RTRlib Handbook

Release 0.4

Sebastian Meiling, Marcel Röthke, Colin Sames

Feb 03, 2019

Contents

This is the official User Handbook of the RTRlib, it provides guidance on how to use the library for development and gives an overview on a variety of tools that utilize the RTRlib. Further information can be found on the RTRlib website[1] and its source code repository on Github[2].

[1] Project website – https://rtrlib.realmv6.org
[2] Source code on Github – https://github.com/rtrlib/rtrlib

CHAPTER 1

About

1.1 In a Nutshell

RTRlib is a C library that implements the client side of the RPKI-RTR protocol as well as route origin validation. Basically, it maintains data from an RPKI cache server (e.g., Routinator) and allows to verify whether an autonomous system (AS) is the legitimate origin AS, based on the fetched valid ROA data. It is prepared for BGPsec path validation.

RTRlib powers RPKI in BGP software routers such as FRR and is the base for monitoring tools. A Python binding is available. The basis RTRlib package includes the library and lightweight command line tools.

1.2 Why do I need the RTRlib?

RTRlib gives easy and highly efficient access to cryptographically valid RPKI data without relying on a specific cache server or RPKI validator implementation. To prevent single point of failures, it handles failover between multiple cache servers.

Not only developers of routing software but also network operators benefit from RTRlib. Developers can integrate the RTRlib into their BGP daemon to extend their implementation towards RPKI. Network operators may use the RTRlib to develop monitoring tools (e.g., to evaluate the performance of caches or to validate BGP data).

1.3 License

This software is free, open source and licensed under MIT.

1.4 Supported RFCs

The current version implements RFC 6811 and RFC 8210.

1.5 Community

If you run into a problem with RTRlib or you have a feature request, please create an issue on Github. We are also happy to accept your pull requests. For general discussion and exchanging operational experiences we provide a mailing list. More details about RTRlib are available on the project website.

CHAPTER 2

Background

The global deployment of a *Resource Public Key Infrastructure* (RPKI *[1]*) is one step towards securing the Internet routing through cryptography. The RPKI allows the holder of a distinct IP prefix to authorize certain autonomous systems (AS) to originate corresponding routes, which is cryptographically verifiable through *Route Origination Authorizations* (ROAs) that are stored in the RPKI.

RPKI enabled routers do not store such ROAs themselves, but only the validated content of those. To achieve high scalability as well as limit resource utilization on BGP routers, the validation of ROAs is performed by trusted RPKI cache servers, which are deployed at the network operator site. The RPKI-RTR protocol defines a standard mechanism to maintain exchange of the prefix origin AS relations between the cache server and routers. In combination with a BGP prefix origin validation scheme a router is able to verify received BGP updates without suffering from cryptographic complexity.

The RTRlib is a lightweight C library that implements the RPKI-RTR protocol for the client end (i.e., routers) and the proposed prefix origin validation scheme. The RTRlib provides functions to establish a connection to a single or multiple trusted caches using TCP and SSH transport connections, and further allows to determine the validation state of prefix to origin AS relations.

Fig. 2.1 shows a typical RPKI deployment, where trusted cache servers collect ROAs from global RPKI repositories of the RIRs, such as RIPE and APNIC. Each local RPKI cache periodically updates and verifies the stored ROAs, and pushes the preprocessed data to connected RPKI enabled BGP routers using the RTR protocol.

2.1 Further Reading

Detailed insights on the implementation of the RTRlib and its performance can be found in *[2]*. Further information is available in the standard specifications and protocols in RFCs 6810 *[3]* and 6811 *[4]*, to which the RTRlib complies. Even more background material on BGP security extensions can be found in *[5]*, *[6]*, and *[7]*

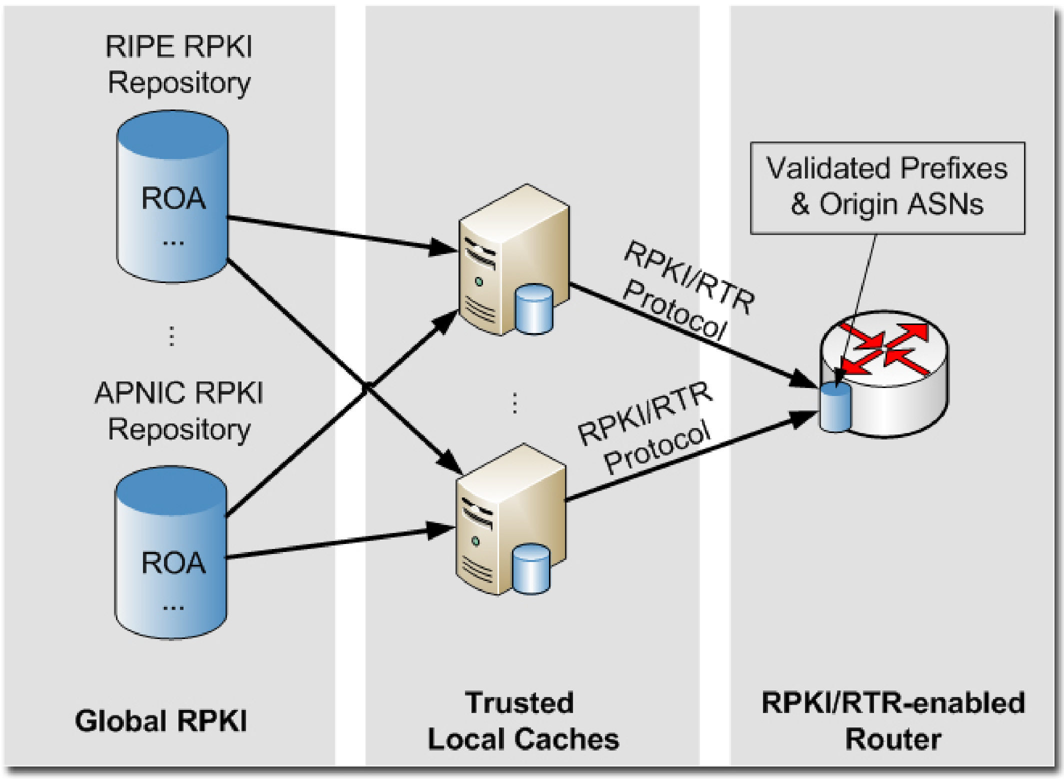

Fig. 2.1: Overview on a typical RPKI deployment, showing global RPKI repositories, trusted cache servers, and RPKI enabled BGP routers.

