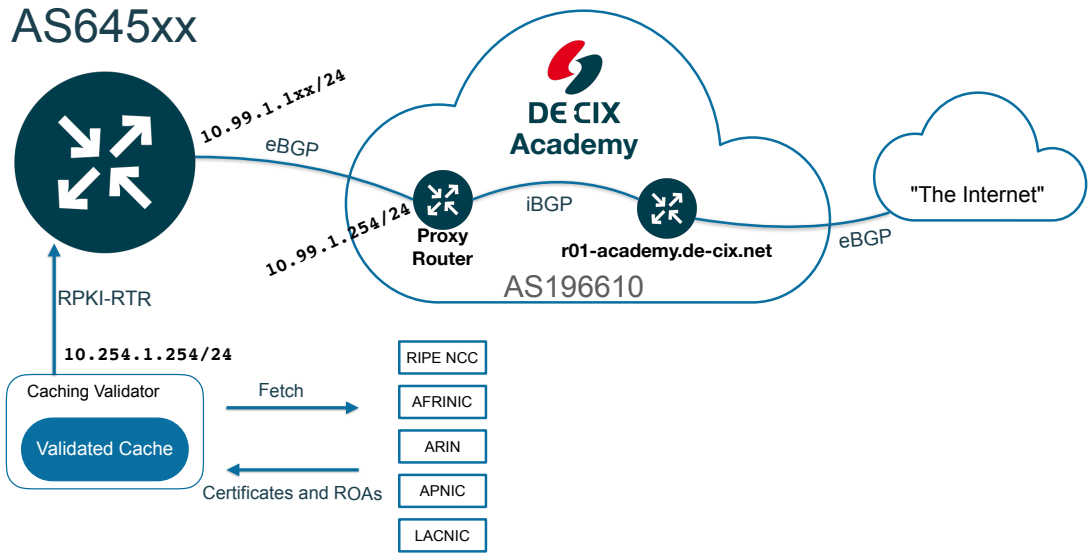


Figure 1: Network Setup

Figure 1 shows the network topology for this experiment. Upstream to AS196610 via a proxy router delivers real prefixes from the Internet. The caching validator runs in a separate container.

2 Introduction

RPKI allows the holder of a resource (an IPv4/IPv6 prefix) to cryptographically prove that it is really the holder and allows via defining *ROA*s how that prefix can be announced via BGP.

A *ROA* is a triple containing the following values:

- The prefix itself (network plus prefix length)
- An AS which is allowed to originate that prefix
- A maximum prefix length for which BGP announcements are allowed, this can be the same as the prefix length of the network (in this case the announcement of more specifics for this prefix is not allowed).

To use RPKI and ROAs you need a host running a *RPKI validator*. This validator fetches resource certificates and ROAs from RIRs, checks their signatures and is then available being contacted by routers using RPKI-RTR protocol.

Routers simply receive a list of validated prefixes, their allowed originating AS number and a range of allowed networks masks. This can be used to check prefixes received via eBGP. Result of this check is one of three possible values:

Valid: A ROA for this prefix exists, the originating AS matches and the announced prefix length is covered by the ROA.

Invalid: A ROA for this prefix exists, but either it is for a different originating AS number or the prefix length is *not* covered (too specific).

Unknown: For this prefix a ROA does not exist. This result is also returned if no validator is reachable by the router. Most of the prefixes announced via eBGP fall into this category.

3 Starting RPKI

RPKI can be stopped and started *outside* the config mode. Use the following commands:

rpki start to start RPKI

rpki stop to stop RPKI

4 Config statements

To configure RPKI you can use in *config mode* (as always: Use "configure" to enter):

RPKI changes the config context to *rpki* mode. All following commands are to be entered in this mode. Use *exit* to leave.

rpki cache [tcp|ssh <ipv4 address> <portnumber> preference <number>] defines *<ipv4 address>* as a validator. Your router will contact it via TCP on port *<portnumber>*. Using the *<preference>* parameter you can configure multiple validators and give them a preference.

rpki cache [tcp|ssh <hostname> <portnumber> preference <number>] alternative version where the name of the validator is used instead of its IPv4 address.

This is the only command you need to configure a validator. For optional parameters please see the FRRouting documentation.

4.1 Within route-maps

There is only one match statement for RPKI.

4.1.1 Match statements

You can match against the three possible results of RPKI validation:

match rpki *valid* returns true if the prefix has a covering ROA and originating AS and netmask evaluate ok.

match rpki *invalid* returns true if there is a ROA but either originating AS is wrong or announced netmask is not covered by the ROA.

match rpki *notfound* returns true if there is no ROA for the prefix.

5 show commands

The show statements are to list validators and the prefix tables received from them:

show rpki cache-server lists all the configured validators.

show rpki cache-connections lists all the active connections to validators.

show rpki <prefix> Looks if prefix (IPv4 or IPv6) is in the validated prefix table and shows its entry.

show rpki prefix-table Lists the whole validated prefix table. Output can be long.

