RPKI Hints, Top Tips, and FAQs

On this page I'm collecting how to do various RPKI bits and pieces. Us
documentation is incomplete, or just plain useless.

These are my build notes, and I keep them current in so far as I use t
for the NSRC Global R&E Routing Table report and for RouteViews. YM

NB always use the most up to date/current versions of the validators -
quickly, and the older versions can often be quite buggy, if not unstab

Here is the list (so far):

- AS0 TALs
- Routinator 3000 validator from NLnetLabs
- FORT validator from NIC Mexico
- RPKI-client validator
- StayRTR
- Cisco IOS-XE
- Cisco IOS-XR
- Juniper
- BIRD
- FRR

The tips and tricks for the validator builds discussed below all are for
work on Ubuntu 18.04 and Ubuntu 20.04 (neither of which is supported
supported until April 2027). If you are installing a fresh container or V
you are wedded to the Ubuntu family).

AS0 TALs

Two of the Regional Internet Registries have supplied Trust Anchor Lo
address space that they hold.

If you want to use these TALs, you can read more:

- APNIC's AS0 TAL
- LACNIC's AS0 TAL

Generally, to use these TALs, place each in a separate file (e.g. apnic-as0.tal
the usual place where you keep your TALs - it depends on your validat

NLnetLabs Routinator

Nothing to say here, the instructions just work, the validator installs s
the instructions are followed. The current version of Routinator is 0.1

If using Debian/Ubuntu as I do, then just use the supplied package an
manager. The instructions for how to install are documentation abs excellen

The installer will set up systemd as necessary that Routinator starts automatic
Remember to modify /etc/routinator/routinator.conf so that Routinator listens
IPv4 (and IPv6) ports of the system - and you can enable the default s
port 8323. A working configuration file would look like this:

```
repository-dir = "/var/lib/routinator/rpki-cache"
rtr-listen = ["x.x.x.x:3323", "[x:x:x:x::x]:3323"]
http-listen = ["x.x.x.x:8323", "[x:x:x:x::x]:8323"]
```

As from Routinator 0.12.0, the 5 RIR TALs are distributed built-in to t
any need to read and agree to the ARIN RPA.

At time of writing, though, if you want to use the AS0 TALs from LACN
extra-tals-dir option mentioned in the configuration guide only works
in the Routinator configuration file that ships as part of the package.
modification to the systemd setup /etc/systemd/system/routinator.service like this:

```
[Service]
ExecStart=
ExecStart=/usr/bin/routinator --config=/etc/routinator/routinator.conf
extra-tals-dir="/var/lib/routinator/tals" --syslog server
```

where the /var/lib/routinator/tals contains the two AS0 TALs you want to u
ExecStart= is needed to overwrite the one that is shipped in
/lib/system/system/routinator.service. Once this bug is fixed you can remove this
systemd setting and put extra-tals-dir definition in the configuration file, like thi

```
repository-dir = "/var/lib/routinator/rpki-cache"
extra-tals-dir = "/var/lib/routinator/tals"
rtr-listen = ["x.x.x.x:3323", "[x:x:x:x::x]:3323"]
http-listen = ["x.x.x.x:8323", "[x:x:x:x::x]:8323"]
```

FORT

FORT is the validator developed by NIC Mexico. Project home page. At time of
writing, version 1.6.7 has been released and fixes many issues presen
from version 1.6.3, FORT requires Ubuntu 24.04 as it requires libjanss
only comes with libjansson4 2.13.1-1.1build3 will only support FORT v

FORT is available as part of Ubuntu 22.04 packaging, but it is an olde
Ubuntu 24.04, the FORT shipped is version 1.6.1-1build3. For this rea
produced package.

FORT is not quite so easy to install, but still relatively simple as long
their Github repo closely.

