The RPKI Documentation

Alex Band and the RPKI Community

Oct 12, 2020
# Using RPKI Data

8.1 Connecting to the Trust Anchor ............................................. 31  
8.2 Fetching and Verifying ...................................................... 31  
8.3 Validating Routes ............................................................ 32  
8.4 Local Overrides ............................................................... 33  
8.5 Feeding Routers ............................................................... 33  

# Router Support

9.1 Hardware Solutions .......................................................... 35  
9.2 Software Solutions .......................................................... 35  

# Resources

10.1 Books ........................................................................... 37  
10.2 Insights and Statistics ..................................................... 37  
10.3 Operational Experiences .................................................. 38  
10.4 Examples of BGP Hijacks .................................................. 38  
10.5 IETF Documents ............................................................. 38

# Software Projects

11.1 Relying Party Software .................................................... 39  
11.2 Certificate Authority Software .......................................... 39  
11.3 Supporting Tools ............................................................ 40

# Krill

12.1 Before You Start .................................................................. 43  
12.2 Install and Run ................................................................. 45  
12.3 Get Started with Krill ........................................................ 51  
12.4 RIR and NIR Interactions .................................................. 59  
12.5 Using the CLI ................................................................. 61  
12.6 Using the API ................................................................. 78  
12.7 Monitoring ................................................................. 80  
12.8 Running a Publication Server .......................................... 81  
12.9 Running a Test Environment .......................................... 85  
12.10 Running with Docker ..................................................... 91  
12.11 Krill Manager .............................................................. 94

# Routinator

13.1 Installation ...................................................................... 125  
13.2 Installation Notes ............................................................. 128  
13.3 Initialisation ................................................................. 131  
13.4 Running Interactively ..................................................... 133  
13.5 Running as a Daemon ..................................................... 135  
13.6 Configuration .............................................................. 138  
13.7 Monitoring ................................................................. 141  
13.8 Manual Page ............................................................... 142

# RTRlib

14.1 About ........................................................................ 153  
14.2 Installation ................................................................ 154  
14.3 RTRlib Command Line Tools ......................................... 157

# RIPE NCC RPKI Validator 3.1

15.1 Installation ................................................................ 159  
15.2 RPKI Validator .............................................................. 162  
15.3 RPKI-RTR Server ........................................................... 162
Welcome to the documentation of the Resource Public Key Infrastructure (RPKI), the community-driven technology based on open standards that is aimed at making Internet routing more secure. If you are new to this documentation, we recommend that you read the introduction page to get an overview of what this documentation has to offer.

The table of contents below and in the sidebar should let you easily access the documentation for your topic of interest. You can also use the search function in the top left corner.

Note: This documentation is an open source project maintained by the RPKI team at NLnet Labs, with contributions from the network operator community around the world. We always appreciate your feedback and improvements.

You can submit an issue or pull request on the GitHub repository, or post a message on the RPKI mailing list. If you are interested in providing a translation for this project, please read this guide to get started.

The main documentation is organised into the following sections:
Welcome to the documentation of the Resource Public Key Infrastructure (RPKI). Here, we aim to offer an overview of the RPKI technology itself, as well as some of the tools that are being developed for it. Any software implementer is welcome to add documentation for their tools to this project.

This page gives a broad overview of the RPKI and how it can help make Internet routing using the Border Gateway Protocol (BGP) more secure. This way, you will learn how RPKI can benefit your organisation, as well as helping others to be more secure on the Internet.

1.1 About this Documentation

This documentation is continuously written, corrected and edited by the RPKI team at NLnet Labs. An initial version was written by Alex Band, Tim Bruijnzeels and Martin Hoffmann. Over time, additions from the network operator community, researchers and interested parties around the world were contributed. The documentation is edited via text files in the reStructuredText markup language and then compiled into a static website/offline document using the open source Sphinx and ReadTheDocs tools.

Note: You can contribute to the RPKI documentation by opening an issue or sending patches via pull requests on the GitHub source repository.

All the contents are under the permissive Creative Commons Attribution 3.0 (CC-BY 3.0) license, with attribution to “The RPKI team at NLnet Labs and the RPKI community”.

1.2 About Resource Public Key Infrastructure

RPKI allows holders of Internet number resources to make verifiable statements about how they intend to use their resources. To achieve this, it uses a public key infrastructure that creates a chain of resource certificates that follows the same structure as the way IP addresses and AS numbers are handed down.
RPKI is used to make Internet routing more secure. It is a community-driven system in which open source software developers, router vendors and all five Regional Internet Registries (RIRs) participate, i.e. ARIN, APNIC, AFRINIC, LACNIC and RIPE NCC.

Currently, RPKI is used to let the legitimate holder of a block of IP addresses make an authoritative statement about which AS is authorised to originate their prefix in the BGP. In turn, other network operators can download and validate these statements and make routing decisions based on them. This process is referred to as route origin validation (ROV). This provides a stepping stone to provide path validation in the future.

1.3 Organisation of this Documentation

This documentation is organised into three main sections:

• The General section contains this introduction as well as information about the licensing, authors, etc. It also contains the FAQ and the Quick Help.

• The RPKI Technology section explains the RPKI technology and standards in order for you to get a good sense of the requirements and moving parts. It will help you choose the right RPKI solution for your organisation, with regards to generating, publishing and using RPKI data.

• The RPKI Tools section is about various open source projects that are maintained to support RPKI.
2.1 RPKI Mechanism

2.1.1 What is RPKI and why was it developed?

The global routing system of the Internet consists of a number of functionally independent actors (Autonomous Systems) which use BGP (Border Gateway Protocol) to exchange routing information. The system is very dynamic and flexible by design. Connectivity and routing topologies are subject to change. Changes easily propagate globally within a few minutes. One weakness of this system is that these changes cannot be validated against information existing outside of the BGP protocol itself.

RPKI is a way to define data in an out-of-band system such that the information that are exchanged by BGP can be validated to be correct. The RPKI standards were developed by the IETF (Internet Engineering Task Force) to describe some of the resources of the Internet’s routing and addressing scheme in a cryptographic system. These information are public, and anyone can get access to validate their integrity using cryptographic methods.

2.1.2 I thought we were all using the IRR to check route origin, why do we need RPKI now?

If you’ve been involved in default-free zone Internet engineering for any length of time, you’re probably familiar with RPSL, a routing policy specification language originally defined in RFC 2280 back in 1998. While RPSL has created considerable early enthusiasm and has seen some traction, the Internet was rapidly growing at the time, and the primary focus was on data availability rather than data trustworthiness. Everyone was busy opportunistically documenting the
minimal policy that was necessary to “make things work” with the policy specification language parsing scripts of everyone else so that something would finally ping!

Over time, this has created an extensive repository of obsolete data of uncertain validity spread across dozens of route registries around the world. Additionally, the RPSL language and supporting tools have proven to be too complex to consistently transpose policy into router configuration language - resulting in most published RPSL data being neither sufficiently accurate and up to date for filtering purposes, nor sufficiently comprehensive or precise for being the golden master in router configuration.

RPKI aims to complement and expand upon this effort focusing primarily on trustworthiness, timeliness, and accuracy of data. RPKI ROAs are hierarchically delegated by RIRs based on strict criteria, and are cryptographically verifiable. This offers the Internet community an opportunity to build an up to date and accurate information of IP address origination data on the Internet.

2.1.3 Why are we investing in RPKI, isn’t it easier to just fix the Internet Routing Registry (IRR) system?

The main weakness of the IRR is that it is not a globally deployed system and it lacks the authorisation model to make the system water tight. The result is that out of all the information on routing intent that is published, it is difficult to determine what is legitimate, authentic data and what isn’t. RPKI solves these two problems, as you can be absolutely sure that an authoritative, cryptographically verifiable statement can be made by any legitimate IP resource holder in the world.

2.1.4 Is it true that BGP4 is just not up to the task any longer?

Unfortunately it’s practically impossible to replace BGP right now. We should, however, work on fixing the broken parts and improving the situation.

2.1.5 As RPKI relies on X.509 PKI, isn’t this the same problem with untrustworthy SSL/TLS Certificate Authorities all over again?

Instead of relying on a large number of CAs subject to variable auditing standards which come pre-installed in a browser or an operating system, RPKI relies on just five Trust Anchors, run by the Regional Internet Registries. These are well established and openly governed organisations. Each operator that wishes to get an RPKI resource certificate already has a contractual relationship with one or more of the RIRs.

2.1.6 What is the value of RPKI based BGP Origin Validation without Path Validation?

While Path Validation is a desirable characteristic, the existing RPKI origin validation functionality addresses a large portion of the problem surface.

Existing operational and economic incentives ensure that the most important prefixes for each network are seen via the shortest AS path possible. One such example are network operators setting a higher local preference for prefixes learned via an Internet exchange or private peers (“peerlock”). This reduces the risk that an invalid route could win the BGP route selection process even if it originates from an impersonated but correct origin AS.

For transit providers, direct interconnections and short AS paths are a defining characteristic, positioning them ideally to act on RPKI data and accept only valid routes for redistribution.
Furthermore, operational experience suggests that the vast majority of route hijacks are unintentional rather than malicious, and are caused by ‘fat-fingering’, where an operator accidentally originates a prefix they are not the holder of. Origin Validation would mitigate many of these problems.

While a malicious party willing to intentionally impersonate the origin AS could still take advantage of the lack of Path Validation in some circumstances, widespread RPKI Origin Validation implementation would make such instances easier to pinpoint and address.

2.1.7 When comparing the ROA data set to the announcements my router sees, what are possible outcomes?

In short, routes can have the state Valid, Invalid, or NotFound (a.k.a. Unknown).

- Valid: The route announcement is covered by at least one ROA
- Invalid: The prefix is announced from an unauthorised AS or the announcement is more specific than is allowed by the maximum length set in a ROA that matches the prefix and AS
- NotFound: The prefix in this announcement is not covered (or only partially covered) by an existing ROA

To understand how more specifics, less specifics and partial overlaps are treated, please refer to section 2 of RFC 6811.

2.1.8 I’ve heard the term “route leak” and “route hijack”. What’s the difference?

A route leak is a propagation of one or more routing announcements that are beyond their intended scope. That is an announcement from an Autonomous System (AS) of a learned BGP route to another AS is in violation of the intended policies of the receiver, the sender, and/or one of the ASes along the preceding AS path.

A route hijack is the unauthorised origination of a route.

Note that in either case, the cause may be accidental or malicious and in either case, the result can be path detours, redirection, or denial of services. For more information, please refer to RFC 7908.

2.1.9 If a ROA is cryptographically invalid, will it make my route invalid?

An invalid ROA means that the object did not pass cryptographic validation and is therefore discarded. The statement about routing that was made within the ROA is simply not taken into consideration. An invalid route on the other hand, is the result of a valid ROA, specifically one that had the outcome that a prefix is announced from an unauthorised AS or the announcement is more specific than is allowed by the maximum length set in a ROA that matches the prefix and AS.

2.2 Operations and Impact

2.2.1 Will my router have a problem with all of this cryptographic validation?

No, routers do not do any cryptographic operations to perform Route Origin Validation. The signatures are checked by external software, called Relying Party software or RPKI Validator, which feeds the processed data to the router over a light-weight protocol. This architecture causes minimal overhead for routers.
2.2.2 Does RPKI reduce the BGP convergence speed of my routers?

No, filtering based on an RPKI validated cache has a negligible influence on convergence speed. RPKI validation happens in parallel with route learning (for new prefixes which aren’t yet in cache), and those prefixes will be marked as valid, invalid, or notfound (and the correct policy applied) as the information becomes available.

2.2.3 Why do I need rsync on my system to use a validator?

In the original standards, rsync was defined as the main means of distribution of RPKI data. While it has served the system well in the early years, rsync has several downsides:

- When RPKI relying party software is used on a client system, it has a dependency on rsync. Different versions and different supported options, such as --contimeout, cause unpredictable results. Furthermore, calling rsync is inefficient. It’s an additional process and the output can only be verified by scanning the disk.
- Scaling becomes more and more problematic as the global RPKI data set grows and more operators download and validate data, as with rsync the server in involved in processing the differences.

To overcome these limitations the RRDP protocol was developed and standardised in RFC 8182, which relies on HTTPS. RRDP was specifically designed for scaling and allows CDNs to participate in serving the RPKI data set globally, at scale. In addition, HTTPS is well supported in programming languages so development of relying party software becomes easier and more robust.

Currently, RRDP is implemented on the server side by the ARIN, RIPE NCC and APNIC. Most RPKI Validator implementations either already have RRDP support, or have it on the short term roadmap.

2.2.4 The five RIRs provide a Hosted RPKI system, so why would I want to run a Delegated RPKI system myself instead?

The RPKI system was designed to be a distributed system, allowing each organisation to run their own CA and publish the certificate and ROAs themselves. The hosted RIR systems are in place to offer a low entry barrier into the system, allowing operators to gain operational experience before deciding if they want to run their own CA.

For many operators, the hosted system will be good enough, also in the long term. However, organisations who for example don’t want to be dependent on a web interface for management, who manage address space across multiple RIR regions, or have BGP automation in place that they would like to integrate with ROA management, can all choose to run a CA on their own systems.

2.2.5 Should I run a validator myself, when I can use an external data source I found on the Internet?

The value of signing the authoritative statements about routing intent by the resource holder comes from being able to validate that the data is authentic and has not been tampered with in any way.

When you outsource the validation to a third party, you lose the certainty of data accuracy and authenticity. Conceptually, this is similar to DNSSEC validation, which is best done by a local trusted resolver.

Section 3 of RFC 7115 has an extensive section on this specific topic.

2.2.6 How often should I fetch new data from the RPKI repositories?

According to section 3 of RFC 7115 you should fetch new data at least every 4 to 6 hours. At the moment, the publication of new ROAs in the largest repositories takes about 10-15 minutes. This means fetching every 15-30 minutes is reasonable, without putting unnecessary load on the system.
2.2.7 What if the RPKI system becomes unavailable or some other catastrophe occurs, will my (signed) prefixes become unreachable to others? Will other prefixes my routers learned over BGP become unreachable for me?

RPKI provides a positive statement on routing intent. If all RPKI validator instances become unavailable and all certificates and ROAs expire, the validity state of all routes will fall back to NotFound, as if RPKI were never used. Routes with this state should be accepted according to section 5 of RFC 7115, as this state will unfortunately be true for the majority of routes.

2.2.8 What if the Validator I use crashes and my router stops getting a feed. What will happen to the prefixes I learn over BGP?

All routers that support Route Origin Validation allow you to specify multiple Validators for redundancy. It is recommended that you run multiple instances, preferably from independent publishers and on separate subnets. This way you rely on multiple caches.

In case of a complete failure, all routes will fall back to the NotFound state, as if Origin Validation were never used.

2.2.9 I don’t want to rely on the RPKI data set in all cases, but I want to have my own preferences for some routes. What can I do?

You can always apply your own, local overrides on specific prefixes/announcements and override the RPKI data you fetch from the repositories. Specifying overrides is in fact standardised in RFC 8416, “Simplified Local Internet Number Resource Management with the RPKI (SLURM)”.

2.2.10 Is there any point in signing my routes with ROAs if I don’t validate and filter myself?

Yes, signing your routes is always a good idea. Even if you don’t validate yourself someone else will, or in worst case someone else might try to hijack your prefix. Imagine what could happen if you haven’t signed your prefixes…

2.3 Miscellaneous

2.3.1 Why isn’t the ARIN RPKI TAL like other public key files?

Unlike the other RIRs, which distribute their TAL publicly, ARIN has a policy requiring users to explicitly agree to terms and conditions concerning its TAL. Note that this policy is not without controversy as discussed here and here on the NANOG list.

Job Snijders made a video explaining his perspective on the ARIN TAL. Christopher Yoo and David Wishnick authored a paper titled Lowering Legal Barriers to RPKI Adoption.

Ben Cox performed various RPKI measurements and concluded that the ARIN TAL is used far less than TALs from their RIR counter parts. This has led to a situation where ROAs created under the ARIN TAL offer less protection against BGP incidents than other RIRs. State of RPKI: Q4 2018.
2.3.2 What is the global adoption and data quality of RPKI like?

There are several initiatives that measure the adoption and data quality of RPKI:

- RPKI Analytics, by NLnet Labs
- Global certificate and ROA statistics, by RIPE NCC
- Cirrus Certificate Transparency Log, by Cloudflare
- The RPKI Observatory, by nusenu
- RPKI Deployment Monitor, by NIST

2.3.3 I want to use the RPKI services from a specific RIR that I’m not currently a member of. Can I transfer my resources?

The RPKI services that each RIR offers differ in conditions, terms of service, availability and usability. Most RIRs have a transfer policy that allow their members to transfer their resources from one RIR region to another. Organisations may wish to do this so that they bring all resources under one entity, simplifying management. Others may do this because they are are looking for a specific set of terms with regards to the holdership of their resources. Please check with your RIR for the possibilities and conditions for resource transfers.

2.3.4 Will RPKI be used as a censorship mechanism allowing governments to make arbitrary prefixes unroutable on a whim?

Unlikely. In order to suppress a prefix, it would be necessary to both revoke the existing ROA (if one is present) and publish a conflicting ROA with a different origin.

These characteristics make using RPKI as a mechanism for censorship a rather convoluted and uncertain way of achieving this goal, and has broad visibility (as the conflicting ROA, as well as the Regional Internet Registry under which it was issued, will be immediately accessible to everyone). A government would be much better off walking into the data center and confiscate your equipment.

2.3.5 What are the long-term plans for RPKI?

With RPKI Route Origin Validation being deployed in more and more places, there are several efforts to build upon this to offer out-of-band Path Validation. Autonomous System Provider Authorisation (ASPA) currently has the most traction in the IETF, defined in these drafts: draft-azimov-sidrops-aspa-profile and draft-azimov-sidrops-aspa-verification.
If you’re reading this page, chances are you find yourself in a situation where you’ve been told by someone that your RPKI ROAs make your routes invalid and you don’t know what that means. The aim of the content on this page is to point you in the right direction and provide further resources that can be of assistance. This page is not meant for experts, and many technicalities will be glossed over in order to be able to provide easy to understand answers for all knowledge levels.

3.1 What is RPKI or ROA?

RPKI stands for Resource Public Key Infrastructure, ROA stands for Route Origin Authorisation.

3.2 What do they do?

They provide a method for the originator of a route to assert they are the correct originator and that other originators are not valid.

3.3 How does it work?

The “root” assigner of all IP space (v4+v6) is IANA. They have delegated this space to one of the RIRs (ARIN, RIPE NCC, APNIC, LACNIC, and AFRINIC). In turn, those RIRs assign the space to other entities. Each RIR has a portal where the owner of the space can assert the origination ASN, which then generates a ROA for that particular combination of route and origination ASN. This ROA is then published out by the RIR so that anyone can view them.

3.4 What is in a ROA?

A ROA is a signed statement that consists of a prefix, a maximum prefix length, and originating ASN.
3.5 What happens next?

Any operator is free to get that list of ROAs from the RIRs and use that to tell their routers to take action based on the ROA. A particular announcement will generally have one of three states:

**NotFound (a.k.a. Unknown)** This is the default state if no ROA has been made for the announcement. It is expected that all operators will allow these routes to be installed in their routers.

**Valid** This is the state if the ROA and route announcement matches. It is expected that all operators will allow these routes to be installed in their routers. It is possible they may up-preference these routes.

**Invalid** This is the state if the ROA and route announcement are different. They either differ in originating ASN or is more specific than is allowed by the maximum prefix length that is set in the ROA. If an operator is using RPKI in a strict fashion, odds are good that this announcement will not be installed into their routers.

3.6 What can I do about my route having an Invalid state?

The only entity that can make any changes to the ROA is the RIR-listed owner of the IP space. Most likely the owner of the IP space has created their ROAs in the Hosted RPKI interface of the RIR, which is part of their respective member portals:

- **AFRINIC**: https://my.afrinic.net
- **APNIC**: https://myapnic.net
- **ARIN**: https://account.arin.net
- **LACNIC**: https://milacnic.lacnic.net
- **RIPE NCC**: https://my.ripe.net

It is important to note that initially, for there to be an RPKI Invalid route, someone must have already entered into one of the above portals and made a ROA for the IP space in question. There is no way for it to have to been done by itself. In other words, there must already be an account at the RIR that is linked to the owner of the IP space.
Resource Public Key Infrastructure (RPKI) revolves around the right to use Internet number resources, such as IP addresses and autonomous system (AS) numbers.

In this PKI, the legitimate holder of a block of IP addresses or AS numbers can obtain a resource certificate. Using the certificate, they can make authoritative, signed statements about the resources listed on it. To understand the structure of RPKI and its usage, we must first look at how Internet number resources are allocated globally.

4.1 Internet Number Resource Allocation

Before being formalised within an organisation, the allocation of Internet number resources, such as IP addresses and AS numbers, had been the responsibility of Jon Postel. At the time, he worked at the Information Sciences Institute (ISI) of the University of Southern California (USC). He performed the role of Internet Assigned Numbers Authority (IANA), which is presently a function of the Internet Corporation for Assigned Names and Numbers (ICANN).

Initially, the IANA function was performed globally, but as the work volume grew due to the expansion of the Internet, Regional Internet Registries (RIRs) were established over the years to take on this responsibility on a regional level. Until the available pool of IPv4 depleted in 2011, this meant that periodically, a large block of IPv4 address space was allocated from IANA to one of the RIRs. In turn, the RIRs would allocate smaller blocks to their member organisations, and so on. IPv6 address blocks and AS numbers are allocated in the same way.

Today, there are five RIRs responsible for the allocation and registration of Internet number resources within a particular region of the world:

- The African Network Information Center (AFRINIC) serves Africa
- The American Registry for Internet Numbers (ARIN) serves Antarctica, Canada, parts of the Caribbean, and the United States
- The Asia-Pacific Network Information Centre (APNIC) serves East Asia, Oceania, South Asia, and Southeast Asia
- The Latin America and Caribbean Network Information Centre (LACNIC) serves most of the Caribbean and all of Latin America
The Réseaux IP Européens Network Coordination Centre (RIPE NCC) serves Europe, the Middle East, Russia, and parts of Central Asia.

In the APNIC and LACNIC regions, Internet number resources are in some cases allocated to National Internet Registries (NIRs), such as NIC.br in Brazil and JPNIC in Japan. NIRs allocate address space to its members or constituents, which are generally organised at a national level. In the rest of world, the RIRs allocate directly to their member organisations, typically referred to as Local Internet Registries (LIRs). Most LIRs are Internet service providers, enterprises, or academic institutions. LIRs either use the allocated IP address blocks themselves, or assign them to End User organisations.

### 4.2 Mapping the Resource Allocation Hierarchy into the RPKI

As illustrated, the IANA has the authoritative registration of IPv4, IPv6 and AS number resources that are allocated to the five RIRs. Each RIR registers authoritative information on the allocations to NIRs and LIRs, and lastly, LIRs record to which End User organisation they assigned resources.

In RPKI, resource certificates attest to the allocation by the issuer of IP addresses or AS numbers to the subject. As a result, the certificate hierarchy in RPKI follows the same structure as the Internet number resource allocation hierarchy, with the exception of the IANA level. Instead, the five RIRs each run a root CA with a trust anchor from which a chain of trust for the resources they each manage is derived.

The IANA does not operate a single root certificate authority (CA). While this was originally a recommendation from the Internet Architecture Board (IAB) to eliminate the possibility of resource conflicts in the system, they reconsidered after operational experience in deployment had caused the RIRs to conclude that the RPKI system would be less brittle using multiple overlapping trust anchors.
Fig. 4.2: The service regions of the five Regional Internet Registries

Fig. 4.3: Internet number resource allocation hierarchy

4.2. Mapping the Resource Allocation Hierarchy into the RPKI
4.3 X.509 PKI Considerations

The digital certificates used in RPKI are based on X.509, standardised in RFC 5280, along with extensions for IP addresses and AS identifiers described in RFC 3779. Because RPKI is used in the routing security context, a common misconception is that this is the Routing PKI. However, certificates in this PKI are called resource certificates and conform to the certificate profile described in RFC 6487.

**Note:** X.509 certificates are typically used for authenticating either an individual or, for example, a website. In RPKI, certificates do not include identity information, as their only purpose is to transfer the right to use Internet number resources.

In addition to RPKI not having any identity information, there is another important difference with commonly used X.509 PKIs, such as SSL/TLS. Instead of having to rely on a vast number of root certificate authorities which come pre-installed in a browser or an operating system, RPKI relies on just five trust anchors, run by the RIRs. These are well established, openly governed, not-for-profit organisations. Each organisation that wishes to get an RPKI resource certificate already has a contractual relationship with one or more of the RIRs.

In conclusion, RPKI provides a mechanism to make strong, testable attestations about Internet number resources. In the next sections, we will look at how this can be used to make Internet routing more secure.
Internet Routing

To understand how RPKI is used to make Internet routing more secure, we must first look at how routing works, what the weaknesses are and which elements RPKI can currently help protect against.

The global routing system of the Internet consists of a number of functionally independent actors called autonomous systems (AS), which use the Border Gateway Protocol (BGP) to exchange routing information.

An autonomous system is a set of Internet routable IP prefixes belonging to a network or a collection of networks that are all managed and supervised by a single entity or organisation. An AS utilises a common routing policy controlled by the entity and is identified by a globally unique 16 or 32-bit number. The AS number (ASN) is assigned by one of the five Regional Internet Registries (RIRs), just like IP address blocks.

The Border Gateway Protocol manages the routed peerings, prefix advertisement and routing of packets between different autonomous systems across the Internet. BGP uses the ASN to uniquely identify each system. In short, BGP is the routing protocol for AS paths across the Internet. The system is very dynamic and flexible by design. Connectivity and routing topologies are subject to change, which easily propagate globally within a few minutes.

Fundamentally, BGP is based on mutual trust between networks. When a network operator configures the routers in their AS, they specify which IP prefixes to originate and announce to their peers. There is no authentication or authorisation embedded within BGP. In principle, an operator can define any ASN as the origin and announce any prefix, also one they are not the holder of.

5.1 BGP Best Path Selection

BGP routing information includes the complete route to each destination. BGP uses the routing information to maintain a database of network reachability information, which it exchanges with other networks. For each prefix in the routing table, BGP continuously and dynamically makes decisions about the best path to reach a particular destination. After the best path is selected, the route is installed in the routing table.

Though there are many factors at play, two of them are the most important to keep in mind throughout the next sections: the preference for shortest path and most specific IP prefix.
5.1.1 Preference for Shortest Path

Out of all the possible routes that a router has in its Routing Information Base (RIB), BGP will always prefer the shortest path to its destination, minimising the amount of hops. When two matching prefixes are announced from two different networks on the Internet, BGP will route traffic to the destination that is topologically closest. This is an important feature of BGP, but when configuration errors occur, it can also be the cause of reachability problems.

![Diagram of network with Client, AS64500, AS64510, AS65540, AS65550, and AS65536]

Fig. 5.1: When the announcement of a prefix is an exact match, the shortest path wins

5.1.2 Preference for Most Specific Prefix

Regardless any local preference, path length or any other attributes, when building the forwarding table, the router will always select most specific IP prefix available. This behaviour is important, but creates the possibility for almost any network to attract someone else’s traffic by announcing an overlapping more specific.

With this in mind, there are several problems that can arise as a result of this behaviour.
Fig. 5.2: Regardless of the path length, the announcement of a more specific prefix always wins
5.2 Routing Errors

Routing errors on the Internet can be classified as route leaks or route hijacks. RFC 7908 provides a working definition of a BGP route leak:

A route leak is the propagation of routing announcement(s) beyond their intended scope. That is, an announcement from an Autonomous System (AS) of a learned BGP route to another AS is in violation of the intended policies of the receiver, the sender, and/or one of the ASes along the preceding AS path. The intended scope is usually defined by a set of local redistribution/filtering policies distributed among the ASes involved. Often, these intended policies are defined in terms of the pair-wise peering business relationship between autonomous systems.

A route hijack, also called prefix hijack, or IP hijack, is the unauthorised origination of a route.

Note: Route leaks and hijacks can be accidental or malicious, but most often arise from accidental misconfigurations. The result can be redirection of traffic through an unintended path. This may enable eavesdropping or traffic analysis and may, in some cases, result in a denial of service or black hole.

Routing incidents occur every day. While several decades ago outages and redirections were often accidental, in recent years they have become more malicious in nature. Some notable events were the AS 7007 incident in 1997, Pakistan’s attempt to block YouTube access within their country, which resulted in taking down YouTube entirely in 2008, and lastly, the almost 1,300 addresses for Amazon Route 53 that got rerouted for two hours in order to steal cryptocurrency, in 2018.

