Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

Towards a full implementation of RPKI in the mini-Internet

Semester Thesis

Author: Sandro Lutz

Tutor: Tobias Bühler, Thomas Holterbach

Supervisor: Prof. Dr. Laurent Vanbever

March to June 2021

Abstract

This thesis completes the implementation of all components of the Resource Public Key Infrastructure (RPKI) architecture into the mini-Internet platform so that it can be used for educational purposes. The auto-configuration feature is extended to cover RPKI components. The RPKI related configuration includes an extended routing policy to perform Route Origin Validation (ROV) at the routers. Additionally, some general improvements are introduced to the mini-Internet platform to reduce resource usage and startup time.

Contents

Chapter 1

Introduction

1.1 Motivation

The Internet evolved into a critical infrastructure spanning around the globe. The Border Gateway Protocol (BGP) is still an important protocol in today's Internet for routing between Autonomous Systems (ASes) although it was not designed with security in mind and has many flaws and vulnerabilities [1]. The Resource Public Key Infrastructure (RPKI) was created to address some of those flaws. In recent years, the number of ASes issuing Route Origin Authorizations (ROAs) has increased significantly. This indicates the increasing importance of RPKI for network operators [2].

Most routing incidents are caused by misconfiguration. During the famous incident in 2008 the Pakistani Telecom hijacked some of YouTube's IP addresses. They started to announce an IP prefix belonging to YouTube with a longer prefix and attracted traffic destined to YouTube from everywhere on the globe [3]. Another massive incident was observed in April 2021 where more than 31'000 routes were hijacked within a few minutes [4]. RPKI can reduce the impact of such events even with partial deployment of Route Origin Validation (ROV).

The mini-Internet is a platform developed at the Networked Systems Group at ETH Zürich. Its primary purpose is to teach the students how the Internet works in practice. The platform creates a virtual Internet built with Docker containers which represent various devices such as switches, routers, hosts and others. The groups of students become operators of their own virtual AS and configure the devices accordingly. The students can learn the various protocols and networking knowledge in an interactive way. They can observe the effect of their own actions in the virtual environment [5]. While there has been some prior work in order to support RPKI with the mini-Internet platform, it was not in a finished state and could not be used for lectures [6].

1.2 Task and goals

The objective of this thesis was to complete the implementation of RPKI in the mini-Internet platform which was already started in another semester thesis [6] so that it can be used by students projects like the routing project during the Communication Networks lecture.

1. Solve issues observed with the previous thesis which prevented the intended approach from working.
2. Automate the configuration of all components of RPKI during startup of the mini-Internet based on the configuration files which define the topology.

3. Create a default routing policy to be configured when Route Origin Validation is used.
4. Provide a workflow for the students working on the Routing Project which is part of the Communication Networks lecture at ETH Zürich.

1.3 Overview

Chapter 2 introduces the concepts behind the used technologies starting with BGP and RPKI in Section 2.1 and 2.2. Afterwards we describe some software which was used for the RPKI implementation. We introduce the concepts of the mini-Internet platform in Section 2.4. In Section 2.5 we show some related work. We discuss the implementation in Chapter 3. In Section 3.1 we discuss how the implementation works in general. Afterwards we describe the technical details and challenges we faced in Section 3.2. Then we show an example workflow for the students during the Routing Project in Section 3.3. In Section 4.1 of Chapter 4 we verify whether the RPKI implementation is functional. Then we show simulations of some hijack attacks in Section 4.2. In Chapter 5 we summarise the thesis and show some ideas for further development of the mini-Internet platform.

Chapter 2

Background and Related Work

Besides the main subject – Resource Public Key Infrastructure (RPKI) – understanding of the mini-Internet is very important for this thesis. We will briefly explain the relevant aspects of both in this chapter. Afterwards, we will explain some software applications needed during this thesis.

2.1 BGP

The Border Gateway Protocol (BGP) is a distributed protocol used as a standard to enable routing between ASes in today's Internet.

Every AS sends announcements to its neighbors for every IP prefix the AS can reach. An announcement contains a network prefix (e.g. 129.132.0.0/16), the AS path and some more metrics. The AS path is a list of AS numbers. The last number is the AS where the announcement originated. Whenever the announcement is forwarded by an AS to another, the forwarding AS prepends its own number to the AS path. If all connected ASes are configured correctly, after some time, every AS should have the information how to reach every device it is somehow connected to.

Issues with BGP

Today's Internet routing heavily depends on BGP. The protocol is highly flexible but it also has some fundamental flaws. Every AS can decide on its own which announcements should be forwarded to its neighbors and which announcements received from its neighbors it wants to accept. The downside of the protocol is that it does not have any mechanisms that protect against a variety of attacks and mistakes. Most relevant for this thesis are BGP route hijacking attacks.

BGP route hijacking takes place when an AS creates an announcement for an IP prefix it does not own so that other ASes route the traffic for that IP prefix to the wrong AS. Such an event can happen unintentionally by a configuration mistake or intentionally by a malicious actor. The motivations and its consequences are manifold. The following list shows some effects a BGP route hijack can have:

- Traffic has a higher latency caused by a longer path. This can have a severe impact on real-time applications.
- When a malicious actor eavesdrops on specific traffic the latency might also be affected. The hijacking AS needs at least one route which is not affected by the hijacking where the traffic can be forwarded so that it is delivered to the intended destination.

- If there is no route from the hijacking AS to the original AS, this produces a dead end (black hole). The traffic is just dropped and never gets delivered to its intended destination. This is a kind of Denial-of-Service attack as a complete IP prefix is not reachable anymore from a region or the whole world.
- Traffic gets routed through a weak link not prepared to handle the amount of traffic which results in many/all dropped packets. This is another Denial-of-Service attack.

More details about possible attacks and other security implications can be found in RFC 4272 [1].

2.2 RPKI

The Resource Public Key Infrastructure (RPKI) is an important addition to the Internet ecosystem to partially address the issues described in Section 2.1. The architecture of RPKI is described in the RFC 6810 [7] and the specification itself is spread to the RFC documents RFC 6481-6493. This complex subject is well explained in the open source documentation maintained by the RPKI Team at NLNet Labs found at "https://rpki.readthedocs.io" [8] where most of the following summary is based on.

The concept of RPKI is very similar to the Internet X.509 Public Key Infrastructure specified in RFC 5280 [9]. X.509 public key certificates are widely used in many internet protocols including TLS. The major difference to RPKI is that the X.509 certificate are extended with the fields for IP prefixes and AS numbers.

The goal of RPKI is to provide additional data for verifying BGP announcements. This cannot be done within the BGP protocol itself because it lacks the required information. Most of the BGP route hijacks are caused by misconfigurations. With BGP announcement validation made possible by RPKI these kinds of routing issues could be prevented from spreading across the Internet. RPKI mainly consists of three components. We will briefly discuss all of them.

2.2.1 Certificate Authority

The Certificate Authority is the entity which signs certificates of subordinate CAs and issues Route Origin Authorizations (ROAs). These certificates are based on X.509 extended with IP addresses and AS identifiers. Instead of having a vast number of root CAs, there are just five of them in RPKI. As we see in the first row of Figure 2.2, they are run by the five Regional Internet Registries (RIRs) which manage the allocation and registration of Internet number resources including IP prefixes and Autonomous System (AS) numbers in their region of the world [10]. A root CA is also called Trust Anchor (TA) which issues a self-signed certificate for all Internet resources, namely the whole IP address space and all AS numbers. That certificate is used to sign certificates of other CAs to which the TA has delegated some IP prefixes and AS numbers. This results in a tree structure as we see in Figure 2.2 where every CA except the Trust Anchor has at least one parent and an arbitrary amount of children. A correctly signed certificate certifies the ownership of the resources noted in the certificate. For these resources, the Certificate Authority can issue Route Origin Authorizations.

There are two implementation models for RPKI. All five RIRs offer the service to run the CA for other organizations on their infrastructure. This implementation model is called hosted RPKI. The RIRs offer a web interface and some of them also an API to manage ROAs. It is the more accessible solution as there is no need to maintain the CA software nor any hardware. This service also includes publication of ROAs to the RIR's Publication Server. This implementation model