First step is to grab the deb from their archive:

```
wget
https://github.com/NICMx/FORT-validator/releases/download/1.6.8/fort
_amd64.deb
```

and then install it:

```
sudo apt install ./fort_1.6.8-1_amd64.deb
```

Note that apt install installs a systemd file and starts FORT running automatically
TCP/323 as the listener port - you may want to customise this, and to
file /etc/fort/config.json. This is the configuration file that I use:

```
{
	"tal": "/etc/fort/tal",
	"local-repository": "/var/lib/fort",
	"slurm": "/etc/fort/slurm/",
	"server": {
		"port": "3323"
	},
	"log": {
		"output": "syslog"
	}
}
```

All I have done is modify the port that the server listens on.

The package ships with 4 of the 5 Trust Anchor Locators, so to get the
need to run:

```
sudo fort --init-tals --tal=/etc/fort/tal
```

You will be asked to confirm that you have read the Terms and Conditi

```
...
Jan 26 03:50:46 DBG: Done. Total bytes transferred: 466
Jan 26 03:50:46 DBG: HTTP result code: 200
Successfully fetched '/etc/fort/tal/apnic.tal'!

Attention: ARIN requires you to agree to their Relying Party Agreemer
before you can download and use their TAL.
Please download and read https://www.arin.net/resources/manage/rpki
If you agree to the terms, type 'yes' and hit Enter: yes
Jan 26 03:50:51 DBG: HTTP GET:
https://www.arin.net/resources/manage/rpki/arin.tal
Jan 26 03:50:51 DBG: Done. Total bytes transferred: 487
Jan 26 03:50:51 DBG: HTTP result code: 200
Successfully fetched '/etc/fort/tal/arin.tal'!

Jan 26 03:50:51 DBG: HTTP GET:
https://www.lacnic.net/innovaportal/file/4983/1/lacnic.tal
```

```
...
```

One thing that I found is that FORT crashes on start up following the
the letter. The issue was that the /var/lib/fort folder is owned by root and not by the fort user. Easy to fix

```
sudo chown fort:fort /var/lib/fort
```

Then restart FORT:

```
sudo systemctl restart fort
```

and it should run successfully. You should see something like this when
fort:

```
● fort.service - FORT RPKI validator
     Loaded: loaded (/usr/lib/systemd/system/fort.service; enabled; pre
enabled)
     Active: active (running) since Mon 2024-10-07 22:58:03 AEST; 29s
       Docs: man:fort(8)
             https://nicmx.github.io/FORT-validator/
   Main PID: 148150 (fort)
      Tasks: 27 (limit: 38225)
     Memory: 680.6M (peak: 680.7M)
        CPU: 27.801s
     CGroup: /system.slice/fort.service
             ├─148150 /usr/bin/fort --configuration-file
/etc/fort/config.json

Oct 07 22:58:03 fort systemd[1]: Started fort.service - FORT RPKI val
```

You can check by using:

```
  195 ?        Ssl   95:13 /usr/bin/fort --configuration-file
/etc/fort/config.json
```

and netstat -tulpn (up to Ubuntu 18.04) or ss -tulpn (from Ubuntu 20.04 onwards) to get:

```
tcp    LISTEN   0        128                0.0.0.0:3323
0.0.0.0:*
```

The systemd that ships with FORT doesn't have a restart on failure setti
we do this so that if/when FORT crashes, it will automatically restart.

Create a new /etc/systemd/system/fort.service (where is where to create user-de
systemd), and in it put:

```
[Service]
Restart=
Restart=on-failure
```

(If you edit the /lib/systemd/system/fort.service it will be overwritten on the next upg
FORT, so it is best create the user entry instead.)

And your new FORT installation is now ready for service!

RPKI-client

rpki-client is just a validator - it does not have the functionality to accep
We'll come to that later on (we'll need to use a fork of Cloudßare's now
unmaintained GoRTR).

rpki-client has now been packaged and is available across most mainstre
platforms. Including as part of the Ubuntu 22.04 and later distribution
version in Ubuntu is old (version 7.6 on 22.04, 9.0 on 24.04). At the t
release rpki-client version 9.8. There is a current version of rpki-client on the Ubuntu Snap
Store, but not every sysadm wishes to use yet another package manag

My preference is to stay up to date on Ubuntu, and so we have to buil
rpki-client maintainers don't build their own deb package, or pre-build pa
with Routinator, given that Ubuntu maintainers seem to be unable to k
well.