5.3 Mitigation of Routing Errors

One weakness of BGP is that routing errors cannot be easily be deduced from information within the protocol itself. For this reason, network operators have to carefully gauge what the intended routing policy of their peers is. As a result, it is imperative that networks employ filters to only accept legitimate traffic and drop everything else.

There are several well known methods to achieve this. Certain backbone and private peers require a valid Letter of Agency (LOA) to be completed prior to allowing the announcement or re-announcement of IP address blocks. A more widely accepted method is the use of Internet Routing Registry (IRR) databases, where operators can publish their routing policy. Both methods allow other networks to set up filters accordingly.

5.4 The Internet Routing Registry

The Internet Routing Registry (IRR) is a distributed set of databases allowing network operators to describe and query for routing intent. The IRR is used as a verification mechanism of route origination and is widely, though not universally, deployed to prevent accidental or intentional routing disturbances.

The notation used in the IRR is the Routing Policy Specification Language (RPSL), which was originally defined in RFC 2280 in 1998. RPSL is a very expressive language, allowing for an extremely detailed description of routing policy. While IRR usage had created considerable early enthusiasm and has seen quite some traction, the Internet was rapidly growing at the time. This meant that the primary focus was on data availability rather than data trustworthiness.

In later years, it was considered a good practice to extensively document how incoming and outgoing traffic was treated by the network, but nowadays the most prevalent usage is to publish and query for route objects, describing from which ASN a prefix is intended to be originated:
As explained earlier, only the Regional Internet Registries have authoritative information on the legitimate holder of an Internet number resource. This means that the entries in their IRR databases are authenticated, but they are not in any of the other routing registries. Over time, this has created an expansive repository of obsolete data of uncertain validity, spread across dozens of routing registries around the world.

Additionally, the RPSL language and supporting tools have proven to be too complex to consistently transpose policy into router configuration language. This resulted in most published RPSL data being neither sufficiently accurate and up to date for filtering purposes, nor sufficiently comprehensive or precise for being the golden master in router configuration.

In conclusion, the main weakness of the IRR is that it is not a globally deployed system and it lacks the authorisation model to make the system water tight. The result is that out of all the information on routing intent that is published, it is difficult to determine what is legitimate, authentic data and what isn’t.

RPKI solves these problems, as you can be absolutely sure that an authoritative, cryptographically verifiable statement can be made by any legitimate IP resource holder in the world. In the next sections we will look at how this is achieved.
Now that we’ve looked at how the RPKI structure is built and understand the basics of Internet routing, we can look at how RPKI can be used to make BGP more secure.

RPKI provides a set of building blocks allowing for various levels of protection of the routing system. The initial goal is to provide route origin validation, offering a stepping stone to providing path validation in the future. Both origin validation and path validation are documented IETF standards. In addition, there are drafts describing autonomous system provider authorisation, aimed at providing a more lightweight, incremental approach to path validation.

### 6.1 Route Origin Validation

With route origin validation (ROV), the RPKI system tries to closely mimic what route objects in the IRR intend to do, but then in a more trustworthy manner. It also adds a couple of useful features.

Origin validation is currently the only functionality that is operationally used. The five RIRs provide functionality for it, there is open source software available for creation and publication of data, and all major router vendors have implemented ROV in their platforms. Various router software implementations offer support for it, as well.

#### 6.1.1 Route Origin Authorisations

Using the RPKI system, the legitimate holder of a block of IP addresses can use their resource certificate to make an authoritative, signed statement about which autonomous system is authorised to originate their prefix in BGP. These statements are called Route Origin Authorisations (ROAs).

The creation of a ROA is solely tied to the IP address space that is listed on the certificate and not to the AS numbers. This means the holder of the certificate can authorise any AS to originate their prefix, not just their own autonomous systems.

**Maximum Prefix Length**

In addition to the origin AS and the prefix, the ROA contains a maximum length (maxLength) value. This is an attribute that a route object in RPSL doesn’t have. Described in RFC 6482, the maxLength specifies the maximum...
length of the IP address prefix that the AS is authorised to advertise. This gives the holder of the prefix control over the level of deaggregation an AS is allowed to do.

For example, if a ROA authorises a certain AS to originate 192.0.1.0/24 and the maxLength is set to /25, the AS can originate a single /24 or two adjacent /25 blocks. Any more specific announcement is unauthorised by the ROA. Using this example, the shorthand notation for prefix and maxLength you will often encounter is 192.0.1.0/24-25.

Warning: According to RFC 7115, operators should be conservative in use of maxLength in ROAs. For example, if a prefix will have only a few sub-prefixes announced, multiple ROAs for the specific announcements should be used as opposed to one ROA with a long maxLength.

Liberal usage of maxLength opens up the network to a forged origin attack. ROAs should be as precise as possible, meaning they should match prefixes as announced in BGP.

In a forged origin attack, a malicious actor spoofs the AS number of another network. With a minimal ROA length, the attack does not work for sub-prefixes that are not covered by overly long maxLength. For example, if, instead of 10.0.0.0/16-24, one issues 10.0.0.0/16 and 10.0.42.0/24, a forged origin attack cannot succeed against 10.0.666.0/24. They must attack the whole /16, which is more likely to be noticed because of its size.

6.1.2 Route Announcement Validity

When a network operator creates a ROA for a certain combination of origin AS and prefix, this will have an effect on the RPKI validity of one or more route announcements. Once a ROA is validated, the resulting object contains an IP prefix, a maximum length, and an origin AS number. This object is referred to as validated ROA payload (VRP).

When comparing VRPs to route announcements seen in BGP, RFC 6811 describes their possible statuses, which are:

Valid The route announcement is covered by at least one VRP. The term covered means that the prefix in the route announcement is equal, or more specific than the prefix in the VRP.

Invalid The prefix is announced from an unauthorised AS, or the announcement is more specific than is allowed by the maxLength set in a VRP that matches the prefix and AS.

NotFound The prefix in this announcement is not, or only partially covered by a VRP.
Anyone can download and validate the published certificates and ROAs and make routing decisions based on these three outcomes. In the *Using RPKI Data* section, we’ll cover how this works in practice.

### 6.2 Path Validation

Currently, RPKI only provides origin validation. While BGPsec path validation is a desirable characteristic and standardised in [RFC 8205](https://tools.ietf.org/html/rfc8205), real-world deployment may prove limited for the foreseeable future. However, RPKI origin validation functionality addresses a large portion of the problem surface.

For many networks, the most important prefixes can be found one AS hop away (coming from a specific peer, for example), and this is the case for large portions of the Internet from the perspective of a transit provider - entities which are ideally situated to act on RPKI data and accept only valid routes for redistribution.

Furthermore, the vast majority of route hijacks are unintentional, and are caused by ‘fat-fingering’, where an operator accidentally originates a prefix they are not the holder of.

Origin validation would mitigate most of these problems, offering immediate value of the system. While a malicious party could still take advantage of the lack of path validation, widespread RPKI implementation would make such instances easier to pinpoint and address.

With origin validation being deployed in more and more places, there are several efforts to build upon this to offer out-of-band path validation. Autonomous system provider authorisation (ASPA) currently has the most traction in the IETF, and is described in these drafts: [draft-azimov-sidrops-aspa-profile](https://tools.ietf.org/html/draft-azimov-sidrops-aspa-profile) and [draft-azimov-sidrops-aspa-verification](https://tools.ietf.org/html/draft-azimov-sidrops-aspa-verification).
Implementation Models

RPKI is designed to allow every resource holder to generate and publish cryptographic material on their own systems. This is commonly referred to as delegated RPKI. To offer a turn-key solution, each RIR also offers a hosted RPKI system in their member portals. Both models have their own advantages, based on the specific requirements of the organisation using the system.

No matter what implementation model you choose, it always a good idea to publish ROAs for your BGP announcements. Even when you are still evaluating how to deploy RPKI within your organisation, the benefits are immediate. Others can already filter based on what you publish, offering protection for you and other Internet users. For example, in case someone inadvertently announces your address space from their AS, it will be flagged as Invalid and dropped by everyone who has deployed route origin validation.

Important: Once you start authorising announcements with RPKI, it is imperative that ROAs are created for all route origins from the prefixes you hold, including more specifics announced by other business units or customers. In addition, RPKI should become a standard part of operations, ensuring staff is trained and ROAs are continually monitored and maintained.

7.1 Hosted RPKI

In 2008, when the five RIRs committed to start offering RPKI services, it was clear that there would be an early adopters phase for a considerable amount of time. Given the past experiences with IPv6 and DNSSEC uptake, the RIRs decided to offer a hosted RPKI solution to lower the entry barrier into the technology. This way, organisations could easily get operational experience with the technology, without having to manage a certificate authority themselves.

Hosted RPKI offers a fair balance between ease-of-use, maintenance and flexibility. It allows users to log into their RIR member portal and request a resource certificate, which is securely hosted on the servers of the RIR. All cryptographic operations, such as key roll overs, are automated. The certificates and ROA are published in repositories hosted by the RIR. In short, there is nothing that the user has to manage, apart from creating and maintaining ROAs.

The functionality and user interfaces of the hosted RPKI implementations vary greatly across the five RIRs. Despite these variations, if you are an organisation with a single ASN and a handful of statically announced IP address blocks that are not delegated to customers, hosted RPKI is sufficient for most use cases.
Fig. 7.1: Example of the Hosted RPKI interface of the RIPE NCC
### 7.1.1 Functional differences across RIRs

This section provides an overview of the functionality each RIR provides to help users manage RPKI, which is summarised in the table below.

First, the table indicates if the RPKI system supports setting up delegated RPKI, so users can run their own certificate authority if they want. An RIR may also offer a publication server for users running delegated RPKI. When using the hosted RPKI system, there is an overview if multiple users can be authorised to manage ROAs, and whether they can authenticate using two-factors.

To make management of ROAs easier, some systems provide a list of all announcements with certified address space that are seen by BGP route collectors, such as the RIPE Routing Information Service (RIS). ROAs have an explicit start and end validity date, but in some cases it is possible to automatically renew the ROAs, so that they are valid for as long as there is an entry in the web interface. In addition, it may be possible to synchronise the management of “route” objects in the IRR with the ROAs that are created. An application programming interface (API) may be provided to make batch processing easier.

To improve retrieval of published RPKI data by relying party software, the RPKI Repository Delta Protocol (RRDP) protocol (RFC 8182) was developed. Support for this standard is listed as well.

Lastly, nonrepudiation refers to the inability for a party to dispute or deny having performed an action.

<table>
<thead>
<tr>
<th></th>
<th>APNIC</th>
<th>AFRINIC</th>
<th>ARIN</th>
<th>LACNIC</th>
<th>RIPE NCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for delegated RPKI</td>
<td>Yes</td>
<td>Yes¹</td>
<td>Yes</td>
<td>Yes²</td>
<td>Yes</td>
</tr>
<tr>
<td>Publication service for delegated RPKI</td>
<td>Yes³</td>
<td>No</td>
<td>No⁴</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi-user support</td>
<td>Yes</td>
<td>No</td>
<td>No⁵</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Two-factor authentication</td>
<td>Yes</td>
<td>No</td>
<td>Yes⁶</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>BGP route collector suggestions</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-renew ROAs</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes⁷</td>
<td>Yes</td>
</tr>
<tr>
<td>Match “route” objects with ROAs</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>API</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Publication via RRDP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Nonrepudiation</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

A final differentiator is the publication interval of each RIR repository. Please keep in mind that once a ROA is created by a user in one of the hosted systems, it can take between several minutes up to multiple hours before the object is published and available for download, depending on the RIR system you use.

### 7.2 Delegated RPKI

Operators who prefer more control and have better integration with their systems can run their own child CA. This model is usually referred to as delegated RPKI.

In this model, the certificate authority that manages object signing is functionally separated from the publication of cryptographic material. This means that an organisation can run a CA and either publish themselves, or delegate this responsibility to a third party, such as a hosting company or cloud provider.

---

¹ Available in the test environment only.
² Available upon request.
³ Available upon request.
⁴ On the roadmap
⁵ Requires a client X.509 certificate to use RPKI.
⁶ Requires a ROA Request Key Pair.
⁷ Explicit opt-in feature.
There may be various reasons for organisations to choose this model. For example, this may be useful for organisations that need to be able to delegate RPKI to their customers or different business units, so that they can run a CA on their systems and manage ROAs themselves.

Alternatively, enterprises who manage large amounts of address space across various RIRs, may not want to manage ROAs in up to five different web interfaces. Instead, they might prefer to be operationally independent from the RIR and manage everything from within one package that is tightly integrated with IP address management and provisioning systems.

Lastly, in the LACNIC and APNIC regions there are several National Internet Registries who provide registration services on a national level to their members and constituents. They also need to be operationally independent and run a certificate authority as a child of their RIR.
CHAPTER 8

Using RPKI Data

Validation is a key part of any public key infrastructure. The value from signing comes with validation, and should always be done by the party relying on the data. If validation is outsourced to a third party, you can never be certain if the data is complete, or tampered with in any way.

Operators who want to deploy route origin validation in their BGP decision making process have to fetch and validate all of the published RPKI data. As with any PKI, you have to start with one or more entities you are prepared to trust. In the case of RPKI, these are the five Regional Internet Registries.

8.1 Connecting to the Trust Anchor

When you want to retrieve all RPKI data, you connect to the trust anchor that each RIR provides. The root certificate contains pointers to its children, which contain pointers to their children, and so on. These certificates, and other cryptographic material such as ROAs, can be published in the repository that the RIR provides, or a repository operated by an organisation who either runs delegated RPKI themselves, or hosts a repository as a service. As a person who wants to fetch and validate the data, formally known as a relying party, it is not a concern where data is published. By simply connecting to the trust anchor, the chain of trust is followed automatically.

The RIR trust anchor is found through a static trust anchor locator (TAL), which is a very simple file that contains a URL to retrieve the trust anchor and a public key to verify its authenticity. The reason the TAL exists is because it’s very likely that the contents of the self signed root certificate change, due to resource transfers between RIRs. By using a TAL, the data in the trust anchor can change, without it needing to be redistributed.

8.2 Fetching and Verifying

Various open source relying party software packages, also known as RPKI validators, are available in order to download, verify and process RPKI data. Please note that most RPKI validators come preinstalled with TALs for all RIRs except the one for ARIN, as they require users to first review and agree to their Relying Party Agreement.

When the validator runs, it will start retrieval at each of the RIR trust anchors and follows the chain of trust to fetch all published certificates and ROAs. Fetching data was originally done via rsync but RIRs and software developers
are gradually migrating to the RPKI Repository Delta Protocol (RRDP) for retrieval, standardised in RFC 8182. This protocol uses HTTPS, which makes development and implementation easier, and opens up possibilities for Content Delivery Networks to participate in serving RPKI data. Work to deprecate rsync altogether is ongoing in the IETF.

Once the data has been downloaded, the validator will verify the signatures on all objects and output the valid route origins as a list. Each object in this list contains an IP prefix, a maximum length, and an origin AS number. This object is referred to as validated ROA payload (VRP). The collection of VRPs is known as the validated cache.

**Note:** Objects that do not pass cryptographic verification are discarded. Any statements made about route origins are not considered, as if a ROA was never published. As a result, they will not affect any route announcements.

Please note that objects that do not pass cryptographic verification are sometimes referred to as ‘invalid ROAs’, but we like to avoid this term because validity is used elsewhere in a different context.

Fetching and verification of data should be performed periodically, in order to process updates. Though the standards recommend retrieval at least once every 24 hours, current operational practice recommends that processing updates every 30 to 60 minutes is reasonable.

### 8.3 Validating Routes

As explained in the *Route Origin Validation* section, when comparing VRPs to the route announcements seen in BGP, it will have an effect on their RPKI validity state. They can be:

- **Valid** The route announcement is covered by at least one VRP. The term *covered* means that the prefix in the route announcement is equal, or more specific than the prefix in the VRP.

- **Invalid** The prefix is announced from an unauthorised AS, or the announcement is more specific than is allowed by the maxLength set in a VRP that matches the prefix and AS.

- **NotFound** The prefix in this announcement is not, or only partially covered by a VRP.

Please carefully note the use of the word *validity*. Because RPKI revolves around signing and verifying cryptographic objects, it’s easy to confuse this term with the validity state of a BGP announcement. As mentioned, it can occur that a ROA doesn’t pass cryptographic verification, for example because it expired. As a result, it is discarded and will not affect any BGP announcement. In turn, only a validated ROA payload—sometimes referred to as ‘valid ROA’—can make a BGP announcement Valid or Invalid.

A route announcement may be covered by several VRPs. For example, there may be a VRP for the aggregate announcement, which overlaps with a customer announcement of a more specific prefix from a different AS. A route announcement will be Valid as long as there is one covering VRP that authorises it.

Based on the three validity outcomes, operators can make an informed decision what to do with the BGP route announcements they see. As a general guideline, announcements with Valid origins should be preferred over those with NotFound or Invalid origins. Announcements with NotFound origins should be preferred over those with Invalid origins.

As origin validation is deployed incrementally, the amount of IP address space that is covered by a ROA will gradually increase over time. Therefore, accepting the NotFound validity should be done for the foreseeable future.

**Important:** For route origin validation to succeed in its objective, operators should ultimately drop all BGP announcements that are marked as Invalid. Before taking this step, organisations should first analyse the effects of doing this, to avoid unintended results. Initially accepting Invalid announcements and giving them a lower preference, as well as tagging them with a BGP community is a good first step to measure this.
8.4 Local Overrides

Sometimes there is an operational need to accept Invalid announcements temporarily. Local overrides allow you to manage your own exceptions to the validated cache. This ensures that you remain in full control of the VRPs used by your routers. For example, if an Invalid origin is the result of a misconfigured ROA, you may accept it until the operator in question has resolved the issue. A format named SLURM is available for this, which is standardised in RFC 8416.

SLURM provides several ways to achieve exceptions. First, you can add a VRP specifically for the affected route by specifying the correct ASN, prefix and maximum length. Secondly, you can filter out an existing VRP, thereby moving the route back to NotFound state. In general, the former is the safer way, as it deals better with changing ROAs. Lastly, it is possible to allow all routes from a certain ASN or prefix. It is advised to use overrides with care, as liberal usage may have unintended consequences.

8.5 Feeding Routers

The validated cache can be fed directly into RPKI-capable routers via the RPKI to Router Protocol (RPKI-RTR), described in RFC 8210. Many routers, including Cisco, Juniper, Nokia, as well as BIRD and OpenBGPD support processing the validated cache. Alternatively, most validators can export the cache in various useful formats for processing outside of the router, in order to set up filters.

Note that your router does not perform any of the cryptographic validation, this is all handled by the relying party software. In addition, using RPKI causes minimal overhead for routers and has a negligible influence on convergence speed. Validation happens in parallel with route learning for new prefixes which are not yet in the cache. Those prefixes will be marked as Valid, Invalid, or NotFound as the information becomes available, after which the correct policy is applied.

Please keep in mind that the RPKI validator software you run in your network fetches cryptographic material from the outside world. To do this, it needs at least ports 873 and 443 open for rsync and HTTPS, respectively. In most cases, the processed data is fed to a router via RPKI-RTR over a clear channel, as it’s running in your local network. Currently, only Cisco IOS-XR provides a practical means to secure transports for RPKI-RTR, using SSH.

It is recommended to run multiple validator instances as a failover measure. The router will use the union of RPKI data from all validators to which they are connected. This means that (temporary) differences in the validated cache
produced by the validators, for example due to differing fetching intervals, does not pose a problem.

In the *Router Support* section we will look at which routers support route origin validation, and how to get started with each.
Several router vendors participated in the development of the RPKI standards in the IETF, ensuring the technology offered an end-to-end solution for route origin validation. The RPKI to Router protocol (RPKI-RTR) is standardised in RFC 6810 (v0) and RFC 8210 (v1). It is specifically designed to deliver validated prefix origin data to routers. This, as well as origin validation functionality, is currently available in on various hardware platforms and software solutions.

### 9.1 Hardware Solutions

**Important:** The versions listed here are the earliest ones where RPKI support became available. However, a newer version may be required to get recommended improvements and bug fixes. Please check your vendor documentation and knowledge base.

- **Juniper** — [Documentation](#) Junos version 12.2 and newer. Please read PR1461602 and PR1309944 before deploying.
- **Cisco** — [Documentation](#) IOS release 15.2 and newer, as well as Cisco IOS/XR since release 4.3.2.
- **Nokia** — [Documentation](#) SR OS 12.0.R4 and newer, running on the 7210 SAS, 7250 IXR, 7750 SR, 7950 XRS and the VSR.
- **Arista** — [Blog post](#) EOS 4.24.0F and newer
- **MikroTik** — [RouterOS v7 BETA forum thread](#) · [RPKI forum thread](#) 7.0beta7 and newer

### 9.2 Software Solutions

Various software solutions have support for origin validation:

- **BIRD**
In some solutions, such as OpenBGPD, RPKI-RTR is not available but the same result can be achieved through a static configuration. The router will periodically fetch the validated cache and allow operators to set up route maps based on the result. Relying party software such as Routinator and rpki-client can export validated data in a format that OpenBGPD can parse.

**RTRlib** is a C library that implements the client side of the RPKI-RTR protocol, as well as route origin validation. RTRlib powers RPKI in BGP software routers such as FRR. In a nutshell, it maintains data from RPKI relying party software and allows to verify whether an autonomous system (AS) is the legitimate origin AS, based on the fetched valid ROA data. **BGP-SRx** by NIST is a prototype that can perform similar functions.
This page provides an overview of projects that support RPKI. It includes statistics, measurements projects and presentations about operational experiences. Finally, there is an overview of all work in the Internet Engineering Task Force relevant to RPKI.

The *Software Projects* page an overview of all available tools for using RPKI.

### 10.1 Books

- **BGP RPKI: Instructions for use**, by Flavio Luciani & Tiziano Tofoni (PDF)
- **Juniper Day One: Deploying BGP Routing Security**, by Melchior Aelmans & Niels Raijer (PDF)

### 10.2 Insights and Statistics

There are several initiatives that measure the adoption and data quality of RPKI:

- Global country stats, with AS and IP prefix analysis, by NL.net Labs
- Cirrus Certificate Transparency Log, by Cloudflare
- Global certificate and ROA statistics, by RIPE NCC
- RPKI Deployment Monitor, by NIST
- The RPKI Observatory, by nusenu
- RPKI connection test, by RIPE Labs
10.3 Operational Experiences

Using RPKI with IXP Manager  Documentation to set up Routinator, OctoRPKI and the RIPE NCC Validator with BIRD 2.x  
Use Routinator with Cisco IOS-XR  Blog post by Fabien Vincent  
Wikimedia RPKI Validation Implementation  Documentation by Arzhel Younsi describing RPKI validator and router configuration  
Dropping RPKI invalid routes in a service provider network  Lightning talk by Nimrod Levy - AT&T, NANOG 75, February 2019  
RPKI and BGP: our path to securing Internet Routing  Blog post by Jérôme Fleury & Louis Poinsignon - Cloudflare, September 2018  
RPKI For Managers  Presentation by Niels Raijer - Fusix Networks, NLNOG Day 2018, September 2018  
RPKI at IXP Route Servers  Presentation by Nick Hilliard - INEX, RIPE 78, May 2019

10.4 Examples of BGP Hijacks

How Verizon and a BGP Optimizer Knocked Large Parts of the Internet Offline Today  Cloudflare Blog, 24 June 2019  
BGP / DNS Hijacks Target Payment Systems  Oracle Internet Intelligence, 3 August 2018  
Shutting down the BGP Hijack Factory  Oracle Dyn, 10 July 2018  
Suspicious event hijacks Amazon traffic for 2 hours, steals cryptocurrency  Ars Technica, 24 April 2018  
Popular Destinations rerouted to Russia  BGPmon, 12 December 2017  
Insecure routing redirects YouTube to Pakistan  Ars Technica, 25 February 2008

10.5 IETF Documents

Most of the original work on RPKI standardisation for both origin and path validation was done in the Secure Inter-Domain Routing (sidr) working group. After the work was completed, the working group was concluded.

Since then, the SIDR Operations (sidrops) working group was formed. This working group develops guidelines for the operation of SIDR-aware networks, and provides operational guidance on how to deploy and operate SIDR technologies in existing and new networks.

All relevant drafts and standards can be found in the archives of these two working groups, with a few exceptions, such as draft-ietf-grow-rpki-as-cones.
Software Projects

This section provides an overview of all well known open source projects that support RPKI. It includes Relying Party software for validating RPKI data, Certificate Authority software to run RPKI on your own infrastructure and supporting tools that help deployment and integration.

11.1 Relying Party Software

**Dragon Research Labs Validating Cache**  Software to fetch and validate RPKI certificates and serve them to routers by Dragon Research Labs, written in the Python programming language.

**Fort Validator**  MIT-licensed Relying Party software by NIC.mx, written in C.

**OctoRPKI**  Cloudflare’s Relying Party software, written in the Go programming language.

**RIPE NCC RPKI Validator**  Full-featured RPKI relying party software, written by the RIPE NCC in the Java programming language.

**Routinator**  RPKI relying party software written by NLnet Labs in the Rust programming language, designed to have a small footprint and great portability.

**rpki-client(8)**  rpki-client is written in C as part of the OpenBSD project, and has been ported to various Linux distributions. Designed to be secure and simple to use.

**RPSTIR**  Relying Party Security Technology for Internet Routing (RPSTIR) software, initially written by Raytheon BBN Technologies in the C programming language, now maintained by ZDNS.

11.2 Certificate Authority Software

**Dragon Research Labs Certificate Authority**  RPKI Certificate Authority software by Dragon Research Labs, written in the Python programming language.

**Krill**  RPKI Certificate Authority software by NLnet Labs, written in the Rust programming language.
11.3 Supporting Tools

**BGP-SRx** SRx is an open source reference implementation and research platform by the National Institute for Standards and Technology (NIST). It is intended for investigating emerging BGP security extensions and supporting protocols such as RPKI Origin Validation and BGPSec Path Validation.

**GoRTR** An open-source implementation of RPKI to Router protocol (RFC 6810) using the Go programming language. This project is maintained by Louis Poinsignon at Cloudflare.

**pmacct** pmacct is a small set of multi-purpose passive network monitoring tools. It can account, classify, aggregate, replicate and export forwarding-plane data, i.e. IPv4 and IPv6 traffic; collect and correlate control-plane data via BGP and BMP; collect and correlate RPKI data; collect infrastructure data via Streaming Telemetry.

The pmacct toolset can perform RPKI Origin Validation and present the outcome as a property in the flow aggregation process. Because it separates out the various types kinds of (invalid) BGP announcements, operators can a good grasp on how their connectivity to the rest of the Internet would look like after deploying a “invalid == reject” policy.

**rpki-ov-checker** rpki-ov-checker is an open source utility to quickly analyse BGP RIB dumps and the potential impact of deploying “invalid is reject” routing policies.

**RTRlib** The RTRlib implements the client-side of the RPKI-RTR protocol (RFC 6810, RFC 8210) and BGP Prefix Origin Validation (RFC 6811). This also enables the maintenance of router keys, which are required to deploy BGPSec.

RTRlib was originally founded by researchers from the Computer Systems & Telematics group at Freie Universität Berlin and researchers from the INET research group at Hamburg University of Applied Sciences, under the supervision of Matthias Wahlisch and Thomas Schmidt. It is now a community project.
Krill is a free, open source Resource Public Key Infrastructure (RPKI) daemon, featuring a Certificate Authority (CA) and publication server, written by NL.net Labs.

You are welcome to ask questions or post comments and ideas on our RPKI mailing list. If you find a bug in Krill, feel free to create an issue on GitHub. Krill is distributed under the Mozilla Public License 2.0.

Krill is intended for:

- Organisations who hold address space from multiple Regional Internet Registries (RIRs). Using Krill, ROAs can be managed seamlessly for all resources within one system.

- Organisations that need to be able to delegate RPKI to their customers or different business units, so that they can run their own CA and manage ROAs themselves.

- Organisations who do not wish to rely on the web interface of the hosted systems that the RIRs offer, but require RPKI management that is integrated with their own systems using a common UI or API.

Using Krill, you can run your own RPKI Certificate Authority as a child of one or more parent CAs, usually a Regional Internet Registry (RIR) or National Internet Registry (NIR). With Krill you can run under multiple parent CAs seamlessly and transparently. This is especially convenient if your organisation holds address space in several RIR regions, as it can all be managed as a single pool.

Krill can also act as a parent for child CAs. This means you can delegate resources down to children of your own, such as business units, departments, members or customers, who, in turn, manage ROAs themselves.

Lastly, Krill features a publication server so you can either publish your certificate and ROAs with a third party, such as your NIR or RIR, or you publish them yourself. Krill can be managed with a web user interface, from the command line and through an API.