CHAPTER 3

Installation

Most Linux distributions as well as Apple macOS support RTRlib. The RTRlib software package includes the library and basic ready-to-use command line tools that show some of the RTRlib features.

3.1 Apple macOS

For macOS we provide a *Homebrew tap* to easily install the RTRlib. First, setup Homebrew[1] and then install the RTRlib package:

```
brew tap rtrlib/pils
brew install rtrlib
```

3.2 Archlinux

For Archlinux we maintain two PKGBUILDs in the Archlinux User Repository, `rtrlib`[2] and `rtrlib-git`[3]. `rtrlib` includes the latest official RTRlib release, `rtrlib-git` includes the current git master.

You can either use your favourite `aur` helper or execute the following commands:

```
sudo pacman --needed base-devel

# for the latest release
wget https://aur.archlinux.org/cgit/aur.git/snapshot/rtrlib.tar.gz
tar xf rtrlib
cd rtrlib
```

(continues on next page)

[1] Homebrew – http://brew.sh
[2] https://aur.archlinux.org/packages/rtrlib/
[3] https://aur.archlinux.org/packages/rtrlib-git/

(continued from previous page)

```
# for the git version
wget https://aur.archlinux.org/cgit/aur.git/snapshot/rtrlib-git.tar.gz
tar xf rtrlib-git
cd rtrlib-git

# for both
makepkg -sci
```

3.3 Debian

RTRlib is part of the official Debian package repository since Buster[4] and can be installed using `apt`. The following packages are available:

librtr0 includes the basis library.

librtr0-dev includes header files etc. for developers.

rtr-tools includes basic command line tools based on RTRlib.

librtr0-dbgsym includes debugging symbols.

librtr-doc includes offline documentation.

To install the minimal set of packages required for development, execute the following command:

```
apt install librtr0 librtr-dev
```

If you just want to use the RTRlib command line tools, run

```
apt install librtr0 rtr-tools
```

3.4 Gentoo

The FRR routing project maintains a gentoo overlay[5] that contains an `ebuild` for the RTRlib. First, setup `layman`[6], then install rtrlib with the following commands:

```
# If this doe not work try layman -f
layman -a frr-gentoo
emerge rtrlib
```

3.5 From Source

The source code repository of RTRlib includes everything that you need to implement or run applications based on the RTRlib, and to use the RTRlib command line tools.

The RTRlib source code consists of the following subdirectories:

[4] Buster is currently in testing and scheduled for release Mid 2019.
[5] https://github.com/FRRouting/gentoo-overlay
[6] https://wiki.gentoo.org/wiki/Layman

- `cmake/` CMake modules
- `doxygen/` Example code and graphics used in the Doxygen documentation
- `rtrlib/` Header and source code files of the RTRlib
- `tests/` Function tests and unit tests
- `tools/` Contains `rtrclient` and `rpki-rov`

3.5.1 Getting Started

To build and install the RTRlib from source, you need the following common software:

`cmake` version >= 2.6 to build the system.

`libssh` version >= 0.5.0 to establish SSH transport connections (optional but highly recommended).

Additional optional requirements are:

`cmocka` to run RTRlib unit tests

`doxygen` to build the RTRlib API documentation

3.5.2 Building

The easiest way to get the source code is to download either the latest RTRlib release from https://github.com/rtrlib/rtrlib/releases/latest or the current master from https://github.com/rtrlib/rtrlib/archive/master.zip, and then unpack:

```
unzip rtrlib-master.zip
cd rtrlib-master
# or alternatively, clone the current git master
git clone https://github.com/rtrlib/rtrlib/
cd rtrlib
```

Then, build the library and command line tools using `cmake`. We recommend an *out-of-source* build:

```
# inside the main RTRlib source code directory
mkdir build && cd build
cmake -D CMAKE_BUILD_TYPE=Release ../
make
sudo make install
```

To enable debug symbols and messages, change the `cmake` command to:

```
cmake -D CMAKE_BUILD_TYPE=Debug ../
```

If the build command fails with any error, please consult the RTRlib README[7] and Wiki[8], you may also join our *mailing list*[9] or open an issue on Github[10].

[7] README – https://github.com/rtrlib/rtrlib/blob/master/README
[8] Wiki – https://github.com/rtrlib/rtrlib/wiki
[9] Mailing list – https://groups.google.com/forum/#!forum/rtrlib
[10] Issue tracker – https://github.com/rtrlib/rtrlib/issues

3.5.3 Additional `cmake` Options and Targets

If you did not install `libssh` in the default directories, you can run `cmake` with the following parameters:

```
-D LIBSSH_LIBRARY=<path-to-libssh.so>
-D LIBSSH_INCLUDE=<include-directory>
```

To configure explicitly a directory where to place the RTRlib during installation, you can pass the following argument to `cmake`:

```
-D CMAKE_INSTALL_PREFIX=<path>
```

For developers, we provide a pre-build API documentation online[11] which documents the API of the latest release. Alternatively, and if `Doxygen` is available on your system, you can build the documentation locally as follows:

```
make doc
```

To execute the build-in tests provided by the RTRlib package, run:

```
make test
```

[11] API reference – https://rtrlib.realmv6.org/doxygen/latest

CHAPTER 4

Development with RTRlib

4.1 Overview

The RTRlib shared library is installed to `/usr/local/lib` by default, and its headers files to `/usr/local/include`, respectively. To write an application in C/C++ using the RTRlib, include the main header file into the code:

```
#include "rtrlib/rtrlib.h"
```

The name of the corresponding shared library is *rtr*. To link an application against the RTRlib, pass the following parameter to the compiler:

```
-lrtr
```

If the linker reports an error such as `cannot find -lrtr`, probably the RTRlib was not installed to a standard location. In this case, pass its location as an absolute path to the compiler, add parameter:

```
-L</path/to/librtr/>
```

On Linux you can alternatively try to update the linker cache instead, run:

```
ldconfig
# verify with
ldconfig -p | grep rtr
```

4.2 Step-by-Step Example

The RTRlib package includes two command line tools, the `rtrclient` and the `rpki-rov`, see also tools. The former connects to a single RTR cache server via TCP or SSH and prints validated prefix origin data to STDOUT. You can use this tool to get first experiences with the RPKI-RTR protocol. With the latter you can validate arbitrary prefix origin AS relations against records received from a connected RPKI cache. Both tools are located in the `tools/` directory. Having a look into the source code of these tools will help to understand and integrate the RTRlib into applications.