Initial Preparation

Before you attempt to download rpki-client, it is not true that you need
other packages in place. These are automake, autoconf, make, git itself, libtool, libexpat,
libssl-dev, libtls-dev, rsync and zlib1g-dev. This is all quite easy using the Ubuntu
manager (and some of these packages may already be installed by def

```
sudo apt install automake autoconf make git libtool libexpat1-dev libs
libtls-dev zlib1g-dev
```

And now we are ready to build rpki-client

Building rpki-client

Easiest is just to install rpki-client from the Github repository:

```
git clone https://github.com/rpki-client/rpki-client-portable.git
```

and then:

```
cd rpki-client-portable
./autogen.sh
./configure --with-tal-dir=/etc/rpki \
   --with-base-dir=/var/lib/rpki-client \
   --with-output-dir=/var/db/rpki-client
```

```
make
```

The autogen.sh script ßxes the conßg set up, ready to run, which will produce the Makeßle. Note that we are specifying where our TALs go, where the temp Ubuntu norms), and where the output ßle storing the VRPs goes (again

Hidden in the oßcial instructions is a comment that rpki-client is a normal user if start root. So we need to create that normal user:

```
sudo groupadd _rpki-client
sudo useradd -g _rpki-client -s /sbin/nologin -d /nonexistent -c "rpki-
client user" _rpki-client
```

Now we can install RPKI-client:

```
sudo make install
```

which will install the client in /usr/local/sbin and the 5 TALs in /etc/rpki, as well as create the c and output directories needed. It will also copy the 5 RIR tiles /etc/rpki/constraints prevent overclaiming of resources by the 5 RIRs.

Now the client can be run. There is no daemon option, it simply runs a it has ßnished downloading all the VRPs (around 10-15 minutes depen that's okay. Try running the client:

```
sudo /usr/local/sbin/rpki-client
```

You'll see errors about various CAs or ßles not being accessible - tha check in /var/db/rpki-client folder you will see a ßle once the above rpki-run of client completes. This is the conßguration you'd need if you are going run a standalone RtR client instead, so we will need JSON output inst

Once rpki-client completes, we can now set it up to run automatically. To d /etc/cron.hourly rpki-client, and in it we put:

```
#!/bin/bash

# run RPKI-client every hour
# - default output location is /var/db/rpki-client
# - -j option means json output, suitable for stayrtr

/usr/local/sbin/rpki-client -j > /tmp/rpki-client.log 2>&1
```

thaT128.441 5nhe5g 0Tr0n

```
sudo make install
```

```

```

```

```

```

```

```
fi
```

Log out. And then log in again. (Easiest way to activating the updated PATH)

And now check you have a working Go environment:

```
go version
```

If the version shows what you installed, you are set!

Building StayRTR

Easiest way to install StayRTR is to build from the Github repo.

```
git clone https://github.com/bgp/stayrtr.git
cd stayrtr
make build-all
```

which builds stayrtr as well as rtrmon and rtrdump (the latter used for testing purposes).

Copy the resulting binaries to /usr/local/bin:

```
cd dist
sudo cp -p stayrtr-v0.6.4-1-g4bad963-linux-x86_64 /usr/local/bin/stay
sudo cp -p rtrdump-v0.6.4-1-g4bad963-linux-x86_64 /usr/local/bin/rtrd
sudo cp -p rtrmon-v0.6.4-1-g4bad963-linux-x86_64 /usr/local/bin/rtrmo
```

StayRTR has lots of options, but the ones we need are these:

```
 -bind string
       Bind address (default ":8282")
 -cache string
       URL of the cached JSON data (default
"https://console.rpki-client.org/vrps.json")
```

We don't need to use the public RPKI-client JSON source, given we ha
created RPKI-client.