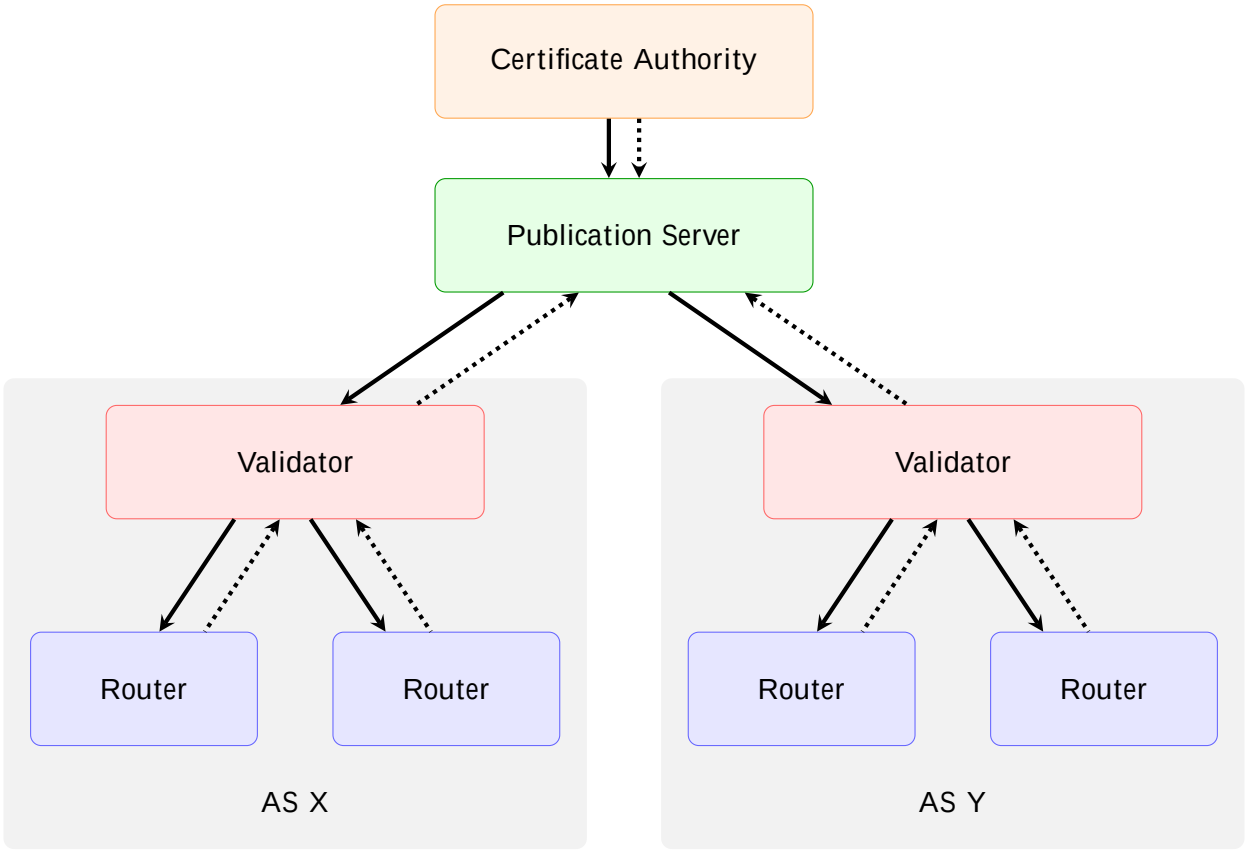


Figure 2.1: Shows the relevant components for RPKI in a simplified form with one Certificate Authority (CA), a Publication Server (PB) and multiple RPKI Validators. The solid arrows show the direction of the data flow and the dotted arrows indicate how the communication is initiated.

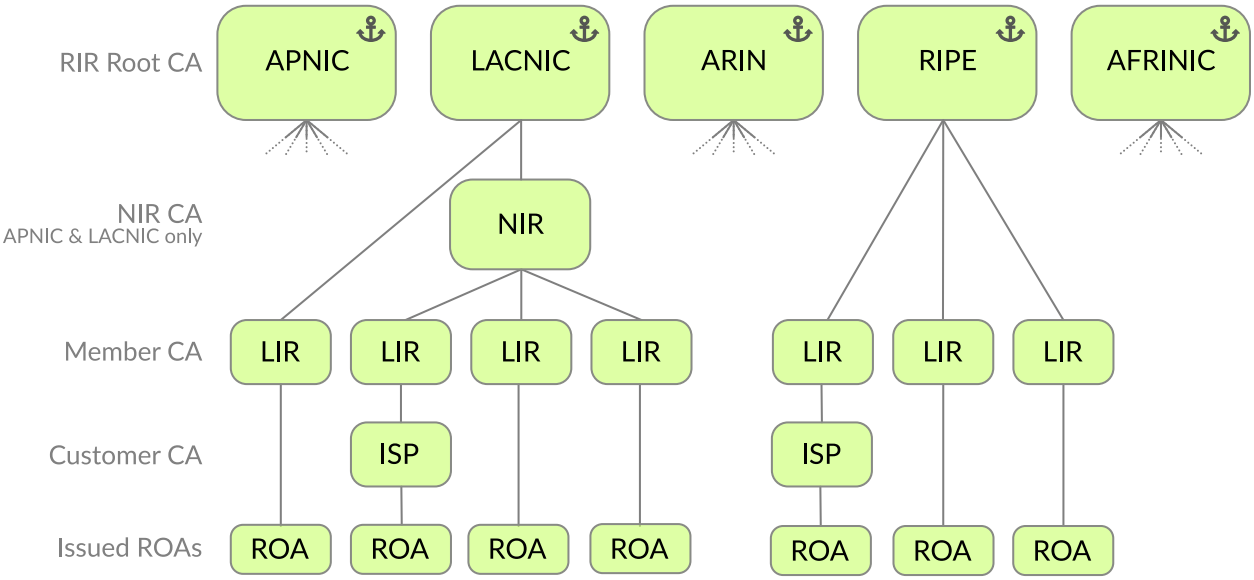


Figure 2.2: Overview of the CA hierarchy in RPKI [8].

is the more accessible one in contrast to delegated RPKI. Delegated RPKI allows an organization to run its CA on its own hardware. This is useful for globally operating organizations to manage ROAs for resources delegated from different RIRs.

A Route Origin Authorization (ROA) is a signed attestation that a certain AS is allowed to announce an IP prefix. Besides the IP prefix and the AS number, it also has a maximum length value (maxLength). This specifies the maximum length of the IP prefix that the AS is authorized to advertise. They also have an explicit start and end validity date. The ROAs are distributed through a Publication Server to the Relying Parties as explained below.

2.2.2 Publication Server

A Publication Server stores certificates, ROAs, revocation lists and other RPKI related data and makes them available to the public through rsync and RRDP. Rsync is an open source project which provides fast incremental file transfer [11]. RRDP was later added as an option based on HTTPS. It is specified in RFC 8182. RRDP is the recommended protocol to use with new setups because it allows the use of Content Delivery Networks (CDN) and caching infrastructure [12]. In contrast to the Certificate Authorities, Publication Servers are not organized in any form and every Relying Party must fetch the data from all Publication Servers. Therefore the number of Publication Servers should be kept as low as possible to reduce the load for the Relying Parties which have to fetch the data from every Publication Server. The IETF Standard also allows to set up caching servers to reduce the load on the Publication Servers. It is essential for RPKI that the Publication Servers are always online and reachable. Therefore it is possible with most RIRs to use their Publication Server even if the CA is run on your own hardware so that an organisation has the flexibility of an own CA but not the challenging requirement to provide a Publication Server without downtime.

2.2.3 Relying Party

A Relying Party (RP), also called RPKI Validator, is a software which performs the cryptographic validation of ROAs and is required for any AS to perform Route Origin Validation (ROV). For a RP to retrieve all RPKI data, it connects to the Trust Anchor of each RIR. Their self-signed certificates contain pointers to their children which also have pointers to their own children. Every CA certificate also contains a pointer to the Publication Server where the CA publishes its data.

The Trust Anchors are found through a static Trust Anchor Locator (TAL) file which simply contains an URL to the Trust Anchor and a public key to verify its authenticity. These files allow updating the Trust Anchor's certificate data without requiring all RPs to update its configuration. The TAL files are often distributed directly with the Relying Party software itself.

Once the RPKI data has been downloaded, the Relying Party verifies the ROAs with the downloaded chain of keys. All validated ROAs are then made available to the routers of the AS through the RPKI Router Protocol (RPKI-RTR). The routers do not need to do any cryptographic operations as the ROAs were already validated by the Relying Party. This means that the load on the routers for doing Route Origin Validation is not significantly higher and the existing hardware can be used.

2.2.4 Route Origin Validation

With the three key components explained above we have validated Route Origin Authorizations in the cache of the RPKI Validator. This validated data is brought to the routers of the AS using the RPKI Router Protocol (RPKI-RTR). It is a lightweight protocol based on HTTP which allows

to transfer only the delta between a past state and the current cache state and thus minimizes the time a router needs to update its RPKI data.

Once the router has populated its RPKI prefix table, it can use that data to filter BGP announcements using Route Origin Validation (ROV). A validation for an announcement is returning either one of the following three states:

- Valid – The announcement is covered at least by one ROA. These announcements are accepted and usually get a high local-preference assigned.
- Invalid – The announced prefix is found in the prefix table but the originating AS is not authorized or the announcement is more specific than the ROA allows with its `maxLength` value. For RPKI to succeed in its objective, those announcements should be dropped. Sometimes, they just get a low local-preference instead especially in an early adoption phase of ROV.
- Not Found – The prefix of the announcement is not fully covered by a ROA. Even though the number of ASes using RPKI is growing, most of the validations result in that state. Therefore these announcements should not be dropped but assigned a lower local-preference than the "valid" state and a higher value than the "invalid" state (if not dropped).

2.3 Software

We use the same software as in the previous semester thesis [10]. It proved to be suitable for our intended use-case. Because all tools are open source projects, it is also very easy to apply small changes to the applications as needed. We will briefly discuss all of them.

2.3.1 Krill

For simulating the Certificate Authority, the open source software Krill was used. The software is written in the programming language Rust and is under active development by NLNet Labs [13]. The main reason this software was used over alternatives was its testing mode. The testing mode allows to have a built-in Trust Anchor. This is exactly what is needed in the virtual network environment of the mini-Internet as we cannot rely on an externally hosted Trust Anchor of one of the Regional Internet Registries such as RIPE NCC. Another important feature of Krill is its web interface to issue ROAs. With the recent release of v0.9.0, the web interface and the API underneath support multiple users with different permissions perfectly suited for the usage within the mini-Internet. The software also includes a Publication Server where the CAs can publish the issued ROAs. For normal operations, a running instance of Krill is either hosting one or more CAs or one Publication Server but not both at the same time. The intention behind this setup is to allow higher up-time guarantee for the Publication Server as it should be reachable at any time. This constraint is not required for hosting a Certificate Authority. So even if an organisation wants to host both services on their own, it allows more flexibility regarding the hosting of those services.