Any application using the RTRlib will have to setup a RTR connection manager that handles synchronization with one (or multiple) trusted RPKI cache server(s). The following provides an overview on important code segments.

First, create a RTR transport socket, for instance using TCP as shown in Listing 4.1.

Listing 4.1: Create a RTR transport socket

```
struct tr_socket tr_tcp;
struct rtr_socket rtr_tcp;
char tcp_host[] = "rpki-validator.realmv6.org";
char tcp_port[] = "8282";

struct tr_tcp_config tcp_config = {
    tcp_host,    // cache server host
    tcp_port,    // cache server port
    NULL         // source address, empty
};

tr_tcp_init(&tcp_config, &tr_tcp);
rtr_tcp.tr_socket = &tr_tcp;
```

Afterwards, create a group of RTR cache servers with preference *1*. In this example (see Listing 4.2), it includes only a single cache instance.

Listing 4.2: Create a group of RTR caches

```
rtr_mgr_group groups[1];
groups[0].sockets = malloc(sizeof(struct rtr_socket*));
groups[0].sockets_len = 1;
groups[0].sockets[0] = &rtr_tcp;
groups[0].preference = 1;
```

Now initialize the RTR connection manager (Listing 4.3) providing a pointer to a configuration object, the preconfigured group(s), number of groups, a refresh interval, an expiration interval, and retry interval, as well as distinct callback functions. In this case, a refresh interval of 30 seconds, a 600s expiration timeout, and a 600s retry interval will be defined. Afterwards, start the RTR Connection Manager.

Listing 4.3: Initialize the RTR connection manager.

```
struct rtr_mgr_config *conf;
int ret = rtr_mgr_init(&conf, groups, 1, 30, 600, 600,
                       pfx_update_fp, spki_update_fp, status_fp, NULL);

rtr_mgr_start(conf);
```

As soon as an update has been received from the RTR-Server, the callback function will be invoked. In this example, *update_cb* (see Listing 4.4) is called which prints the prefix, its minimum, and maximum length, as well as the corresponding origin AS.

Listing 4.4: RTR connection manager update callback

```
static void update_cb(struct pfx_table* p, const pfx_record rec, const bool added){
    char ip[INET6_ADDRSTRLEN];
    if(added)
        printf("+ ");
    else
        printf("- ");
    ip_addr_to_str(&(rec.prefix), ip, sizeof(ip));
    printf("%-18s %3u-%-3u %10u\n", ip, rec.min_len, rec.max_len, rec.asn);
}
```

With a running RTR connection manager, you can also execute validation queries. For instance, validate the relation of prefix *10.10.0.0/24* and its origin AS 12345 as shown in Listing 4.5.

Listing 4.5: Validate a prefix to origin AS relation

```
struct lrtr_ip_addr pref;
lrtr_ip_str_to_addr("10.10.0.0", &pref);
enum pfxv_state result;
const uint8_t mask = 24;
rtr_mgr_validate(conf, 12345, &pref, mask, &result);
```

For a clean shutdown and exit of the application, first stop the RTR Connection Manager, and secondly release any memory allocated (see Listing 4.6).

Listing 4.6: RTR connection manager cleanup

```
rtr_mgr_stop(conf);
rtr_mgr_free(conf);
free(groups[0].sockets);
```

4.3 Complete RTRlib Example

The code in Listing 4.7 shows a fully functional RPKI validator using the RTRlib. It includes all parts explained in the previous section, and shows how to setup multiple RPKI cache server connections using either TCP or SSH transport sockets. For the latter, the RTRlib has to be build and installed with *libssh* support.

Listing 4.7: A complete code example for the RTRlib

```
#include <stdio.h>
#include <stdlib.h>
#include "rtrlib/rtrlib.h"

int main(){
    //create a SSH transport socket
    char ssh_host[]      = "123.231.123.221";
    char ssh_user[]      = "rpki_user";
    char ssh_hostkey[]   = "/etc/rpki-rtr/hostkey";
    char ssh_privkey[]   = "/etc/rpki-rtr/client.priv";
    struct tr_socket tr_ssh;
```

(continues on next page)

(continued from previous page)

```
    struct tr_ssh_config config = {
        ssh_host,       //IP
        22,             //Port
        NULL,           //Source address
        ssh_user,
        ssh_hostkey,    //Server hostkey
        ssh_privkey,    //Private key
    };
    tr_ssh_init(&config, &tr_ssh);

    //create a TCP transport socket
    struct tr_socket tr_tcp;
    char tcp_host[] = "rpki-validator.realmv6.org";
    char tcp_port[] = "8282";

    struct tr_tcp_config tcp_config = {
        tcp_host, //IP
        tcp_port, //Port
        NULL      //Source address
    };
    tr_tcp_init(&tcp_config, &tr_tcp);

    //create 3 rtr_sockets and associate them with the transprort sockets
    struct rtr_socket rtr_ssh, rtr_tcp;
    rtr_ssh.tr_socket = &tr_ssh;
    rtr_tcp.tr_socket = &tr_tcp;

    //create a rtr_mgr_group array with 2 elements
    struct rtr_mgr_group groups[2];

    //The first group contains both TCP RTR sockets
    groups[0].sockets = malloc(sizeof(struct rtr_socket*));
    groups[0].sockets_len = 1;
    groups[0].sockets[0] = &rtr_tcp;
    groups[0].preference = 1;       //Preference value of this group

    //The seconds group contains only the SSH RTR socket
    groups[1].sockets = malloc(1 * sizeof(struct rtr_socket*));
    groups[1].sockets_len = 1;
    groups[1].sockets[0] = &rtr_ssh;
    groups[1].preference = 2;

    //create a rtr_mgr_config struct that stores the group
    struct rtr_mgr_config *conf;

    //initialize all rtr_sockets in the server pool with the same settings
    int ret = rtr_mgr_init(&conf, groups, 2, 30, 600, 600, NULL, NULL, NULL, NULL);

    //start the connection manager
    rtr_mgr_start(conf);

    //wait till at least one rtr_mgr_group is fully synchronized with the server
    while(!rtr_mgr_conf_in_sync(conf)) {
        sleep(1);
```

(continues on next page)