You can choose to run Krill as a standalone application or run it together with Krill Manager, a tool that brings together all of the puzzle pieces needed to administer and run Delegated RPKI with Krill as a highly available scalable service.
Welcome to Krill

Let's start by creating your RPKI Certificate Authority (CA). It will be used to configure Delegated RPKI with one or multiple parent CAs, usually your Regional or National Internet Registry.

The handle you choose will identify your CA in interactions with parent and child CAs. It will not be published in the RPKI. Please choose a handle that helps others recognize your organization. Once set, the handle cannot be changed.

* CA Handle
Krill Manager includes Docker, Gluster, NGINX, Rsyncd, as well as Prometheus and Fluentd outputs for monitoring and log analysis. The integrated setup wizard allows for seamless TLS configuration, optionally using Let’s Encrypt, as well as automated updating of the application itself and all included components.

Krill with Krill Manager is available for free as a 1-Click App on the AWS Marketplace and the DigitalOcean Marketplace.

12.1 Before You Start

RPKI is a very modular system and so is Krill. Which parts you need and how you fit them together depends on your situation. Before you begin with installing Krill, there are some basic concepts you should understand and some decisions you need to make.

12.1.1 The Moving Parts

With Krill there are two fundamental pieces at play. The first part is the Certificate Authority (CA), which takes care of all the cryptographic operations involved in RPKI. Secondly, there is the publication server which makes your certificate and ROAs available to the world.

In almost all cases you will need to run the CA that Krill provides under a parent CA, usually your Regional Internet Registry (RIR) or National Internet Registry (NIR). The communication between the parent and the child CA is initiated through the exchange of two XML files, which you need to handle manually: a child request XML and a parent response XML. This involves generating the request file, providing it to your parent, and giving the response file back to your CA.

After this initial exchange has been completed, all subsequent requests and responses are handled by the parent and child CA themselves. This includes the entitlement request and response that determines which resources you receive on your certificate, the certificate request and response, as well as the revoke request and response.

**Important:** The initial XML file exchange is the only manual step required to get started with Delegated RPKI. All other requests and responses, as well as re-signing and renewing certificates and ROAs are automated. As long as Krill is running, it will automatically update the entitled resources on your certificate, as well as reissue certificates, ROAs and all other objects before they expire or become stale. Note that even if Krill does go down, you have 8 hours to bring it back up before data starts going stale.

Whether you also run the Krill publication server depends on if you can, or want to use one offered by a third party. For the general wellbeing of the RPKI ecosystem, we would generally recommend to publish with your parent CA, if available. Setting this up is done in the same way as with the CA: exchanging a publisher request XML and a repository response XML.

12.1.2 Publishing With Your Parent

If you can use a publication server provided by your parent, the installation and configuration of Krill becomes extremely easy. After the installation has completed, you perform the XML exchange twice and you are done.

Krill is designed to run continuously, but there is no strict uptime requirement for the CA. If the CA is not available you just cannot create or update ROAs. This means you can bring Krill down to perform maintenance or migration, as long as you bring it back up within 8 hours to ensure your cryptographic objects are re-signed before they go stale.
Note: This scenario illustrated here also applies if you use an RPKI publication server offered by a third party, such as a cloud provider.

At this time, only Brazilian NIR NIC.br offer a publication server for their members. Several RIRs have this functionality on their roadmap. This means that in most cases, you will have to publish yourself.

### 12.1.3 Publishing Yourself

Krill features a publication server, disabled by default, but which can be used to host a server for yourself, and others, such as customers or business units who run their own Krill CAs as children under your CA, and to whom you have delegated resource certificates.

If you run Krill as a publication server, you will be faced with running a public service with all related responsibilities, such as uptime and DDoS protection. So, this option is not recommended if you don’t have a clear need to run your own server.

Read more about this option in Running a Publication Server

### 12.1.4 System Requirements

The system requirements for Krill are quite minimal. The cryptographic operations that need to be performed by the Certificate Authority have a negligible performance and memory impact on any modern day machine.

When you publish ROAs yourself using the Krill publication server in combination with Rsyncd and a web server of your choice, you will see traffic from several hundred relying party software tools querying every few minutes. The total amount of traffic is also negligible for any modern day situation.

Tip: For reference, NLnet Labs runs Krill in production and serves ROAs to the world using a 2 CPU / 2GB RAM / 60GB disk virtual machine. Although we only serve four ROAs and our repository size is 16KB, the situation would not be different if serving 100 ROAs.
12.2 Install and Run

Before you can start to use Krill you will need to install, configure and run the Krill application somewhere. Please follow the steps below and you will be ready to Get Started with Krill or start Running a Test Environment.

12.2.1 Installation

Getting started with Krill is quite easy either building from Cargo or running with Docker. In case you intend to serve your RPKI certificate and ROAs to the world yourself or you want to offer this as a service to others, you will also need to have a public Rsyncd and HTTPS web server available.

Krill can also be set up as a highly available, scalable service using Krill Manager. A 1-Click App on the DigitalOcean Marketplace can set up Krill with all required components, along with integration points for monitoring and log analysis.

Quick Start

For recent Debian and Ubuntu releases you can download, install and run a .deb package from the NLnet Labs package repository.

**Note:** If you had previously installed Krill using cargo install krill you should first use cargo uninstall krill before installing a .deb package. Otherwise the cargo installed binaries for krill and krillc may take precedence in your shell $PATH which could be confusing.

1. Add the line below that corresponds to your operating system to /etc/apt/sources.list or /etc/apt/sources.list.d/:

   ```
   deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ stretch main
   deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ buster main
   deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ xenial main
   deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ bionic main
   deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ focal main
   ```

2. Add the repository signing key to the list of trusted keys:

   ```
   wget -qO- https://packages.nlnetlabs.nl/aptkey.asc | sudo apt-key add -
   ```

3. Install and start Krill:

   ```
   sudo apt update
   sudo apt-get install krill
   # review / edit /etc/krill.conf
   sudo systemctl enable --now krill
   ```

Alternatively, you can build from sources. Assuming you have a newly installed Debian or Ubuntu machine, you will need to install the C toolchain, OpenSSL, curl and Rust. You can then install Krill using Cargo.

After the installation has completed, first create a data directory in a location of your choice. Next, generate a basic configuration file specifying a secret token and make sure to refer to the data directory you just created. Finally, start Krill pointing to your configuration file.
apt install build-essential libssl-dev openssl pkg-config curl
curl --proto 'https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
cargo install krill
mkdir ~/.data
krillc config simple --token correct-horse-battery-staple --data ~/.data/ > ~/.data/ ~-krill.conf
krill --config ~/.data/krill.conf

Krill now exposes its user interface and API on https://localhost:3000 using a self-signed TLS certificate. You can go to this address in a web browser, accept the certificate warning and start configuring your RPKI Certificate Authority. A Prometheus endpoint is available at /metrics.

If you have an older version of Rust and Krill, you can update via:

```bash
rustup update
cargo install --force krill
```

**Note:** Using a fully qualified domain name, configuring a real TLS certificate such as Let’s Encrypt, running on a different port and exposing Krill securely to other machines is all possible, but goes beyond the scope of this Quick Start.

### Installing with APT/dpkg

Pre-built Debian/Ubuntu packages are available for recent operating system versions on x86_64 platforms. These can be installed using the standard `apt`, `apt-get` and `dpkg` commands as usual.

Unlike with installing with Cargo there is no need to have Rust or a C toolchain installed. Additionally the packages come with systemd service files for easy start/stop of the Krill daemon and with short Linux man pages.

**Note:** For the oldest platforms, Ubuntu 16.04 LTS and Debian 9, the packaged krill binary is statically linked with OpenSSL 1.1.0 as this is the minimum version required by Krill and is higher than available in the official package repositories for those platforms.

To install Krill from the NLnet Labs package repository:

1. Run `cargo uninstall krill` if you previously installed Krill with Cargo.
2. Add the line below that corresponds to your operating system to `/etc/apt/sources.list` or `/etc/apt/sources.list.d`:

```
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ stretch main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ buster main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ xenial main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ bionic main
```

2. Add the repository signing key to the list of trusted keys:

```
wget -qO- https://packages.nlnetlabs.nl/aptkey.asc | sudo apt-key add -
```

3. Install Krill using `sudo apt-get update` and `sudo apt-get install krill`. 

46 Chapter 12. Krill
4. Review the generated configuration file at /etc/krill.conf. **Pay particular attention** to the service_uri and auth_token settings. Tip: The configuration file was generated for you using the krillc config simple command.

5. Once happy with the settings use `sudo systemctl enable --now krill` to instruct systemd to enable the Krill service at boot and to start it immediately.

The krill daemon runs as user krill and stores its data in /var/lib/krill. You can manage the Krill daemon using the following commands:

- Review the Krill logs with `journalctl -u krill`, or view just the most recent entries with `systemctl status krill`.
- Stop Krill with `sudo systemctl stop krill`.
- Learn more about Krill using `man krill` and `man krillc`.
- Upgrade Krill by running `apt-get update` and `apt-get install krill`.

**Note:** Due to issue #280, when upgrading with `apt-get` it is currently necessary to restart Krill manually after upgrade with `sudo systemctl restart krill`. This issue will be resolved in the next major release.

### Installing with Cargo

There are three things you need for Krill: Rust, a C toolchain and OpenSSL. You can install Krill on any Operating System where you can fulfil these requirements, but we will assume that you will run this on a UNIX-like OS.

**Rust**

The Rust compiler runs on, and compiles to, a great number of platforms, though not all of them are equally supported. The official Rust Platform Support page provides an overview of the various support levels.

While some system distributions include Rust as system packages, Krill relies on a relatively new version of Rust, currently 1.40 or newer. We therefore suggest to use the canonical Rust installation via a tool called rustup.

To install rustup and Rust, simply do:

```bash
curl --proto '="https"' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Alternatively, visit the official Rust website for other installation methods.

You can update your Rust installation later by running:

```
rustup update
```

For some platforms, rustup cannot provide binary releases to install directly. The Rust Platform Support page lists several platforms where official binary releases are not available, but Rust is still guaranteed to build. For these platforms, automated tests are not run so it’s not guaranteed to produce a working build, but they often work to quite a good degree.

One such example that is especially relevant for the routing community is OpenBSD. On this platform, patches are required to get Rust running correctly, but these are well maintained and offer the latest version of Rust quite quickly.

Rust can be installed on OpenBSD by running:

```
pkg_add rust
```
Another example where the standard installation method does not work is CentOS 6, where you will end up with a long list of error messages about missing assembler instructions. This is because the assembler shipped with CentOS 6 is too old.

You can get the necessary version by installing the Developer Toolset 6 from the Software Collections repository. On a virgin system, you can install Rust using these steps:

```
sudo yum install centos-release-scl
sudo yum install devtoolset-6
scl enable devtoolset-6 bash
curl https://sh.rustup.rs -sSf | sh
source $HOME/.cargo/env
```

**C Toolchain**

Some of the libraries Krill depends on require a C toolchain to be present. Your system probably has some easy way to install the minimum set of packages to build from C sources. For example, `apt install build-essential` will install everything you need on Debian/Ubuntu.

If you are unsure, try to run `cc` on a command line and if there’s a complaint about missing input files, you are probably good to go.

**OpenSSL**

Your system will likely have a package manager that will allow you to install OpenSSL in a few easy steps. For Krill, you will need `libssl-dev`, sometimes called `openssl-dev`. On Debian-like Linux distributions, this should be as simple as running:

```
apt install libssl-dev openssl pkg-config
```

**Building**

The easiest way to get Krill is to leave it to cargo by saying:

```
cargo install krill
```

If you want to update an installed version, you run the same command but add the `-f` flag, a.k.a. force, to approve overwriting the installed version.

The command will build Krill and install it in the same directory that cargo itself lives in, likely `$HOME/.cargo/bin`. This means Krill will be in your path, too.

**12.2.2 Generate Configuration File**

After the installation has completed, there are just two things you need to configure before you can start using Krill. First, you will need a data directory, which will store everything Krill needs to run. Secondly, you will need to create a basic configuration file, specifying a secret token and the location of your data directory.

The first step is to choose where your data directory is going to live and to create it. In this example we are simply creating it in our home directory.
mkdir ~/data

Krill can generate a basic configuration file for you. We are going to specify the two required directives, a secret token and the path to the data directory, and then store it in this directory.

```
krillc config simple --token correct-horse-battery-staple --data ~/data/ > ~/data/krill.conf
```

**Note:** If you wish to run a self-hosted RPKI repository with Krill you will need to use a different `krillc config` command. See *Running a Publication Server* for more details.

You can find a full example configuration file with defaults in the GitHub repository.

### 12.2.3 Start and Stop the Daemon

There is currently no standard script to start and stop Krill. You could use the following example script to start Krill. Make sure to update the `DATA_DIR` variable to your real data directory, and make sure you saved your `krill.conf` file there.

```bash
#!/bin/bash
KRILL="krill"
DATA_DIR="/path/to/data"
KRILL_PID="${DATA_DIR}/krill.pid"
CONF="${DATA_DIR}/krill.conf"
SCRIPT_OUT="${DATA_DIR}/krill.log"

nohup $KRILL -c $CONF >$SCRIPT_OUT 2>&1 &
echo $! > $KRILL_PID
```

You can use the following sample script to stop Krill:

```bash
#!/bin/bash
DATA_DIR="/path/to/data"
KRILL_PID="${DATA_DIR}/krill.pid"

kill `cat $KRILL_PID`
```

### 12.2.4 Proxy and HTTPS

Krill uses HTTPS and refuses to do plain HTTP. By default Krill will generate a 2048 bit RSA key and self-signed certificate in `ssl` in the data directory when it is first started. Replacing the self-signed certificate with a TLS certificate issued by a CA works, but has not been tested extensively. By default Krill will only be available under `https://localhost:3000`.

If you need to access the Krill UI or API (also used by the CLI) from another machine you can use use a proxy server such as NGINX or Apache to proxy requests to Krill. This proxy can then also use a proper HTTPS certificate and production grade TLS support.

**Proxy Krill UI**

The Krill UI and assets are hosted directly under the base path `/`. So, in order to proxy to the Krill UI you should proxy ALL requests under `/` to the Krill back-end.
Note that although the UI and API are protected by a token, you should consider further restrictions in your proxy setup - like restrictions on source IP, or you may want to have your own authentication added.

**Proxy Krill as Parent**

If you delegated resources to child CAs then you will need to ensure that these children can reach your Krill. Child requests for resource certificates are directed to the `/rfc6492` under the `service_uri` that you defined in your configuration file.

Note that contrary to the UI you should not add any additional authentication mechanisms to this location. RFC 6492 uses cryptographically signed messages sent over HTTP and is secure. However, verifying messages and signing responses can be computationally heavy, so if you know the source IP addresses of your child CAs, you may wish to restrict access based on this.

**Proxy Krill as Publication Server**

If you are running Krill as a Publication Server, then you should read [here](#) how to do the Publication Server specific set up.

**Warning:** We recommend that you do **not** make Krill available to the public internet unless you really need remote access to the UI or API, or you are serving as parent CA or Publication Server for other CAs.

### 12.2.5 Backup and Restore

To back-up Krill:

- Stop Krill
- Backup your data directory
- Start Krill

We recommend that you stop Krill because there can be a race condition where Krill was just in the middle of saving its state after performing a background operation. We will most likely add a process in future that will allow you to back up Krill in a consistent state while it is running.

To restore Krill just put back your data directory and make sure that you refer to it in the configuration file that you use for your Krill instance.

### 12.2.6 Used Disk Space

Krill stores all of its data under the `DATA_DIR`. For users who will operate a CA under an RIR / NIR parent the following sub-directories are relevant:

<table>
<thead>
<tr>
<th>Dir</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl</td>
<td>Contains the HTTPS key and cert used by Krill</td>
</tr>
<tr>
<td>cas</td>
<td>Contains the history of your CA in raw JSON format</td>
</tr>
<tr>
<td>rfc6492</td>
<td>Contains all messages exchanged with your parent</td>
</tr>
<tr>
<td>rfc8181</td>
<td>Contains all messages exchanged with your repository</td>
</tr>
</tbody>
</table>
The space used by the latter two directories can grow significantly over time. We think it may be a good idea to have an audit trail of all these exchanges. However, if space is a concern you can safely archive or delete the contents of these two directories.

In a future version of Krill we will most likely only store the exchanges where either an error was returned, or your Krill instance asked for a change to be made at the parent side: like requesting a new certificate, or publishing an object. The periodic exchanges where your CA asks the parent for its entitlements will then no longer be logged.

### 12.2.7 Krill Upgrades

It is our goal that future versions of Krill will continue to work with the configuration files and saved data from version 0.4.1 and above. However, please read the changelog to be sure.

The normal process would be to:

- Install the new version of Krill
- Stop the running Krill instance
- Start Krill again, using the new binary, and the same configuration

Note that after a restart you may see a message like this in your log file:

```
2020-01-28 13:41:03 [WARN] [krill::commons::eventsourcing::store] Could not deserialize snapshot json '/root/krill/data/pubd/0/snapshot.json', got error: 'missing field `stats` at line 296 column 1'. Will fall back to events.
```

You can safely ignore this message. Krill is telling you that the definition of a struct has changed and therefore it cannot use the `snapshot.json` file that it normally uses for efficiency. Instead, it needs to build up the current state by explicitly re-applying all the events that happened to your CA and/or publication server.

### 12.3 Get Started with Krill

Before you can start managing your own ROAs you need to do a one time setup where you:

- create your CA
- connect to Publication Server
- connect to Parent CA (typically a Regional or National Internet Registry)

This can be easily achieved using the user interface. Connecting to the Publication Server and Parent CA is done by exchanging a couple of XML files. After this initial setup, and you can simply manage your ROAs.

If you are using the defaults you can access the user interface in a browser on the server running Krill at `https://localhost:3000`. By default, Krill generates a self-signed TLS certificate, so you will have to accept the security warning that your browser will give you.

If you want to access the UI, or use the CLI, from another computer, you can either set up a reverse proxy on your server running Krill, or set up local port forwarding with SSH, for example:

```
ssh -L 3000:localhost:3000 user@krillserver.example.net
```

Here we will guide you through the set up process using the UI, but we will also link to the relevant subcommands of the command line interface (CLI)
12.3.1 Login

The login will ask you to enter the secret token you configured for Krill.

![Enter your secret token to access Krill](image)

Fig. 12.2: Enter your secret token to access Krill

If you are using the CLI you will need to specify the token using the \textit{--token} option. Because the CLI does not have a session, you will need to specify this for each command, or you set the the \texttt{KRILL\_CLI\_TOKEN} environment variable and save yourself the trouble of repeating it.

12.3.2 Create your Certification Authority

Next, you will see the Welcome screen where you can create your Certification Authority (CA). It will be used to configure delegated RPKI with one or multiple parent CAs, usually your Regional or National Internet Registry.

The handle you select is not published in the RPKI but used as identification to parent and child CAs you interact with. Please choose a handle that helps others recognise your organisation. Once set, the handle cannot be changed.

If you are using the CLI you can create your CA using the subcommand \texttt{krillc add}.

12.3.3 Repository Setup

Before Krill can request a certificate from a parent CA, it will need to know where it will publish. You can add a parent before configuring a repository for your CA, but in that case Krill will postpone requesting a certificate until you have done so.
12.3. Get Started with Krill

Fig. 12.3: Enter a handle for your Certification Authority
In order to register your CA as a publisher, you will need to copy the RFC 8183 Publisher Request XML and supply it to your Publication Server. You can retrieve this file with the CLI subcommand `krillc repo request`, or you can simply use the UI:

![Image of the UI showing the publisher request](image)

Fig. 12.4: Copy the publisher request XML or download the file

Your publication server provider will give you a repository response XML. You can use the CLI subcommand `krillc repo update` to tell add this configuration to your CA, or you can simply use the UI:

12.3.4 Parent Setup

After successfully configuring the repository, the next step is to configure your parent CA. You will need to present your CA’s RFC 8183 Child Request XML file to your parent. You can get this file using the CLI subcommand `krillc parents request`, or you can simply use the UI:

Your RIR or NIR will provide you with a parent response XML. You can use the CLI subcommand `krillc parents add` for this, or you can simply paste or upload it using the UI:

12.3.5 ROA Configuration

After successfully setting up the parent exchange, you are now running delegated RPKI. You can start creating ROAs for the resources you see in the pane on the right.
Fig. 12.5: Paste or upload the repository response XML.
Fig. 12.6: Copy the child request XML or download the file
Fig. 12.7: Paste or upload the parent response XML
Fig. 12.8: The ROAs screen displaying all resources and configured ROAs
Click the Add ROA button, then fill in the authorised ASN and one of your prefixes in the form. The maximum prefix length will automatically match the prefix you entered to follow best operational practices, but you can change it as desired.

Fig. 12.9: Adding a new ROA

If you prefer to use the CLI then you can manage ROAs using the subcommand `krillc roas`.

### 12.4 RIR and NIR Interactions

In almost all cases, you will interact with one or more Regional Internet Registries (RIRs) or National Internet Registries (NIRs) when setting up delegated RPKI.

The fundamental principle is the same with each of them: the RIR or NIR needs to establish who you are, which resources you are entitled to and where your RPKI certificate and ROAs will be published.

Your identity, permissions and entitlements are all managed by the registry and exposed via their respective member portals. The rest of the information is exchanged in two XML files. You will need to provide a child request generated by Krill, and in return you will receive a parent response that you need to give back to Krill. See Parent Setup for more details.
12.4.1 Hosted Publication Server

Your RIR or NIR may also provide an RPKI publication server. You are free to publish your certificate and ROAs anywhere you like, so a third party may provide an RPKI publication server as well. To use this service you will need to do an additional exchange. You need to generate and provide a publisher request file and in return you will receive a repository response.

Using an RPKI publication server relieves you of the responsibility to keep a public rsync and web server running at all times to make your certificate and ROAs available to the world.

Of the five RIRs, only APNIC currently offers RPKI publication as a service for their members, upon request. Most other RIRs have it on their roadmap. NIC.br, the Brazilian NIR, provides an RPKI repository server for their members as well. This means that in most cases you will have to publish your certificate and ROAs yourself, as described in the Running a Publication Server section.

12.4.2 Member Portals

If you hold resources in one or more RIR or NIR regions, you will need to have access to the respective member portals and the permission to configure delegated RPKI.

**AFRINIC**
https://my.afrinic.net

**APNIC**
https://myapnic.net

**ARIN**
https://account.arin.net

**LACNIC**
https://milacnic.lacnic.net

**RIPE NCC**
https://my.ripe.net

Most RIRs have a few considerations to keep in mind.

**AFRINIC**

AFRINIC have delegated RPKI available in their test environment, but it's not operational yet. Work to bring it to production is planned for later in 2020.

**APNIC**

If you are already using the hosted RPKI service provided by APNIC and you would like to switch to delegated RPKI, there is currently no option for this with MyAPNIC. Please open a ticket with the APNIC help desk to resolve this.

Please note that APNIC offers RPKI publication as a service upon request. It is highly recommended to make use of this, as it relieves you of the need to run a highly available repository yourself.

**LACNIC**

Although LACNIC offers delegated RPKI, it is not possible to configure this in their member portal yet. While the procedures are still being defined, please open a ticket via hostmaster@lacnic.net to get started.
RIPE NCC Certification Service Terms and Conditions

Introduction

Article 1 – Definitions
In the Terms and Conditions, the following terms shall be understood to have the meanings assigned to them below:

Type of Certificate Authority
- Hosted
- Non-Hosted

By clicking on 'I accept' below you confirm that you have read, understood and agree to the RIPE NCC Certification Service Terms and Conditions.

RIPE NCC

When you are a RIPE NCC member who does not have RPKI configured, you will be presented with a choice if you would like to use hosted or non-hosted RPKI.

If you want to set up delegated RPKI with Krill, you will have to choose non-hosted. If you are already using the hosted service and you would like to switch, then there is currently no option for that in the RIPE NCC portal.

Make a note of the ROAs you created and then send an email to rpki@ripe.net requesting your hosted CA to be deleted, making sure to mention your registration id. After deletion, you will land on the setup screen from where you can choose non-hosted RPKI.

12.5 Using the CLI

12.5.1 Introduction

Every function of Krill can be controlled from the command line interface (CLI). The Krill CLI is a wrapper around the Krill API which is based on JSON over HTTPS.

Note that you can use the CLI from another machine, but then you will need to set up a proxy server in front of Krill and make sure that it has a real TLS certificate.

To use the CLI you need to invoke krillc followed by one or more subcommands, and some arguments. Help is built-in:

```shell
$ :ref:`krillc help<cmd_krillc_help>` [subcommand..]

USAGE:
(continues on next page)```
krillc <subcommand..> [FLAGS] [OPTIONS]

FLAGS:
--api Only show the API call and exit. Or set env: KRILL_CLI_API=1
-h, --help Prints help information
-V, --version Prints version information

OPTIONS:
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
-f, --format <type> Report format: none | json | text (default) | xml. Or set env: KRILL_CLI_FORMAT
-s, --server <URI> The full URI to the krill server. Or set env: KRILL_CLI_SERVER
-t, --token <string> The secret token for the krill server. Or set env: KRILL_CLI_TOKEN

Setting Defaults

As noted in the OPTIONS help text above you can set default values via environment variables for the most common arguments that need to be supplied to krillc subcommands. When setting environment variables note the following requirements:

- KRILL_CLI_SERVER must be in the form https://<host:port>/.
- KRILL_CLI_MY_CA must consist only of alphanumeric characters, dashes and underscores, i.e. a-zA-Z0-9_.

For example:

export KRILL_CLI_TOKEN="correct-horse-battery-staple"
export KRILL_CLI_MY_CA="Acme-Corp-Intl"

If you do use the command line argument equivalents, you will override whatever value you set in the ENV. Krill will give you a friendly error message if you did not set the applicable ENV variable, and don’t include the command line argument equivalent.

Diagnosing Problems

You can show the history of all the things that happened to your CA using the krillc history command.

12.5.2 Reference

The reference below documents the available krillc subcommands. Flags and options that are the same for most subcommands are omitted in this section in order to focus on the most important aspects of each subcommand.

Tip: Click on a subcommand name to jump to the help for that subcommand.

USAGE:
krillc [SUBCOMMAND]

SUBCOMMANDS:
  action Show details for a specific CA action.
  add Add a new CA.
The RPKI Documentation

bulk
- Manually trigger refresh/republish/resync for all cas

children
- Manage children for a CA in Krill.

cfg
- Creates a configuration file for krill and prints it to STDOUT.

health
- Perform an authenticated health check

help
- Prints this message or the help of the given subcommand(s)

history
- Show full history of a CA.

info
- Show server info

issues
- Show issues for CAs.

keyroll
- Perform a manual key-roll in Krill.

list
- List the current CAs.

parents
- Manage parents for this CA.

publishers
- Manage publishers in Krill.
epo
- Manage the repository for your CA.

roas
- Manage ROAs for your CA.

show
- Show details of a CA.

krillc action

Show details for a specific historic CA action.

USAGE:
  krillc action [FLAGS] [OPTIONS] --key <action key string>

OPTIONS:
  -c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_
  --CLI_MY_CA
  --key <action key string> The action key (as shown in the history).

krillc add

API Call: POST /v1/cas

Add a new CA.

When adding a CA you need to choose a handle, essentially just a name. The term “handle” comes from RFC 8183 and is used in the communication protocol between parent and child CAs, as well as CAs and publication servers.

The handle you select is not published in the RPKI but used as identification to parent and child CAs you interact with. You should choose a handle that helps others recognise your organisation. Once set, the handle cannot be changed as it would interfere with the communication between parent and child CAs, as well as the publication repository.

When a CA has been added, it is registered to publish locally in the Krill instance where it exists, but other than that it has no configuration yet. In order to do anything useful with a CA you will first have to add at least one parent to it, followed by some Route Origin Authorisations and/or child CAs.

USAGE:
  krillc add [FLAGS] [OPTIONS]

OPTIONS:
  -c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_
  --CLI_MY_CA

12.5. Using the CLI 63
Note: The CA name may consist of alphanumerics, dashes and underscores, i.e. a-zA-Z0-9_.

krillc bulk

Manually trigger refresh/publish/resync for all CAs.

**USAGE:**

```
krillc bulk [SUBCOMMAND]
```

**SUBCOMMANDS:**

- **help**  Prints this message or the help of the given subcommand(s)
- **publish**  Force that all CAs create new objects if needed (in which case they will also sync)
- **refresh**  Force that all CAs ask their parents for updated certificates
- **sync**  Force that all CAs sync with their repo server

krillc bulk publish

Force that all CAs create new objects if needed (in which case they will also sync).