Krill is running in a mixed setup when the test mode is activated. This means that it automatically starts the configured CAs and a Publication Server with the same instance. In Figure 2.3 we see the internal logical parts of the software as a block diagram. This setup is easier to handle for performing tests and for the mini-Internet it does not matter whether it is combined into one or split to two services. This test mode is called testbed and includes some more features besides the Trust Anchor and starting in mixed mode. It automatically adds another CA called "testbed" subordinate to the Trust Anchor and enables additional API endpoints with the same prefix. The web interface also includes a page where the user can create new CAs or add CAs as children to

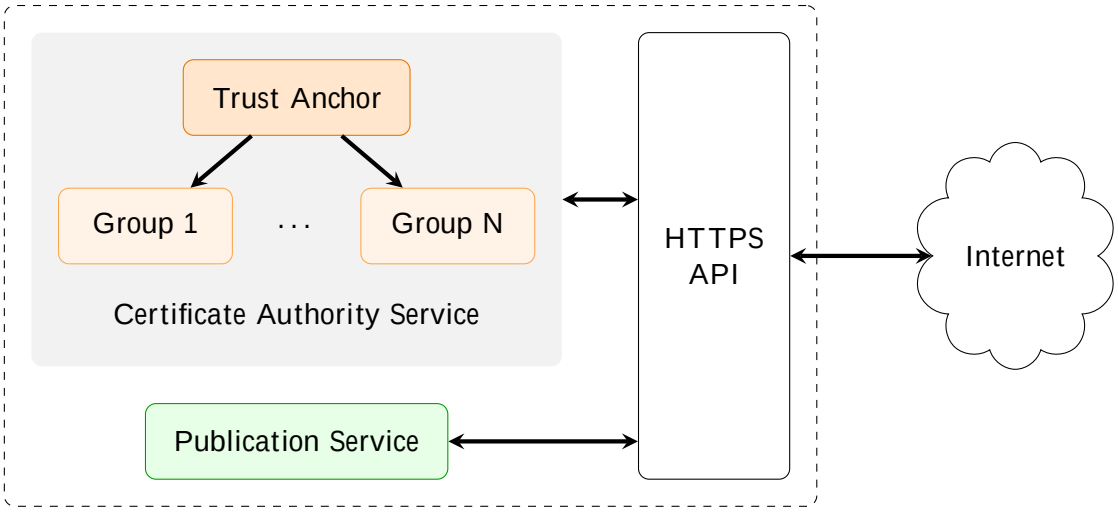


Figure 2.3: Overview of the internal structure of the Krill server when configured in test mode.

other CAs or remove a parent relation without any authentication. This feature is not intended to be used by the students and should not be accessible for them. As we do not expose Krill directly to the connected networks but through the HAProxy instance, we managed to prevent access to those specific API endpoints and therefore disabled the feature.

The CAs behave as if they are not hosted with the same instance in order to follow the out-of-band setup protocol specified in RFC 8183 [14]. This means that all CAs hosted on a Krill instance are completely independent from each other and during their setup, the XML files must be generated and referred to with the Krill CLI commands to set up a parent-child relation and configure the Publication Server for a CA.

2.3.2 Routinator

Routinator 3000 is an open source RPKI Relying Party software. It is also written in Rust and under active development by the same team at NLNet Labs [15]. We use the software as an RPKI Validator deployed to every AS. It is crucial that the RPKI Validator instance is trustworthy. Therefore, it is recommended for an AS to host their own instance. The validator software fetches all ROA entries from the Publication Servers and verifies the cryptographic signatures of the certificates and ROAs. After this validation, the cryptographic part is left out for sending the ROA entries to the routers using the RPKI-RTR protocol.

The Routinator CLI offers commands to directly perform a validation of a given IP Prefix and AS number combination or updating and validating the RPKI data manually from the Publication Servers. When started as a daemon, the update procedure for the RPKI data is performed automatically on a configured interval.

The software offers also a web interface for showing the validated RPKI data and issuing validation requests for an IP Prefix and AS number combination.

2.3.3 HAProxy

HAProxy is an open-source proxy server for TCP and HTTP-based applications. It is written in C and is maintained by the HAProxy Technologies LLC and the community. A proxy server is an intermediary service. It receives requests from clients and forward them eventually to the requested

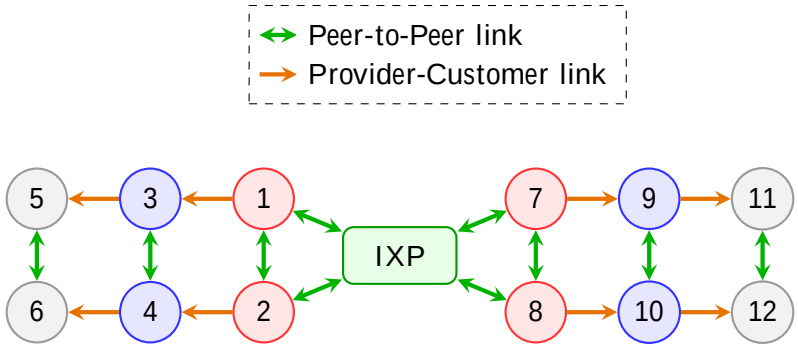


Figure 2.4: Overview of a typical mini-Internet topology.

service. The clients do not have to know how to reach the service directly which allows the operator to keep the service infrastructure secret and apply some rules before forwarding the request to the actual service. HaProxy allows to configure access control lists (ACLs) to protect some URLs or block access altogether. It also o□ers to balance the load on multiple backends allowing so serve more concurrent requests [16].

2.4 The mini-Internet

The mini-Internet is an open source platform providing a virtual network environment mimicking the real Internet. It is developed at the Networked Systems Group at ETH Zurich. The platform is designed mainly for teaching the students how the di□erent protocols work in practice. It is configurable for an Internet topology of choice. It can have a layer two network with switches and a layer three network with di□erent ASes. We see in Figure 2.4 an example of a layer three network configuration with ASes and an IXP. Each component – routers, host, DNS server, switches, etc. – in the mini-Internet is run in a Docker container. Aside from the core components of the Internet, there are some additional helper services in the mini-Internet which are intended to help students with their projects. Every AS has an SSH container which allows the students to connect to the various devices within their virtual AS. To simplify things a little bit, there is a DNS server which is connected directly to every AS. This ensures that the students can get DNS working without relying on other groups. The platform gathers data and visualizes gathering and visualisation is performed by additional containers to give the students some feedback what is going on in the virtual mini-Internet. In Figure 2.4 we see an example topology we can create with the platform.

There are multiple configuration files which are relevant for configuring a specific topology. We briefly explain the most relevant files.

- `AS_config.txt` – This file contains the the main configuration for all ASes of the topology such as the AS number, the configuration files which describe the internal topology of the AS and whether the devices of the AS should be configured automatically during startup.
- `external_links_config.txt` – This file is where the external links between the ASes are defined. Every AS on an external link has one of the roles assigned – Provider, Customer or Peer together with an IP prefix for assigning an IP address to the router's network interfaces and some other metrics.

- `internal_links_config.txt` – Internal links are AS specific and therefore referenced by the file "AS_config.txt". ASes in the mini-Internet can have different internal structures and different number of routers.
- `router_config.txt` – The router config is also referenced by the file `AS_config.txt` as it is AS specific and must match to the config of `internal_links_config.txt` of the same AS. In this file, the configuration of every router within the AS is specified very simplified. Every router can have a connection to one of the special service containers such as DNS, MATRIX_TARGET, MATRIX or MEASUREMENT. There is also one host connected to every router. The docker image for the host container is also specified in this file.

A more complete description of the mini-Internet is found in [5].

2.5 Related Work

The Resource Public Key Infrastructure is an IETF Standard defined over various RFC publications as described in Section 2.2. It is difficult to understand the basics of RPKI by reading these very technical documents. We appreciate the great work of the RPKI Team of NLNet Labs for summarising the basics in their online documentation found at https://rpki.readthedocs.io [8]. Their documentation greatly lowers the hurdle to get started with RPKI.

A longitudinal study of RPKI deployment and invalid route origins shows that during the first two years after the initial deployment about 20.76% of the RPKI-covered BGP announcements were invalid. The study also observes that the percentage of BGP announcements covered by RPKI is consistently increasing [17].

Most Regional Internet Registries (RIR) have an FAQ section about RPKI on their website with a focus on operational matters, like for example ARIN [18]. This educational content provided by the RIRs was of great importance for the network operators to understand the RPKI architecture and how to use it through the web interfaces provided by the RIRs. The writer of the longitudinal study conclude that the educational content provided by the RIRs was of great importance for the network operators to understand the RPKI architecture. They also made the observation that some network operators are likely confused about how the `maxLenth` attribute should be used [17].

Kathará is a similar platform for network emulation to the mini-Internet platform. The platform is also based on Docker containers but follows a more general approach. Kathará is very flexible in regards of configurability and might be more usable for research purposes than the mini-Internet platform [19].

A very limited implementation of RPKI in the mini-Internet was done in another semester thesis by Denis Mikhaylov [6]. The result was neither scalable nor production ready, but proved to be a great starting point to reach the goal of this thesis of fully integrating RPKI in a scalable and configurable way in the mini-Internet.

Chapter 3

Implementation

First, we will introduce our approach of RPKI we implemented within the mini-Internet in Section 3.1. Then we explain the details of the implementation and the challenges we faced in Section 3.2. Afterwards we show how the workflow for student projects using RPKI could look like in Section 3.3.

3.1 Approach

The work done in the previous semester thesis [6] was a great starting point. The approach we followed in this thesis is however slightly different. Our goal was to implement RPKI in the mini-Internet as similar as possible to the deployment in the real internet.

In order to get a fully functional RPKI implementation in the mini-Internet, we have to simulate all key components as described in Section 2.2. Our implementation can be used to simulate all variations of RPKI explained in Section 2.2.