(continued from previous page)

```
    }

    //validate the BGP-Route 10.10.0.0/24, origin ASN: 12345
    struct lrtr_ip_addr pref;
    lrtr_ip_str_to_addr("10.10.0.0", &pref);
    enum pfxv_state result;
    const uint8_t mask = 24;
    rtr_mgr_validate(conf, 12345, &pref, mask, &result);

    //output the result of the prefix validation above
    //to showcase the returned states.
    char buffer[INET_ADDRSTRLEN];
    lrtr_ip_addr_to_str(&pref, buffer, sizeof(buffer));

    printf("RESULT: The prefix %s/%i ", buffer, mask);
    switch(result) {
        case BGP_PFXV_STATE_VALID:
            printf("is valid.\n");
            break;
        case BGP_PFXV_STATE_INVALID:
            printf("is invalid.\n");
            break;
        case BGP_PFXV_STATE_NOT_FOUND:
            printf("was not found.\n");
            break;
        default:
            break;
    }

    // cleanup before exit
    rtr_mgr_stop(conf);
    rtr_mgr_free(conf);
    free(groups[0].sockets);
    free(groups[1].sockets);
}
```

CHAPTER 5

RTRlib Python Binding

The RTRlib is also available for scripting in Python using the *RTRlib Python binding*[1]. This section gives a quick overview on the usage of the Python binding. An even more detailed documentation on the API and further usage examples can be found on the corresponding *readthedocs.io*[2] page.

5.1 Installation

The RTRlib Python binding runs on Linux and Apple macOS like the C library. It supports both Python 2 and Python 3, in any recent release, detailed requirements and install instructions are described here.

5.1.1 Getting Started

The Python binding for the RTRlib has several dependencies. For compilation it requires the following external packages to be installed:

- Python, version 2.7 or 3.x
- C Compiler
- RTRlib C library

To use the Python binding, the following Python packages have to be installed as well:

- cffi>=1.4.0
- enum34
- six

If you are using *virtualenv*, these are installed automatically during the install step, otherwise you have to use your platforms package management tool or just run `pip install -r requirements.txt`.

[1] RTRlib Python binding – https://github.com/rtrlib/python-binding
[2] ReadTheDocs – https://python-rtrlib.readthedocs.io

5.1.2 Building and Installation

The setup process of the RTRlib Python binding is straight forward and complies to well-known Python standards. First, download the source code from Github:

```
git clone https://github.com/rtrlib/python-binding.git
cd python-binding
```

And second, build and install the package using Python commands:

```
python setup.py build
python setup.py install
```

5.2 Step-by-Step Example

The following code listings show how to implement a simple RPKI validator based on the RTRlib Python binding. The functionality basically reflects the `rpki-rov` tool shipped with the RTRlib C library (see *RTRlib ROV Validator*).

First, import required Python packages as shown in Listing 5.1; namely *rtrlib*, but also some future imports in case of Python 2.

Listing 5.1: Import RTRlib package

```python
# uncomment future imports, required for Python 2
#from __future__ import print_function
from rtrlib import RTRManager, PfxvState
```

Afterwards, initialized and start an instance of the *RTRManager*, see Listing 5.2, mandatory parameters are *host* and *port* of a trusted RPKI cache server.

Listing 5.2: Setup and run RTRManager

```python
mgr = RTRManager('rpki-validator.realmv6.org', 8282)
mgr.start()
```

As soon as the *RTRManager* is up and running, it can validate any prefix to origin AS relation as shown in Listing 5.3. The return value in result contains the corresponding validation state, i.e., *valid*, *invalid*, or *not_found*; other return values indicate an error during validation.

Listing 5.3: Validate prefix to origin AS relation

```python
result = mgr.validate(12345, '10.10.0.0', 24)
if result == PfxvState.valid:
    print('Prefix Valid')
elif result == PfxvState.invalid:
    print('Prefix Invalid')
elif result == PfxvState.not_found:
    print('Prefix not found')
else:
    print('Invalid response')
```

CHAPTER 6

RTRlib Command Line Tools

The RTRlib software package includes two lightweight command line tools to showcase some of the RTRlib features. `rtr-client` connects to an RPKI cache server, fetches and maintains the valid ROA payloads, and prints the received data. `rpki-rov` allows to verify whether an autonomous system is the legitimate origin AS of an IP prefix, based on RPKI data.

If you want to use these command line tools, you need an RPKI-RTR connection to an RPKI cache server (e.g., Routinator). For those who do not have access to a cache server, we provide a public cache with *hostname* `rpki-validator.realmv6.org` and *port* `8282`.

6.1 RTRlib RTR Client

`rtrclient` is part of the default RTRlib software package. This command line tool connects to an RPKI cache server and prints the received valid ROA payloads to standard out.

To establish a connection to RPKI cache servers, the client can use *TCP* or *SSH* transport sockets. To run the program you have to specify the transport protocol as well as the hostname and port of the RPKI cache server; additionally you can set several options. To get a complete reference over all options for the command simply run `rtrclient` in a shell.

Listing 6.1 shows how to connect the `rtrclient` to a cache server as well as 10 lines of the resulting output. It shows IPv4 and IPv6 prefixes secured by ROAs, the allowed prefix lengths, and the legitimate origin AS numbers. Each line represents either a ROA that was added (+) or removed (−) from the selected RPKI cache server. The RTRlib client will receive and print such updates until the program is terminated, i.e., by `ctrl + c`.

Listing 6.1: Output of the rtrclient tool.

```
rtrclient tcp -k -p rpki-validator.realmv6.org 8282
Prefix                                      Prefix Length         ASN
+ 89.185.224.0                                 19 -  19         24971
+ 180.234.81.0                                 24 -  24         45951
+ 37.32.128.0                                  17 -  17        197121
```

(continues on next page)

(continued from previous page)

```
+ 161.234.0.0                                          16 -  24          6306
+ 85.187.243.0                                         24 -  24         29694
+ 2a02:5d8::                                           32 -  32          8596
+ 2a03:2260::                                          30 -  30        201701
+ 2001:13c7:6f08::                                     48 -  48         27814
+ 2a07:7cc3::                                          32 -  32         61232
+ 2a05:b480:fc00::                                     48 -  48         39126
```

6.2 RTRlib ROV Validator

`rpki-rov` is also part of the RTRlib software package. This simple command line interface allows to verify whether an autonomous system is allowed to announce a specific IP prefix, based on data received from an RPKI cache server.

To run the program, you must provide two parameters, `hostname` and `port` of a known RPKI cache server. Then, you can interactively validate IP prefixes by typing `prefix`, `prefix length`, and `origin ASN` separated by spaces. Press `ENTER` to run the validation. The result will be shown instantly below the input.

Note: `rpki-rov` can validate IPv4 and IPv6 prefixes by default.