**USAGE:**

```
krillc bulk publish [FLAGS] [OPTIONS]
```

krillc bulk refresh

Force that all CAs ask their parents for updated certificates.

**USAGE:**

```
krillc bulk refresh [FLAGS] [OPTIONS]
```

krillc bulk sync

**API Call:**  `POST /v1/bulk/cas/sync/repo`

Force that all CAs sync with their repo server.

If your CAs have somehow become out of sync with their repository, then they will automatically re-sync whenever there is an update like a renewal of manifest and crl (every 8 hours), or whenever ROAs are changed. However, you can force that all Krill CAs re-sync with this command.

**USAGE:**

```
krillc bulk sync [FLAGS] [OPTIONS]
```
**krillc children**

Manage children for a CA in Krill.

**USAGE:**

```
krillc children [SUBCOMMAND]
```

**SUBCOMMANDS:**

- **add**    Add a child to a CA.
- **help**   Prints this message or the help of the given subcommand(s).
- **info**   Show info for a child (id and resources).
- **remove** Remove an existing child from a CA.
- **response** Get the RFC8183 response for a child.
- **update** Update an existing child of a CA.

**krillc children add**

**API Call:** See: POST /v1/cas/{parent_ca_handle}/children

Add a child to a CA.

**To add a child, you will need to:**

1. Choose a unique local name (handle) that the parent will use for the child
2. Choose initial resources (asn, ipv4, ipv6)
3. Present the child’s RFC 8183 request

The default response is the RFC 8183 parent response XML file. Or, if you set --format json you will get the plain API response.

If you need the response again, you can use the *krillc children response* command.

**USAGE:**

```
krillc children add [FLAGS] [OPTIONS] --child <name> --request <<XML file>>
```

**FLAGS:**

- **--api**    Only show the API call and exit. Or set env: KRILL_CLI_API=1
- **-h, --help**    Prints help information
- **-V, --version**    Prints version information

**OPTIONS:**

- **-a, --asn <AS resources>**    The delegated AS resources: e.g. AS1, AS3-4
- **-c, --ca <name>**    The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
- **--child <name>**    The name of the child CA you wish to control.
- **-f, --format <type>**    Report format: none|json|text (default). Or set env: KRILL_CLI_FORMAT
- **--request <<XML file>>**    The location of the RFC8183 Child Request XML file.

**krillc children info**

Show info for a child (id and resources).
# The RPKI Documentation

**krillc children info**

**Usage:**
```
krillc children info [FLAGS] [OPTIONS] --child <name>
```

**Options:**
- `-c, --ca <name>` The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
- `--child <name>` The name of the child CA you wish to control.

**krillc children remove**

Remove an existing child from a CA.

**Usage:**
```
krillc children remove [FLAGS] [OPTIONS] --child <name>
```

**Options:**
- `-c, --ca <name>` The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
- `--child <name>` The name of the child CA you wish to control.

**krillc children response**

**API Call:** GET /v1/cas/{parent_ca_handle}/children/{child_ca_handle}/contact

Get the RFC8183 response for a child.

**Usage:**
```
krillc children response [FLAGS] [OPTIONS] --child <name>
```

**Options:**
- `-c, --ca <name>` The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
- `--child <name>` The name of the child CA you wish to control.

**krillc children update**

Update an existing child of a CA.

**Usage:**
```
krillc children update [FLAGS] [OPTIONS] --child <name>
```

**Options:**
- `-a, --asn <AS resources>` The delegated AS resources: e.g. AS1,\_\_\_\_\_\_
- `--AS3-4` The name of the child CA you wish to control.
- `-c, --ca <name>` The name of the CA you wish to control.
- `--child <name>` The name of the child CA you wish to control.
- `--idcert <DER encoded certificate>` The child's updated ID certificate
- `-4, --ipv4 <IPv4 resources>` The delegated IPv4 resources: e.g. 192.168.0.0/16
- `-6, --ipv6 <IPv6 resources>` The delegated IPv6 resources: e.g. 2001:db8::/32"
**krillc config**

Creates a configuration file for kroll and prints it to STDOUT.

**USAGE:**

```
krillc config [SUBCOMMAND]
```

**SUBCOMMANDS:**

- `help`  Prints this message or the help of the given subcommand(s)
- `repo`  Use a self-hosted repository (not recommended)
- `simple`  Use a 3rd party repository for publishing

**krillc config simple**

Creates a configuration file that configures Krill to be used with external repositories.

**USAGE:**

```
krillc config simple [FLAGS] [OPTIONS]
```

**OPTIONS:**

- `-d, --data <path>`  Override the default path (./data/) for the data directory (must end with slash).
- `-l, --logfile <path>`  Override the default path (./krill.log) for the log file.

**krillc config repo**

Creates a configuration file that enables a self-hosted repository within Krill that CAs can be configured to publish to instead of publishing to an external repository.

**Warning:** Running your own repository service is not recommended. For more information about using the self-hosted repository see [*Running a Publication Server*](#).

**USAGE:**

```
krillc config repo [FLAGS] [OPTIONS] --rrdp <uri> --rsync <uri>
```

**OPTIONS:**

- `-d, --data <path>`  Override the default path (./data/) for the data directory (must end with slash).
- `-l, --logfile <path>`  Override the default path (./krill.log) for the log file.
- `--rrdp <uri>`  Specify the base https URI for your RRDP (excluding notify.xml), must end with '/'.
- `--rsync <uri>`  Specify the base rsync URI for your repository. must end with '/'.

---

**12.5. Using the CLI**
**krillc health**

Perform an authenticated health check. Verifies that the specified Krill server can be connected to, is able to verify the specified token and is, at least thus far, healthy.

Can be used in automation scripts by checking the exit code:

```plaintext
<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the Krill server appears to be healthy.</td>
</tr>
<tr>
<td>non-zero</td>
<td>incorrect server URI, token, connection failure or server error.</td>
</tr>
</tbody>
</table>
```

**krillc help**

Prints the version of *krillc* and the complete list of possible subcommands with a short explanatory text for each one.

**krillc history**

Show full history of a CA. Using this command you can show the history of all the things that happened to your CA.

**USAGE:**

```
krillc history [FLAGS] [OPTIONS]
```

**FLAGS:**

- `--full` Show history including publication.

**OPTIONS:**

- `--after <<RFC 3339 DateTime>>` Show commands issued after date/time in RFC 3339 format, e.g. 2020-04-09T19:37:02Z
- `--before <<RFC 3339 DateTime>>` Show commands issued after date/time in RFC 3339 format, e.g. 2020-04-09T19:37:02Z
- `-c, --ca <name>` The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
- `--offset <<number>>` Number of results to skip
- `--rows <<number>>` Number of rows (max 250)

**Example:**

```
$ krillc history
2020-06-07T20:33:21Z::Update repo to server at: https://localhost:3000/rfc8181/ca::success
2020-06-07T20:34:18Z::Add parent 'ripencc' as 'RFC 6492 Parent' ::cmd-ca-parent-add::OK
2020-06-07T20:34:19Z::Update entitlements under parent 'ripencc': 0 => asn: 0 blocks, v4: 1 blocks, v6: 1 blocks ::cmd-ca-parent-entitlements::OK
2020-06-07T20:34:20Z::Update received cert in RC '0', with resources 'asn: 0 blocks, v4: 1 blocks, v6: 1 blocks' ::cmd-ca-rcn-receive::OK
2020-06-07T20:36:28Z::Update ROAs add: 2 remove: '0' ::cmd-ca-roas-updated::OK
```
krillc info

Show server info. Prints the version of the Krill server and the date and time that it was last started, e.g.:

```bash
USAGE:
   krillc info [FLAGS] [OPTIONS]
```

Example:

```bash
$ krillc info
Version: 0.7.3
Started: 2020-07-06T14:24:02+00:00
```

krillc issues

Show issues for CAs.

```bash
USAGE:
   krillc issues [FLAGS] [OPTIONS]
```

```bash
OPTIONS:
   -c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
```

krillc keyroll

Perform a manual key-roll in Krill.

```bash
USAGE:
   krillc keyroll [SUBCOMMAND]
```

SUBCOMMANDS:

- `activate` Finish roll for all keys held by this CA.
- `help` Prints this message or the help of the given subcommand(s)
- `init` Initialise roll for all keys held by this CA.

krillc keyroll activate

Finish roll for all keys held by this CA.

```bash
USAGE:
   krillc keyroll activate [FLAGS] [OPTIONS]
```

```bash
OPTIONS:
   -c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
```
**krillc keyroll init**

Initialise roll for all keys held by this CA.

```
USAGE:
   krillc keyroll init [FLAGS] [OPTIONS]

OPTIONS:
   -c, --ca <name>     The name of the CA you wish to control. Or set env: KRILL CLI_MY_CA
```

**krillc list**

API Call: GET /v1/cas

List the current CAs.

```
USAGE:
   krillc list [FLAGS] [OPTIONS]
```

**krillc parents**

Manage parents for this CA.

```
USAGE:
   krillc parents [SUBCOMMAND]

SUBCOMMANDS:
   add       Add a parent to this CA.
   contact   Show contact information for a parent of this CA.
   help      Prints this message or the help of the given subcommand(s)
   remove    Remove an existing parent from this CA.
   request   Show RFC8183 Publisher Request XML
   update    Update an existing parent of this CA.
```

**krillc parents add**

API Call: POST /v1/cas/ca/parents

Add a parent to this CA.

```
USAGE:
   krillc parents add [FLAGS] [OPTIONS] --parent <name> --response <<XML file>>

OPTIONS:
   -c, --ca <name>     The name of the CA you wish to control. Or set env: KRILL CLI_MY_CA
   -p, --parent <name> The local name by which your ca refers to this parent.
   -r, --response <<XML file>> The RFC8183 Parent Response XML
```
Note that you can use any local name for `--parent`. This is the name that Krill will show to you. Similarly, Krill will use your local CA name which you set in the `KRILL_CLI_MY_CA` ENV variable. However, the parent response includes the names (or handles as they are called in the RFC) by which it refers to itself, and your CA. Krill will make sure that it uses these names in the communication with the parent. There is no need for these names to be the same.

Note that whichever handle you choose, your CA will use the handles that the parent response included for itself and for your CA in its communication with this parent. I.e. you may want to inspect the response and use the same handle for the parent (parent_handle attribute), and do not be surprised or alarmed if the parent refers to your ca (child_handle attribute) by some seemingly random name. Some parents do this to ensure unicity.

In case you have multiple parents you may want to refer to them by names that make sense in your context.

**krillc parents contact**

Show contact information for a parent of this CA.

**krillc parents remove**

Remove an existing parent from this CA.

**krillc parents request**

API Call: `GET /v1/cas/{ca_handle}/child_request.json`

Show [RFC 8183](https://tools.ietf.org/html/rfc8183) Publisher Request XML for the named CA. This XML is needed when registering the CA as a child of another CA. For more information see [Parent Setup](https://tools.ietf.org/html/rfc8183).

---

### curl command line syntax

**Usage:**

```
krillc parents contact [FLAGS] [OPTIONS] --parent <name>
```

**Options:**

- `-c, --ca <name>` The name of the CA you wish to control. Or set env: `KRILL_CLI_MY_CA`
- `-p, --parent <name>` The local name by which your ca refers to this parent.

---

**Usage:**

```
krillc parents remove [FLAGS] [OPTIONS] --parent <name>
```

**Options:**

- `-c, --ca <name>` The name of the CA you wish to control. Or set env: `KRILL_CLI_MY_CA`
- `-p, --parent <name>` The local name by which your ca refers to this parent.

---

**Usage:**

```
krillc parents request [FLAGS] [OPTIONS]
```

**Options:**

- `-c, --ca <name>` The name of the CA you wish to control. Or set env: `KRILL_CLI_MY_CA`
**krillc parents update**

Update the known information about existing parent of this CA. Note there is no good description of this in the RFCs for the moment. You should not need this option in practice. However, this will allow to replace the parent response for one of your parents.

**Usage:**

```
krillc parents update [FLAGS] [OPTIONS] --parent <name> --response <<XML file>>
```

**Options:**

- `-c, --ca <name>` The name of the CA you wish to control. Or set env: `KRILL_CLI_MY_CA`
- `-p, --parent <name>` The local name by which your ca refers to this parent.
- `--response <<XML file>>` The RFC8183 Parent Response XML

**krillc publishers**

Manage publishers in Krill.

**Usage:**

```
krillc publishers [SUBCOMMAND]
```

**Subcommands:**

- `add` Add a publisher.
- `help` Prints this message or the help of the given subcommand(s)
- `list` List all publishers.
- `remove` Remove a publisher.
- `response` Show RFC8183 Repository Response for a publisher.
- `show` Show details for a publisher.
- `stale` List all publishers which have not published in a while.
- `stats` Show publication server stats.

**krillc publishers add**

Add a publisher. In order to add a publisher you have to get its RFC 8183 Publisher Request XML, and hand it over to the server.

**Usage:**

```
krillc publishers add [FLAGS] [OPTIONS] --request <file>
```

**Options:**

- `-p, --publisher <handle>` Override the publisher handle in the XML.
- `-r, --request <file>` RFC8183 Publisher Request XML file containing a certificate (tag is ignored)

**krillc publishers list**

List all publishers under the Publication Server.
krillc publishers remove

Remove a publisher. If you do, then all of its content will be removed as well and the publisher will no longer be allowed to publish.

**Warning:** You can do this without the publisher’s knowledge, nor consent. You should check with the publisher whether they no longer need your Publication Server (perhaps they migrated to another, or disabled their CA).

krillc publishers response

Show RFC8183 Repository Response for a publisher.

Example:

```bash
$ krillc publishers response --publisher ca
  <repository_bpki_ta> repository server id certificate base64 </repository_bpki_ta>
</repository_response>
```

krillc publishers show

Show details for a publisher, including the files that they published.

The default text output just shows the handle of the publisher, the hash of its identity certificate key, and the rsync URI jail under which the publisher is allowed to publish objects.

The JSON response includes a lot more information, including the files which were published and the full ID certificate used by the publisher.
OPTIONS:
- p, --publisher <handle> The handle (name) of the publisher.

**krillc publishers stale**

List all publishers which have not published in a while.

**USAGE:**

```
krillc publishers stale [FLAGS] [OPTIONS] --seconds <seconds>
```

OPTIONS:
- --seconds <seconds> The number of seconds since last publication.

**krillc publishers stats**

Show publication server stats.

**USAGE:**

```
krillc publishers stats [FLAGS] [OPTIONS]
```

**krillc repo**

Manage the repository for your CA.

**USAGE:**

```
krillc repo [SUBCOMMAND]
```

**SUBCOMMANDS:**
- help Prints this message or the help of the given subcommand(s)
- request Show RFC8183 Publisher Request.
- show Show current repo config.
- state Show current repo state.
- update Change which repository this CA uses.

**krillc repo request**

Show the RFC 8183 Publisher Request XML for a CA. You will need to hand this over to your repository so that they can add your CA.

**USAGE:**

```
krillc repo request [FLAGS] [OPTIONS]
```

OPTIONS:
- -c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_
- CLI_MY_CA

Example:
### krillc repo request

Show the repository configuration for your CA.

**USAGE:**

```
krillc repo show [FLAGS] [OPTIONS]
```

**OPTIONS:**

```
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
```

**Example:**

```
$ krillc repo show
Repository Details:
  type: remote
  service uri: https://krill-ui-dev.do.nlnetlabs.nl/rfc8181/Acme-Corp-Intl
  base_uri: rsync://krill-ui-dev.do.nlnetlabs.nl/repo/Acme-Corp-Intl/
  rpki_notify: https://krill-ui-dev.do.nlnetlabs.nl/rrdp/notification.xml
```

### krillc repo state

Show which repository server your CA is using, as well as what it has published at the location. Krill will issue an actual list query to the repository and give back the response, or an error in case of issues.

**USAGE:**

```
krillc repo state [FLAGS] [OPTIONS]
```

**OPTIONS:**

```
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
```

**Example:**

```
$ krillc repo state
Available and publishing objects:
  407cb2f9c72ae0a65badb49c073fc9df79c170336b3c14734e8e2ec9e1c106ca rsync://krill-ui-dev.do.nlnetlabs.nl/repo/Acme-Corp-Intl/0/5796A18D9A941AB72D78C820C5F0837B1CB30694.mft
  ad2afbf2fdddfc0212df4c9b4bc1834b1d9c1a2ece1e982ec08e9e9ef92aa8ca912 rsync://krill-ui-dev.do.nlnetlabs.nl/repo/Acme-Corp-Intl/0/79f8fd1c1c42e1e029a65011e28415515602b12f6e850799adeb1fc515339b roa
  6f7029b9e9a8c96eb1c5c388bd6f1cbcaac6cd24885e666f4cb5a218647beff2 rsync://krill-ui-dev.do.nlnetlabs.nl/repo/Acme-Corp-Intl/0/31302e0302e302e302f3dc22d3232203d3e203634343936.roa
```

(continues on next page)
krillc repo update

Change which repository this CA uses.

You can change which repository server is used by your CA. If you have multiple CAs you will have to repeat this for each of them.

Changing repositories is actually more complicated than one might think, but fortunately it’s all automated. When you ask Krill to change, the following steps will be executed:

- check that the new repository can be reached, and this ca is authorised
- regenerate all objects using the URI jail given by the new repository
- publish all objects in the new repository
- request new certificates from (all) parent CA(s) including the new URI
- once received, do a best effort to clean up the old repository

In short, Krill performs a sanity check that the new repository can be used, and then tries to migrate there in a way that will not lead to invalidating any currently signed objects.

To start a migration you can use the following.

```
USAGE:
   krillc repo update [FLAGS] [OPTIONS]

OPTIONS:
   -c, --ca <name>   The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
   -r, --response <file>  The location of the RFC8183 Publisher Response XML file. Defaults to reading from STDIN
```

krillc roas

Manage ROAs for your CA.

Krill lets users create Route Origin Authorisations (ROAs), the signed objects that state which Autonomous System (AS) is authorised to originate one of your prefixes, along with the maximum prefix length it may have.

```
USAGE:
   krillc roas [SUBCOMMAND]

SUBCOMMANDS:
   help  Prints this message or the help of the given subcommand(s)
   list  Show current authorizations.
   update  Update authorizations.
```

krillc roas list

Show current authorizations.
krillc roas list

Usage:
krillc roas list [FLAGS] [OPTIONS]

Options:
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA

Example:
You can list ROAs in the following way:

```
$ krillc roas list
192.0.2.0/24 => 64496
2001:db8::/32-48 => 64496
```

**krillc roas update**

API Call: POST /v1/cas/ca/routes

Update authorizations.

You can update ROAs through the command line by submitting a plain text file with the following format:

```
# Some comment
# Indented comment
A: 10.0.0.0/24 => 64496
A: 10.1.0.0/16-20 => 64496 # Add prefix with max length
R: 10.0.3.0/24 => 64496 # Remove existing authorization
```

Usage:
krillc roas update [FLAGS] [OPTIONS] --delta <<file>>

Options:
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA

--delta <<file>> Provide a delta file using the following format:

```
# Some comment
# Indented comment
A: 192.168.0.0/16 => 64496 # inline comment
A: 192.168.1.0/24 => 64496
R: 192.168.3.0/24 => 64496
```

Example:

```
$ krillc roas update --delta ./roas.txt
```

**krillc show**

API Call: GET /v1/cas/{ca_handle}

Show details of a CA.
The RPKI Documentation

**Usage:**

```
krillc show [FLAGS] [OPTIONS]
```

**Options:**

```
-c, --ca <name> The name of the CA you wish to control. Or set env: KRILL_CLI_MY_CA
```

**Example:**

```
$ krillc show --ca ca
Name: ca

Base uri: rsync://localhostrepo/ca/
RRDP uri: https://localhost:3000/rrdp/notification.xml

ID cert PEM:
----BEGIN CERTIFICATE-----
MIIDPDCCAiSgAwIBAgIBATANBgkqhkiG9w0BAQsFADAzMTEwLwYDVQQDEyg2NTA1RDA4RUI5MTk5NkJFNkFERDNGOEYyQzUzQTUxNTg4RTYT4NDJCMCAXDTE5MTIwMzEy
.. zKtG5esZ+zg48If6jBgDyyONXEICowcjrx1Y5fnjHhL0jstMlTuITgYuRoGIK2KzQ+qLiXg2G+8s8u/lPW7PVYg==
----END CERTIFICATE-----

Hash: 9f1376b2e1c8052c1b5d94467f8708935224c518effbe746778fb2215e

Total resources:
   ASNs:
   IPv4: 10.0.0.0/8
   IPv6: 2001:db8::/32

Parents:
Handle: ripencc Kind: RFC 6492 Parent

Resource Class: 0
Parent: ripencc
State: active Resources:
   ASNs:
   IPv4: 10.0.0.0/8
   IPv6: 2001:db8::/32
Current objects:
553A7C2E751CA0B04B49CB72E30EB5684F861987.crl
553A7C2E751CA0B04B49CB72E30EB5684F861987.mft

Children:
<none>
```

## 12.6 Using the API

The Krill API is a primarily JSON based REST-like HTTPS API with bearer token based authentication.

### 12.6.1 Getting Help

- Consult the Interactive API documentation (courtesy of ReDoc)
- Follow the API links in the *Krill CLI documentation*, e.g. API Call: GET /v1/cas
- Check out the API hints built-in to the *Krill CLI*, e.g.

  ```
  $ krillc list --api
  GET:
  https://<your.domain>/api/v1/cas
  Headers:
  Authorization: Bearer *****
  ```

### 12.6.2 Generating Client Code

The *OpenAPI Generator* can generate *Krill API client code* in many languages from the *Krill v0.7.3 OpenAPI specification*.3

### 12.6.3 Sample Application

Below is an example of how to write a small *Krill client application* in *Python 3* using a *Krill API client library* produced by the *OpenAPI Generator*. To try out this sample you’ll need *Docker* and *Python 3*.

1. Save the following as `/tmp/krill_test.py`, replacing `<YOUR XXX>` values with the correct access token and domain name for your *Krill server*. This example assumes that your *Krill instance API endpoint* is available on port 443 using a valid TLS certificate.

   ```python
   # Import the OpenAPI generated *Krill client library*
   import krill_api
   from krill_api import *

   # Create a configuration for the client library telling it how to connect to
   # the *Krill server*
   krill_api_config = krill_api.Configuration()
   krill_api_config.access_token = '<YOUR KRILL API TOKEN>'
   krill_api_config.host = 'https://{}/api/v1'.format('<YOUR KRILL FQDN>')
   krill_api_config.verify_ssl = True
   krill_api_config.assert_hostname = False
   krill_api_config.cert_file = None

   # Create a *Krill API client*
   krill_api_client = krill_api.ApiClient(krill_api_config)

   # Get the client helper for the Certificate Authority set of *Krill API endpoints*
   krill_ca_api = CertificateAuthoritiesApi(krill_api_client)

   # Query *Krill* for the list of configured CAs
   print(krill_ca_api.list_cas())
   ```

2. Run the following commands in a shell to generate a *Krill client library*:

   ```
   # prepare a working directory
   GENDIR=/tmp/gen
   VENNDIR=/tmp/venv
   mkdir -p $GENDIR

   # fetch the *Krill OpenAPI specification document*
   wget -O $GENDIR/openapi.yaml https://raw.githubusercontent.com/NLnetLabs/krill/v0.7.3/doc/openapi.yaml
   ```

(continues on next page)
The RPKI Documentation

# use the OpenAPI Generator to generate a Krill client library from the krill OpenAPI specification

docker run --rm -v $GENDIR:/local \
  openapitools/openapi-generator-cli generate \
  -i /local/openapi.yaml \
  -g python \
  -o /local/out \
  --additional-properties=packageName=krill_api

# install the generated library where your Python 3 can find it

python3 -m venv $VENVDIR
source $VENVDIR/bin/activate
pip3 install wheel
pip3 install $GENDIR/out/

3. Run the sample application:

$ python3 /tmp/krill_test.py
{'cas': [{'handle': 'ca'}]}

Tip: To learn more about using the generated client library, consult the documentation in $GENDIR/out/README.md.

Warning: Future improvements to the Krill OpenAPI specification may necessitate that you re-generate your client library and possibly also alter your client program to match any changed class and function names.

12.7 Monitoring

The HTTPS server in Krill provides endpoints for monitoring the application. A data format specifically for Prometheus is available and dedicated port 9657 has been reserved.

On the /metrics path, Krill will expose several data points:

- A timestamp when the daemon was started
- The number of CAs Krill has configured
- The number of children for each CA
- The number of ROAs for each CA
- Timestamps when publishers were last updated
- The number of objects in the repository for each publisher
- The size of the repository, in bytes
- The RRDP serial number

This is an example of the output of the /metrics endpoint:
# HELP krill_server_start timestamp of last krill server start
# TYPE krill_server_start gauge
krill_server_start 1582189609

# HELP krill_repo_publisher number of publishers in repository
# TYPE krill_repo_publisher gauge
krill_repo_publisher 1

# HELP krill_repo_rrdp_last_update timestamp of last update by any publisher
# TYPE krill_repo_rrdp_last_update gauge
krill_repo_rrdp_last_update 1582700400

# HELP krill_repo_rrdp_serial RRDP serial
# TYPE krill_repo_rrdp_serial counter
krill_repo_rrdp_serial 128

# HELP krill_repo_objects number of objects in repository for publisher
# TYPE krill_repo_objects gauge
krill_repo_objects{publisher="acme-corp-intl"} 6

# HELP krill_repo_size size of objects in bytes in repository for publisher
# TYPE krill_repo_size gauge
krill_repo_size{publisher="acme-corp-intl"} 16468

# HELP krill_repo_last_update timestamp of last update for publisher
# TYPE krill_repo_last_update gauge
krill_repo_last_update{publisher="acme-corp-intl"} 1582700400

# HELP krill_cas number of cas in krill
# TYPE krill_cas gauge
krill_cas 1

# HELP krill_cas_roas number of roas for CA
# TYPE krill_cas_roas gauge
krill_cas_roas{ca="acme-corp-intl"} 4

# HELP krill_cas_children number of children for CA
# TYPE krill_cas_children gauge
krill_cas_children{ca="acme-corp-intl"} 0

The monitoring service has several additional endpoints on the following paths:

/stats/info Returns the Krill version and timestamp when the daemon was started in a concise format

/stats/cas Returns the number of ROAs and children each CA has

/stats/repo Returns details on the internal repository, if configured

12.8 Running a Publication Server

**Important:** It is highly recommended to use an RPKI publication server provided by your parent CA, if available. This relieves you of the responsibility to keep a public rsync and web server available at all times.

If you need to run your own Publication Server using Krill, then we recommend that you use a separate Krill instance.
The RPKI Documentation

acting as a repository only. This setup allows for much easier reconfiguration (more on this below), and it allows that
other CAs - for example a delegated CA for one of your business units also publish at this same Publication Server.

![Diagram](image_url)

Fig. 12.11: Running a publication server for yourself and your children

12.8.1 Configuring a Krill Repository

**Note:** The Krill UI is not currently aimed at using Krill as a repository server. For example when visiting the UI of a
Krill instance intended for use only as a repository and not as a CA, it will still prompt you on first use to create a CA.
There is also no support via the UI for managing the repository, for example it is not possible via the UI to complete a
child request to register with the repository.

Krill can be set up to run as a Publication Server through its configuration file. If enabled, the Publication Server is
created on start-up. After this any updates to the configuration will *NOT* be reflected in the Publication Server.

For this reason you should double check the values used for the public URIs to your repository server carefully before
the set-up. Using a dedicated Krill instance for the Publication Server will allow you to simply destroy and replace the
instance should it have been misconfigured.

The easiest way to make a configuration file is by using `krillc config` to generate the required configuration for you.
For example:

```bash
krillc config repo \
  --server "https://rfc8183.example.net/" \
  --token correct-horse-battery-staple \
```
Make sure that the `--server` option reflects a base URI that your Krill CA publication clients can reach, and make sure that this URI is exposed using a proxy server that has a proper HTTPS certificate configured.

Make sure that the `--rrdp` and `--rsync` options match the configuration of your “Repository Servers” which make your repository available over HTTPS and rsync to Relying Parties.

**Note:** It would have been better to make the Publication Server configuration something that should be done run-time, as this would match more intuitively with the fact that the `server`, `rrdp` and `rsync` URIs cannot be changed through the configuration file.

In a future release of Krill we may do exactly that. But, even if we do it would be ill advised to allow changing these URIs at run time, as there would be no way for the Krill Publication Server to inform its publishers about any change.

So, in short, this needs to be set up correctly once. If it turns out to be wrong, then a new Publication Server should be set up and any existing publishers should be migrated as described below.