Certificate Authority and Publication Server

We need at least one Trust Anchor and several other Certificate Authorities. We use the software *Krill* (See Section 2.3.1) to simulate one or more Certificate Authorities. In order to prevent any breaking changes to the topology configuration files of the mini-Internet, the auto-configuration feature only supports one RPKI deployment variant we see the most suitable for the *Routing Project*

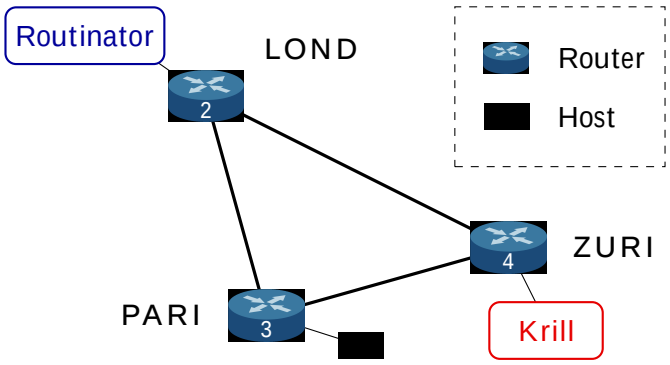


Figure 3.1: Internal structure of an AS hosting the Trust Anchor (TA) and an RPKI Validator to perform Route Origin Validation (ROV) on its routers.

```
# Example ROA:
#   IP prefix: 8.0.0.0/8
#   maxLength: 9
#   AS number: 8

# Add a ROA:
A: 8.0.0.0/8-9 => 8
# Remove an existing ROA:
R: 8.0.0.0/8-9 => 8

# If the maxLength equals the IP prefix length (8 in our example),
# it can be simplified:
A: 8.0.0.0/8 => 8
R: 8.0.0.0/8 => 8
```

Listing 1: Shows example entries for the krill delta files for modifying ROAs on startup of the mini-Internet topology.

of the lecture Communication Networks at ETH Zürich. The auto-configurable variant has only one Trust Anchor. Every AS (called *group* in teaching context) has its own Certificate Authority which has the Trust Anchor as a parent. The CAs are deployed as hosted RPKI, so there is only one Krill instance needed. The Krill instance is started in its own container configured in test mode to enable the Trust Anchor. We do not need another instance for running a Publication Server because Krill configured in test mode already includes one. We see in Figure 3.1 that it is connected to a router replacing the host normally connected there. The Trust Anchor Locator (TAL) file is extracted automatically during startup for later use.

The passwords used for SSH access and login for the Krill server are generated individually for every group and are stored in the file `groups/passwords.txt`. Two additional passwords for special Krill user accounts are also generated and stored in the file `groups/krill_passwords.txt`. Every Krill instance will have an account for every group with the scheme *groupX@ethz.ch*. An additional account *readonly@ethz.ch* is provided which has read-only access to all Certificate Authorities found on the Krill instance. This way, the students can check the ROAs of other groups without having a functioning Routinator instance running within their AS. The account *admin@ethz.ch* has admin privileges for all Certificate Authorities found on the Krill instance.

For every AS which has the auto-configuraton feature enabled, the startup script creates Route Origin Authorizations (ROAs) for the assigned IP prefix (`X.0.0.0/8` where X stands for the AS number). It is also possible to create custom ROAs on startup for any CA. A delta file `config/roas/gX.txt` has to be created where X stands for the AS number. In Listing 1 we see some example entries for adding and removing ROAs.

The delta file is applied after the auto-configuration, so it is possible to remove the ROAs added by the auto-configuration with a delta file.

Krill also provides a web interface for managing ROAs from the web browser. The web interface is accessible on any network interface of the Krill container with HTTP on port 3080 and with HTTPS on port 3000.

Relying Party

For an AS to be able to perform Route Origin Validation, the RPKI data has to be fetched and validated. We use Routinator 3000 (See Section 2.3.2) to simulate the Relying Party Software. Routinator must be deployed to any AS which should perform ROV. As we see in Figure 3.1 each AS's Routinator service is started in its own container and is connected to a router of that AS replacing the host normally connected there. The TAL files extracted from the started Trust Anchor instances are mounted to the Routinator instance automatically. Routinator does not need any additional configuration as it automatically tries to fetch the data starting at the Trust Anchors provided by the mounted TAL files.

Routinator also allows to modify some RPKI data locally according to a JSON file. Its content is structured according to the Simplified Local Internet Number Resource Management (SLURM) scheme and is specified in RFC 8416 [20].

The exceptions can be configured in the file `/root/rpki_exceptions.json`. An example configuration file is found in Appendix A.

Route Origin Validation

The validation of BGP announcements is done by the routers. They are running the software suite provided by the FRRouting project which is managed by the Linux Foundation [21]. The daemon configuration file is mounted from `config/daemons`. In order to load the RPKI module with the BGP daemon on startup, the following line has to be changed within that file:

```
bgpd_options="-A 127.0.0.1 -M rpki"
```

Default Routing Policy

For every AS which has auto-configuration enabled and contains a Routinator instance, Route Origin Validation (ROV) is configured on every router during startup. Table 3.1 shows the default routing policy applied to routers of ASes which should be configured automatically during startup without ROV disabled. The highest local preference value is assigned to BGP announcements received from a customer and the lowest local preference is assigned to announcements received from a provider.

Role in Business-Relation	Local Preference
Provider	100
IXP	50
Peer	50
Customer	20

Table 3.1: Shows the local preference value assigned to BGP announcements for the different roles within the business relationship to the neighboring AS if Route Origin Validation (ROV) is disabled.

If an AS has at least one Routinator instance connected to any of its routers, RPKI related configuration is applied to all routers of that AS. Table 3.2 shows what local preference values are applied for different cases. When the Route Origin Validation (ROV) returns with `invalid`, the BGP announcement is dropped. For RPKI to show its full potential, `invalid` BGP announcements must be dropped [8]. The role and value order is kept the same as in Table 3.1. BGP announcements with ROV state "valid" get the highest local preference as we can see in Table 3.2.

The announcements with ROV state ”not-found” get slightly lower values than the default values without ROV. This means that we prefer any valid route over one with ROV state `not-found` independent of the business relation with our neighbors. The minor difference between the last two columns of Table 3.2 ensures that the local preference value identifies precisely which rule was applied.

The last column in Table 3.2 is for a route-map rule which does not have any RPKI match statement at all. This rule is necessary because the match for the ROV state `not-found` is only true when there is an active connection to a RPKI Validator and the RPKI data is up-to-date. Upon startup, no routes are configured yet and therefore the connection to the RPKI Validator will fail initially. We also have the same issue with the RPKI Validator connecting to the Krill server. If we do not have a route-map without an RPKI match statement, all BGP announcements would be dropped by the router which will prevent that any routes between the ASes are ever accepted. The observed behavior might be specific for the FRRouting software suite and might not apply to other solutions. We just just took the local preference values as specified in Table 3.1 allowing to keep the default route-map rules if ROV is not configured on startup already, which solved the problem.

Role in Business-Relation	Local Preference			
	ROV valid	ROV invalid	ROV not-found	no ROV
Provider	200	*drop*	90	100
IXP	150	*drop*	40	50
Peer	150	*drop*	40	50
Customer	120	*drop*	10	20

Table 3.2: Shows the local preference value assigned to BGP announcements for the different roles within the business relationship to the neighboring AS if Route Origin Validation (ROV) is enabled.

3.2 Implementation Details

In this section we will discuss the changes we made to the mini-Internet to implement RPKI more in-depth. The mini-Internet is based on Docker containers. Some fundamental changes to the Docker images will be explained first. We will see how the additional Docker images are structured. It is then followed by a description of how the RPKI services are configured.

3.2.1 Docker Images

We introduced two new Docker images for the mini-Internet platform. We will briefly discuss them in this Section.

Krill

The Dockerfile uses the multi-stage feature of Docker. It builds multiple images in sequence during the build process which allows to keep all build related packages in the build stage. We can just copy the binaries from the build stage to the final image in order to minimize the size of the Docker image.

We had to build Krill from source because a small patch had to be applied to the source files of Krill so that all configuration steps could be fully automated. Users can be configured

statically within an additional section of the Krill configuration file. The Krill CLI offers a command which prepares the configuration line for a new user based on the parameters and user input. Unfortunately, the password prompt did not accept piped input. A small patch allowed us to completely automate the configuration of the Krill server during startup.

Krill also offers a web interface besides the HTTPS API specified in the RFC documents. It allows CA operators to create and delete ROA entries, configure parent CAs and Publication Servers.

Krill generates its own self-signed TLS certificate for secure communication over HTTPS. For improved security and performance, the documentation suggests to put a reverse-proxy in front of the Krill server. We included HAProxy directly within the Docker image to keep the general approach to have one container per virtual device. A very similar approach was already used in the previous thesis [6], but there was an issue with TLS certificates which limited the deployment of Krill and Routinator to the same container only.

The issue we were facing was that the certificate was self-signed and issued for localhost only. To overcome this problem, we first generate our own root TLS certificate which is able to sign subordinate certificates. Then we issue individual certificates for every Krill instance signed with that root certificate which is valid for the IP address and the domain names for that specific Krill instance. The root certificate then has to be installed on every other container which needs to access Krill over HTTPS so that the TLS certificate is accepted.

The port 3080 is configured with the reverse-proxy to allow communication with the Krill service over unencrypted HTTP. Every software not enforcing HTTPS is then able to access the service without encryption. Such a configuration is not recommended for production use as the data transferred over the network is not encrypted. We chose this configuration so that we can offer the students a simple way to access the Krill web interface for configuring ROAs without having to accept any untrusted TLS certificates.

Routinator

We had to build Routinator from source because some non-default feature flags were required for our setup. As explained above for the Krill Docker image, we had to introduce our own root certificate for issuing TLS certificates for the Krill proxy server. Routinator uses the Rust package `rustls` for handling TLS. This package manages its own trust store containing root certificates which are trusted. With the feature flag `native-tls`, the application uses the system-wide trust store managed by the operating system. This allows us to install the previously generated root certificate into the trust store. This is necessary because Routinator only allows to connect with HTTPS to the Krill Publication Server.

The configuration file for Routinator requires the following line because we use some non-existing TLDs such as `.group1` for the Trust Anchor domain.

```
allow-dubious-hosts = true
```

The Docker image contains the required configuration file needed to run within the mini-Internet.

3.2.2 Integration into the mini-Internet

The integration of RPKI into the mini-Internet is done in two steps similar to how the scripts for the other part of the platform are structured in the mini-Internet. These two steps are separated into two Bash scripts `setup/rpki_config.sh` and `setup/rpki_setup.sh`. The first script is executed

before any container is started and the second one after all container are running already and the default router configuration has been applied. In the first step, some configuration files are generated based on the mini-Internet topology configuration files and in the second step, tasks are run which required the containers to be running already.

Configuration Files

All configuration files are generated with the assumption that there is only one Krill instance. If there should be multiple Krill servers such as multiple Trust Anchors or delegated RPKI implementation, some additional options would have to be added to the configuration files.