Listing 6.2 shows the validation results of all RPKI-enabled RIPE RIS beacons. The output consists of three columns, which are separated by pipes (`|`):

`<input query> | <ROAs> | <validation result>.`

The validation results are `0` for *valid*, `1` for *not found*, and `2` for *invalid*.

In case of a *valid* and *invalid* prefix-AS pair, the output shows the matching ROAs for the given prefix and AS number. If multiple ROAs for a prefix exist, they are listed in a row separated by commas (`,`).

Listing 6.2: Output of rpki-rov showing validation results of multiple prefixes.

```
rpki-rov rpki-validator.realmv6.org 8282
93.175.146.0 24 12654
93.175.146.0 24 12654|12654 93.175.146.0 24 24|0
2001:7fb:fd02:: 48 12654
2001:7fb:fd02:: 48 12654|12654 2001:7fb:fd02:: 48 48|0
93.175.147.0 24 12654
93.175.147.0 24 12654|196615 93.175.147.0 24 24|2
2001:7fb:fd03:: 48 12654
2001:7fb:fd03:: 48 12654|196615 2001:7fb:fd03:: 48 48|2
84.205.83.0 24 12654
84.205.83.0 24 12654||1
2001:7fb:ff03:: 48 12654
2001:7fb:ff03:: 48 12654||1
```

CHAPTER 7

Third Party Tools Using RTRlib

In the following sections we give an overview on several software tools, which utilize the RTRlib and its features. These tools range from low level shell commands to easy-to-use browser plugins. For all tools we provide small usage examples; where ever appropriate we will use the *RIPE RIS Beacons*[1] (see Table 7.1) with well known RPKI validation results to show case the tool.

Table 7.1: RIPE RIS beacons for RPKI tests

IP Prefix	Valid Origin	Result
93.175.146.0/24	AS12654	valid
2001:7fb:fd02::/48	AS12654	valid
93.175.147.0/24	AS196615	invalid AS
2001:7fb:fd03::/48	AS196615	invalid AS
84.205.83.0/24	None	not found
2001:7fb:ff03::/48	None	not found

Note: for all prefixes RPKI validation results are based on origin AS 12654 that is owned by RIPE. Most examples also require a connection to a trusted RPKI cache server, for that we provide a public cache with *hostname* `rpki-validator.realmv6.org` and *port* `8282`.

7.1 RPKI Validator Browser Plugin

The RPKI Validator plugin for web browsers allows to check the RPKI validation of visited URLs, i.e., the associated IP prefix and origin AS of the URL. A small icon indicates the validation state of the visited URL, which is either valid ( ), invalid ( ), or not found ( ).

The plugin is available as an add-on (or extension) for the web browsers Firefox and Chrome . While the *Firefox add-on*[2] is available through the add-on store, Chrome users have to download and install the extension themselves as follows:

[1] https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/current-ris-routing-beacons

[2] Firefox add-on – https://addons.mozilla.org/en-US/firefox/addon/rpki-validator/

1. download the *Chrome extension*[3] from GitHub
2. open a new tab in Chrome and enter `chrome://extensions`
3. activate *Developer Mode* via the checkbox in the top right
4. click the *Load unpacked extension* button and navigate to the source

The screenshots show the results of the RPKI Validator browser plugin for Firefox (*valid* Fig. 7.1, *invalid* Fig. 7.2, and *not found* Fig. 7.3) for certain websites .

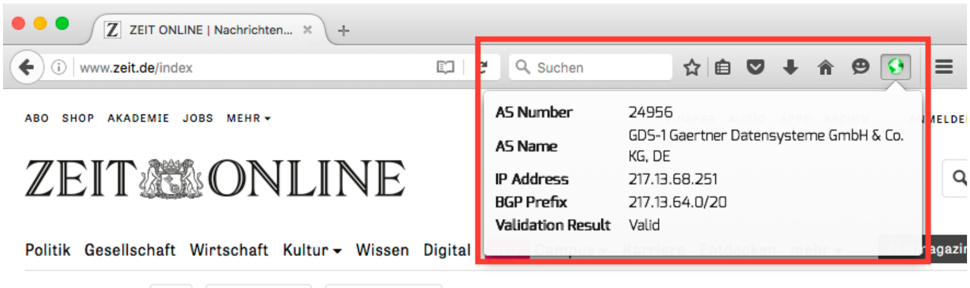

Fig. 7.1: Screenshot of RPKI Validator plugin in Firefox showing result *valid*.

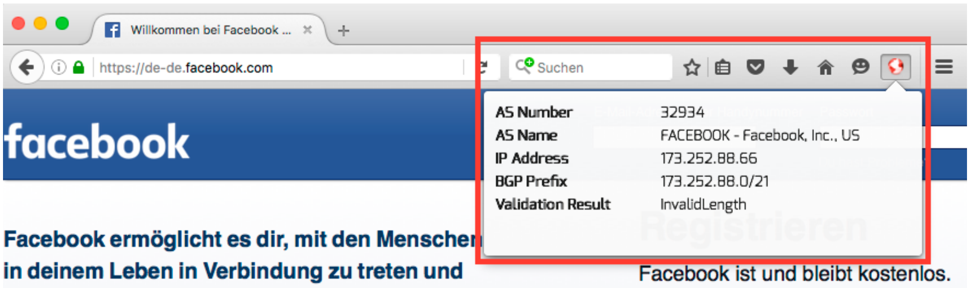

Fig. 7.2: Screenshot of RPKI Validator plugin in Firefox showing result *invalid*.

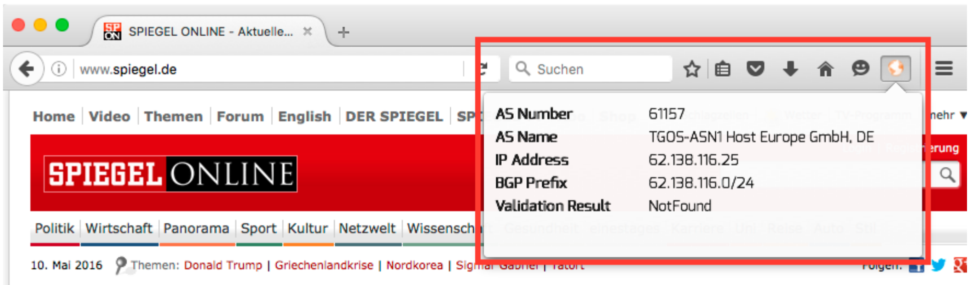

Fig. 7.3: Screenshot of RPKI Validator plugin in Firefox showing result *not found*.