### 12.8.2 Proxy for Remote Publishers

Krill runs the RFC8181 Publication Server. Remote publishers, CAs which use your Publication Server, will need to connect to this under the `/rfc8181` path under the `service_uri` that you specified in your server.

Make sure that you set up a proxy server such as NGINX, Apache, etc. which uses a valid HTTPS certificate, and which proxies `/rfc8181` to Krill.

Note that you should not add any additional authentication mechanisms to this location. RFC 8181 uses cryptographically signed messages sent over HTTP and is secure. However, verifying messages and signing responses can be computationally heavy, so if you know the source IP addresses of your publisher CAs, you may wish to restrict access based on this.

### 12.8.3 Configuring Repository Servers

To actually serve the published content to Rsync and RRDP clients you will need to run your own repository servers using tools such as Rsyncd and NGINX.

Krill will write the repository files under the data directory specified in its configuration file:

```bash
$DATA_DIR/repo/rsync/current/  Contains the files for Rsync
$DATA_DIR/repo/rrdp/           Contains the files for HTTPS (RRDP)
```

You can share the contents of these directories with your repository servers in various ways. It is possible to have a redundant shared file system where the Krill Publication Server can write, and your repository servers can read. Alternatively, you can synchronise the contents of these directories in another way, such as rsyncing them over every couple of minutes.

If you are using a shared file system, please note that the `rsync /current` directory cannot be the mount point. Krill tries to write the entire repository to a new folder under `$DATA_DIR/repo/rsync` and then renames it. This is done to minimise issues with files being updated while relying party software is fetching data.
The next step is to configure your rsync daemons to expose a ‘module’ for your files. Make sure that the Rsync URI including the ‘module’ matches the \texttt{rsync\_base} in your Krill configuration file. Basic configuration can then be as simple as:

\begin{verbatim}
$ cat /etc/rsyncd.conf
uid = nobody
gid = nogroup
max connections = 50
socket options = SO_KEEPALIVE
[repo]
path = /var/lib/krill/data/repo/rsync/current/
comment = RPKI repository
read only = yes
\end{verbatim}

**RRDP**

For RRDP you will need to set up a web server of your choice and ensure that it has a valid TLS certificate. Next, you can make the files found under, or copied from $\texttt{DATA\_DIR/repo/rrdp}$ available here. Make sure that the public URI to the RRDP base directory matches the value of \texttt{rrdp\_service\_uri} in your \texttt{krill.conf} file, or the \texttt{--rrdp} option if you generated the configuration.

If desired, you can also use a CDN in front of this server to further reduce your load and uptime requirements. If you do, make sure that the public URI matches the directive in \texttt{krill.conf}, because this will be used in your RPKI certificate.

**RFC 8181 (publication protocol)**

Make sure that your Krill Publication Server can be reached by your Krill CA clients. The best way to do this, is by setting up a web server, similar to the RRDP set up above, which proxies access to URIs starting with \texttt{/rfc8181} under the hostname you specified with the \texttt{--server} option through to your Krill Publication Server.

### 12.8.4 Publishing in the Repository

As there is no UI support for this, you will need to use the command line interface using the \texttt{krillc publisher} subcommand to manage publishers.

This subcommand will allow you to add your Krill CA client’s RFC8181 Publisher Request XML, and obtain a Repository Response XML for it. From the client CA’s perspective this part of the process is exactly as described \textit{here}.

To add the Krill CA client XML to your server use the following:

\begin{verbatim}
$ krillc publishers add --request <path-to-xml> [--publisher publisher]
\end{verbatim}

If \texttt{--publisher} is not specified then the publisher identifier handle will be taken from the XML. Handles need to be unique. So, you may want or need to override this - especially if you provide your Publication Server as a service to others.

If successful this will show the response XML. But, you can also get this response XML for a configured publisher using the following:
12.8.5 Migrating the Repository

If you find that there is an issue with your repository or, for example, you want to change its domain name, you can set up a new Krill instance for the new repository. When you are satisfied that the new one is correct, you can migrate your CA to it by adding them as a publisher under the new repository server, and then updating your CA to use the new repository.

Updating the repository of your Krill CAs is currently not possible using the UI, but you can archive this through the command line interface connecting to your Krill instance that hosts your CA.

First you will need to get your CA’s Publication Request XML using the following:

$ krillc repo request

You then need to give this XML to your Publication Server, be it provided by a third party or managed by yourself as described above. After receiving the Repository Response XML you can then update your CA’s repository using:

$ krillc repo update -response <path-to-xml>

Krill will then make sure that objects are moved properly, and that a new certificate is requested from your parent(s) to match the new location. This scenario would also apply when your RIR starts providing a repository service. You can update your CA to start publishing there instead.

12.9 Running a Test Environment

If you want to get operational experience with Krill before configuring a production parent, you can run with an embedded Trust Anchor (TA) which you can give any address space you want. You can generate your own Trust Anchor for it, which can be added to your Relying Party software in order to validate the objects you have published locally.

**Note:** The Krill TA is intended for test purposes only and as such there are some limitations when using it:

- The Krill Trust Anchor Locator (TAL) points to the TA certificate using an https:// URI. It does not include an rsync:// URI.
- Only Relying Party software which supports such an HTTPS TAL can be used with the Krill TA.
- When enabling the embedded TA use_ta = true the Krill daemon assumes that you also set repo_enabled = true to enable an embedded repository in this Krill instance. **Tip:** The *krillc config repo* command does this for you.
- The TA claims ownership of all possible ASNs, IPv4 addresses and IPv6 addresses. It is not currently possible to restrict this to a subset of these resources.

12.9.1 Setting up the Configuration

For testing we will assume that you will run your own Krill repository inside a single Krill instance, using ‘localhost’ in the repository URIs. You have to set the following environment variable to re-assure Krill that you are running a test environment, or it will refuse the use of ‘localhost’:
The RPKI Documentation

$ export KRILL_TEST="true"

For convenience you may wish to set the following variables, so that you don’t have to repeat command line arguments for these:

$ export KRILL_CLI_SERVER="https://localhost:3000/"
$ export KRILL_CLI_TOKEN="correct-horse-battery-staple"
$ export KRILL_CLI_MY_CA="ca"

Note: Replace “correct-horse-battery-staple” with a token of your own choosing! If you don’t the UI will kindly remind you that “You should not get your passwords from https://xkcd.com/936/”.

You can now generate a krill configuration file using the following command:

$ krillc config repo
  --token $KRILL_CLI_TOKEN
  --rrdp https://localhost:3000/rrdp/
  --rsync rsync://localhost/repo/ > /tmp/krill.conf
  --data /tmp/krill_data

12.9.2 Enable the Embedded Trust Anchor (TA)

To run Krill in test mode you can set “use_ta” to “true” in your krill.conf, or use an environment variable:

$ export KRILL_USE_TA="true"
$ krill -c /tmp/krill.conf &

Verify that the TA now exists:

$ krillc list
ta

Use the following to show more details of the embedded TA:

$ krillc show --ca ta
Name: ta

Base uri: rsync://localhost/repo/ta/
RRDP uri: https://localhost:3000/rrdp/notification.xml

ID cert PEM:
-----BEGIN CERTIFICATE-----
MIIDPDCCAiSgAwIBAgIBATANBgkqhkiG9w0BAQsFADAzMTEwLwYDVQQDEyg2MUE1
QzYuMTEwLzEwNjYxMTEwLzEwNjYxMTEwLzEwNjYx
Yge7Bo1ITINX8XBzdTr91TgUKreDEG1Nh6sY0ONJW9rQxZIsD1dTeBoPSQKCDXk
D13RgMxQSjycIfAeIBo9yg==
-----END CERTIFICATE-----

Hash: 85041ff6bf2d6edf4e02c716e8be9f4dd49e2cc8aa578213556072bab75575ee

Total resources:
  ASNs: AS0-AS4294967295
  IPv4: 0.0.0.0/0
IPv6: ::/0

Parents:
Handle: ta Kind: This CA is a TA

Resource Class: 0
Parent: ta
State: active
Resources:
ASNs: AS0-AS4294967295
IPv4: 0.0.0.0/0
IPv6: ::/0
Current objects:
1529A3C0E47EA38C1101DECDD6330E932E3AB98F.crl
1529A3C0E47EA38C1101DECDD6330E932E3AB98F.mft

Children:
<none>

12.9.3 Example Usage with a TA

In this example we show you how to create a CA, register it with the embedded repository and as a child of the TA, and how to publish ROAs.

Create a CA

$ krillc add

Verify that now both TA and CA exist:

$ krillc list
ta
ta
ca

Register the CA with a repository

You can do the CA part of this using the UI.

But, if your CAs and your test publication server are all running in the same Krill instance you can quickly do the full set up using the CLI.

$ krillc repo request > publisher_request.xml

$ krillc publishers add
  --publisher $KRILL_CLI_MY_CA
  --request publisher_request.xml > repository_response.xml

$ krillc repo update --response repository_response.xml

Use the TA as the Parent of the CA

When using an embedded TA for testing then you will first need to add your new CA “ca” to it. The steps below are not specific to the TA, the same steps must be taken when registering any CA with a parent CA.
Step 1: Obtain the RFC 8183 request XML

$ krillc parents request > myid.xml

Step 2: Add the CA as a Child of the TA

In this example we need to override the ENV variable in order to refer to the TA and not the CA, and we need to indicate that we want to add this child to the CA “ta”. The following command will add the child, and the RFC 8183 XML from the “ta”:

$ krillc children add --ca ta --child ca --ipv4 "10.0.0.0/8" --ipv6 "2001:DB8::/32" --request myid.xml > parent-res.xml

If you need the response again, you can ask the “ta” again:

$ krillc children response --ca "ta" --child "ca"

Step 3: Add the TA as the Parent of the CA

$ krillc parents add --parent myta --response ./parent-res.xml

Now you should see that your “child” is certified:

$ krillc show
Name: ca
Base uri: rsync://localhostrepo/ca/
RRDP uri: https://localhost:3000/rrdp/notification.xml

ID cert PEM:
-----BEGIN CERTIFICATE-----
MIIDPDCCAiSgAwIBAgIBATANBgkqhkiG9w0BAQsFADAzMTExMTExMRMwQDA0MDQw
.. zKtG5esZ+g48ihf6j8gDyyONXEICowcjrx1Y5fnjHhL0jsTuITgYuRoGIK2KzQ+
+qLiXg2G+8s8u/1PW7PVYg==
-----END CERTIFICATE-----
Hash: 9f1376b2e1c8052c1b5d94467f8708935224c518effbe7a1c0e967578fb2215e

Total resources:
  ASNs:
    IPv4: 10.0.0.0/8
    IPv6: 2001:DB8::/32

Parents:
Handle: myta Kind: RFC 6492 Parent

Resource Class: 0
Parent: myta
State: active
Resources:
  ASNs:
The RPKI Documentation

IPv4: 10.0.0.0/8
IPv6: 2001:db8::/32

Current objects:
553A7C2E751CA0B04B49CB72E30EB5684F861987.crl
553A7C2E751CA0B04B49CB72E30EB5684F861987.mft

Children:
<none>

Add and List ROAs

$ cat >./roas.txt <<EOF
A: 10.0.0.0/24 => 64496
A: 10.1.0.0/16-20 => 64496
EOF

$ krillc roas update
--delta ./roas.txt

$ krillc roas list
10.1.0.0/16-20 => 64496
10.0.0.0/24 => 64496

Review your CA History

$ krillc history

time::command::key::success
2020-06-07T20:33:21Z::Update repo to server at: https://localhost:3000/
→rfc8181/ca ::command--1591562001--1--cmd-ca-repo-update::OK
2020-06-07T20:34:18Z::Add parent 'myta' as 'RFC 6492 Parent' ::command--
→1591562058--2--cmd-ca-parent-add::OK
2020-06-07T20:34:19Z::Update entitlements under parent 'myta': 0 => asn: 0
→blocks, v4: 1 blocks, v6: 1 blocks ::command--1591562059--3--cmd-ca-parent-
→entitlements::OK
2020-06-07T20:34:20Z::Update received cert in RC '0', with resources 'asn:
→0 blocks, v4: 1 blocks, v6: 1 blocks' ::command--1591562060--4--cmd-ca-rcn-
→receive::OK
2020-06-07T20:36:28Z::Update ROAs add: 2 remove: '0' ::command--1591562188--5-
→--cmd-ca-roas-updated::OK

Using Routinator with the Test TA

While there are many Relying Party tools, when testing with the Krill TA as noted above you will need an RP that supports a TAL file that contains an HTTPS URI.

One such RP is NLnet Labs Routinator. However, before you can use Routinator with Krill you will need to either setup Krill on a proper domain name with a matching TLS certificate issued by a trusted authority, or issue your own certificate and force Routinator to trust it.

12.9. Running a Test Environment
The RPKI Documentation

**Issue Own TLS Certificate**

```
$ mkdir /tmp/own_cert
$ cd /tmp/own_cert
$ ISSUER="/C=NL/L=Amsterdam/O=Your Organisation Name"
$ SUBJECT="/C=NL/L=Amsterdam/O=Your Organisation Name/CN=localhost"
$ SAN="DNS:localhost"
$ openssl req -new
   -newkey rsa:4096 -keyout issuer.key
   -x509 -out issuer.crt
   -days 365 -nodes -subj "$ISSUER"
$ openssl req -new -out subject.csr
   -newkey rsa:4096 -keyout subject.key
   -days 365 -nodes -subj "$SUBJECT"
$ echo "subjectAltName=$SAN" > subject.ext
$ openssl x509
   -in subject.csr -req -out subject.crt -extfile subject.ext
   -CA issuer.crt -CAkey issuer.key -CAcreateserial
   -days 365
```

**Reconfigure Krill**

```
$ kill $(cat /tmp/krill_data/krill.pid)
$ cp /tmp/own_cert/subject.crt /tmp/krill_data/ssl/cert.pem
$ cp /tmp/own_cert/subject.key /tmp/krill_data/ssl/key.pem
$ export KRILL_TEST="true"
$ export KRILL_USE_TA="true"
$ krill -c /tmp/krill.conf &
```

**Initialize Routinator**

To point Routinator at our test Krill TA we must download the TAL and store it where Routinator can find it. Also, to ensure that we don’t interfere with any existing Routinator cache on your computer let’s create a temporary cache directory for Routinator to use.

```
$ mkdir -p /tmp/routinator/{tals,rpki-cache}
$ wget -q --no-check-certificate -O /tmp/routinator/tals/ta.tal \
   https://localhost:3000/ta/ta.tal
```

**Run Routinator**

To successfully use Routinator with the Krill TA we must specify the following command line options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository-dir</td>
<td>Location of the Routinator cache directory.</td>
</tr>
<tr>
<td>tal-dir</td>
<td>Location of the Krill TA file</td>
</tr>
<tr>
<td>rrdp-root-cert</td>
<td>Location of the certificate of the authority that issued the Krill TLS certificate</td>
</tr>
<tr>
<td>allow-dubious-hosts</td>
<td>Do NOT skip the Krill localhost repository</td>
</tr>
</tbody>
</table>

The full command to invoke Routinator and the output showing our test ROAs is then:
### 12.10 Running with Docker

This page explains the additional features and differences compared to running Krill with Cargo that you need to be aware of when running Krill with Docker.

#### 12.10.1 Get Docker

If you do not already have Docker installed, follow the platform specific installation instructions via the links in the Docker official “Supported platforms” documentation.

#### 12.10.2 Fetching and Running Krill

The `docker run` command will automatically fetch the Krill image the first time you use it, and so there is no installation step in the traditional sense. The `docker run` command can take many arguments and can be a bit overwhelming at first.

The command below runs Krill in the background and shows how to configure a few extra things like log level and volume mounts (more on this below).

```
$ docker run -d --name krill -p 127.0.0.1:3000:3000 \
-e KRILL_LOG_LEVEL=debug \
-e KRILL_FQDN=rpki.example.net \
-e KRILL_AUTH_TOKEN=correct-horse-battery-staple \
-e TZ=Europe/Amsterdam \
-v krill_data:/var/krill/data/ \
-v /tmp/krill_rsync:/var/krill/data/repo/rsync/ \
nlnetlabs/krill
```

**Note:** The Docker container by default uses UTC time. If you need to use a different time zone you can set this using the `TZ` environment variable as shown in the example above.

#### 12.10.3 Admin Token

By default Docker Krill secures itself with an automatically generated admin token. You will need to obtain this token from the Docker logs in order to manage Krill via the API or the `krillc` CLI tool.

```
$ docker logs krill 2>&1 | fgrep token
docker-krill: Securing Krill daemon with token <SOME_TOKEN>
```
You can pre-configure the token via the `auth_token` Krill config file setting, or if you don’t want to provide a config file you can also use the Docker environment variable `KRILL_AUTH_TOKEN` as shown above.

### 12.10.4 Running the Krill CLI

#### Local

Using a Bash alias with `<SOME_TOKEN>` you can easily interact with the locally running Krill daemon via its command-line interface (CLI):

```bash
$ alias krillc='docker exec \\
-e KRILL_CLI_SERVER=https://127.0.0.1:3000/ \\
-e KRILL_CLI_TOKEN=correct-horse-battery-staple \\
nlnetlabs/krill krillc'

$ krillc list -f json
{
  "cas": []
}
```

#### Remote

The Docker image can also be used to run `krillc` to manage remote Krill servers. Using a shell alias simplifies this considerably:

```bash
$ alias krillc='docker run --rm \\
-e KRILL_CLI_SERVER=https://rpki.example.net/ \\
-e KRILL_CLI_TOKEN=correct-horse-battery-staple \\
-v /tmp/ka:/tmp/ka nlnetlabs/krill krillc'

$ krillc list -f json
{
  "cas": []
}
```

Note: The `-v` volume mount is optional, but without it you will not be able to pass files to `krillc` which some subcommands require, e.g.

```bash
$ krillc roas update --ca my_ca --delta /tmp/delta.in
```

### 12.10.5 Service and Certificate URIs

The Krill `service_uri` and `rsync_base` config file settings can be configured via the Docker environment variable `KRILL_FQDN` as shown in the example above. Providing `KRILL_FQDN` will set both `service_uri` and `rsync_base`.

### 12.10.6 Data

Krill writes state and data files to a data directory which in Docker Krill is hidden inside the Docker container and is lost when the Docker container is destroyed.
Persistence

To protect the data you can write it to a persistent Docker volume which is preserved even if the Krill Docker container is destroyed. The following fragment from the example above shows how to configure this:

```
docker run -v krill_data:/var/krill/data/
```

Access

Some of the data files written by Krill to its data directory are intended to be shared with external clients via the rsync protocol. To make this possible with Docker Krill you can either:

- Mount the rsync data directory in the host and run rsyncd on the host, OR
- Share the rsync data with another Docker container which runs rsyncd

Mounting the data in a host directory:

```
docker run -v /tmp/krill_rsync:/var/krill/data/repo/rsync
```

Sharing via a named volume:

```
docker run -v krill_rsync:/var/krill/data/repo/rsync
```

12.10.7 Logging

Krill logs to a file by default. Docker Krill however logs by default to stderr so that you can see the output using the `docker logs` command.

At the default `warn` log level Krill doesn’t output anything unless there is something to warn about. Docker Krill however comes with some additional logging which appears with the prefix `docker-krill:`. On startup you will see something like the following in the logs:

```
docker-krill: Securing Krill daemon with token ba473bac-021c-4fc9-9946-6ec109befec3
docker-krill: Configuring /var/krill/data/krill.conf ..
docker-krill: Dumping /var/krill/data/krill.conf config file
```

12.10.8 Environment Variables

The Krill Docker image supports the following Docker environment variables which map to the following `krill.conf` settings:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Equivalent Krill config setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRILL_AUTH_TOKEN</td>
<td>auth_token</td>
</tr>
<tr>
<td>KRILL_FQDN</td>
<td>service_uri and rsync_base</td>
</tr>
<tr>
<td>KRILL_LOG_LEVEL</td>
<td>log_level</td>
</tr>
<tr>
<td>KRILL_USE_TA</td>
<td>use_ta</td>
</tr>
</tbody>
</table>

To set these environment variables use `-e` when invoking `docker`, e.g.:
12.10.9 Using a Config File

Via a volume mount you can replace the Docker Krill config file with your own and take complete control:

```
docker run -v /tmp/krill.conf:/var/krill/data/krill.conf
```

This will instruct Docker to replace the default config file used by Docker Krill with the file `/tmp/krill.conf` on your host computer.

12.11 Krill Manager

Krill Manager is a tool for running Krill as a highly available scalable service. It brings together all of the puzzle pieces needed to administer and run Delegated RPKI with Krill.

Krill Manager includes Docker, Gluster, NGINX, Rsyncd, as well as Prometheus and Fluentd outputs for monitoring and log analysis. The integrated setup wizard allows for seamless TLS configuration, optionally using Let’s Encrypt, as well as automated updating of the application itself and all included components.

Krill with Krill Manager is available for free as a 1-Click App on the AWS Marketplace and the DigitalOcean Marketplace.

You can watch an introduction to the capabilities of Krill Manager in the video below. It walks through setting up Krill and all additional components using the 1-Click App, configure the Certificate Authority to run Delegated RPKI under a Regional Internet Registry and create ROAs, all in just 6 minutes real-time.

12.11.1 Updates

News

To stay informed about new releases of Krill and Krill Manager and to learn about documentation updates and other announcements, subscribe to the NLnet Labs RPKI mailing list. Detailed release notes are available on GitHub.

Feedback

If you would like to suggest an improvement or to report a problem, please create an issue in the Krill or Krill Manager GitHub issue tracker as appropriate.

Upgrading

Krill Manager is able to upgrade itself and the components that it manages such as Krill, NGINX and Rsync.

To upgrade Krill Manager do one of the following:

- Use the `krillmanager upgrade` CLI command.
- Answer YES when notified by the `krillmanager init` command that a new version is available.
- Automatically by `krillmanager init` if a new version is available and Initial Setup is not yet complete.
12.11.2 Prerequisites

**Required**

To use Krill Manager you will need the following:

- A DigitalOcean or AWS account *(to create the virtual machine that will run Krill Manager, Krill, et al.)*
- The ability to create DNS subdomain A records *(to point one or more domains at the new VM).*

**Optional**

- Your own TLS certificate and key file(s) (in PEM format) for the domain(s) that you wish to use with Krill. When using your own TLS certificate files you will need to upload them to the VM *before* performing the initial setup, e.g.:
  
  ```
  scp /local/path/to/certificate.pem username@<IP address>:/tmp/
  ```

  **Tip:** It is not necessary to use your own TLS certificates as Krill Manager can obtain for you a Let’s Encrypt TLS certificate per configured domain. Krill Manager will ensure that Let’s Encrypt certificates are renewed before they expire.

- Connection details and credentials for an AWS S3 compatible service to which host and application logs can be uploaded periodically.

12.11.3 Initial Setup

First, log into the virtual machine you created using SSH. Note that for AWS Marketplace EC2 instances you have use the username `ubuntu` and for DigitalOcean Marketplace droplets the username `root`.

Next, run the `krillmanager init` command to launch the interactive setup wizard.

**Important:** The wizard covers the most common cases. It does NOT yet support clustered or advanced log streaming setups or overriding the Krill or NGINX or RsyncD configuration. Such setups are possible but not yet via the wizard.

**Exiting the Wizard**

Press CTRL–C at any time to abort the wizard. You can use the `krillmanager init` command later to run the wizard again.

**Automatic Upgrade**

On first launch Krill Manager will automatically upgrade itself to a newer version if available:

```
# krillmanager init
A new version is available (v0.2.0 => v0.2.2).
Automatically upgrading to newer version v0.2.2..
Checking for newer version
Fetching newer version (v0.2.2)
```
Running post-upgrade actions
Upgrading host files
Upgrading dependencies
[###################-------------------------------] 38% Upgrading dependencies

**Note:** This is an upgrade of Krill Manager, which might not include an upgrade of Krill, it may just be an upgrade to the Manager itself and/or to one or more of the other components such as NGINX or Rsync. The version number shown is the version number of Krill Manager, not of Krill.

**Manual Upgrade**

If you have previously completed the wizard and later run `krillmanager init` again, if a new version is available you will be offered the choice to upgrade:

```
# krillmanager init
A new version is available (v0.2.0 => v0.2.2).
> Would you like to upgrade? [YES/NO]: YES
```

The default action is to upgrade, but you can continue without upgrading.

**Step by Step**

Once any available upgrade is complete you will be presented with the welcome page of the wizard. Below you will find help for each of the possible wizard pages to help you along the way:

**Wizard: Welcome**

The first page of the wizard welcomes you, summarises the steps ahead, and offers a few useful tips.

```
KRILL SETUP WIZARD: Welcome [next: Restore from backup (Optional)]
```

(continues on next page)
Welcome to the Krill setup wizard. You will be guided through the following topics:

- Restore from backup (Optional)
- Publication mode
- CA name
- Domains
- Authentication
- Logging
- HTTPS certificates

Tips:
- For more information see: https://marketplace.digitalocean.com/apps/krill
- To redo this process later, invoke the following command: krillmanager init
- To abort this process without making any changes: press CTRL-C

Press any key to continue:

Wizard: Restore from Backup

A Krill Manager backup is a complete copy of all of the application settings and data files for applications managed by Krill Manager, e.g. including Krill settings, NGINX certificate and key files, a record of your wizard choices, the RRDP and Rsync content files, application logs, etc.

If you previously made such a backup using the krillmanager backup command on this or another Krill Manager instance (and in the latter case transported the backup archive to this instance) you can choose at this point to set up this Krill Manager instance using the data in the backup archive, instead of answering wizard questions to setup Krill Manager from scratch.

With or Without Existing Backups

In most cases the restore from backup wizard page will look like this:

KRILL SETUP WIZARD: Restore from backup (Optional)  [next: Publication mode]

Would you like to:
- Setup Krill by answering questions, OR
- Restore from a previously made 'krillmanager backup' archive?

> Type: INITIALIZE, or FILE to supply an archive file path?:

If however you have previously used the krillmanager backup command on this Krill Manager instance, you will see the backup archives the command created listed on this page:

KRILL SETUP WIZARD: Restore from backup (Optional)  [next: Publication mode]

Detected backup archives:

(continues on next page)
Would you like to:
- Setup Krill by answering questions, OR
- Restore from a previously made 'krillmanager backup' archive?

> Type: INITIALIZE, or FILE to supply an archive file path?:

Krill Manager has no way of knowing about backup archives that you might have copied to the filesystem from an external storage location or from another server and so also provides an option for you to specify the path to the backup archive manually.

**Initialise or Restore**

Enter one of:

- INITIALIZE to skip this page and continue setting up Krill Manager from scratch.
- N where N is the number of a listed backup that you would like to restore.
- FILE to specify the path to a backup archive to restore, e.g. that you have previously copied to the system with the `scp` command.

**Note:** If you choose a backup to restore from the wizard will complete the restore process and then exit with a status summary of the running Krill Manager instance.

**Wizard: Publication Mode**

**Tip:**

See also the following Krill documentation:

- *Before You Start*
- *RIR and NIR Interactions*
- *Running a Publication Server*

**Krill can operate in one of two modes:**

- Publish with a 3rd party
- Publish in its own repository

The publication mode wizard page lets you choose which of these modes Krill will be configured for:

Would you like to publish with a 3rd party (e.g. NIC.br), or run your own publication server?

(continues on next page)
Info:
- If you answer YES you will need to use your repository provider's portal to obtain details needed to permit Krill to publish to the repository.
- If instead you answer NO, Krill’s embedded repository functionality will be enabled and the repository data will be served for you from HTTP/RRDP and Rsync servers that will run on this Droplet.

Warning: We advise against running your own repository as each additional repository server is one more server that Relying Parties must contact, and thus you should ensure that your repository server remains available and reachable at all times.

> Would you like to publish with a 3rd party? [YES/NO]:

3rd Party Mode

In 3rd party mode your Krill instance will only be a Certificate Authority and you will need to configure it to publish any ROAs with an external repository, e.g. that of a parent such as NIC.br.

Answering YES will enable 3rd party mode.

Self-Publishing Mode

In self-publishing mode the ROA objects created by your Krill instance will be made available by Krill Manager to Internet clients via the RRDP and Rsync protocols.

Answering NO will enable self-publishing mode.

Note: The wizard may need to ask you for additional information in later pages in order to complete the setup for self-publishing mode.

Wizard: CA Name

Normally with Krill when first visiting the web UI you will be prompted to enter the name of your Certificate Authority.

Krill Manager streamlines this process by asking you for the Certificate Authority name during the Krill Manager wizard. Once the wizard is complete Krill Manager will automatically create a CA in Krill by the name that you give here:

```
KRILL SETUP WIZARD: CA name

What name would you like to use for your Certificate Authority?

Info: A Certificate Authority will be created in Krill for you using the name that you specify.

> Certificate Authority name:
```

From Get Started with Krill:
The handle you select is not published in the RPKI but used as identification to parent and child CAs you interact with. Please choose a handle that helps others recognise your organisation. Once set, the handle cannot be changed.