The Krill configuration file is based on the default configuration file shipped with the software which contains a short description for every configurable property. Besides some URLs for the testbed operation, an auth_token must be set. This token is generated automatically and is added to the environment variables of the Krill container such that any `krillc` command is authenticated automatically against the instance of Krill running within the same container. Additionally, the `auth_type` was set to "config-file" so that the multi-user feature is enabled and the user account information is loaded directly from the configuration file. The default configuration expects a Krill `auth_token` so the web interface would also request a token for authentication. With this configuration change, we get a username and password login form instead.

The passwords used for SSH access and login for the Krill server are generated individually for every group and are stored in the file `groups/passwords.txt`. Two additional passwords for special Krill user accounts are also generated and stored in the file `groups/krill_passwords.txt`. Every Krill instance will have an account for every group with the scheme *groupX@ethz.ch*. We also create an additional account *readonly@ethz.ch* which has read-only access to all Certificate Authorities found on the Krill instance. The students may check the ROAs of other groups without having a functioning Routinator instance running within their AS. The account *admin@ethz.ch* has admin privileges for all Certificate Authorities found on the Krill instance. The accounts and their passwords are the same for all Krill instances and are added to the Krill configuration files after the containers have been started. The lines are generated with a Krill CLI command as follows where X is replaced with the actual group number:

```
{
    echo $group_password | \
    krillc config user --id "groupX@ethz.ch" \
        -a "role=readwrite" -a "inc_cas=groupX" | \
    grep "groupX"
} >> path/to/groupX/krill.conf
```

This is done in the second step because of the need to use the Krill CLI, the containers already have to be running and that command has to be executed within a Krill container.

The configuration file for Routinator does not need any instance-specific values. Therefore we can use the default configuration from the Docker image as explained in Section 3.2.1.

Afterwards, we reconfigure the routers of all ASes which also run an instance of Routinator and are set to be configured automatically during startup. If an AS does not have a Routinator instance connected to any of their routers, the configuration of the routers for RPKI operation is skipped.

3.2.3 General Improvements

We applied some general improvements not directly related to RPKI to improve the performance of the mini-Internet platform in general.

Docker Images

The Docker images used with the mini-Internet were rather large. In order to improve the download time for the Docker images, we redesigned how the Docker images are built. We wanted to share as many layers as possible so they do not have to be downloaded separately for every image but can be shared between images.

We built a new image which should be used as the base image for all other Docker images of the platform. It is based on Alpine Linux which is well suited for Docker images as it is based on BusyBox which is very minimalistic and therefore very small but still comes with a complete package manager. The package manager helps to effortlessly install additional software. We added packages such as bash, vim, nano and a set of network debugging utilities to the image. The default shell is set to Bash instead of the Almquist shell because the students are more familiar with Bash.

Based on the previous image, we built another base image. It additionally comes with a pre-configured process management system named Supervisor [22]. This image is intended to be used for Docker images within the mini-Internet platform which needs to start multiple services within a single container. The Supervisor daemon ensures that the process is restarted if it is terminated or has crashed. If a configured service fails to start multiple times in a row, a custom bash script is executed which stops the Supervisor daemon and therefore stops the whole container ensuring that when the container is running, every service is running correctly within that container.

Another service which is configured in the second base image is for logging purposes. It ensures that the logging output from all started processes within the container are prefixed with the service on every line. This simplifies debugging as it allows to just use the command `docker logs` even after the container has crashed or was stopped.

Startup Time Improvement

The startup of a mini-Internet topology took rather long. We wanted to reduce the startup time by executing for-loops cycles in parallel if their execution does not depend on a prior cycle. Listing 2 shows the content of the helper file we created to simplify the migration to parallel executed for-loop cycles. The variable `NR_TASKS` specifies how many for-loop cycles are allowed to be executed in parallel. We set this value to the number of CPU cores of the system. We can source the created helper file in other script files as we see in Listing 3.

3.3 Student Project Workflow

We want to show how the workflow could look like for the students doing the Routing Project for the Communication Networks lecture at ETH Zürich.

The workflow in regards of RPKI is not heavily depending on the topology, as long as there is only one Trust Anchor, every AS uses hosted RPKI and there is a Routinator instance running in every AS.

The students have to solve two tasks. They have to create a ROA for their IP prefix in the RPKI system. They also have to configure RPKI at their routers and update the route-maps to enable ROV. Both tasks are independent from each other. We briefly explain what the students should do for those tasks.

```
# Set N_TASKS to the number of processor cores
# (twice as much if CPU has hyperthreading)
N_TASKS=$(grep -c ^processor /proc/cpuinfo)

function wait_if_n_tasks_are_running {
    # allows to execute up to N_TASKS jobs in parallel
    if [[ $(jobs -r -p | wc -l) -ge $N_TASKS ]]; then
        # If N_TASKS are running already, wait for the next task to terminate.
        wait -n
    fi
}
```

Listing 2: Shows the content of the file setup/_parallel_helper.sh. The function wait_if_n_tasks_are_running waits for the next task to finish if there are already N_TASKS concurrent tasks running.

```
source "${DIRECTORY}/setup/_parallel_helper.sh"

for ((k=0;k<5;k++)); do
    (
        echo "Task ${k}: Starting the task..."
        sleep 2
        echo "Task ${k}: Task completed!"
    ) &

    wait_if_n_tasks_are_running
done

wait
```

Listing 3: Shows an example Bash script executing the for-loop cycles in parallel with a maximum of N_TASKS concurrent tasks.

Create a Route Origin Authorization

The Krill web interface should be used by the students to manage their ROAs. There are several ways to access the web interface from their own device. If they already managed to connect their device to the L2 network within their AS through VPN, they could access the Krill container directly through the regular route within the mini-Internet. This scenario is very similar to the real mini-Internet. When they lose connection to the AS hosting the Krill server, they cannot access the web interface anymore and thus cannot manage their ROAs. It is also possible to access the web interface over SSH which does not depend on the success of other groups for previous tasks. The students are already familiar with SSH access to manage their virtual devices. Listing 4 shows how to add the port forwarding option to the SSH command for connecting to the mini-Internet. The value for krill-ip-address must be given with the instructions. Then they can access the Krill web interface through the web browser on their own devices by visiting http://localhost:3080/.

```
$ ssh -p X -L 3080: krill-ip-address :3080 root@mini-inter.net
```

Listing 4: Shows the SSH command which allows to access the Krill web interface from the student's own device on http://localhost:3080/ . The task instructions must state what krill-ip-address is (e.g 158.1.11.2).

Figure 3.2: Shows the Krill web interface from the student's perspective. They can create, view and delete the Route Origin Authorizations for the assigned resources in the right column.

We see the web interface as the students would see in Figure 3.2. There is a list of existing ROAs published by the group's CA. On the right side we see all Internet resources for which the CA is allowed to issue ROAs for.

The students should create a ROA for their pre x by clicking on the button "Add ROA". A form pops up where the IP pre x, maxLength value and AS number must be speci ed. The form reports an error if the student enters any resources which do not belong to this CA. When they succeeded in adding a ROA the interface displays the newly issued ROA, as shown in Figure 3.2. If the students successfully issued a ROA for their IP pre x it is automatically published to the linked Publication Server.

Con gure FRRouting for Route Origin Validation

The students have to con gure the RPKI cache server on every router of their AS where they want to perform ROV. Listing 5 shows the sequence of commands required in the router shell to con gure the RPKI cache server. The IP address for the cache server must be given with the instructions. The last two instructions in Listing 5 show whether there is currently a connection to the con gured server and what RPKI data has been loaded.

They must add rules to the existing input route-map LOCALPREFIN for the three ROV outcomes "valid", "invalid" and "not-found". The students should already know the commands to do so. The

```
router# conf t
router(config)# rpki
router(config-rpki)# rpki cache cache-server-ip-address 3323 pref 1
router(config-rpki)# exit
router(config)# exit
router# show rpki cache-connection
router# show rpki prefix-table
```

Listing 5: Shows the sequence of commands to con gure an RPKI cache server on a router. The IP address for the cache server must be given with the instructions.

```
router# conf t
router(config)# route-map LOCAL_PREF_IN permit 4
router(config-route-map)# match rpki valid
router(config-route-map)# set community 1:10
router(config-route-map)# set local-preference 200
router(config-route-map)# exit
router(config)# route-map LOCAL_PREF_IN permit 6
router(config-route-map)# match rpki notfound
router(config-route-map)# set community 1:10
router(config-route-map)# set local-preference 90
router(config-route-map)# exit
router(config)# route-map LOCAL_PREF_IN deny 8
router(config-route-map)# match rpki invalid
router(config-route-map)# exit
router(config)# exit
```

Listing 6: Shows the sequence of commands to con gure the input route-map to match the routing policy given in Table 3.2. In this example sequence, the community values and local preferences are set for the business relation role "Provider".

routing policy rules from Table 3.2 is applied with the commands found in Listing 6.

In order for the new rules to be applied, the BGP session has to be refreshed. This can be enforced with the command:

```
router# clear ip bgp *
```

Chapter 4

Evaluation

The Resource Public Key Infrastructure adds two additional Docker images to the mini-Internet platform which results in more data to be downloaded for starting a topology. In order to improve the startup time and resource usage, the new Docker images bring a reduction in size by up to 89%. The most significant reduction was observed with the Docker image for the routers which was reduced from 1.3 GB down to 147 MB.

We achieved another significant reduction of the startup time by adding parallelization for certain for-loops. The reduction for the topology used for verification (See Section 4.1.1) containing nine ASes was 30%.

In this Chapter, we will verify in Section 4.1 that the RPKI implementation works correctly. Afterwards, we test some attack scenarios learned from a paper [23] in Section 4.2.

4.1 Verification of the RPKI Implementation

As a first step, we verify that the RPKI implementation in the mini-Internet works correctly. We will show that the different states of the Route Origin Validation (ROV) are detected and handled correctly. We briefly introduce the topology used to evaluate the RPKI implementation in Section 4.1.1 followed by the evaluation of all possible ROV states within that topology in Section 4.1.2.