7.2 RPKI READ

The *RPKI Realtime Dashboard* (*RPKI READ*[4]) aims to provide a consistent (and live) view on the RPKI validation state of currently announced IP prefixes. That is, it verifies relation of an IP prefix and its BGP origin AS (autonomous system) utilizing the RPKI.

The RPKI READ monitoring system has two parts:

[3] Chrome Extension – https://github.com/rtrlib/chrome-extension
[4] RPKI READ – https://rpki-read.realmv6.org/

1. the backend, storing latest validation results in a database, and
2. the (web) frontend, displaying these results as well as an overview of statistics derived from them.

The backend connects to a live BGP stream, e.g. of a BGPmon[5] instance or via BGPstream[6]. It then parses received BGP messages and extracts IP prefixes and origin AS information. These prefix to origin AS relations are validated using the RTRlib validator to query a trusted RPKI cache server.

The RPKI READ frontend presents a dashboard like interface showing a live overview of the RPKI validation state of all currently advertised IP prefixes observed by a certain BGP source (see Fig. 7.4). Further, the frontend provides detailed statistics and also allows the user to search for validation results of distinct prefixes or all prefixes originated by a certain AS.

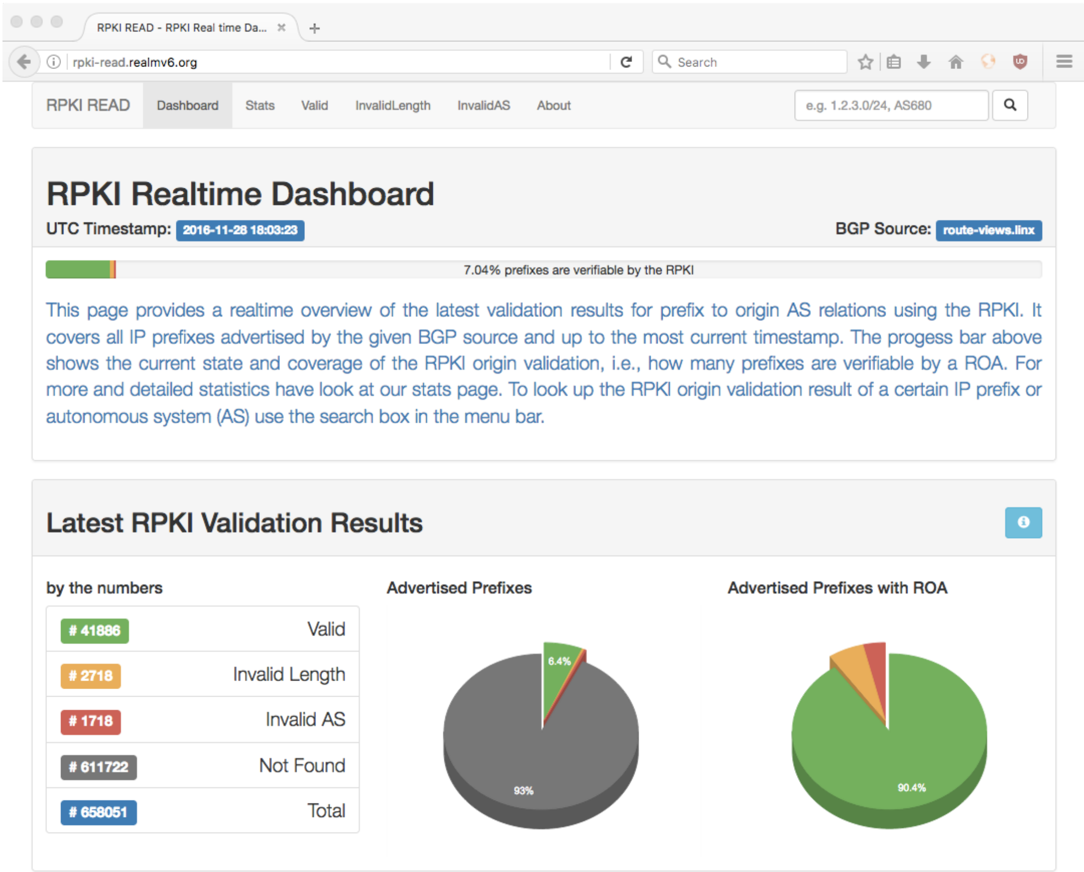

Fig. 7.4: Screenshot of the RPKI READ web frontend

7.3 RPKI MIRO

The RPKI *Monitoring and Inspection of RPKI Objects* (*RPKI MIRO*[7]) aims for easy access to RPKI certificates, revocation lists, ROAs etc. to give network operators more confidence in their data. Though, RPKI is a powerful tool, its success depends on several aspects. One crucial piece is the correctness of the RPKI data. Though, the RPKI data is public, it still might be hard to inspect outside of shell-like environments.

[5] BGPmon – http://www.bgpmon.io/
[6] BGPstream – https://bgpstream.caida.org/
[7] RPKI MIRO – http://rpki-miro.realmv6.org/

The main objective of RPKI MIRO is to provide an extensive but painless insight into the published RPKI content. RPKI MIRO is a monitoring application that consists of three parts:

1. standard functions to collect RPKI data from remote repositories,
2. a browser to visualize RPKI objects, and
3. statistical analysis of the collected objects.