**The CA name:**

- Can be used with the Krill API to manipulate the CA.
- Will be shown in the Krill web UI.
- Will be visible to child CAs.
- Will appear as a component in URIs contained in RRDP snapshot XML and delta XML content.
- Will be used as a component in the Rsync repository path for fetching content with the Rsync protocol.
- Will appear inside .roa and .mft objects served via RRDP and Rsync.

**Wizard: Domains**

Krill Manager needs to know which domain names your clients will be expected to use in order to contact your Krill services. The domains that you need to specify in this step are influenced by the choice you made in the **Wizard: Publication Mode** step:

- **3rd Party Mode:** only a single domain name for the Krill UI and API is required.
  
  > Krill domain: ca.demo.krill.cloud  
  > RRDP domain: rrdp.demo.krill.cloud  
  > Rsync domain: rsync.demo.krill.cloud

- **Self-Publishing Mode:** additional domains for RRDP and Rsync will be requested.

  > Krill domain: ca.demo.krill.cloud  
  > RRDP domain: rrdp.demo.krill.cloud  
  > Rsync domain: rsync.demo.krill.cloud

**Warning:** The domain names that you enter in this page of the wizard should already be configured to point at your Krill Manager IP address.
Note: Later in the process the wizard will offer to obtain Let’s Encrypt certificates on your behalf for the Krill and RRDP domains that you supply on this page of the wizard.

Domain Validity

Krill Manager will attempt to lookup the DNS records for the given domain names in order to verify that they are valid. If not found, Krill Manager will warn you.

If you are sure that the domain name is correct but DNS propagation has not completed yet, or for some other reason you would like to proceed, Krill Manager allows you to ignore the lookup failure:

Which domain name(s) will Krill, RRDP and Rsync on this Droplet be reachable at?

Warning: These domain names may be the same, or different. You should ensure that they are registered in DNS and resolve to this Droplet.

> Krill domain: foo.bar
DNS lookup for this domain did not return any results

> Are you sure you wish to use this domain? [YES/NO]:

Wizard: Authentication

The Krill UI and API are secured by an authentication token. In this step of the wizard you can choose the token to use, or accept a token generated by Krill Manager for you:

Please choose a token which you will use to authenticate both with the Krill API and the Krill web portal, or accept the default.

> Authentication token: 529f463d-70c1-4b01-af33-c40ee8fbfa8a

Wizard: Logging

See also:

- Amazon S3 product details & Getting your security credentials
- DigitalOcean Spaces product details & AWS S3 Compatibility
- Wikipedia: S3 API and competing services
- Logging in Krill Manager

If you have an account with a 3rd party S3-like service such as DigitalOcean Spaces, Krill Manager can use it to store copies of the logs from your host operating systemd journal and the various Krill Manager operated services, including Krill RFC exchange logs.
Would you like logs (e.g. Krill logs and RFC protocol messages, NGINX and RsyncD access and error logs, and operating system logs) to be uploaded automatically to an AWS S3 compatible provider (e.g. DigitalOcean Spaces)?

For information about these services and to sign up and create a storage “bucket” please visit the service provider web pages, e.g.:

> Would you like to upload logs to an AWS S3 compatible service? [YES/NO]:

Enter:

- NO to skip this page and continue with the wizard.
- YES to provide your S3-like service connection details.

**Warning:** If you do not choose to upload logs they are still available to you but in the event that the host suffers a failure you will lose these logs unless you capture them as part of a periodic backup process.

**Tip:** You can re-run krillmanager init later to enable log upload. However, note that only new logs from that moment on will be uploaded.

## Providing Connection Details

After answering YES you will be prompted to enter the S3-like service connection and authentication details. You will need to obtain these from your S3-like service service provider.

The wizard will try to detect the environment that it is running in and provide sensible default values where possible, e.g. in the example below the S3 Endpoint value was set by the wizard based on the fact that the Droplet on which Krill Manager was running was located in the DigitalOcean ams3 region.

> Would you like to upload logs to an AWS S3 compatible service? [YES/NO]: YES

Please provide the connection details for your bucket.

You may find the official documentation for your AWS S3 provider helpful, e.g.:
- https://developers.digitalocean.com/documentation/spaces/#aws-s3-compatibility

> Bucket Name: krillmanagerdemo
> Bucket Directory: logs
> S3 Endpoint: ams3.digitaloceanspaces.com
> Access Key: **********************
> Secret Key: *******************************************

Attempting to list contents of bucket krillmanagerdemo to verify credentials..
Invoking S3 client..
Success!
Once the wizard has the connection and authentication details it will attempt to verify them by trying to list the contents of the destination S3 bucket.

In the event that the connection and/or authentication details are incorrect the wizard will output error messages instead of *Success!* and you will be returned to the initial yes/no question where you can either choose to try again or continue without log uploading at this time.

**Video Guide**

**Advanced Configuration**

For more information about what is logged, how, to where and how to configure the logging setup beyond what is possible with the wizard, consult the Krill Manager *Logging* documentation.

**Wizard: HTTPS Certificates**

**See also:**

- *Proxy and HTTPS*
- *Let’s Encrypt HTTP-01 challenge*

As advised in *Proxy and HTTPS*, Krill Manager makes Krill available to the Internet via the NGINX proxy server. When running in self-publishing mode (see *Wizard: Publication Mode*) NGINX is also used to offer the RRDP protocol to Relying Party clients.

To secure the connection NGINX requires a TLS certificate, either provided by you or requested on your behalf from *Let’s Encrypt* by the Krill Manager wizard.

For each domain that requires a TLS certificate (either just one domain for Krill, or if using a separate domain for RRDP then that domain too) the wizard checks whether or not it already has a certificate and asks how you would like to proceed:

```plaintext
KRILL SETUP WIZARD: HTTPS certificates [next: Verifying domains]

Domain: ca.demo.krill.cloud

Checking for certificate files for domain:
  Certificate: Not found
  Private key: Not found

An HTTPS certificate and corresponding private key file are required for this domain.

How would you like to proceed? Enter one of:
  - NEW: to request (or renew) a Lets Encrypt certificate, OR
  - OWN: to supply your own certificate files from /tmp, OR

> NEW or OWN:
```

Enter one of:
The RPKI Documentation

- **NEW** to request a new Let’s Encrypt certificate.
- **OWN** to supply your own certificate files.
- **USE** to use the existing certificate that the wizard found, if any.

**Using Let’s Encrypt Certificates**

When using Let’s Encrypt issued certificates Krill Manager will ensure that they are renewed before they expire.

**Warning:** When using your own certificates, instead of Krill Manager obtained Let’s Encrypt certificates, you are responsible for replacing the certificate files before the certificates expire.

**DNS and Firewall Requirements**

For Let’s Encrypt to issue a TLS certificate the following requirements must be met:

- A DNS A record for the domain name must point to the Krill Manager IP address.
- The DNS A record must have sufficiently propagated around the global DNS network such that multiple Let’s Encrypt probe locations around the world can all resolve the name correctly.
- Port 80 on the Krill Manager instance must be open, both on the host and on any cloud firewall or proxy layer (e.g. load balancer) in front of the Krill Manager instance.

**IP Address Verification**

Prior to requesting a Let’s Encrypt certificate the wizard will ask you to confirm that DNS lookup results for the domain look correct.

```plaintext
KRILL SETUP WIZARD: HTTPS certificates [next: Applying settings]
-----------------------------------------------
Domain: ca.demo.krill.cloud

To respond to the Lets Encrypt HTTP-01 challenge, a standalone certbot web server will be started on this Droplet on port 80.

info: In order for Lets Encrypt to issue a certificate for this domain there must be a DNS A record pointing either to:

- the IP address of this Droplet: 198.51.100.2, OR
- the IP address of a proxy such as a load balancer or CDN

From this Droplet the DNS lookup result for the domain is:
ca.demo.krill.cloud. 59 IN A 198.51.100.2

> Are you sure you want to continue? [YES/NO]:
```

**Let’s Encrypt Request Log**

If you approve the wizard will then contact Let’s Encrypt:
> Are you sure you want to continue? [YES/NO]: YES
Deleting any existing Lets Encrypt certificate files for this domain
Deleting any self-signed/provided certificate files for this domain
Stopping NGINX if running
Requesting Lets Encrypt certificate for domain demo.krill.cloud
letsencrypt: Saving debug log to /var/log/letsencrypt/letsencrypt.log
letsencrypt: Plugins selected: Authenticator standalone, Installer None
letsencrypt: Registering without email!
letsencrypt: Obtaining a new certificate
letsencrypt: Performing the following challenges:
letsencrypt: http-01 challenge for demo.krill.cloud
letsencrypt: Waiting for verification...
letsencrypt: Cleaning up challenges
letsencrypt: IMPORTANT NOTES:
letsencrypt: - Congratulations! Your certificate and chain have been saved at:
letsencrypt: /etc/letsencrypt/live/ca.demo.krill.cloud/fullchain.pem
letsencrypt: Your key file has been saved at:
letsencrypt: /etc/letsencrypt/live/ca.demo.krill.cloud/privkey.pem
letsencrypt: Your cert will expire on 2020-07-07. To obtain a new or tweaked
letsencrypt: version of this certificate in the future, simply run certbot
letsencrypt: again. To non-interactively renew *all* of your certificates, run
letsencrypt: "certbot renew"
letsencrypt: - Your account credentials have been saved in your Certbot
letsencrypt: configuration directory at /etc/letsencrypt. You should make a
letsencrypt: secure backup of this folder now. This configuration directory will
letsencrypt: also contain certificates and private keys obtained by Certbot so
letsencrypt: making regular backups of this folder is ideal.
letsencrypt: - If you like Certbot, please consider supporting our work by:
letsencrypt: Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
letsencrypt: Donating to EFF: https://eff.org/donate-le

Press any key to continue:

In this example the request succeeded. If any problems occurred the log would instead indicate the reason for the failure.

Once you press a key to continue you will be returned to the start of the HTTPS Certificates wizard page. The wizard will verify if it now has a certificate for the domain and if so will give you the option to USE it:

<table>
<thead>
<tr>
<th>KRILL SETUP WIZARD: HTTPS certificates</th>
<th>[next: Verifying domains]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain: ca.demo.krill.cloud</td>
<td></td>
</tr>
<tr>
<td>Checking for certificate files for domain:</td>
<td></td>
</tr>
<tr>
<td>Certificate: Found</td>
<td></td>
</tr>
<tr>
<td>Private key: Found</td>
<td></td>
</tr>
<tr>
<td>This certificate was issued for: subject=CN = ca.demo.krill.cloud</td>
<td></td>
</tr>
<tr>
<td>This certificate was issued by: issuer=C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3</td>
<td></td>
</tr>
<tr>
<td>How would you like to proceed? Enter one of:</td>
<td></td>
</tr>
<tr>
<td>- USE: Use this certificate, OR</td>
<td></td>
</tr>
<tr>
<td>- NEW: to request (or renew) a Lets Encrypt certificate, OR</td>
<td></td>
</tr>
<tr>
<td>- OWN: to supply your own certificate files from /tmp, OR</td>
<td></td>
</tr>
<tr>
<td>&gt; NEW, OWN, or USE:</td>
<td></td>
</tr>
</tbody>
</table>

12.11. Krill Manager
Wizard: Verifying Domains

Before applying all settings and starting the services, the wizard will run a basic check to verify that each configured domain resolves and can be connected to:

```plaintext
KRILL SETUP WIZARD: Verifying domains [next: Applying settings]

Running a basic check that each configured domain resolves to this Droplet and can be connected to.
- ca.demo.krill.cloud: OKAY
- rrdp.demo.krill.cloud: OKAY
- rsync.demo.krill.cloud: OKAY
```

Wizard: Applying Settings

At this stage the wizard has everything it needs to generate application configuration files based on the settings chosen in the earlier wizard pages and to launch the applications:

```plaintext
KRILL SETUP WIZARD: Applying settings [next: Setup complete]

Generating Krill configuration file
Preparing NGINX configuration
Preparing RSYNCD configuration
Creating network krill_default
Creating service krill_cert_renewer
Creating service krill_host_metrics
Creating service krill_krill
Creating service krill_nginx
Creating service krill_nginx_metrics
Creating service krill_rsyncd
Waiting for services to become ready..
[###################################-----] 88% Starting services..

Once the applications are running the wizard will configure the CA Name you requested (assuming no CA exists already), and in self-publishing mode the embedded Krill repository will be configured for use by the newly created CA:

```plaintext
Waiting for services to become ready..
Creating CA 'Acme-Corp-Intl'..
Registering CA 'Acme-Corp-Intl' with the embedded repository..
```

Wizard: Setup Complete

Once everything is setup the wizard will report the status of the running services and the locations at which the services can be found:

```plaintext
KRILL SETUP WIZARD: Setup complete [next: END]

Service status summary:
(continues on next page)```
All services appear to be running.

Krill and related services should now be available as follows:

- Krill Web Portal: https://ca.demo.krill.cloud/ (token: 4741d1f8-e317-488e-8c8a-a36e0cb16bf1)
- RRDP URI : https://rrdp.demo.krill.cloud/rrdp/
- Rsync URI : rsync://rsync.demo.krill.cloud/repo/
- Prometheus monitoring endpoints:
  - Krill : http://ca.demo.krill.cloud:9657/metrics
  - NGINX : http://ca.demo.krill.cloud:9113/metrics
  - Docker : http://ca.demo.krill.cloud:9323/metrics
  - O/S : http://ca.demo.krill.cloud:9100/metrics
  - Gluster : http://ca.demo.krill.cloud:8080/metrics

Please consult the documentation for guidance on administering and monitoring these services.

Thanks,

The NLnet Labs RPKI team.

Press any key to continue:

---

**Verify that Krill is Running**

Use the [Krill Web Portal](https://ca.demo.krill.cloud/) link and token to login to the Krill UI where you should see your newly created Certificate Authority and the details required to link your CA to a parent:

Refer to the [Krill Documentation](https://docs.krill.cloud) to learn more about Krill.

**Next Steps**

Click Next or return to the index to continue learning about Krill Manager.

**12.11.4 Using the CLI**

Krill Manager is controlled via a command line interface (CLI) tool called `krillmanager`, separate to the `krillc` tool that can be used to manage a Krill server. This page documents how to use both in the context of a Krill Manager instance.

`krillc`

On a Krill Manager machine you can invoke the `krillc` command just as if you had installed Krill yourself. However, what you are actually invoking is a special wrapper provided by Krill Manager which simplifies and tailors the use of the `krillc` command to the Krill Manager context. You can read more about this in the `krillmanager krillc` documentation below.
**krillmanager**

Krill Manager supports the following commands:

```
# krillmanager --help

Usage: COMMAND [ARGUMENTS]
```

A tool for managing NLnet Labs Krill and related services.

Commands:

- **backup**  Backup Krill and supporting services state
- **certs**  List the TLS certificates in use by NGINX
- **help**  Display this message
- **init**  (Re)initialize DNS, TLS and Krill settings
- **krillc**  Execute Krill CLI commands
- **logs**  Show the service container logs
- **renew**  Renew expiring NGINX Lets Encrypt certificates
- **restart**  Restart Krill and supporting services
- **restore**  Restore Krill and supporting services state from a backup
- **start**  Start Krill and supporting services
- **status**  Show the status of the service containers
- **stop**  Stop Krill and supporting services
- **upgrade**  Upgrade Krill and supporting services
Querying the Version

```
# krillmanager --version
v0.3.2 [Krill: v0.7.3]
```

This tells you that Krill Manager is version 0.3.2, and that it deploys version 0.7.3 of Krill.

**Command: backup**

Creates a tar archive on the host filesystem containing all configuration files and data for Krill Manager and the components that it manages. This includes NGINX certificate files and Krill embedded repository data files. It does NOT include log files.

The path to the created archive will be printed to the terminal on completion of the backup. The backup archive can be restored later using the `krillmanager restore` command.

**Warning:** In order to avoid impacting your system the archive is made while all applications are running. There is a very small chance that a Krill data file will be inconsistently captured in the backup.

**Command: certs**

This command outputs information both about the certificates in use by NGINX, and the certificates being managed by the Lets Encrypt certbot tool.

**Command: help**

Displays the usage summary.

**Command: init**

Runs the (re)configuration wizard. See *Initial Setup*.

The `init` command supports some useful options for test and clustered scenarios that are not available via the interactive wizard:

```
# krillmanager [--use-lets-encrypt-staging] [--private] init
```

The `--use-lets-encrypt-staging` option causes any Let’s Encrypt certificate requests to be made to the Let’s Encrypt staging environment rather than the production environment. This can be useful to avoid hitting Let’s Encrypt rate limits in the production environment through repeated testing.

The `--private` option causes a self-signed certificate to be issued to NGINX for serving the RRDP FQDN. This might be of interest if running Krill Manager behind a proxy which itself has the real RRDP certificate.
Command: krillc

This command invokes the Krill CLI tool krillc.

Tip: You can also invoke this command as just krillc without the krillmanager prefix, just like in the krillc documentation.

In a Krill Manager instance there is no krillc binary installed on the host. Instead this command runs a throw away Krill Docker container and invokes the krillc binary contained within.

Normally invoking krillc requires also defining environment variables or passing command line arguments to tell krillc where Krill is and how to authenticate with it. With Krill Manager this is taken care of for you automatically. If needed you can override the defaults using command line arguments in order to interact with a separate external instance of Krill.

Krill Manager also simplifies the interaction with the host filesystem by automatically remapping any paths to input files supplied on the command line so that they work when krillc accesses them from within the Docker container.

Command: logs

This command outputs the Docker service logs for key Krill Manager components. If invoked without any arguments it displays a usage tip:

```
# krillmanager logs
Usage: krillmanager logs <krill|nginx|rsyncd> [-f] [--tail=n]
```

The -f argument tells the command to keep following the log output.
The --tail argument tells the command to show only n lines of prior log output.

Command: renew

This command forces the Lets Encrypt certbot agent to attempt to renew any Let’s Encrypt certificates that it is managing. If the certificates are renewed the NGINX instances will be signalled to reload the certificate files without causing any downtime.

Note: It shouldn’t be necessary to use this command as it is triggered automatically once a day.

Command: restart

This command is an alias for stop followed by start.
Command: restore

This command restores a backup made previously by the `backup` command.

The restored data will be processed by the current Krill Manager version which may be newer than the version that created the backup. Any incompatibilities should be handled automatically by the restore process.

If Krill and related services were running when the restore process started Krill Manager will stop them prior to restore and start them again afterwards. Otherwise you will need to use the `start` command to start the services after restore.

Note: If the domain names referred to in the backup archive do not resolve to the external public IP address of the machine being restored to, the DNS setup or configuration in the archive may be incorrect. Krill Manager will advise against proceeding with the restore in this case. A valid scenario in which this can occur is when using a CDN for RRDP in which case the FQDN resolves to the CDN endpoint and not to the instance directly.

Command: start

Deploy all Krill Manager managed components according to the configuration settings chosen when the `init` command was last run.

Command: status

Display a status report indicating which of the Krill Manager components are running. It also shows a recap of key URIs that can be used to work with the Krill Manager instance.

Command: stop

Terminate all Krill Manager components.

Warning: This will cause clients to receive connection refused errors.

Command: upgrade

Check to see if a newer version of Krill Manager is available and if so offer to upgrade to it.

Note: A newer version of Krill Manager doesn’t necessarily contain a newer version of Krill.
12.11.5 Monitoring

See also:

Known issues:

- Wrong version of node_exporter
- Missing rsync metrics
- Limited Docker metrics - use cAdvisor

The available Prometheus endpoints for monitoring Krill Manager components can be determined using the `krillmanager status` command:

```bash
# krillmanager status
... ...
- Prometheus monitoring endponts:
  - Krill : http://<YOUR DOMAIN>:9657/metrics
  - NGINX : http://<YOUR DOMAIN>:9113/metrics
  - Docker : http://<YOUR DOMAIN>:9323/metrics
  - O/S : http://<YOUR DOMAIN>:9100/metrics
  - Gluster : http://<YOUR DOMAIN>:8080/metrics
  - Fluentd : http://<YOUR DOMAIN>:24231/metrics
```

Note: Fluentd metrics are available from Krill Manager v0.2.2.
Note: In cluster mode the per-node metrics (NGINX, Docker, O/S and Gluster) should be queried on the node you are interested in, Krill Manager does NOT aggregate cluster metrics for you.

Tip: Krill metrics can be queried on any cluster node, NGINX will fetch them from Krill on whichever cluster node the single Krill instance is running.

Visualisation

To visualise the monitoring endpoint metrics deploy your own Prometheus and Grafana servers, e.g. using these DigitalOcean Marketplace Apps:


Alternatively, if you don’t mind losing your monitoring and alerting if your server has problems, you could deploy Prometheus and Grafana on your Krill server like this.

Add stanzas like the following to the scrape_configs section of the prometheus.yml file on the Prometheus server and restart Prometheus:

```yaml
scrape_configs:
  ...
  ...
  ...
  - job_name: 'krill'
    static_configs:
      - targets: ['<YOUR DOMAIN>:9657']
  - job_name: 'nginx'
    static_configs:
      - targets: ['<YOUR DOMAIN>:9113']
```

Add http://<PROMETHEUS DOMAIN OR IP>:9090 as a datasource to Grafana.

Then import Grafana Dashboards by ID, e.g.:

- https://grafana.com/grafana/dashboards/1860 (Node Exporter Full)
- https://grafana.com/grafana/dashboards/11199 (NGINX by nginxinc)

Alerting

Grafana can be configured to send notifications to a variety of destination types when alert conditions are met.

12.11.6 Logging

In Krill Manager when we refer to logs we primarily refer to a series of (mainly) unstructured messages, not to metrics such as counters and guages exposed by Prometheus endpoints.

On a Krill Manager host journald is the primary log subsystem and Docker container logs are routed to the journal via the Docker journald logging driver.
Log Viewing

- Host logs can be viewed in the usual way with `journalctl` and via files stored in `/var/log/`.
- Primary Krill Manager logs can be viewed with the `krillmanager logs` command.
- Other Krill Manager logs can be viewed with the `docker service logs` command.

Tip: In cluster mode `krillmanager logs` and `docker service logs` can be used to view logs even if the source container is on a slave cluster node.

Log Aggregation, Upload & Analysis

Using FluentD Krill Manager can:

- aggregate journal logs across all cluster nodes together.
- stream journal logs to an AWS S3 compatible storage service.
- stream journal logs to one of many 3rd party services for external processing and analysis.

Using s3cmd Krill Manager can:

- upload Krill RFC audit log files to an AWS S3 compatible storage service.

Note: FluentD and s3cmd related Krill Manager Docker services are only created if log uploading was enabled during **Initial Setup**.

Upload Frequency

RFC protocol exchange logs are uploaded hourly. All other logs are uploaded at least every 10 minutes, more frequently if there is a lot of logging activity.

Force Flush

If needed you can force FluentD to flush its buffers which should cause it to stream any data it has pending to the destination, e.g. S3 compatible storage or a custom destination that you have configured:

1. Use `docker service ps krill_log_uploader` to find the server running the log upload container.
2. SSH to the server running the log upload container.
3. Use `docker ps` to find the the container ID or name of the `krill_log_uploader` container.
4. Use `docker kill -s USR1 <container PID/name>` to send the flush signal to FluentD.
5. Use `docker logs <container PID/name>` to see that the flush was received and if it caused any upload activity, e.g.:

```
$ docker service logs --raw z1c6ksk6zvdx | fgrep flush
2020-04-21 08:44:25 +0000 [info]: #0 force flushing buffered events
2020-04-21 08:44:25 +0000 [info]: #0 flushing all buffer forcibly
```
Log Retention

When log upload is enabled, local copies of Krill RFC audit logs are deleted after two days as these logs can become quite large. All other logs are rotated according to the default journald behaviour and logrotate configuration.

Log Bucket Structure

When using the default s3.conf fluentd config file, uploaded logs are structured like so:

```
/<Bucket Directory>/rfc_trail
/<Bucket Directory>/YYYY/MM/DD/HH/<hostname>/<service>.<N>.gz
/<Bucket Directory>/YYYY/MM/DD/HH/<hostname>/<container>/<instance id>.<N>.gz
```

Where `<Bucket Directory>` is the value you provided to the wizard.

Log File format

The format of the files is dependent on the type of log file:

- `rfc_trail` log files are in a Krill internal binary format.
- `<service>` log files are in JSON format.
- `<container>` log file are in JSON format with additional fields.

This SSHD log message shows a `<service>` log line example:

```
{
  "hostname": "demomaster",
  "source": "syslog",
  "syslog_id": "sshd",
  "ts_epoch_ms": "1586277165425045",
  "message": "Invalid user test from 104.236.250.88 port 49112"
}
```

This NGINX access log message shows a `<container>` log line example:

```
{
  "hostname": "demomaster",
  "source": "journal",
  "syslog_id": "6ef2b2bf3eba9",
  "ts_epoch_ms": "1586278786997270",
  "container": "krill_nginx.w2ia8pd3b2kxqm77uwypooghm.01v5trgdykgea09y1hsd5",
image": "krillmanager/http-server:v0.1.0@sha256:f88c52b73abf86c3223dcf4c0cc3ff8351f61e74ee307aa8c420c9e0856678f7"
}
```

Custom Behaviour

**Warning:** When providing custom configuration files you should use the `krillmanager edit` command to create and edit configuration files so that the changes are properly replicated across all cluster nodes.
Customising Log Streaming

Files in `/fluentd-conf/*.conf` can be edited with `krillmanager edit` to configure fluentd according to your own design, streaming logs to any of the many 3rd party services that fluentd supports. Configuration elements should be placed inside a label stanza like so:

```xml
<label @ready>
  <match **>
    @type s3
  ..
</match>
</label>
```

When working with Fluentd configuration files note the following useful commands:

```
# Reload the Fluentd configuration:
docker service restart krill_log_uploader --force

# Flush Fluentd output buffers:
docker kill -s SIGUSR1 <krill_log_uploader container name/id>
```

See also:
- fluentd: List of Data Outputs
- fluentd: Input / Output Plugins

Diagnosing Streaming Problems

Krill Manager v0.2.2 added a Fluentd Prometheus metrics endpoint on port 24231 at `/metrics`. The statistics published at this endpoint can help identify whether events are being received and handled by the expected Fluentd output plugins.