4.1.1 Topology

We prepared a topology very similar to the topology used for testing in the previous thesis [6] allowing to compare the evaluation results of the two theses. The topology consists of 9 ASes including one IXP as show in Figure 4.1. The most significant difference to the topology of [6] is that there are multiple Routinator instances, one in each AS, instead of just one Routinator instance in the special AS 1 which was connected as a peer to every other AS.

All ASes except AS 10 have a Routinator instance connected as a host to the router LON. The Krill instance is situated in AS 1 which is a tier 1 AS. It is connected as a host to the router ZURI. The external connections between different ASes follow the same rules for all ASes in the topology. The ASes 5-8 have a peering connection to the IXP (AS 10) as shown in Figure 4.1. This connection is set up at the router ZURI as indicated in Figure 4.2. The customers of an AS are interconnected to the router PARI and the providers are wired up at the router ZURI.

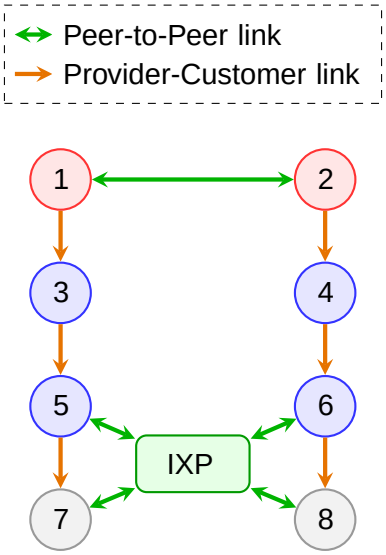


Figure 4.1: Shows the mini-Internet topology used to verify that the RPKI implementation is functional. The topology has all major components and con gurations possible including Tier 1 (red), Tier 2 (blue) and Tier 3 (grey) ASes and an IXP.

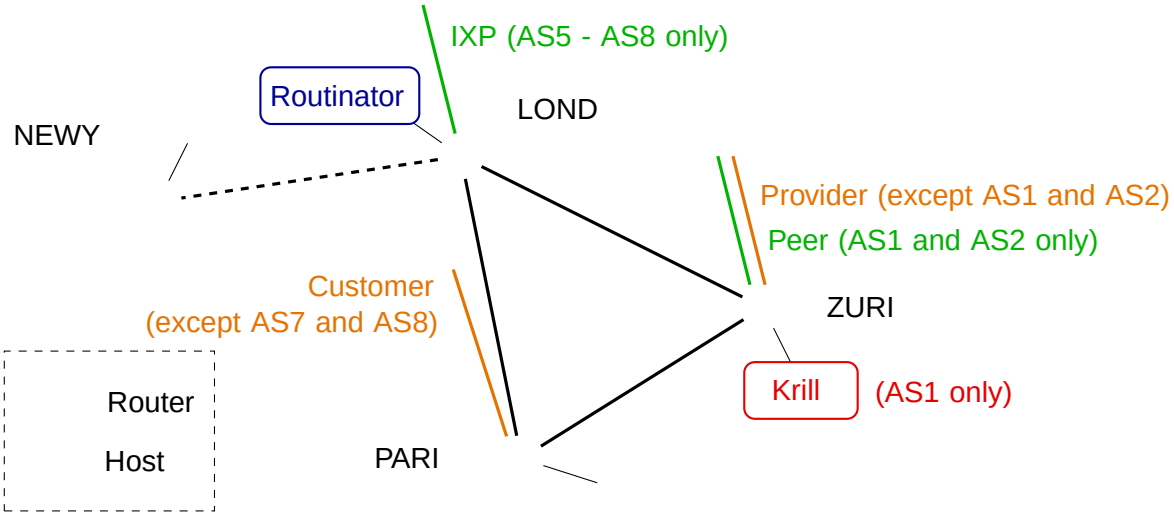


Figure 4.2: Shows the internal topology of routers and hosts within each AS and the external links according to Figure 4.1. Routinator replaces the host at the routerLONDKrill is only present in AS1 where it replaces the host at the routerZURI

```
# Shows the BGP routing table of the router.
vtysh -c 'show ip bgp'

# Shows the IP prefix table fetched from the RPKI Validator cache.
vtysh -c 'show rpki prefix-table'
```

Listing 7: List of commands used to show the effect of the modifications made to RPKI.

4.1.2 Evaluation

We will test the behavior of the topology described in Section 4.1.1 where all devices are configured automatically with the default configuration as discussed in Chapter 3 for different RPKI data sets. Our focus for this evaluation is on AS7 which is a Tier 3 AS. For readability reasons we just show the BGP routes and RPKI prefix table of the router ZURI by using the commands found in Listing 7.

Without RPKI

We want to show how the BGP routing table looked like for AS7 before any of the RPKI related parts were applied. We slightly modified the topology from Section 4.1.1 to remove all components of the RPKI implementation for this analysis. We simply replaced the hosts running the Krill and Routinator services with a regular host Docker image.

The BGP routing table for router NEWY is shown in Listing 8. It clearly shows a route for every IP prefix announced by the other ASes and assigns the local preferences according to Table 3.1.

```
AS7_ZURIrouter# show ip bgp
BGP table version is 10, local router ID is 7.152.0.1, vrf id 0
     Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.2.1                             20      0 5 3 1 i
*> 2.0.0.0/8        179.0.2.1                             20      0 5 3 1 2 i
*> 3.0.0.0/8        179.0.2.1                             20      0 5 3 i
*> 4.0.0.0/8        179.0.2.1                             20      0 5 3 1 2 4 i
*> 5.0.0.0/8        179.0.2.1                0            20      0 5 i
*>i6.0.0.0/8        7.151.0.1                0            50      0 6 i
*                     179.0.2.1                           20      0 5 6 i
* i7.0.0.0/8        7.151.0.1                0    100             0 i
* i                   7.154.0.1                0    100             0 i
* i                   7.153.0.1                0    100             0 i
*>                    0.0.0.0                  0                32768 i
*>i8.0.0.0/8        7.151.0.1                0            50      0 8 i
*                     179.0.2.1                           20      0 5 8 i

Displayed  8 routes and 13 total paths
```

Listing 8: Shows that the local preferences in the BGP routing table of router NEWY in AS7 are assigned exactly as stated in Table 3.1.

With default RPKI configuration

We started the topology as described in Section 4.1.1. The startup script has automatically configured the Krill server located in AS 1 and the Routinator instances found in every AS with the default rules as discussed in Chapter 3. The first command in Listing 9 shows a fully populated RPKI prefix table and with the second command we see that the expected local preference values were assigned as defined in Table 3.2.

```
AS7_ZURIrouter# show rpki prefix-table
host: 7.101.0.1 port: 3323
RPKI/RTR prefix table
Prefix                                  Prefix  Length    Origin-AS
1.0.0.0                                    8 -     8              1
4.0.0.0                                    8 -     8              4
6.0.0.0                                    8 -     8              6
5.0.0.0                                    8 -     8              5
2.0.0.0                                    8 -     8              2
3.0.0.0                                    8 -     8              3
7.0.0.0                                    8 -     8              7
8.0.0.0                                    8 -     8              8
Number of IPv4 Prefixes: 8
------------------------------------------------------------------------------
AS7_ZURIrouter# show ip bgp
BGP table version is 33, local router ID is 7.152.0.1, vrf id 0
     Network          Next Hop            Metric LocPrf Weight Path
*>  1.0.0.0/8        179.0.2.1                     120        0 5 3 1 i
*>  2.0.0.0/8        179.0.2.1                     120        0 5 3 1 2 i
*>  3.0.0.0/8        179.0.2.1                     120        0 5 3 i
*>  4.0.0.0/8        179.0.2.1                     120        0 5 3 1 2 4 i
*>  5.0.0.0/8        179.0.2.1                0    120        0 5 i
*>i6.0.0.0/8        7.151.0.1                0    150        0 6 i
*                     179.0.2.1                     120        0 5 6 i
* i7.0.0.0/8        7.154.0.1                0    100        0 i
* i                   7.153.0.1                0    100        0 i
* i                   7.151.0.1                0    100        0 i
*>                    0.0.0.0                  0         32768 i
*>i8.0.0.0/8        7.151.0.1                0    150        0 8 i
*                     179.0.2.1                     120        0 5 8 i

Displayed  8 routes and 13 total paths
```

Listing 9: The first command shows that the RPKI prefix table of router ZURIin AS7 is populated correctly and with the second command we see that the local preference values are assigned to the BGP routes as stated in Table 3.2.

With incorrect ROA

For testing the behavior when the ROV returns the state invalid, we modified the ROA for AS5 by creating the delta file config/roas/g5.txt (See Listing 1 for details) with the following content:

```
R: 5.0.0.0/8 => 5
A: 5.0.0.0/8 => 0
```

AS number set to zero means that nobody is authorized to announce the IP prefix. AS5 keeps announcing its IP prefix nonetheless. Every AS doing ROV should apply its routing policy for matching the invalid state. Listing 10 shows that the route for the network 2.0.0.0/8 was filtered as it does not have a local preference value set. The BGP announcement of AS5 is seen as invalid because according to the RPKI data, AS5 is not authorized to announce that IP prefix.

```
AS7_ZURIrouter# show ip bgp neighbor 179.0.2.1 received-routes
BGP table version is 30, local router ID is 7.152.0.1, vrf id 0
     Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.2.1                     120        0 5 3 1 i
*> 2.0.0.0/8        179.0.2.1                     120        0 5 3 1 2 i
*> 3.0.0.0/8        179.0.2.1                     120        0 5 3 i
*> 4.0.0.0/8        179.0.2.1                     120        0 5 3 1 2 4 i
*> 5.0.0.0/8        179.0.2.1                0                0 5 i
*> 6.0.0.0/8        179.0.2.1                     120        0 5 6 i
*> 7.0.0.0/8        179.0.2.1                     120        0 5 7 i
*> 8.0.0.0/8        179.0.2.1                     120        0 5 8 i

Total number of prefixes 8 (1 filtered)
```

Listing 10: The list of all routes received at router ZURI in AS7 shows that the route for network 5.0.0.0/8 was filtered as it has no local preference value set.