We run StayRTR like this:

```
/usr/local/bin/stayrtr -bind :3323 -cache /var/db/rpki-client/json
```

which will at least let us test that it works. Run it and see what happe
command line looking like this:

```
INFO[0000] new cache file: Updating sha256 hash  ->
e0a14ea955e183e2719dcfbee0e9429b34581972c6ad5f6e9e064ee1396ca
INFO[0001] New update (307007 uniques, 307007 total prefixes).
```

```
sudo systemctl enable stayrtr
```

which then displays:

```
Created symlink /etc/systemd/system/multi-user.target.wants/stayrtr.s
! /lib/systemd/system/stayrtr.service.
```

and then we can run StayRTR, like this:

```
sudo systemctl start stayrtr
```

Once it is running, check that it is working by running:

```
sudo systemctl status stayrtr
```

and you should see something like this:

```
%˘ stayrtr.service - StayRTR RPKI to Router Server
     Loaded: loaded (/usr/lib/systemd/system/stayrtr.service; enabled;
preset: enabled)
     Active: active (running) since Thu 2024-08-15 16:57:07 +06; 31s a
       Docs: https://github.com/bgp/stayrtr
   Main PID: 44045 (stayrtr)
      Tasks: 5 (limit: 4614)
     Memory: 379.9M (peak: 459.4M)
        CPU: 3.805s
     CGroup: /system.slice/stayrtr.service
             %% 44045 /usr/local/bin/stayrtr -bind :3323 -cache /var/db/r
client/json

Aug 15 16:57:06 rpki systemd[1]: Starting stayrtr.service - StayRTR R
Router Server...
Aug 15 16:57:07 rpki systemd[1]: Started stayrtr.service - StayRTR RP
Router Server.
Aug 15 16:57:10 rpki stayrtr[44045]: time="2024-08-15T16:57:10+06:0
level=info msg="New update (602077 uniques, 602151 total prefixes, 7
3 router>
Aug 15 16:57:11 rpki stayrtr[44045]: time="2024-08-15T16:57:11+06:0
level=info msg="Update added, new serial 0"
Aug 15 16:57:11 rpki stayrtr[44045]: time="2024-08-15T16:57:11+06:0
level=info msg="StayRTR Server started (sessionID:61374, refresh:36
retry:600, ex>
```

and you can also run the ps command to see something like:

```
  44045 ?        Ssl    0:03 /usr/local/bin/stayrtr -bind :3323 -cache
/var/db/rpki-client/json
```

And that's it. Enjoy your new StayRTR installation.

Cisco IOS-XE Hints

This section shows the basic configuration needed to get route origin
Cisco IOS-XE platforms.

Most commentary is for IOS-XE 16.x. Older versions will also likely wo
but their RPKI implementation is somewhat old and buggy.

IOS-XE Configuration with Validator

Setting up a Cisco IOS-XE router to talk with a validator is a one line

```
router bgp <ASN>
 bgp rpki server tcp <ip address> port <port> refresh 3600
```

where <ip address> is the IP address (IPv4 or IPv6) of the validator, <
validator listens on, and 3600 is the RFC8210 recommended refresh in
router will use to ask the validator if there is new/updated validation

Doing this will download the VRPs from the specified validator(s), and
RIB) with the prefixes validation state (whether valid, invalid, or not-f
See caveats below.

To find out what is in the validation database (IPv4 and IPv6 command

```
show ip bgp rpki table
show bgp ipv6 rpki table
```

and to find out the status of the connection to the validator:

```
show ip bgp rpki servers
```

Cisco IOS-XE Caveats

Cisco IOS-XE has many defaults which are non-standard and will be po
operators:

- Cannot specify a source-interface for the router to validator conne
- Automatically activates route origin validation (can be turned off!)
- Automatically drops invalids (can be turned off!)
- Locally originated prefixes are always marked as valid (cannot be

where <ip address> is the IP address (IPv4 or IPv6) of the validator, <

The above enables origin validation, and still allows invalid preßxes i

Once you are ready to drop invalids, as per recommended best practic

```
router bgp <ASN>
  address-family ipv4 unicast
    no bgp bestpath origin-as allow invalid
  address-family ipv6 unicast
    no bgp bestpath origin-as allow invalid
```