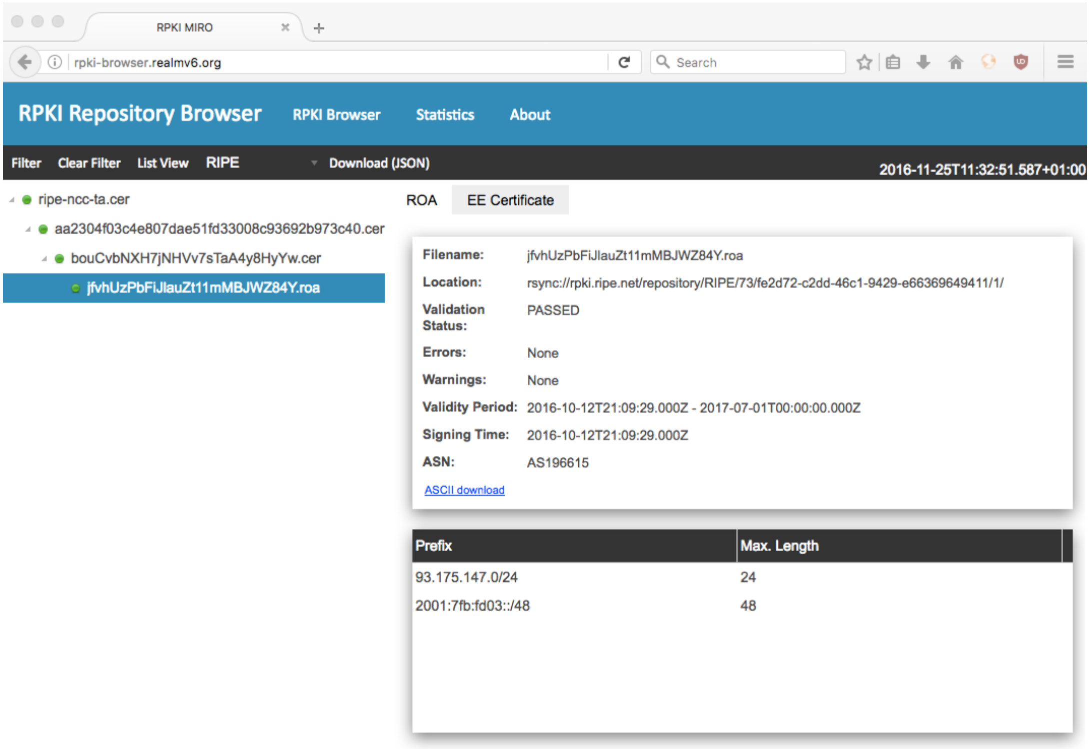

Fig. 7.5: Screenshot of the RPKI MIRO web interface.

Using RPKI MIRO you can lookup any IP prefix and its associated ROA, e.g. the RIPE RIS beacon `93.175.147.0/24`. Open a browser and goto URL http://rpki-browser.realmv6.org, in the menu switch from `AFRINIC` to `RIPE` and set a filter for the prefix `93.175.147.0/24` with attribute `resource`. Expand the ROA tree view on the left side to get the corresponding ROA for the beacon prefix, the resulting web view should look like the screenshot in Fig. 7.5.

7.4 RPKI RBV

The RPKI *RESTful BGP Validator* (*RPKI RBV*[8]) is web application that provides a RESTful API to validate IP prefix to origin AS relations. The validation service can be accessed via a plain and simple web page (see also Fig. 7.6) or directly using its RESTful API.

RBV provides two distinct APIs to run RPKI validation queries, the APIs allow RESTful GET queries with the following syntax and formatting of the URL path:

1. `/api/v1/validity/<asn>/<prefix>/<masklen>`
2. `/api/v2/validity/<host>`

[8] RPKI RBV – https://rpki-rbv.realmv6.org/

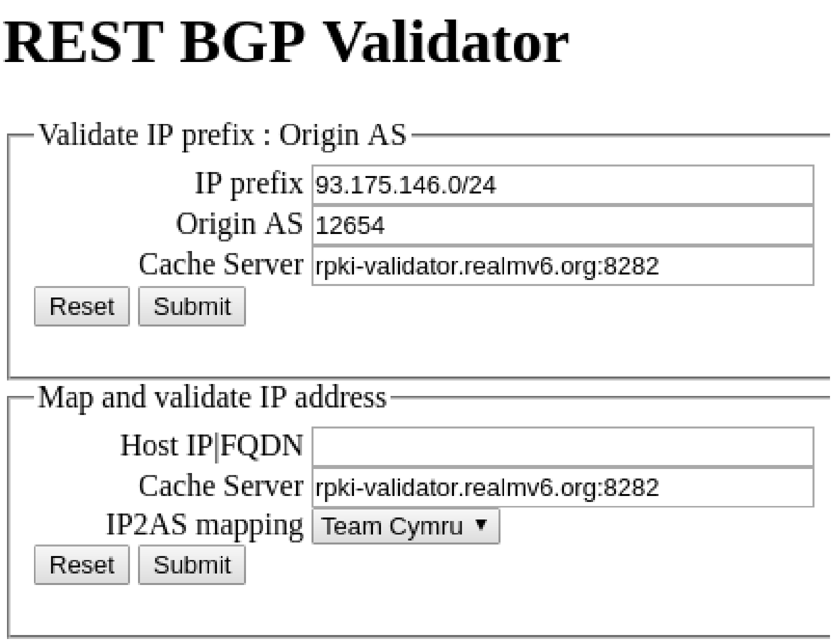

Fig. 7.6: Screenshot of the RPKI RBV web interface

Note: the AS number in `<asn>` has to be prepended with *AS*; and `<host>` can either be an IP address or a DNS hostname. To test the APIs type the following queries for the RIPE RIS beacon `93.175.146.0/24` into the address bar of your favorite web browser:

```
rpki-rbv.realmv6.org/api/v1/validity/AS12654/93.175.146.0/24
rpki-rbv.realmv6.org/api/v2/validity/93.175.146.1
```

The result will be a JSON object as shown in Listing 7.1.

Listing 7.1: Sample JSON output of RPKI RBV


```
{
    "validated_route": {
        "info": {
            "origin_country": "EU",
            "origin_asname": "RIPE-NCC-RIS-AS Reseaux IP Europeens Network Coordination Centre (RIPE NCC), EU"
        },
        "route": {
            "prefix": "93.175.146.0/24",
            "origin_asn": "AS12654"
        },
        "validity": {
            "state": "Valid",
            "code": 0,
            "description": "At least one VRP Matches the Route Prefix",
            "VRPs": {
                "unmatched_as": [],
                "unmatched_length": [],
                "matched": [{
                    "prefix": "93.175.146.0/24",
                    "max_length": "24",
                    "asn": "AS12654"
```


(continues on next page)

(continued from previous page)

```
                        } ]
                    }
                }
            }
        }
```

For detailed instruction how to install and set up the API visit the *RBV Repository* on GitHub[9].

7.5 Other Third-Party Tools

The *RIPE Tools and Resources*[10] webpage provides an (almost) complete overview on other tools related to RPKI and BGP security, in general.

[9] RPKI RBV Github – https://github.com/rtrlib/rbv

[10] https://www.ripe.net/manage-ips-and-asns/resource-management/certification/tools-and-resources/

CHAPTER 8

BGP Routing Daemons with RPKI/RTR

For several Routing Daemons such as *Quagga*[1] and *BIRD*[2] exist RPKI enabled extensions that are based on the RTRlib.

8.1 The BIRD Internet Routing Daemon

To set up BIRD, first download[3] the latest release, unzip and change into the source directory. To build BIRD, run:

```
./configure
make
make install
```

You may need to execute these and any following commands in this handbook as `sudo`. More information on the building process can be found in the README of BIRD.