Without ROA

Similar to the modification before, we created the delta file config/roas/g5.txt with the following content:

```
R: 5.0.0.0/8 => 5
```

Listing 11 shows that the local preference value has changed to 10 for the prefix 5.0.0.0/8 which is the expected local preference value as defined in Table 3.2.

4.2 Simulating Hijack Attacks

RPKI has the biggest impact on the routing in the Internet as long as all ASes are performing Route Origin Validation (ROV). If we have a partial adoption of ROV, there are some side-effects depending on the distribution of ASes performing ROV. We will introduce the topology used for the simulations first. Afterwards, we will show scenarios for collateral benefit (Section 4.2), collateral damage by disconnection (Section 4.2) and collateral damage by traffic being hijacked (Section 4.2).

```
AS7_ZURIrouter# show ip bgp
BGP table version is 29, local router ID is 7.152.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.2.1                        120      0 5 3 1 i
*> 2.0.0.0/8        179.0.2.1                        120      0 5 3 1 2 i
*> 3.0.0.0/8        179.0.2.1                        120      0 5 3 i
*> 4.0.0.0/8        179.0.2.1                        120      0 5 3 1 2 4 i
*> 5.0.0.0/8        179.0.2.1                0        10      0 5 i
*> 6.0.0.0/8        179.0.2.1                        120      0 5 3 1 2 4 6 i
* i7.0.0.0/8        7.154.0.1                0       100      0 i
* i                 7.153.0.1                0       100      0 i
* i                 7.151.0.1                0       100      0 i
*> 7.0.0.0/8        0.0.0.0                  0             32768 i
*> 8.0.0.0/8        179.0.2.1                        120      0 5 3 1 2 4 6 8 i

Displayed  8 routes and 11 total paths
```

Listing 11: Shows the BGP routing table of router ZURI in AS7 when the default RPKI configuration is applied for all ASes in the topology but no ROA was created for the IP prefix 5.0.0.0/8.

Topology

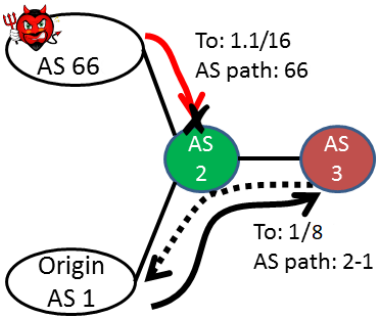
All three examples of collateral e□ects for other ASes use the same topology but with di□erent ROV adoption. Figure 4.3 shows the topologies consisting of three honest ASes 1-3 and one malicious AS66. Every AS has a provider-customer business relation to its neighbors so that they can communicate with any other AS in this topology. Every AS has issued a correct ROA for their IP prefix.

Collateral Benefit

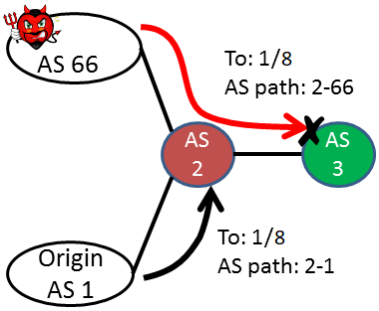
A collateral benefit happens when an AS without ROV does not receive an invalid BGP announcement because another AS did already drop that announcement. We see in Figure 4.3a that AS2 performs ROV but AS3 does not. AS66 sends the BGP announcement `1.1.0.0/16` which we can confirm with the first command in Listing 12. We expect the invalid announcement to be dropped by AS2 and not to propagate any further. We see with the second command in Listing 12 that AS2 performs ROV and drops the invalid BGP announcement as expected. All other ASes in this topology do not receive the invalid announcement sent by AS66. They benefit from the ROV performed by AS2.

Collateral Damage (disconnect)

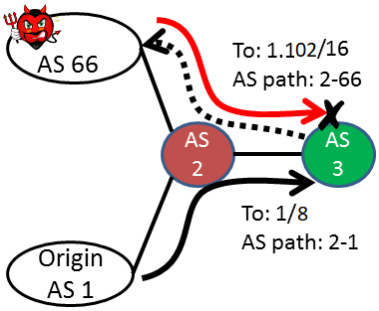
A collateral disconnection damage happens when an AS loses connection to another AS because a third AS does not perform ROV. As indicated in Figure 4.3b, AS66 announces the IP prefix `1.0.0.0/8` to AS2. We see in Listing 13 that AS2 has received two BGP announcement deemed valid for this IP prefix because it does not perform ROV and selected the one from AS66 based on the internal routing policy. AS3 performs ROV and has received only the invalid BGP announcement coming from AS66. We see in the second part of Listing 13 that AS3 drops this route and therefore does not have any path to AS1.



(a) Collateral benefit: AS3 does not receive the invalid announcement because AS2 dropped it.



(b) Collateral damage (disconnect): AS3 receives only an invalid announcement so it gets disconnected from AS1.



(c) Collateral damage (hijack): AS3 drops the invalid announcement but AS2 not performing ROV is on both routes so the traffic of AS3 is re-routed to AS66.

Figure 4.3: Shows collateral benefit and collateral damage for partial ROV adoption on a simple topology. The green AS performs Route Origin Validation, solid arrows represent BGP announcements and dashed arrows represent data packet forwarding. Figures were taken from [23].

```
AS66_ZURIrouter# show ip bgp
BGP table version is 11, local router ID is 66.152.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.2.1                    20      0 2 1 i
*> 1.1.0.0/16       0.0.0.0                   0         32768 i
... (Output truncated)
------------------------------------------------------------------------------
AS2_LONDrouter# show ip bgp neighbor 179.0.2.2 received-routes
BGP table version is 10, local router ID is 2.151.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*> 1.1.0.0/16       179.0.2.2                0             0 66 i
*> 66.0.0.0/8       179.0.2.2                0    200      0 66 i

Total number of prefixes 2 (1 filtered)
```

Listing 12: The output of the first command shows that the the router of AS66 announced the IP prefix `1.1.0.0/16`. The router in AS2 received the announcement from AS66 and filtered it as it has no local preference value assigned.

```
AS2_ZURIrouter# show ip bgp
BGP table version is 13, local router ID is 2.152.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*>i1.0.0.0/8        2.151.0.1                0    100      0 66 i
*                   179.0.0.1                0     20      0 1 i
... (Output truncated)
------------------------------------------------------------------------------
AS3_ZURIrouter# show ip bgp neighbor 179.0.1.1 received-routes
BGP table version is 20, local router ID is 3.152.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.1.1                              0 2 66 i
*> 2.0.0.0/8        179.0.1.1                0    120      0 2 i
*> 3.0.0.0/8        179.0.1.1                     120      0 2 3 i
*> 66.0.0.0/8       179.0.1.1                     120      0 2 66 i

Total number of prefixes 4 (1 filtered)
```

Listing 13: The first command shows that AS2 received two BGP announcements for the IP prefix `1.0.0.0/8` and selects the one from AS66 because of the local preference values. We see with the second command that AS3 drops the announcement for the IP prefix `1.0.0.0/8` because ROV returned with *invalid* hence it has no connection to AS1.

Collateral Damage (hijack)

A collateral hijack damage happens when a valid and an invalid route share same path segments to an AS performing ROV. AS3 in Figure 4.3c performs ROV and AS2 does not. Listing 14 shows in the second half that AS3 only accepted the legitimate BGP announcement for the IP prefix of AS1 contrary to AS2 which deemed the BGP announcement for a longer IP prefix (`1.102.0.0/16`) sent by AS66 valid. If AS3 wants to reach an address within the IP prefix `1.102.0.0/16`, it uses the valid route to forward the traffic to AS2. AS2 forwards the traffic to AS66 as we can see in the routing table in the first part of Listing 14. AS66 hijacks that specific traffic from AS3 to AS1. This results in a black hole with the topology used in our simulation as we can see with the traceroute from AS3 to host `1.102.0.1` located in AS1 in relation to before the malicious attack as shown in Listing 15. In case that AS66 would have an unaffected route to AS1, it could just forward the traffic to AS1 allowing AS66 to monitor the traffic from AS3 to AS1.

```
AS2_LONDrouter# show ip bgp
BGP table version is 7, local router ID is 2.151.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*>i1.0.0.0/8        2.152.0.1                0     20      0 1 i
*> 1.102.0.0/16     179.0.2.2                0    100      0 66 i
... (Output truncated)
------------------------------------------------------------------------------
AS3_ZURIrouter# show ip bgp neighbor 179.0.1.1 received-routes
BGP table version is 18, local router ID is 3.152.0.1, vrf id 0
   Network          Next Hop            Metric LocPrf Weight Path
*> 1.0.0.0/8        179.0.1.1                        120      0 2 1 i
*> 1.102.0.0/16     179.0.1.1                                 0 2 66 i
*> 2.0.0.0/8        179.0.1.1                0    120      0 2 i
*> 3.0.0.0/8        179.0.1.1                     120      0 2 3 i
*> 66.0.0.0/8       179.0.1.1                     120      0 2 66 i

Total number of prefixes 5 (1 filtered)
```

Listing 14: The first part shows that AS2 has accepted two routes for the IP prefix of AS1 where `1.102.0.0/16` is the more specific one announced by AS66. The second command shows that AS3 accepts only the BGP announcement for the IP prefix `1.0.0.0/8` hence it has a connection to AS1 but it has no control over which route AS2 chooses to forward the packets.

```
# ./launch_traceroute.sh 3 1.102.0.1
Hop 1:  3.0.199.1 TTL=0 during transit
Hop 2:  3.0.4.1 TTL=0 during transit
Hop 3:  3.0.1.1 TTL=0 during transit
Hop 4:  179.0.1.1 TTL=0 during transit
Hop 5:  2.0.1.1 TTL=0 during transit
Hop 6:  179.0.0.1 TTL=0 during transit
Hop 7:  1.102.0.1 Echo reply (type=0/code=0)
------------------------------------------------------------------------------
# ./launch_traceroute.sh 3 1.102.0.1
Hop 1:  3.0.199.1 TTL=0 during transit
Hop 2:  3.0.4.1 TTL=0 during transit
Hop 3:  3.0.1.1 TTL=0 during transit
Hop 4:  179.0.1.1 TTL=0 during transit
Hop 5:  2.0.3.2 TTL=0 during transit
Hop 6:  ***
```

Listing 15: Shows a traceroute from AS3 to a host (`1.102.0.1`) in AS1 before and after the malicious attack by AS66.