To display the validation state of preßxes, you can use the following c

```
RP/0/RP0/CPU0:cr1#show bgp origin-as validity ?
  invalid    filter routes with invalid origin-as
  not-found  filter routes with unknown (not found) origin-as
  standby    Display Standby BGP information
  valid      filter routes with valid origin-as
  |          Output Modifiers
  <cr>
```

The sub-options will display all the preßxes ßtting into each category.

Juniper Hints

This section shows the basic conßguration needed to get route origin
JunOS platform. JunOS follows standards, as far as I can tell. All poli
by the operator.

Conßguration with Validator

The conßguration needed for JunOS to talk with two validators looks l

```
routing-options {
  validation {
    group ISP {
      /* validatorA */
      session <ip address validatorA> {
        refresh-time 600;
        hold-time 3600;
        preference 1;
        port <port>;
        local-address <router loopback>;
      }
      /* validatorB */
      session <ip address validatorB> {
        refresh-time 600;
        hold-time 3600;
        preference 2;
```

```
        port <port>;
        local-address <router loopback>;
      }
    }
  }
}
```

where <ip address> is the IP address (IPv4 or IPv6) of the validator, <
validator listens on, and 3600 is the RFC8210 recommended refresh in
router will use to ask the validator if there is new/updated validation

Two validators are recommended - if you only have one, then edit the

This simply creates a validation database on the router - it does not t
what is in the validation database:

```
show validation replication database
```

and to ß nd out the status of the connection to the validator:

```
show validation session detail
```

Flagging validation status in BGP RIB

To indicate validation status in the BGP RIB, the operator needs to im
that. Here is an example:

```
policy-options {
  policy-statement RPKI-validation {
    term VALID {
      from {
        protocol bgp;
        validation-database valid;
      }
      then {
        validation-state valid;
        next policy;
      }
    }
    term INVALID {
      from {
        protocol bgp;
        validation-database invalid;
      }
      then {
        validation-state invalid;
        next policy;
      }
    }
    term UNKNOWN {
```

```
      from {
        protocol bgp;
        validation-database unknown;
      }
      then {
        validation-state unknown;
        next policy;
      }
    }
  }
}
protocols {
  bgp {
    group EBGP {
      type external;
      local-address <local-router>;
      neighbor <ebgp-peer> {
        description "Upstream";
        import [ RPKI-validation Upstream-in ];
        export LocalAS-out;
        peer-as <their-ASN>;
      }
    }
  }
}
```

This is intended to be inserted as a policy on an EBGP peering. The *validation-database* conß guration refers to the database created by the router's validation session w *state* conß guration enters a ß ag in the BGP RIB to indicate the validatio the *Upstream-in* policy is other inbound policy that is on the router, and not

And then the operator can do things like:

```
show route protocol bgp validation-state valid
```

to show valid preß xes as in this example. Other options include invalid and unknown fo

Implementing Route Origin Validation

The ß nal step in JunOS is to implement Route Origin Validation. This replacing the:

```
next policy
```

line in the INVALID term with:

```
reject
```

This will discard invalid preß xes as they arrive on an EBGP speaking

what has been dropped, you can run:

```
show route protocol bgp validation invalid hidden
```

The discarded prefixes are still in Adj-RIB-In and can be seen with the

BIRD Hints

This section shows the basic configuration needed to get route origin
BIRD platform. This will be of most interest to IXPs, as BIRD is the mos
implementation today. The configuration here is for BIRDv2 (2.14 at ti

Configuration with Validator

The configuration needed for BIRD to talk with a validator is:

```
roa4 table r4;
roa6 table r6;

protocol rpki validator1 {
    roa4 { table rpki4; };
    roa6 { table rpki6; };
    remote <ip address1> port <port1>;
    retry keep 600;
    refresh keep 3600;
    expire keep 7200;
}

protocol rpki validator2 {
    roa4 { table rpki4; };
    roa6 { table rpki6; };
    remote <ip address2> port <port2>;
    retry keep 600;
    refresh keep 3600;
    expire keep 7200;
}
```

where <ip address> is the IP address (IPv4 or IPv6) of the validator, <
validator listens on, and 3600 is the RFC8210 recommended refresh in
router will use to ask the validator if there is new/updated validation
validators are shown in this example.