Before any validations with BIRD can be done, it must be configured accordingly. First, a ROA table and the validation function must be added to `/usr/local/etc/bird.conf`. At the top of this file write:

```
roa table rtr_roa_table;

function test_ripe_beacons()
{
    print "Testing ROA";
    print "Should be TRUE TRUE TRUE:",
    " ", roa_check(rtr_roa_table, 84.205.83.0/24, 12654) = ROA_UNKNOWN,
    " ", roa_check(rtr_roa_table, 93.175.146.0/24, 12654) = ROA_VALID,
    " ", roa_check(rtr_roa_table, 93.175.147.0/24, 12654) = ROA_INVALID;
}
```

[1] Quagga – http://www.nongnu.org/quagga/
[2] BIRD – http://bird.network.cz/
[3] BIRD download – http://bird.network.cz/?download

The first line automatically creates a ROA table when the BIRD daemon is started. The function itself checks for three entries in the ROA table and prints the corresponding validity status. The BIRD socket must now be opened. In order to do that type the following command:

```
./bird -c /usr/local/etc/bird.conf -s /tmp/bird.ctl -d
```

With the option `-d` BIRD runs in the foreground. That's necessary to view the output of the `test_ripe_beacons` function. `/tmp/bird.ctl` is the location and name of the socket that will be created. It is required by the `bird-rtrlib-cli` which we will install next.

Open another new terminal. To try out whether BIRD receives actual responses, there is an IPC that runs on the BIRD socket. Clone the *BIRD-RTRlib-CLI* repository on GitHub and build it:

```
git clone https://github.com/rtrlib/bird-rtrlib-cli
cd bird-rtrlib-cli
cmake .
make
```

In case that the RTRlib was not installed into the default directory, run

```
cmake -DRTRLIB_INCLUDE=<rtrlib> -DRTRLIB_LIBRARY=</path/to/rtrlib.
↪[a|so|dylib]> .
make
```

If everything was build correctly, there now should be an executable called `bird-rtrlib-cli`. To see all the options of this program run `./bird-rtrlib-cli --help`. Now connect to the BIRD socket and receive the RPKI data with the following command:

```
./bird-rtrlib-cli -b /tmp/bird.ctl -r rpki-validator.realmv6.org:8282 -t␣
↪rtr_roa_table
```

The options do the following:

`-b`: the location of the BIRD socket.

`-r`: the address and port of the RPKI cache server. Change it if you want to use a different one.

`-t`: the table in which the gathered rpki-data is filled into. We created this one earlier in the bird.conf

After executing this line, you will see that, after establishing a connection to the cache server, the ROA entries are piped into the BIRD ROA table. Head back to the BRID directory and start the BIRD CLI with the following command:

```
sudo ./birdc -s /tmp/bird.ctl
```

All the commands of the CLI can be viewed by typing `?`. To list all the entries from the ROA table enter:

```
bird> show roa
194.3.206.0/24 max 24 as 24954
03.4.119.0/24 max 24 as 38203
```

(continues on next page)

(continued from previous page)

```
200.7.212.0/24 max 24 as 27947
200.7.212.0/24 max 24 as 19114
103.10.79.0/24 max 24 as 45951
...
```

Type `q` to exit. There will be a lot of similar output. The content of the `bird-rtrlib-cli` was successfully written to the ROA table. Search, for example, for the prefix `93.175.146.0/24` and BIRD will return the entry with its corresponding ASN.

```
bird> show roa 93.175.146.0/24
93.175.146.0/24 max 24 as 12654
```

To do the actual validation of the prefixes that were defined in `test_ripe_beacons` execute:

```
bird> eval test_ripe_beacons()
(void)
```

To see the output of the function, switch to the terminal that is running the BIRD daemon. The output will look like:

```
bird: Testing ROA
bird: Should be TRUE TRUE TRUE: TRUE TRUE TRUE
```

After seeing this line, the test function was executed and the prefixes were successfully tested.

8.2 The Quagga Routing Software Suite

The Quagga routing daemon implements IP routing via the protocols OSPF, RIP and BGP. It acts as a router that fetches and shares routing information with other routers. Quagga is mainly dedicated to BGP4. An unofficial release implements support for the RPKI so BGP updates can be verified against a ROA. Doing so requires the support of the RTRlib so Quagga can initialize a connection to a cache server using the RTR protocol.

To install Quagga, clone the Git repository and switch the branch like this:

```
git clone https://github.com/rtrlib/quagga-rtrlib.git
cd quagga-rtrlib
git checkout feature/rtrlib
```

This repository is a fork of the original and implements RPKI support. Before building it, make sure your system meets the perquisites:

- automake: 1.9.6
- autoconf: 2.59
- libtool: 1.5.22
- texinfo: 4.7
- GNU AWK: 3.1.5

If all of these packages are installed, Quagga can be build. Some steps might require `sudo` privileges:

```
./bootstrap
./configure --enable-rpki
make
make install
```

The `--enable-rpki` option tells the configure script to include the RTRlib.

Now that Quagga is built, start the BGP and Zebra daemons. Zebra acts as a process between the package stream of the kernel and daemons like BGP or OSPF. Execute `bgpd` and `zebra`:

```
./bgpd/bgpd
./zebra/zebra
```

To interact with BGPD, connect to it via `vtysh`, a command line interface that gains access to such daemons.

Bibliography

[1] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480, IETF, February 2012.

[2] Matthias Wählisch, Fabian Holler, Thomas C. Schmidt, and Jochen H. Schiller. RTRlib: An Open-Source Library in C for RPKI-based Prefix Origin Validation. In *Proc. of USENIX Security Workshop CSET'13*. Berkeley, CA, USA, 2013. USENIX Assoc. URL: https://www.usenix.org/conference/cset13/rtrlib-open-source-library-c-rpki-based-prefix-origin-validation.

[3] R. Bush and R. Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810, IETF, January 2013.

[4] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. BGP Prefix Origin Validation. RFC 6811, IETF, January 2013.

[5] S. Bellovin, R. Bush, and D. Ward. Security Requirements for BGP Path Validation. RFC 7353, IETF, August 2014.

[6] Matthew Lepinski and Sean Turner. An Overview of BGPsec. Internet-Draft – work in progress 08, IETF, June 2016.

[7] Matthew Lepinski and Kotikalapudi Sriram. BGPsec Protocol Specification. Internet-Draft – work in progress 19, IETF, November 2016.