To show valid/invalid prefixes you need to define a route-map and then can use it in a show command:

```
configure
route-map rpki-invalid
  match rpki invalid
end

show bgp ipv4 route-map rpki-invalid
show bgp ipv6 route-map rpki-invalid
```

6 Task: Reject prefixes with RPKI invalid

- Connect your router to a validator
- create a route-map which matches only invalid prefixes
- create a route-map which matches only valid prefixes
- have a look what (in)valid prefixes you receive from your upstream
- Update the *upstream-in* route-map so prefixes with status *invalid* are no longer accepted.
- check again for *invalid* prefixes.

A Initial Router Configuration

Initially the router connects to AS196610 (DE-CIX Academy AS)

```
hostname rxx
!
interface loopback0
 ip address 172.16.1.xx/32
!
router bgp 645xx
 neighbor upstream peer-group
 neighbor 10.99.1.254 remote-as 196610
 neighbor 10.99.1.254 peer-group upstream
 !
 address-family ipv4 unicast
  neighbor upstream route-map upstream-in in
  neighbor upstream route-map upstream-out out
 exit-address-family
!
route-map upstream-in permit 100
!
route-map upstream-out deny 100
!
end
```

B Final Router Configuration

Route-Maps *rpki-invalid* and *rpki-valid* are for show command only and not referenced (use *show bgp ipv4 route-map invalid* for example).

```
hostname rxx
rpki
  rpki polling_period 3600
  rpki timeout 600
  rpki initial-synchronisation-timeout 30
  rpki cache tcp 10.254.1.254 3323 preference 1
  exit
!
```

```
interface loopback0
 ip address 172.16.1.xx/32
!
router bgp 645xx
 neighbor upstream peer-group
 neighbor 10.99.1.254 remote-as 196610
 neighbor 10.99.1.254 peer-group upstream
 !
 address-family ipv4 unicast
  neighbor upstream route-map upstream-in in
  neighbor upstream route-map upstream-out out
 exit-address-family
!
route-map rpki-invalid permit 100
 match rpki invalid
!
route-map rpki-valid permit 100
 match rpki valid
!
route-map upstream-in deny 50
 match rpki invalid
!
route-map upstream-in permit 100
!
route-map upstream-out deny 100
!
end
```

C Slides

Receive real prefixes via eBGP

- For this we need real prefixes from the internet
- Configuration needed will be preloaded - but for your reference here it is

```
conf t

route-map upstream-in permit 100
route-map upstream-out deny 100

router bgp 645xx
  neighbor upstream peer-group
  address-family ipv4
    neighbor upstream route-map upstream-in in
    neighbor upstream route-map upstream-out out
  exit-address-family
  neighbor 10.99.1.254 remote-as 196610
  neighbor 10.99.1.254 peer-group upstream
```

Have a look at your routing table:

- **show bgp ipv4 summary** - you should receive about 17000 prefixes
- **show bgp ipv4** - beware, output is looong
- **show bgp ipv4 statistics** - perhaps the most useful command for this experiment

Validator: Routinator

- We are using *Routinator* as validator
- It also runs in a docker container
- Just like your router, but on port **7099**

Try:

- `https://containerlab.as196610.net:7099/`
- `https://containerlab.as196610.net:7099/status`
- `https://containerlab.as196610.net:7099/metrics`
- `https://containerlab.as196610.net:7099/csv`
- `https://containerlab.as196610.net:7099/json`

If you run into problems - check these URLs to see if the validator is running.

Connecting to the validator

- a router connects to a validator using the RPKI-RTR protocol via TCP
- we only have to configurate the validator IP address and port-number
- FRRouting uses a special rpki *configuration context* for this

```
configure

rpki
  rpki cache tcp 10.254.1.254 3323 preference 1
end
```

To see if RPKI is working use:

- **show rpki cache-connection**
  shows if connection to the validator is up
- **show rpki cache-server**
  shows all configured validators
- **show rpki prefix-table** (long output!)
  shows the complete table received from the validator
- **show rpki prefix *some prefix***
  looks up *some prefix* in the table of validated prefixes.

You might have to start/stop rpki validation in FRRouting. Use commands "*rpki stop*" and "*rpki start*" (**not** in config mode) for this

Route-Maps for RPKI

- FRRouting does not filter by itself (other routers sometimes do)
- So we need route-maps
- We want one route-map for each RPKI state *valid* and *invalid*

```
conf t

route-map rpki-invalid permit 100
  match rpki invalid

route-map rpki-valid permit 100
  match rpki valid
```

We can now match against these route-maps:

- **show bgp ipv4 route-map rpki-valid**
  shows all the prefixes which are valid in terms of RPKI
- **show bgp ipv4 route-map rpki-invalid**
  shows all prefixes which are invalid - you might check how using:
- **show rpki prefix | include *part of prefix***
  shows only the part of the full rpki prefix table that matches *part of prefix* as a text - quite useful if you want to figure out *why* a prefix is invalid.

Dropping invalids

- *Invalid* prefixes (in terms of RPKI) are bad - so we do not want to accept them. Neither from upstream nor from peering.
- with the match command we know, we can extend our *upstream-in* route-map
- check how many prefixes you receive:
  show bgp ipv4 statistics
- extend the *upstream-in* route map with a *deny* statement:

```
conf t

route-map upstream-in deny 50
  match rpki invalid
```

- **show bgp ipv4 statistics**
  Enter this command *before* and *after* you add the deny-statement to your route-map.

Congratulations!

You just made the Internet a little bit more safe!