This simply creates a validation database in BIRD - it does not touch
in the validation database (IPv4 and IPv6 shown):

```
show route table rpki4
show route table rpki6
```

and to find out the status of the connection to *validator1*:

```
show protocols validator1
```

Implementing Route Origin Validation

The final step in BIRD is to implement Route Origin Validation. This ne
applied on the *outbound* policy on all BGP sessions (internal and external). We b
functions, like below (as used on route-server implementations of BIR

```
# v4 function to check if prefix valid
function is_v4_rpki_invalid () {
  return roa_check(rpki4, net, bgp_path.last_nonaggregated) = ROA_IN
}

# v6 function to check if prefix valid
function is_v6_rpki_invalid () {
  return roa_check(rpki6, net, bgp_path.last_nonaggregated) = ROA_IN
}

# return true if invalid, false if not
function prefix_is_invalid()
{
    if net.type = NET_IP4 then
        if is_v4_rpki_invalid() then return true;
    if net.type = NET_IP6 then
        if is_v6_rpki_invalid() then return true;
    return false;
}
```

And then the *prefix_is_valid* function will be called as part of a larger outbound
example:

```
filter EXPORT
{
  if (prefix_is_bogon()) then reject "[Rejected Prefix: Bogon] ", net;
  if (prefix_is_invalid()) then reject "[Rejected Prefix: Invalid] ", net;
  if (as_path_contains_bogons()) then reject "[Rejected Prefix: Bogon
net;
  if (bgp_path.len > 32) then reject "[Rejected Prefix: Long AS] ", net
  if (bgp_path.len < 1) then reject "[Rejected Prefix: No AS] ", net;
  accept "[Exported Prefix] ", net;
}
```

FRrouting Hints

This section shows the basic configuration needed to get route origin

implementations using FRrouting (FRR). The commentary below assume
FRR (8.1 and 7.5) have slightly different CLI and may not have all the

Configuration with Validator

The configuration needed for FRR to talk with a validator is:

```
rpki
 rpki polling_period 3600
 rpki cache <ip address1> <port1> preference 1
 rpki cache <ip address2> <port2> preference 2
exit
```

Two validators are configured in this example, the preferred one has p
validator (used when the primary has become unreachable) is preferen
following this principle.

FRR automatically populates the BGP RIB with validation status of eac
intervention is needed. To find out what is in the validation database:

```
show rpki prefix-table
```

and to find out the status of the connection to the active validator:

```
show rpki cache-connection
```

To show the BGP RIB with the validation information now flagged, just

```
show bgp
```

command set. To show prefixes that meet any of the validation states:

```
show bgp rpki valid
```

will show all the entries that are *valid*. There is also a command for *invalid* and *notfound*.

Implementing Route Origin Validation

The final step in FRR is to implement Route Origin Validation. This ne
applied to an *outbound* policy on all BGP sessions (internal and external). The f
route-map exporting IPv4 routes. Similar can be done for IPv6.

```
route-map EXPORT deny 5
 description Don't send RPKI Invalids
 match rpki invalid
exit
!
route-map EXPORT deny 15
```

```
 description Drop the IPv4 bogons
 match ip address prefix-list v4bogon
exit
!
route-map EXPORT permit 20
 description Everything else is good
exit
!
```

Back to Home page

From:
https://bgp4all.com.au/pfs/ Philip Smith's Internet Development

Permanent link:
https://bgp4all.com.au/pfs/hints/rpki?rev=1784559286

Last update: 2026/07/20 14:54