Logs from the log streaming and audit trail upload processes can be checked with the following commands:

```
docker service logs --since=2h krill_log_forwarder
docker service logs --since=2h krill_log_uploader
docker service logs --since=2h krill_rfc_trail_uploader
```

If you have configured log uploading to an S3 bucket look for permission errors in these logs. For AWS S3 a sample IAM policy that grants sufficient access for Krill Manager is:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KrillManagerLogUpload",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3::<bucket_name>",
                "arn:aws:s3::<bucket_name>/*
            ]
        }
    ]
}
```

(continues on next page)
Remember to replace `<bucket_name>` with the name of the actual S3 bucket.

**Customising Audit Log Upload**

The `/s3cmd-conf/s3cmd.conf` file can be edited with `krillmanager edit` to take advantage of any additional features of your S3-like service provider that `s3cmd` supports.

**See also:**
- About the `s3cmd` configuration file

**Analysis Examples**

**Manual Log Analysis**

**Tip:** Upload to an AWS S3 compatible service is primarily intended for archival and root cause analysis after an incident. If your intention is to extract interesting metrics or you would like a more visual way to interact with your logs we suggest feeding tools like Grafana Loki or Elastic Search from FluentD.

Assuming that you have configured Krill Manager to store logs in a DigitalOcean Space, you can generate a report of RRDP clients visiting your Krill Manager instance on a particular date like so:

```
532 RIPE NCC RPKI Validator/3.1-2020.01.13.09.31.26
515 reqwest/0.9.19
190 Jetty/9.4.15.v20190215
101 RIPE NCC RPKI Validator/3.1-2019.12.16.15.18.18
81 Routinator/0.7.0
...
```

Such a report can be produced using commands like those below:

```
$ DATE_OF_INTEREST="2020/05/11"
$ S3_BUCKET_NAME="my-bucket-name"
$ export AWS_ACCESS_KEY_ID="your-access-key"
$ export AWS_SECRET_ACCESS_KEY="your-secret-access-key"
$ docker run -it --rm \
  -v /tmp/logs:/mnt/logs \
  -e AWS_ACCESS_KEY_ID \
  -e AWS_SECRET_ACCESS_KEY \
  --entrypoint=s3cmd \
  krillmanager/log-uploader:v0.1.1 \
  get \
    --host-bucket="%(bucket)s.ams3.digitaloceanspaces.com" \
    --rexclude=".*" \
    --rinclude=".*$DATE_OF_INTEREST}`).*/krill_nginx/.*" \
  s3://$S3_BUCKET_NAME//logs/ /mnt/logs/
```
Streaming to Elasticsearch

**Note:** The examples below require Krill Manager v0.2.2 or higher.

Using the Fluentd support integrated into Krill Manager you can stream logs to 3rd party log analysis tools such as EFK (Elasticsearch, Fluentd and Kibana).

When streaming to an external service you can either do that:

- Instead of streaming to an S3 storage backend: replace `s3.conf`.
- In addition to streaming to an S3 storage backend: modify `s3.conf` and add additional Fluentd config files.

Below is an example configuration for sending rsync access logs to Elasticsearch:

```xml
# elastic-search.conf
<label @ready>
  <filter **>
    @type grep
    <regexp>
      key container
      pattern /krill_rsyncd..+/  
    </regexp>
  </filter>

  <filter **>
    @type parser
    key_name message
    reserve_data true
    <parse>
      @type regexp
      expression /^(?<datetime>d+/d+/d+ d+:d+:d+) \[(?<unknown>[^\]]*)\] connect \n      from (?<client_host>[^ ]+) ((?<client_ip>[^\)])*+))$/
    </parse>
  </filter>

  <match **>
    @type elasticsearch
    host elasticsearch.mydomain.com
    port 9200
    logstash_format true
  </match>
</label>
```

118 Chapter 12. Krill
A similar technique can be used to stream NGINX access logs, using the built-in nginx parser in Fluentd. However, if you use a CDN (content delivery network) in front of your Krill Manager instance(s) you’ll want to analyze the CDN provider logs, not the NGINX logs.

To stream rsync access logs to Elasticsearch but also still upload all logs to an S3 compatible storage target, use a copy configuration like so:

```xml
# copy.conf
<label @ready>
  <match **>
    @type copy
    <store>
      @type relabel
      @label @s3
    </store>
    <store>
      @type relabel
      @label @elastic-search
    </store>
  </match>
</label>

# elasticsearch.conf
<label @elastic-search>
  # the remainder of this file is the same as above
</label>

# s3.conf
<label @s3>
  # the remainder of this file is the same as the stock s3.conf file
  # that comes with Krill Manager.
</label>
```

### Installing Additional Fluentd Plugins

Krill Manager comes with the following Fluentd plugins pre-installed:

- fluent-plugin-elasticsearch
- fluent-plugin-prometheus
- fluent-plugin-rewrite-tag-filter
- fluent-plugin-s3
- fluent-plugin-systemd

**Note:** The Elasticsearch plugin is included with Krill Manager from v0.2.2.

```bash
$ CONTAINER_ID=$(sudo docker ps -q --filter "name=krill_log_uploader")
$ sudo docker exec -it $CONTAINER_ID /bin/bash
# gem install fluent-plugin-XXX
```

(continues on next page)
# exit
$ sudo docker commit $(CONTAINER_ID) krillmanager/log-streamer:custom
$ sudo docker service update krill_log_uploader --image krillmanager/log-streamer:custom

**Warning:** An upgrade of Krill Manager may cause the service to revert to a stock Krill Manager image. Repeat the steps above to re-install the missing plugin. You can also request inclusion of the plugin in the next Krill Manager release by submitting an issue to the [Krill Manager GitHub issue tracker](https://github.com).  

## 12.11.7 Cluster Mode

Krill Manager supports running on a cluster of servers but by default assumes that it is not part of a cluster.  

**Warning:** Only the RRDP and Rsync servers will be deployed more than once in the cluster. Krill does not yet support more than one active instance of the Krill CA sharing the same data directory. Running more than one instance of the Krill CA using the same data directory will NOT work.

### Setting up a Cluster

#### Activate Cluster Mode

There is no support in the *Initial Setup* wizard for activating cluster mode, instead it must be done via command line arguments to the wizard.

After deploying N servers running Krill Manager, e.g. N instances of the DigitalOcean Marketplace 1-Click App, execute the following commands via SSH:

```
# open-cluster-ports  # on both master and slave
# krillmanager --slave-ips=<ipv4,ipv4,...> init  # on the master only
# krillmanager --slave=1 init  # on the slaves only
```

**Example:**

In a shell:
```
$ ssh root@slave1.rpki.example.com
# open-cluster-ports
# krillmanager --slave=1 init
...
Slave initialized
```

In another shell:
```
$ ssh root@slave2.rpki.example.com
# open-cluster-ports
# krillmanager --slave=1 init
...
Slave initialized
```

In another shell:
```
$ ssh root@master.rpki.example.com
```

(continues on next page)
# open-cluster-ports

Joining slave at 10.0.0.2 to our GlusterFS cluster
Joining slave at 10.0.0.3 to our GlusterFS cluster

Waiting for all GlusterFS peers to become 'Connected'.

Initializing Swarm manager at <some.public.ip.address>
Sharing Swarm join token via GlusterFS
Waiting for 2 swarm workers to be in status 'Ready'
Waiting for 1 swarm workers to be in status 'Ready'

### Warning:
open-cluster-ports is a simple helper script that opens to the world the ports required for cluster servers to communicate with each other. In a production setup you should restrict access so that these ports are only open between cluster servers and not to the wider Internet, either via ufw or via a cloud firewall.

## Deploy & Configure a Load Balancer

For requests to be able to reach the Krill Manager servers, the load balancer must be configured to forward ports:

### Port Forwarding Rules:

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Required For</th>
</tr>
</thead>
</table>
| 80   | HTTP     | • HTTP -> HTTPS redirect.  
      |          | • Let’s Encrypt HTTP-01 challenge responses. |
| 443  | HTTPS    | • Krill UI  
      |          | • Krill API  
      |          | • RRDP |
| 873  | TCP      | • Rsync |

### TLS Termination:
Either configure your load balancer with a TLS certificate or set it to pass through TLS traffic still encrypted to the cluster servers.

### Health Check:
In order for the load balancer to route traffic only to healthy cluster servers you should configure a health check.

### Proxy Protocol:
Do NOT enable Proxy Mode on your load balancer. See the F.A.Q. item below for more information.

## Configure DNS

In order to request a Let’s Encrypt TLS certificate via Krill Manager the cluster servers need to be reachable via the desired DNS name, e.g. via a DNS A or CNAME record.
F.A.Q.

Should I Use a Cluster?

Whether cluster mode is needed or is the right way to achieve your objectives depends on your particular use case. If using a 3rd party repository and only a few ROAs, then you probably don’t need a cluster.

A cluster provides various benefits including:

1. Higher availability - loss of a cluster server, whether due to an issue or while upgrading, does not cause the service to be down toward customers.

2. Scalability - RRDP and Rsync requests can be served by multiple servers instead of just one.

A cluster also comes with some costs, e.g.:

1. The obvious cost of running more (virtual) hardware.

2. The complexity cost of operating and maintaining a cluster, though Krill Manager greatly reduces this.

How Is Cluster Mode Different To Normal Mode?

The main difference is that instead of having one server running NGINX and RsyncD, in cluster mode every cluster server will run NGINX and RsyncD.

In clustered mode the Gluster volume enables Krill Manager to replicate configuration, TLS certificates, RRDP and Rsync repo contents, etc. to every cluster server.

Why Not Just Use a CDN?

Currently Relying Party software communicate with RPKI repository servers using the Rsync protocol and most also support the RRDP protocol.

Using a CDN (e.g. Fastly as used by the NLnet Labs production Krill deployment) should increase availability, increase capacity and decrease latency, but only for RRDP, not for Rsync. One could argue that Rsync is being rapidly obsoleted by RRDP and it is only a matter of time before Rsync is not used by Relying Parties at all.

Where Should My Cluster Servers Be Located?

Depending on how many 9’s of uptime/availability you are aiming for, you should consider whether your cluster servers are separate enough from each other, e.g. several VMs running on the same server or in the same rack is less robust than spreading the VMs across cloud availability zones or across regions.

Note however that the further apart your cluster servers are from each other the longer it may take Gluster to keep the replicated volume contents consistent.

Also, not all load balancing technologies support wider separation, e.g. a cloud load balancer may be able to balance across VMs in one region but not across regions.

How Can I Balance Traffic Across My Cluster?

You can use a load balancer (e.g. the DigitalOcean Load Balancer), anycast IP, a CDN provider, geographic/latency based DNS, etc.
Is Proxy Protocol supported?

Not yet. Without Proxy Protocol you will likely see the IP addresses of the proxy in your NGINX and RsyncD logs rather than that of the real client.

How Can a Proxy Check the Backend Health?

Krill Manager does not yet offer a dedicated health check endpoint. When using a load balancer or other proxy that supports health checks you are currently limited to testing TCP or HTTP(S) connectivity. For example if using a single DigitalOcean Load Balancer you can check either connectivity to NGINX or to RsyncD but not both. A dedicated Krill Manager health check endpoint would allow you to direct traffic to the cluster server only if all services were green.

What Happens If a Cluster Server Becomes Unreachable?

If your proxy detects that the backend is unreachable then clients (possibly after some delay) will no longer be routed to the “dead” server but will continue to be able to access RRDP and Rsync endpoints on the remaining servers.

If your proxy monitors the health of the backend services and the health check fails then connections to that service will be routed to other “healthy” servers. However, as noted above, the current health check options are not perfect.

If the “unhealthy” cluster server is a slave and the “master” loses its connection to the slave then any Krill Manager components that were running only on that cluster server will be re-launched on a remaining “healthy” cluster server.

If the “unhealthy” cluster server is the “master” then any Krill Manager components that were running only on that cluster server will be lost and you will need to manually fix the Docker Swarm and Gluster clusters. However, note that NGINX and RsyncD run on every cluster server and so clients will still be able to get the last synced RRDP and Rsync data from the remaining “healthy” cluster servers. You may however lose Krill and/or log streaming/uploading services.

Can I Use Plain HTTP Behind a Load Balancer?

No, Krill Manager does not support this.

Can I Use Self-Signed TLS Certificates Behind a Load Balancer?

In the case where the load balancer handles TLS termination, to avoid having to install and renew real certificates on both the load balancer and the cluster servers the --private argument can be used on the master. This will cause Krill Manager to generate self-signed certificates for the cluster NGINX instances. E.g.

```
# krillmanager --slave-ips=<ipv4>,<ipv4>,... --private init
```

How is the cluster established?

1. The master server activates Docker Swarm mode becoming a Swarm Manager.
2. The master server adds the other servers as Gluster peers.
3. The master server creates a Gluster replication volume across the peers. Each peer will have a complete copy of the data written to the volume.
4. The master server writes the Docker Swarm join token to the Gluster volume.
5. The slave servers detect the join token and use it to join the Docker Swarm.

Can I add or remove cluster servers later?

1. Run `open-cluster-ports` and `krillmanager --slave=1 init` as usual on any new slave servers.
2. Run `krillmanager --slave-ips=<ipv4>,<ipv4>,... init` on the master cluster server with the new set of IPv4 cluster slave addresses:
   - Any missing slave IP addresses will cause Krill Manager to forcibly disconnect those slaves from the Gluster cluster.
   - Any new slave IP addresses will be added to the Gluster cluster.
   - The new slaves will then add themselves to the Swarm cluster.
3. Terminate the removed slave servers.

Is the Swarm Manager highly available?

No. This could be done but adds complexity while adding little value. If the manager server is lost the worst case is that the Krill UI and API become unavailable if Krill was running on the Swarm Manager server, RRDP and Rsync endpoints will continue to be available.

Is the Docker Swarm Routing Mesh Used?

No, the NGINX (HTTP(S)/RRDP) and Rsync containers bind directly to the host interface ensuring that IPv6 is supported and eliminating an unnecessary extra proxy hop.
Routinator 3000 is free, open source RPKI Relying Party software written by NLnet Labs in the Rust programming language.

The application is designed to be lightweight and have great portability. This means it can run on any Unix-like operating system, but also works on Microsoft Windows. Due to its lean design, it can run effortlessly on minimalist hardware such as a Raspberry Pi. Monitoring is possible through the built-in Prometheus endpoint. It allows you to build beautiful dashboards for detailed insights.

Routinator connects to the Trust Anchors of the five Regional Internet Registries (RIRs) — APNIC, AFRINIC, ARIN, LACNIC and RIPE NCC — downloads all of the certificates and ROAs in the various repositories, verifies the signatures and makes the result available for use in the BGP workflow. It can perform RPKI validation as a one-time operation and store the result on disk in formats such as CSV, JSON and RPSL, or run as a service that periodically fetches and verifies RPKI data. The data is then served via the built-in HTTP server, or fetched from RPKI-capable routers via the RPKI-RTR protocol.

If you run into a problem with Routinator 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. This is also the place where we will announce releases of the application and updates on the project.

You can follow the adventures of Routinator on Twitter and listen to its favourite songs on Spotify.

## 13.1 Installation

Getting started with Routinator is really easy either building from Cargo or running with Docker.

### 13.1.1 Quick Start

Assuming you have a newly installed Debian or Ubuntu machine, you will need to install rsync, the C toolchain and Rust. You can then install Routinator and start it up as an RTR server listening on 127.0.0.1 port 3323 and HTTP on port 9556:
apt install rsync build-essential
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
cargo install --locked routinator
routinator init
   # Follow instructions provided
routinator server --rtr 192.0.2.13:3323 --http 192.0.2.13:9556

If you have an older version of Rust and Routinator, you can update via:

   rustup update
cargo install --locked --force routinator

If you want to try the master branch from the repository instead of a release version, you can run:

cargo install --git https://github.com/NLnetLabs/routinator.git

13.1.2 Quick Start with Docker

Due to the impracticality of complying with the ARIN TAL distribution terms in an unsupervised Docker environment, before launching the container it is necessary to first review and agree to the ARIN Relying Party Agreement (RPA). If you agree to the terms, you can let the Routinator Docker image install the TALs into a mounted volume that is later reused for the server:

   # Create a Docker volume to persist TALs in
   sudo docker volume create routinator-tals
   # Review the ARIN terms.
   # Run a disposable container to install TALs.
   sudo docker run --rm -v routinator-tals:/home/routinator/.rpki-cache/tals 
      nlnetlabs/routinator init -f --accept-arin-rpa
   # Launch the final detached container named 'routinator' exposing RTR on 
   # port 3323 and HTTP on port 9556
   sudo docker run -d --restart=unless-stopped --name routinator -p 3323:3323 
      -p 9556:9556 -v routinator-tals:/home/routinator/.rpki-cache/tals 
         nlnetlabs/routinator

13.1.3 System Requirements

At this time, the size of the global RPKI data set is about 500MB. Cryptographic validation of it takes Routinator about 2 seconds on a quad-core i7.

When choosing a system to run Routinator on, make sure you have 1GB of available memory and 1GB of disk space. This will give you ample margin for the RPKI repositories to grow over time, as adoption increases.

13.1.4 Getting Started

There are three things you need to install and run Routinator: rsync, a C toolchain and Rust. You can install Routinator on any system where you can fulfil these requirements.

You need rsync because most RPKI repositories currently use it as its main means of distribution. Some of the cryptographic primitives used by Routinator require a C toolchain. Lastly, you need Rust because that’s the programming language that Routinator has been written in.
rsync

Currently, Routinator requires the rsync executable to be in your path. Due to the nature of rsync, it is unclear which particular version you need at the very least, but whatever is being shipped with current Linux and *BSD distributions and macOS should be fine. Alternatively, you can download rsync from its website.

On Windows, Routinator requires the rsync version that comes with Cygwin – make sure to select rsync during the installation phase.

C Toolchain

Some of the libraries Routinator depends on require a C toolchain to be present. Your system probably has some easy way to install the minimum set of packages to build from C sources. For example, `apt install build-essential` will install everything you need on Debian/Ubuntu.

If you are unsure, try to run `cc` on a command line and if there’s a complaint about missing input files, you are probably good to go.

Rust

The Rust compiler runs on, and compiles to, a great number of platforms, though not all of them are equally supported. The official Rust Platform Support page provides an overview of the various support levels.

While some system distributions include Rust as system packages, Routinator relies on a relatively new version of Rust, currently 1.40 or newer. We therefore suggest to use the canonical Rust installation via a tool called rustup.

To install rustup and Rust, simply do:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Alternatively, visit the official Rust website for other installation methods.

You can update your Rust installation later by running:

```
rustup update
```

13.1.5 Building

The easiest way to get Routinator is to leave it to cargo by saying:

```
cargo install --locked routinator
```

If you want to try the master branch from the repository instead of a release version, you can run:

```
cargo install --git https://github.com/NLnetLabs/routinator.git
```

If you want to update an installed version, you run the same command but add the `-f` flag, a.k.a. force, to approve overwriting the installed version.

The command will build Routinator and install it in the same directory that cargo itself lives in, likely `~/.cargo/bin`. This means Routinator will be in your path, too.
13.1.6 Notes

In case you want to build a statically linked Routinator, or you have an Operating System where special care needs to be taken, such as OpenBSD and CentOS, please refer to the Installation Notes section.

13.2 Installation Notes

In certain scenarios and on some platforms specific steps are needed in order to get Routinator working as desired.

13.2.1 Statically Linked Routinator

While Rust binaries are mostly statically linked, they depend on libc which, as least as glibc that is standard on Linux systems, is somewhat difficult to link statically. This is why Routinator binaries are actually dynamically linked on glibc systems and can only be transferred between systems with the same glibc versions.

However, Rust can build binaries based on the alternative implementation named musl that can easily be statically linked. Building such binaries is easy with rustup. You need to install musl and the correct musl target such as x86_64-unknown-linux-musl for x86_64 Linux systems. Then you can just build Routinator for that target.

On a Debian (and presumably Ubuntu) system, enter the following:

```
sudo apt-get install musl-tools
rustup target add x86_64-unknown-linux-musl
cargo build --target=x86_64-unknown-linux-musl --release
```

13.2.2 Platform Specific Instructions

**Tip:** GÉANT has created an Ansible playbook defining a role to deploy Routinator on Ubuntu.

For some platforms, rustup cannot provide binary releases to install directly. The Rust Platform Support page lists several platforms where official binary releases are not available, but Rust is still guaranteed to build. For these platforms, automated tests are not run so it’s not guaranteed to produce a working build, but they often work to quite a good degree.

**OpenBSD**

On OpenBSD, patches are required to get Rust running correctly, but these are well maintained and offer the latest version of Rust quite quickly.

Rust can be installed on OpenBSD by running:

```
pkg_add rust
```

**CentOS 6**

The standard installation method does not work when using CentOS 6. Here, you will end up with a long list of error messages about missing assembler instructions. This is because the assembler shipped with CentOS 6 is too old.

You can get the necessary version by installing the Developer Toolset 6 from the Software Collections repository. On a virgin system, you can install Rust using these steps:
SELinux using CentOS 7

This guide, contributed by Rich Compton, describes how to run Routinator on Security Enhanced Linux (SELinux) using CentOS 7.

1. Start by setting the hostname.

```bash
sudo nmtui-hostname
Hostname will be set
```

2. Set the interface and connect it.

**Note:** Ensure that “Automatically connect” and “Available to all users” are checked.

```bash
sudo nmtui-edit
```

3. Install the required packages.

```bash
sudo yum check-update
sudo yum upgrade -y
sudo yum install -y epel-release
sudo yum install -y vim wget curl net-tools lsof bash-completion yum-utils \
  htop nginx httpd-tools tcpdump rust cargo rsync policycoreutils-python
```

4. Set the timezone to UTC.

```bash
sudo timedatectl set-timezone UTC
```

5. Remove postfix as it is unneeded.

```bash
sudo systemctl stop postfix
sudo systemctl disable postfix
```

6. Create a self-signed certificate for NGINX.

```bash
sudo mkdir /etc/ssl/private
sudo chmod 700 /etc/ssl/private
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \  -keyout /etc/ssl/private/nginx-selfsigned.key \  -out /etc/ssl/certs/nginx-selfsigned.crt
# Populate the relevant information to generate a self signed certificate
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

7. Add in the ssl.conf file to /etc/nginx/conf.d/ssl.conf and edit the ssl.conf file to provide the IP of the host in the server_name field.

8. Replace /etc/nginx/nginx.conf with the nginx.conf file.

9. Set the username and password for the web interface authentication.
The RPKI Documentation

```plaintext
sudo htpasswd -c /etc/nginx/.htpasswd <username>

10. Start Nginx and set it up so it starts at boot.
```n
```plaintext
sudo systemctl start nginx
sudo systemctl enable nginx
```

11. Add the user "routinator", create the /opt/routinator directory and assign it to the "routinator" user and group
```plaintext
sudo useradd routinator
sudo mkdir /opt/routinator
sudo chown routinator:routinator /opt/routinator
```

12. Sudo into the routinator user.
```plaintext
sudo su - routinator
```

13. Install Routinator and add it to the $PATH for user “routinator”
```plaintext
cargo install routinator
vi /home/routinator/.bash_profile
Edit the PATH line to include "/home/routinator/.cargo/bin"
PATH=$PATH:$HOME/.local/bin:$HOME/bin:/home/routinator/.cargo/bin
```

14. Initialise Routinator, accept the ARIN TAL and exit back to the user with sudo.
```plaintext
/home/routinator/.cargo/bin/routinator -b /opt/routinator init -f --accept-arin-rpa
exit
```

15. Create a routinator systemd script using the template below.
```plaintext
sudo vi /etc/systemd/system/routinator.service
[Unit]
Description=Routinator RPKI Validator and RTR Server
After=network.target
[Service]
Type=simple
User=routinator
Group=routinator
Restart=on-Failure
RestartSec=90
ExecStart=/home/routinator/.cargo/bin/routinator -v -b /opt/routinator server
   --http 127.0.0.1:8080 --rtr <IPv4 IP>:8323 --rtr [<IPv6 IP>]:8323
   TimeoutStartSec=0
[Install]
WantedBy=default.target
```

**Note:** You must populate the IPv4 and IPv6 addresses. In addition, the IPv6 address needs to have brackets ‘[ ]’ around it. For example:
```plaintext
/home/routinator/.cargo/bin/routinator -v -b /opt/routinator server
   --http 127.0.0.1:8080 --rtr 172.16.47.235:8323 --rtr [2001:db8::43]:8323
```

16. Configure SELinux to allow connections to localhost and to allow rsync to write to the /opt/routinator directory.
17. Reload the systemd daemon and set the routinator service to start at boot.

```
sudo systemctl daemon-reload
sudo systemctl enable routinator.service
sudo systemctl start routinator.service
```

18. Set up the firewall to permit ssh, HTTPS and port 8323 for the RTR protocol.

```
sudo firewall-cmd --permanent --remove-service=ssh --zone=public
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv4" source address="<IPv4 management subnet>" service name=ssh accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv6" source address="<IPv6 management subnet>" service name=ssh accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv4" source address="<peering router IPv4 loopback subnet>" port port=8323 protocol=tcp accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv6" source address="<peering router IPv6 loopback subnet>" port port=8323 protocol=tcp accept'
```

```
sudo firewall-cmd --reload
```

19. Navigate to https://<IP address of rpki-validator>/metrics to see if it's working. You should authenticate with the username and password that you provided in step 10 of setting up the RPKI Validation Server.

### 13.3 Initialisation

Before running Routinator for the first time, you must prepare its working environment. You do this using the *init* command. This will prepare both the directory for the local RPKI cache, as well as the Trust Anchor Locator (TAL) directory.

By default, both directories will be located under `$HOME/.rpki-cache`, but you can change their locations via the command line options `--repository-dir` and `--tal-dir`.

TALs provide hints for the trust anchor certificates to be used both to discover and validate all RPKI content. The five TALs — one for each Regional Internet Registry (RIR) — are bundled with Routinator and installed by the *init* command.

**Warning:** Using the TAL from ARIN, the RIR for the United States, Canada as well as many Caribbean and North Atlantic islands, requires you to read and accept their Relying Party Agreement before you can use it. Running the *init* command will provide you with instructions.

```
routinator init
```

Before we can install the ARIN TAL, you must have read and agree to the ARIN Relying Party Agreement (RPA). It is available at [ARIN Relying Party Agreement](https://www.arin.net/legal/rpa).
If you agree to the RPA, please run the command again with the --accept-arin-rpa option.

```
routinator init --accept-arin-rpa
```

If you decide you cannot agree to the ARIN RPA terms, the --decline-arin-rpa option will install all TALs except the one for ARIN. If, at a later point, you wish to use the ARIN TAL anyway, you can add it to your current installation using the --force option, to force the installation of all TALs.

### 13.3.1 Performing a Test Run

To see if Routinator has been initialised correctly and your firewall allows the required connections, it is recommended to perform an initial test run. You can do this by having Routinator print a validated ROA payload (VRP) list with the `vrps` subcommand, and using `-v` to increase the log level to INFO to see if Routinator establishes rsync and RRDP connections as expected.

```
routinator -v vrps
```

Now, you can see how Routinator connects to the RPKI trust anchors, downloads the contents of the repositories to your machine, validates it and produces a list of validated ROA payloads in the default CSV format to standard output. From a cold start, this process will take a couple of minutes.

```
routinator -v vrps
rsyncing from rsync://repository.lacnic.net/rpki/.
rsyncing from rsync://rpki.afrinic.net/repository/.
rsyncing from rsync://rpki.apnic.net/repository/.
rsyncing from rsync://rpki.ripe.net/ta/.
rsync://rpki.ripe.net/ta: The RIPE NCC Certification Repository is subject to Terms and Conditions
rsync://rpki.ripe.net/ta: See http://www.ripe.net/lir-services/ncc/legal/certification/repository-tc
rsyncing from rsync://rpki.ripe.net/repository/.
Found valid trust anchor rsync://rpki.afrinic.net/repository/AfriNIC.cer. Processing.
Found valid trust anchor rsync://rpki.arin.net/repository/.
Found valid trust anchor rsync://rpki.arin.net/repository/arin-rpki-ta.cer.
rsyncing from rsync://rpki.apnic.net/member_repository/.
Found valid trust anchor rsync://repository.lacnic.net/rpki/lacnic/rta-lacnic-rpki.cer. Processing.
rsync://rpki.ripe.net/repository: The RIPE NCC Certification Repository is subject to Terms and Conditions
rsync://rpki.ripe.net/repository: See http://www.ripe.net/lir-services/ncc/legal/certification/repository-tc
```
rsyncing from rsync://rpkica.twnic.tw/rpki/.
rsyncing from rsync://rpki-repository.nic.ad.jp/ap/.
rsyncing from rsync://rpki.cnnic.cn/rpki/.

Summary:
afrinic: 338 valid ROAs, 459 VRPs.
lacnic: 2435 valid ROAs, 7042 VRPs.
apnic: 3186 valid ROAs, 21934 VRPs.
ripe: 10780 valid ROAs, 56907 VRPs.
arin: 4964 valid ROAs, 6621 VRPs.
ASN,IP Prefix,Max Length,Trust Anchor
AS43289,2a03:f80:373::/48,48,ripe
AS14464,131.109.128.0/17,17,arin
AS17806,114.130.5.0/24,24,apnic
AS59587,151.232.192.0/21,21,ripe
AS13335,172.68.30.0/24,24,arin
AS6147,190.40.0.0/14,24,lacnic
...

13.4 Running Interactively

Routinator can perform RPKI validation as a one-time operation and print a Validated ROA Payload (VRP) list in various formats, or it can return the validity of a specific announcement. These functions are accessible on the command line via the following sub-commands:

vrps Fetched RPKI data and produces a Validated ROA Payload (VRP) list in the specified format.

validate Outputs the RPKI validity for a specific announcement by supplying Routinator with an ASN and a prefix.

13.4.1 Printing a List of VRPs

Routinator can produce a Validated ROA Payload (VRP) list in five different formats, which are either printed to standard output or saved to a file:

csv The list is formatted as lines of comma-separated values of the prefix in slash notation, the maximum prefix length, the autonomous system number, and an abbreviation for the trust anchor the entry is derived from. The latter is the name of the TAL file without the extension .tal. This is the default format used if the --format or -f option is missing.

csvcompat The same as csv except that all fields are embedded in double quotes and the autonomous system number is given without the prefix AS. This format is pretty much identical to the CSV produced by the RIPE NCC Validator.

csvext This is an extended version of the csv format, which was used by the RIPE NCC RPKI Validator 1.x. Each line contains these comma-separated values: the rsync URI of the ROA the line is taken from (or “N/A” if it isn’t from a ROA), the autonomous system number, the prefix in slash notation, the maximum prefix length, and lastly the not-before and not-after date of the validity of the ROA.

json The list is placed into a JSON object with a single element roas which contains an array of objects with four elements each: The autonomous system number of the network authorised to originate a prefix in asn, the prefix in slash notation in prefix, the maximum prefix length of the announced route in maxLength, and the trust anchor from which the authorisation was derived in ta. This format is identical to that produced by the RIPE NCC Validator except for different naming of the trust anchor. Routinator uses the name of the TAL file without the extension .tal whereas the RIPE NCC Validator has a dedicated name for each.
openbgpd  Choosing this format causes Routinator to produce a roa-set configuration item for the OpenBGPD configuration.

bird   Choosing this format causes Routinator to produce a roa table configuration item for the BIRD configuration.

bird2  Choosing this format causes Routinator to produce a route table configuration item for the BIRD2 configuration.

rpsl   This format produces a list of RPSL objects with the authorisation in the fields route, origin, and source. In addition, the fields descr, mnt-by, created, and last-modified, are present with more or less meaningful values.

summary This format produces a summary of the content of the RPKI repository. For each trust anchor, it will print the number of verified ROAs and VRPs. Note that this format does not take filters into account. It will always provide numbers for the complete repository.