Chapter 5

Conclusion

We fully integrated RPKI into the mini-Internet platform with all primary features needed for the lecture. The RPKI is set up and configured automatically according to the topology configuration files. This allows the operator of the mini-Internet to use RPKI without any additional work required except for minor changes to the configuration files. We also managed to offer a realistic interface for the students to interact with the different components of the RPKI implementation similar to the methods offered in the real Internet. With systematic tests we could verify that the RPKI implementation is functional and that it is possible to simulate attacks observed in today's Internet and demonstrate the effects of partial deployment of ROV.

Additionally we managed to significantly reduce the size of the Docker images so that the download of the Docker images is faster and requires less storage on the system running the mini-Internet topology. We could also improve the startup time by 30%. This was measured with our evaluation topology presented in Section 4.1.1. The additional improvements to the mini-Internet platform are of general nature and as such not limited to RPKI. Therefore they make the whole platform more practical even without using the RPKI feature.

Mini-Internet Platform Development

While the additions implemented in this thesis allow the mini-Internet to use RPKI, there are still future improvements expected. The current implementation of RPKI offers a web interface for every Krill instance with an account having read-only access to every CA hosted on that instance. It enables the students to see what data are actually stored in the RPKI system. This is a great start and very useful as long as there is only one Trust Anchor. If we introduce multiple Trust Anchors or use delegated RPKI so that not all children of the Trust Anchor are hosted on the same instance, it make things a lot more complicated for the students and teaching assistants. An RPKI overview page similar to the connectivity matrix could be a great addition to ease the hurdle to overview a more complex setup of RPKI. It could also offer an automatically generated structural overview figure of the RPKI setup to help the students understand what is going on in the mini-Internet.

Most of the setup scripts for RPKI are prepared to allow configuring and running multiple instances of Krill as Trust Anchor within the mini-Internet. All instances are configured exactly the same way as there are no additional configuration options in the topology configuration files of the mini-Internet just yet. In order to mimic the different Internet regions, some additional configuration values would be needed. It would allow to set up a topology similar to the real Internet offering additional opportunities for education and research.

RPKI is an important first step to improve routing in the Internet. As a next step to improve

routing even further, BGPsec or any other AS-path validation could be added to the mini-Internet platform. The adoption of BGPsec is not very widespread at this time and many routing software, including the FRRouting project, do not yet support BGPsec.

The mini-Internet currently runs on a single server which requires powerful hardware for hosting large topologies. If we want to create even larger topologies, it is crucial that the mini-Internet can be distributed to multiple servers. Distribution to multiple servers would also bring the opportunity for multiple organisations to collaborate within one virtual mini-Internet topology. Many universities offer similar courses to the Communication Networks lecture offered by the Networked Systems Group at ETH Zürich. Distributing a giant topology over multiple servers hosted at different universities would enable the students of those courses to collaborate within the virtual Internet beyond the boundary of their own university giving an even more realistic experience of how the Internet works. If a mini-Internet topology is distributed to different universities, the security aspect should be considered. Because the Docker daemon runs as `root`, the attack surface is large when opening the system to other universities. The container orchestration on the distributed server cluster should be as limited as possible to ensure that if one of the server was hacked, it does not put the other universities' networks at risk.

A visual generator tool to prepare new topologies could improve the general usability. The same tool could also be used to generate the graph images based on the topology configuration files. The Kathará project provides a similar tool for their platform [24].

Research

The main focus of the mini-Internet is on teaching how the Internet actually works. The platform is also suitable for doing research on networking topics as well. It is possible to build a realistic topology mimicking the real Internet structure to analyze some attack scenarios and observe the convergence times and propagation delays in a very controlled environment without the noise measured in the real Internet. For certain research questions, it is also possible to build custom Docker images. For example to support new protocols in the FRRouting software suite to compare to other protocols. In order to use custom Docker images, the names have to be changed in the file `setup/container_setup.sh` manually.

Bibliography

[1] S. L. Murphy, "BGP Security Vulnerabilities Analysis," RFC 4272, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4272.txt

[2] Nist rpki monitor. National Institute of Standards and Technology. Accessed: June 2021. [Online]. Available: https://rpki-monitor.antd.nist.gov/

[3] Youtube hijacking: A ripe ncc ris case study. RIPE Network Coordination Centre. Accessed: June 2021. [Online]. Available: https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study

[4] Youtube hijacking: A ripe ncc ris case study. MANRS Initiative. Accessed: June 2021. [Online]. Available: https://www.manrs.org/2021/04/a-major-bgp-hijack-by-as55410-vodafone-idea-ltd/

[5] T. Holterbach, T. Bühler, T. Rellstab, and L. Vanbever, "An open platform to teach how the internet practically works," 2020.

[6] T. H. Denis Mikhaylov, T. Bühler, and L. Vanbever, "Implementing the resource public key-infrastructure (rpki) in a virtual mini-internet," 2020.

[7] R. Bush and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol," RFC 6810, Jan. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6810.txt

[8] The RPKI team at NLnet Labs and the community. RPKI documentation. Accessed: June 2021. [Online]. Available: https://rpki.readthedocs.io/

[9] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5280.txt

[10] R. Housley, J. Curran, G. Huston, and D. R. Conrad, "The Internet Numbers Registry System," RFC 7020, Aug. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc7020.txt

[11] rsync. The Samba Team. Accessed: June 2021. [Online]. Available: https://rsync.samba.org/

[12] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein, "The RPKI Repository Delta Protocol (RRDP)," RFC 8182, Jul. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8182.txt

[13] The RPKI Team at NLnet Labs. Krill – rpki tools – nlnet labs. Accessed: June 2021. [Online]. Available: https://www.nlnetlabs.nl/projects/rpki/krill/

[14] R. Austein, "An Out-of-Band Setup Protocol for Resource Public Key Infrastructure (RPKI) Production Services," RFC 8183, Jul. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8183.txt

[15] The RPKI Team at NLnet Labs. Routinator – rpki tools – nlnet labs. Accessed: June 2021. [Online]. Available: https://www.nlnetlabs.nl/projects/rpki/routinator/

[16] HAProxy Technologies LLC. Haproxy - the reliable, high performance tcp/http load balancer. Accessed: June 2021. [Online]. Available: https://www.haproxy.org/

[17] T. Chung, E. Aben, T. Bruijnzeels, B. Chandrasekaran, D. Cho□nes, D. Levin, B. M. Maggs, A. Mislove, R. v. Rijswijk-Deij, J. Rula, and N. Sullivan, "Rpki is coming of age: A longitudinal study of rpki deployment and invalid route origins," in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 406–419. [Online]. Available: https://doi.org/10.1145/3355369.3355596

[18] American Registry for Internet Numbers (ARIN). RPKI frequently asked questions. Accessed: June 2021. [Online]. Available: https://www.arin.net/resources/manage/rpki/faq/

[19] Kathará - lightweight container-based network emulation system. Computer Networks and Security Group at Roma Tre University. Accessed: June 2021. [Online]. Available: https://www.kathara.org/

[20] D. Ma, D. Mandelberg, and T. Bruijnzeels, "Simplified Local Internet Number Resource Management with the RPKI (SLURM)," RFC 8416, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8416.txt

[21] FRRouting. The Linux Foundation. Accessed: June 2021. [Online]. Available: https://frrouting.org/

[22] C. McDonough and contributors. Supervisor: A process control system. Accessed: June 2021. [Online]. Available: http://supervisord.org/

[23] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman, "Are we there yet? on rpki's deployment and security," in *The Network and Distributed System Security Symp.*, Feb. 2017, p. 15 pages.

[24] Github - katharaframework/netkit-lab-generator: A client-side javascript tool to configure a kathará or a netkit lab and generate all the files you need and the topology graph. . Accessed: June 2021. [Online]. Available: https://github.com/KatharaFramework/Netkit-Lab-Generator

Appendix A

RPKI Validator Exception file using SLURM

Below you will find an example file for RPKI Validator Exceptions using the SLURM scheme as found in the NLNet Labs documentation for RPKI [8].

```
{
  "slurmVersion": 1,
  "validationOutputFilters": {
   "prefixFilters": [
     {
      "prefix": "192.0.2.0/24",
      "comment": "All VRPs encompassed by prefix"
     },
     {
      "asn": 64496,
      "comment": "All VRPs matching ASN"
     },
     {
      "prefix": "198.51.100.0/24",
      "asn": 64497,
      "comment": "All VRPs encompassed by prefix, matching ASN"
     }
   ],
   "bgpsecFilters": [
     {
      "asn": 64496,
      "comment": "All keys for ASN"
     },
     {
      "SKI": "Zm9v",
      "comment": "Key matching Router SKI"
     },
     {
      "asn": 64497,
```

```
        "SKI": "YmFy",
        "comment": "Key for ASN 64497 matching Router SKI"
      }
    ]
  },
  "locallyAddedAssertions": {
    "prefixAssertions": [
      {
        "asn": 64496,
        "prefix": "198.51.100.0/24",
        "comment": "My other important route"
      },
      {
        "asn": 64496,
        "prefix": "2001:DB8::/32",
        "maxPrefixLength": 48,
        "comment": "My other important de-aggregated routes"
      }
    ],
    "bgpsecAssertions": [
      {
        "asn": 64496,
        "comment" : "My known key for my important ASN",
        "SKI": "<some base64 SKI>",
        "routerPublicKey": "<some base64 public key>"
      }
    ]
  }
}
```