For example, to get the validated ROA payloads in CSV format, run:

```
routinator vrps --format csv
```

```
ASN,IP Prefix,Max Length,Trust Anchor
AS55803,103.14.64.0/23,23,apnic
AS267868,45.176.192.0/24,24,lacnic
AS41152,82.115.18.0/23,23,ripe
AS28920,185.103.228.0/22,22,ripe
AS11845,209.203.0.0/18,24,afrinic
AS63297,23.179.0.0/24,24,arin
...
```

To generate a file with with the validated ROA payloads in JSON format, run:

```
routinator vrps --format json --output authorisedroutes.json
```

```
{ "roas": [ { "asn": "AS12654", "prefix": "93.175.146.0/24", "maxLength": 24, "ta": "ripe" } ] }
```

Filtering

In case you are looking for specific information in the output, Routinator allows filtering to see if a prefix or ASN is covered or matched by a VRP. You can do this using the --filter-asn and --filter-prefix options.

When using --filter-asn, you can use both AS64511 and 64511 as the notation. With --filter-prefix, the result will include VRPs regardless of their ASN and MaxLength. Both filter flags can be combined and used multiple times in a single query and will be treated as a logical "or".

A validation run will be started before returning the result, making sure you get the latest information. If you would like a result from the current cache, you can use the --noupdate or -n option.

Here are some examples filtering for an ASN and prefix in CSV and JSON format:

```
routinator vrps --format csv --filter-asn 196615
ASN,IP Prefix,Max Length,Trust Anchor
AS196615,2001:7fb:fd03::/48,48,ripe
AS196615,93.175.147.0/24,24,ripe
```

```
routinator vrps --format json --filter-prefix 93.175.146.0/24
```

```
{ "roas": [ { "asn": "AS12654", "prefix": "93.175.146.0/24", "maxLength": 24, "ta": "ripe" } ] }
```

```
routinator vrps --format json --filter-prefix 93.175.146.0/24
```

```
{ "roas": [ { "asn": "AS12654", "prefix": "93.175.146.0/24", "maxLength": 24, "ta": "ripe" } ] }
```
### 13.4.2 Validity Checker

You can check the RPKI origin validation status of a specific BGP announcement using the `validate` subcommand and by supplying the ASN and prefix. A validation run will be started before returning the result, making sure you get the latest information. If you would like a result from the current cache, you can use the `--noupdate` or `-n` option.

```
routinator validate --asn 12654 --prefix 93.175.147.0/24
```

Invalid

A detailed analysis of the reasoning behind the validation outcome is printed in JSON format. In case of an Invalid state, whether this because the announcement is originated by an unauthorised AS, or if the prefix is more specific than the maximum prefix length allows. Lastly, a complete list of VRPs that caused the result is included.

```
routinator validate --json --asn 12654 --prefix 93.175.147.0/24
```

```json
{
   "validated_route": {
      "route": {
         "origin_asn": "AS12654",
         "prefix": "93.175.147.0/24"
      },
      "validity": {
         "state": "Invalid",
         "reason": "as",
         "description": "At least one VRP Covers the Route Prefix, but no VRP ASN matches the route origin ASN",
         "VRPs": {
            "matched": [],
            "unmatched_as": [
               {
                  "asn": "AS196615",
                  "prefix": "93.175.147.0/24",
                  "max_length": "24"
               }
            ],
            "unmatched_length": []
         }
      }
   }
}
```

If you run the HTTP service in daemon mode, this information is also available at the `/validity` endpoint.

### 13.5 Running as a Daemon

Routinator can run as a service that periodically fetches RPKI data, verifies it and makes the resulting data set available via the RPKI-RTR protocol and through the built-in HTTP server. You can start the Routinator service using the `server` subcommand.

#### 13.5.1 The HTTP Service

The CSV, JSON, OpenBGPD and RPSL formats that Routinator can produce in interactive mode are available via HTTP if the application is running as a service. You can also check the RPKI origin validation status of a specific
The RPKI Documentation

BGP announcement at the /validity endpoint by supplying the ASN and prefix.

The HTTP server is not enabled by default for security reasons, nor does it have a default host or port. In order to start the HTTP server at 192.0.2.13 and 2001:0DB8::13 on port 8323, run this command:

```
routinator server --http 192.0.2.13:8323 --http [2001:0DB8::13]:8323
```

The application will stay attached to your terminal unless you provide the --detach option. After fetching and validating the data set, the following paths are available:

- /csv Returns the current set of VRPs in csv output format
- /csvext Returns the current set of VRPs in csvext output format
- /json Returns the current set of VRPs in json output format
- /openbgpd Returns the current set of VRPs in OpenBGPD output format
- /bird Returns the current set of VRPs in bird output format
- /bird2 Returns the current set of VRPs in bird2 output format
- /rpsl Returns the current set of VRPs in RPSL output format
- /validity Returns the RPKI origin validation status of a specific BGP announcement by supplying the ASN and prefix in the path, e.g. /validity?asn=12654&prefix=93.175.147.0/24

Please note that this server is intended to run on your internal network and doesn’t offer HTTPS natively. If this is a requirement, you can for example run Routinator behind an NGINX reverse proxy.

Lastly, the HTTP server provides paths that allow you to monitor Routinator itself and the data it processes, so it may be desirable to have HTTP running alongside the RTR server. For more information, please refer to the Monitoring section.

13.5.2 The RTR Service

Routinator supports RPKI-RTR as specified in RFC 8210 as well as the older version described in RFC 6810.

When launched as an RTR server, routers with support for route origin validation (ROV) can connect to Routinator to fetch the processed data. This includes hardware routers such as Juniper, Cisco and Nokia, as well as software solutions like BIRD, GoBGP and others. The processed data is also available in a number of useful output formats, such as CSV, JSON, RPSL and a format specifically for OpenBGPD.

Like the HTTP server, the RTR server is not started by default, nor does it have a default host or port. Thus, in order to start the RTR server at 192.0.2.13 and 2001:0DB8::13 on port 3323, run Routinator using the server command:

```
routinator server --rtr 192.0.2.13:3323 --rtr [2001:0DB8::13]:3323
```

Please note that port 3323 is not the IANA-assigned default port for the protocol, which would be 323. But as this is a privileged port, you would need to be running Routinator as root when otherwise there is no reason to do that. The application will stay attached to your terminal unless you provide the --detach option.

By default, the repository will be updated and re-validated every 10 minutes. You can change this via the --refresh option and specify the interval between re-validations in seconds. That is, if you rather have Routinator validate every 15 minutes, the above command becomes:

```
routinator server --rtr 192.0.2.13:3323 --rtr [2001:0DB8::13]:3323 --refresh=900
```

Communication between Routinator and the router using the RPKI-RTR protocol is done via plain TCP. Below, there is an explanation how to secure the transport using either SSH or TLS.
Secure Transports

These instructions were contributed by wk on Github.

RFC 6810#section-7 defines a number of secure transports for RPKI-RTR that can be used to secure communication between a router and a RPKI relying party.

However, the RPKI Router Implementation Report documented in RFC 7128#section-5 suggests these secure transports have not been widely implemented. Implementations, however, do exist, and a secure transport could be valuable in situations where the RPKI relying party is provided as a public service, or across a non-trusted network.

SSH Transport

SSH transport for RPKI-RTR can be configured with the help of netcat and OpenSSH.

1. Begin by installing the openssh-server and netcat packages.

Make sure Routinator is running as an RTR server on localhost:

```
routinator server --rtr 127.0.0.1:3323
```

2. Create a username and a password for the router to log into the host with, such as rpki.

3. Configure OpenSSH to expose an rpki-rtr subsystem that acts as a proxy into Routinator by editing the /etc/ssh/sshd_config file or equivalent to include the following line:

```
Subsystem rpki-rtr /bin/nc 127.0.0.1:3323
```

4. Restart the OpenSSH server daemon.

5. Set up the router running IOS-XR using this example configuration:

```
router bgp 65534
rpki server 192.168.0.100
username rpki
password rpki
transport ssh port 22
```

TLS Transport

TLS transport for RPKI-RTR can be configured with the help of stunnel.

1. Begin by installing the stunnel package.

2. Make sure Routinator is running as an RTR server on localhost:

```
routinator server --rtr 127.0.0.1:3323
```
3. Acquire (via for example Let’s Encrypt) or generate an SSL certificate. In the example below, an SSL certificate for the domain example.com generated by Let’s Encrypt is used.

4. Create an stunnel configuration file by editing `/etc/stunnel/rpki.conf` or equivalent:

   ```
   [rpki]
   ; Use a letsencrypt certificate for example.com
   cert = /etc/letsencrypt/live/example.com/fullchain.pem
   key = /etc/letsencrypt/live/example.com/privkey.pem
   ; Listen for TLS rpki-rtr on port 323 and proxy to port 3323 on localhost
   accept = 323
   connect = 127.0.0.1:3323
   ```

5. Restart `stunnel` to complete the process.

### 13.6 Configuration

Routinator has a number of default settings, such as the location where files are stored, the refresh interval and the log level. You can view these settings by running:

```
routinator config
```

It will return the list of defaults in the same notation that is used by the optional configuration file, which will be largely similar to this:

```
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
exceptions = []
expire = 7200
history-size = 10
http-listen = []
log = "default"
log-level = "WARN"
refresh = 600
repository-dir = "/Users/routinator/.rpki-cache/repository"
retry = 600
rrdp-proxies = []
rrdp-root-certs = []
rsync-command = "rsync"
rsync-timeout = 300
rtr-listen = []
stale = "warn"
strict = false
syslog-facility = "daemon"
systemd-listen = false
tal-dir = "/Users/routinator/.rpki-cache/tals"
validation-threads = 4
```

You can override these defaults, as well as configure a great number of additional options using either command line arguments or via the configuration file.

To get an overview of all available options, please refer to the `configuration file` section of the Manual Page, which can be also viewed by running `routinator man`.  

---

**The RPKI Documentation**

---
13.6.1 Using a Configuration File

Routinator can take its configuration from a file. You can specify such a config file via the \texttt{-c} option. If you don’t, Routinator will check if there is a \texttt{$HOME/.routinator.conf} and if it exists, use it. If it doesn’t exist and there is no \texttt{-c} option, the default values are used.

For specifying configuration options, Routinator uses a \texttt{ TOML} file. Its entries are named similarly to the command line options. A complete sample configuration file showing all the default values can be found in the repository at \texttt{etc/routinator.conf.example}.

\textbf{Note:} One scenario where a configuration file is required is when you would like to pass certain rsync arguments using the \texttt{rsync-args} setting, which is not available as a command line option. This is because Routinator will try to find out if your rsync understands the \texttt{--contimeout} option and set it if possible.

For example, if you want Routinator to refresh every 15 minutes and run as an RTR server on 192.0.2.13 and 2001:0DB8::13 on port 3323, in addition to providing an HTTP server on port 9556, simply take the output from \texttt{routinator config} and change the \texttt{refresh}, \texttt{rtr-listen} and \texttt{http-listen} values in your favourite text editor:

\begin{verbatim}
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
exceptions = []
expire = 7200
history-size = 10
http-listen = ["192.0.2.13:9556", 
[2001:0DB8::13]:9556"]
log = "default"
log-level = "WARN"  
refresh = 900
repository-dir = "/Users/routinator/.rpki-cache/repository"
retry = 600
rrdp-proxies = []
rrdp-root-certs = []
rsync-command = "rsync"
rsync-timeout = 300
rtr-listen = ["192.0.2.13:3323", 
[2001:0DB8::13]:3323"]
stale = "warn"
strict = false
syslog-facility = "daemon"
systemd-listen = false
tal-dir = "/Users/routinator/.rpki-cache/tals"
validation-threads = 4
\end{verbatim}

After saving this file as \texttt{.routinator.conf} in your home directory, you can start Routinator with:

\begin{verbatim}
routinator server
\end{verbatim}

13.6.2 Applying Local Exceptions

In some cases, you may want to override the global RPKI data set with your own local exceptions. For example, when a legitimate route announcement is inadvertently flagged as \textit{invalid} due to a misconfigured ROA, you may want to temporarily accept it to give the operators an opportunity to resolve the issue.
You can do this by specifying route origins that should be filtered out of the output, as well as origins that should be added, in a file using JSON notation according to the SLURM standard specified in RFC 8416.

A full example file is provided below. This, along with an empty one is available in the repository at /test/slurm.

```json
{
    "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>"
            }
        ]
    }
}
```

(continues on next page)
Use the `-x` option to refer to your file with local exceptions. Routinator will re-read that file on every validation run, so you can simply update the file whenever your exceptions change.

### 13.7 Monitoring

The HTTP server in Routinator provides endpoints for monitoring the application. This means it may be a good idea to run the HTTP server alongside the RTR server.

To launch Routinator in server mode on 192.0.2.13 with RTR running on port 3323 and HTTP on 9556, use the following command:

```
routinator server --rtr 192.0.2.13:3323 --http 192.0.2.13:9556
```

The HTTP service has three monitoring endpoints on the following paths:

- `/version` Returns the version of the Routinator instance
- `/metrics` Exposes a data format specifically for Prometheus, for which dedicated port 9556 is reserved.
- `/status` Returns the information from the `/metrics` endpoint in a more concise format

#### 13.7.1 Metrics

**Update metrics**

- When the last update started and finished
- The total duration of the last update
- The retrieval duration and exit code for each rsync publication point
- The retrieval duration and HTTP status code for each RRDP publication point

**Object metrics**

- The number of valid ROAs per Trust Anchor
- The number of Validated ROA Payloads (VRPs) per Trust Anchor
- The number of stale objects found

**RTR server**

- The current RTR serial number
- The current and total number of RTR connections
- The total amount of bytes sent and received over the RTR connection

**HTTP server**

- The current and total number of HTTP connections
- The total amount of bytes sent and received over the HTTP connection
• The number of HTTP requests

13.7.2 Grafana

Using the Prometheus endpoint it’s possible to build a detailed dashboard using for example Grafana. We provide a template to get started.

![A sample Grafana dashboard](image)

Fig. 13.1: A sample Grafana dashboard

13.8 Manual Page

**routinator** - RPKI relying party software

**Date** 2020-06-15

**Author** Martin Hoffmann

**Copyright** 2019-2020 - NLnet Labs

**Version** 0.7.1

13.8.1 Synopsis

13.8.2 Description

Routinator collects and processes Resource Public Key Infrastructure (RPKI) data. It validates the Route Origin Attestations contained in the data and makes them available to your BGP routing workflow.
It can either run in one-shot mode outputting a list of validated route origins in various formats or as a server for the RPKI-to-Router (RTR) protocol that routers often implement to access the data, or via HTTP.

These modes and additional operations can be chosen via commands. For the available commands, see Commands below.

### 13.8.3 Options

The available options are:

- `-c path, --config=path`
  
  Provides the path to a file containing basic configuration. If this option is not given, Routinator will try to use `~/.routinator.conf` if that exists. If that doesn’t exist, either, default values for the options as described here are used. See Configuration File below for more information on the format and contents of the configuration file.

- `-b dir, --base-dir=dir`
  
  Specifies the base directory to keep status information in. Unless overwritten by the `-r` or `-t` options, the local repository will be kept in the sub-directory `repository` and the TALs will be kept in the sub-directory `tals`.

  If omitted, the base directory defaults to `~/.rpki-cache`.

- `-r dir, --repository-dir=dir`
  
  Specifies the directory to keep the local repository in. This is the place where Routinator stores the RPKI data it has collected and thus is a copy of all the data referenced via the trust anchors.

- `-t dir, --tal-dir=dir`
  
  Specifies the directory containing the trust anchor locators (TALs) to use. Trust anchor locators are the starting points for collecting and validating RPKI data. See Trust Anchor Locators for more information on what should be present in this directory.

- `-x file, --exceptions=file`
  
  Provides the path to a local exceptions file. The option can be used multiple times to specify more than one file to use. Each file is a JSON file as described in RFC 8416. It lists both route origins that should be filtered out of the output as well as origins that should be added.

- `--strict`
  
  If this option is present, the repository will be validated in strict mode following the requirements laid out by the standard documents very closely. With the current RPKI repository, using this option will lead to a rather large amount of invalid route origins and should therefore not be used in practice. See Relaxed Validation below for more information.

- `--stale=policy`
  
  This option defines how deal with stale objects. In RPKI, manifests and CRLs can be stale if the time given in their `next-update` field is in the past, indicating that an update to the object was scheduled but didn’t happen. This can be because of an operational issue at the issuer or an attacker trying to replay old objects.

  There are three possible policies that define how Routinator should treat stale objects.

  - A policy of `reject` instructs Routinator to consider all stale objects invalid. This will result in all material published by the CA issuing this manifest and CRL to be invalid including all material of any child CA.

  - The `warn` policy will allow Routinator to consider any stale object to be valid. It will, however, print a warning in the log allowing an operator to follow up on the issue. This is the default policy if the option is not provided.

  - Finally, the `accept` policy will cause Routinator to quietly accept any stale object as valid.

- `--allow-dubious-hosts`
  
  As a precaution, Routinator will reject rsync and HTTPS URIs from RPKI data with dubious host names. In
particular, it will reject the name localhost, host names that consist of IP addresses, and a host name that contains an explicit port.

This option allows to disable this filtering.

--disable-rsync
If this option is present, rsync is disabled and only RRDP will be used.

--rsync-command=command
Provides the command to run for rsync. This is only the command itself. If you need to provide options to rsync, use the rsync-args configuration file setting instead.

If this option is not given, Routinator will simply run rsync and hope that it is in the path.

--rsync-timeout=seconds
Sets the number of seconds an rsync command is allowed to run before it is terminated early. This protects against hanging rsync commands that prevent Routinator from continuing. The default is 300 seconds which should be long enough except for very slow networks.

--disable-rrdp
If this option is present, RRDP is disabled and only rsync will be used.

--rrdp-timeout=seconds
Sets the timeout in seconds for any RRDP-related network operation, i.e., connects, reads, and writes. If this option is omitted, the default timeout of 30 seconds is used. Set the option to 0 to disable the timeout.

--rrdp-connect-timeout=seconds
Sets the timeout in seconds for RRDP connect requests. If omitted, the general timeout will be used.

--rrdp-local-addr=addr
If present, sets the local address that the RRDP client should bind to when doing outgoing requests.

--rrdp-root-cert=path
This option provides a path to a file that contains a certificate in PEM encoding that should be used as a trusted certificate for HTTPS server authentication. The option can be given more than once.

Providing this option does not disable the set of regular HTTPS authentication trust certificates.

--rrdp-proxy=uri
This option provides the URI of a proxy to use for all HTTP connections made by the RRDP client. It can be either an HTTP or a SOCKS URI. The option can be given multiple times in which case proxies are tried in the given order.

--dirty
If this option is present, unused files and directories will not be deleted from the repository directory after each validation run.

--validation-threads=count
Sets the number of threads to distribute work to for validation. Note that the current processing model validates trust anchors all in one go, so you are likely to see less than that number of threads used throughout the validation run.

-v, --verbose
Print more information. If given twice, even more information is printed.

More specifically, a single --v increases the log level from the default of warn to info, specifying it more than once increases it to debug.

-q, --quiet
Print less information. Given twice, print nothing at all.

A single --q will drop the log level to error. Repeating --q more than once turns logging off completely.
The RPKI Documentation

--syslog
Redirect logging output to syslog.

This option is implied if a command is used that causes Routinator to run in daemon mode.

--syslog-facility=facility
If logging to syslog is used, this option can be used to specify the syslog facility to use. The default is daemon.

--logfile=path
Redirect logging output to the given file.

-h, --help
Print some help information.

-V, --version
Print version information.

13.8.4 Commands

Routinator provides a number of operations around the local RPKI repository. These can be requested by providing different commands on the command line.

init
Prepares the local repository directories and the TAL directory for running Routinator. Specifically, makes sure the local repository directory exists, and creates the TAL directory and fills it with the TALs of the five RIRs.

For more information about TALs, see Trust Anchor Locators below.

-f, --force
Forces installation of the TALs even if the TAL directory already exists.

--accept-arin-rpa
Before you can use the ARIN TAL, you need to agree to the ARIN Relying Party Agreement (RPA). You can find it at https://www.arin.net/resources/manage/rpki/rpa.pdf and explicitly agree to it via this option. This explicit agreement is necessary in order to install the ARIN TAL.

--decline-arin-rpa
If, after reading the ARIN Relying Party Agreement, you decide you do not or cannot agree to it, this option allows you to skip installation of the ARIN TAL. Note that this means Routinator will not have access to any information published for resources assigned under ARIN.

vrps
This command requests that Routinator update the local repository and then validate the Route Origin Attestations in the repository and output the valid route origins, which are also known as Validated ROA Payload or VRPs, as a list.

-o file, --output=file
Specifies the output file to write the list to. If this option is missing or file is - the list is printed to standard output.

-f format, --format=format
The output format to use. Routinator currently supports the following formats:

csv The list is formatted as lines of comma-separated values of the prefix in slash notation, the maximum prefix length, the autonomous system number, and an abbreviation for the trust anchor the entry is derived from. The latter is the name of the TAL file without the extension .tal.

This is the default format used if the -f option is missing.
csvcompat The same as csv except that all fields are embedded in double quotes and the autonomous system number is given without the prefix AS. This format is pretty much identical to the CSV produced by the RIPE NCC Validator.

csvext An extended version of csv each line contains these comma-separated values: the rsync URI of the ROA the line is taken from (or “N/A” if it isn’t from a ROA), the autonomous system number, the prefix in slash notation, the maximum prefix length, the not-before date and not-after date of the validity of the ROA.

This format was used in the RIPE NCC RPKI Validator version 1. That version produces one file per trust anchor. This is not currently supported by Routinator – all entries will be in one single output file.

json The list is placed into a JSON object with a single element roas which contains an array of objects with four elements each: The autonomous system number of the network authorized to originate a prefix in asn, the prefix in slash notation in prefix, the maximum prefix length of the announced route in maxLength, and the trust anchor from which the authorization was derived in ta. This format is identical to that produced by the RIPE NCC RPKI Validator except for different naming of the trust anchor. Routinator uses the name of the TAL file without the extension .tal whereas the RIPE NCC Validator has a dedicated name for each.

openbgpd Choosing this format causes Routinator to produce a roa- set configuration item for the OpenBGPD configuration.

bird Choosing this format causes Routinator to produce a roa table configuration item for the BIRD configuration.

bird2 Choosing this format causes Routinator to produce a roa table configuration item for the BIRD2 configuration.

rpsl This format produces a list of RPSL objects with the authorization in the fields route, origin, and source. In addition, the fields descr, mnt-by, created, and last-modified, are present with more or less meaningful values.

summary This format produces a summary of the content of the RPKI repository. For each trust anchor, it will print the number of verified ROAs and VRPs. Note that this format does not take filters into account. It will always provide numbers for the complete repository.

none This format produces no output whatsoever.

-n, --noupdate The repository will not be updated before producing the list.

--complete If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.

-a asn, --filter-asn=asn Only output VRPs for the given ASN. The option can be given multiple times, in which case VRPs for all provided ASNs are provided. ASNs can be given with or without the prefix AS.

-p prefix, --filter-prefix=prefix Only output VRPs with an address prefix that covers the given prefix, i.e., whose prefix is equal to or less specific than the given prefix. This will include VRPs regardless of their ASN and max length. In other words, the output will include all VRPs that need to be considered when deciding whether an announcement for the prefix is RPKI valid or invalid.

The option can be given multiple times, in which case VRPs for all prefixes are provided. It can also be combined with one or more ASN filters. Then all matching VRPs are included. That is, filters combine as “or” not “and.”
**validate**

This command can be used to perform RPKI route origin validation for a route announcement. Routinator will determine whether the provided announcement is RPKI valid, invalid, or not found.

```bash
-a asn, --asn=asn
```

The AS number of the autonomous system that originated the route announcement. ASNs can be given with or without the prefix AS.

```bash
-p prefix, --prefix=prefix
```

The address prefix the route announcement is for.

```bash
-j, --json
```

A detailed analysis on the reasoning behind the validation is printed in JSON format including lists of the VPRs that caused the particular result. If this option is omitted, Routinator will only print the determined state.

```bash
-n, --noupdate
```

The repository will not be updated before performing validation.

```bash
--complete
```

If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.

**server**

This command causes Routinator to act as a server for the RPKI-to-Router (RTR) and HTTP protocols. In this mode, Routinator will read all the TALs (See Trust Anchor Locators below) and will stay attached to the terminal unless the `-d` option is given.

The server will periodically update the local repository, every ten minutes by default, notify any clients of changes, and let them fetch validated data. It will not, however, reread the trust anchor locators. Thus, if you update them, you will have to restart Routinator.

You can provide a number of addresses and ports to listen on for RTR and HTTP through command line options or their configuration file equivalent. Currently, Routinator will only start listening on these ports after an initial validation run has finished.

It will not listen on any sockets unless explicitly specified. It will still run and periodically update the repository. This might be useful for use with `vrps` mode with the `-n` option.

```bash
-d, --detach
```

If present, Routinator will detach from the terminal after a successful start.

```bash
--rtr=addr:port
```

Specifies a local address and port to listen on for incoming RTR connections.

Routinator supports both protocol version 0 defined in RFC 6810 and version 1 defined in RFC 8210. However, it does not support router keys introduced in version 1. IPv6 addresses must be enclosed in square brackets. You can provide the option multiple times to let Routinator listen on multiple address-port pairs.

```bash
--http=addr:port
```

Specifies the address and port to listen on for incoming HTTP connections. See HTTP Service below for more information on the HTTP service provided by Routinator.

```bash
--listen-systemd
```

The RTR listening socket will be acquired from systemd via socket activation. Use this option together with systemd’s socket units to allow a Routinator running as a regular user to bind to the default RTR port 323.

Currently, all TCP listener sockets handed over by systemd will be used for the RTR protocol.
**--refresh=seconds**
The amount of seconds the server should wait after having finished updating and validating the local repository before starting to update again. The next update will be earlier if objects in the repository expire earlier. The default value is 600 seconds.

**--retry=seconds**
The amount of seconds to suggest to an RTR client to wait before trying to request data again if that failed. The default value is 600 seconds, as recommended in RFC 8210.

**--expire=seconds**
The amount of seconds an RTR client can keep using data if it cannot refresh it. After that time, the client should discard the data. Note that this value was introduced in version 1 of the RTR protocol and is thus not relevant for clients that only implement version 0. The default value, as recommended in RFC 8210, is 7200 seconds.

**--history=count**
In RTR, a client can request to only receive the changes that happened since the last version of the data it had seen. This option sets how many change sets the server will at most keep. If a client requests changes from an older version, it will get the current full set.

Note that routers typically stay connected with their RTR server and therefore really only ever need one single change set. Additionally, if RTR server or router are restarted, they will have a new session with new change sets and need to exchange a full data set, too. Thus, increasing the value probably only ever increases memory consumption.

The default value is 10.

**--pid-file=path**
States a file which will be used in daemon mode to store the processes PID. While the process is running, it will keep the file locked.

**--working-dir=path**
The working directory for the daemon process. In daemon mode, Routinator will change to this directory while detaching from the terminal.

**--chroot=path**
The root directory for the daemon process. If this option is provided, the daemon process will change its root directory to the given directory. This will only work if all other paths provided via the configuration or command line options are under this directory.

**--user=user-name**
The name of the user to change to for the daemon process. If this option is provided, Routinator will run as that user after the listening sockets for HTTP and RTR have been created. The option has no effect unless --detach is also used.

**--group=group-name**
The name of the group to change to for the daemon process. If this option is provided, Routinator will run as that group after the listening sockets for HTTP and RTR have been created. The option has no effect unless --detach is also used.

**update**
Updates the local repository by resyncing all known publication points. The command will also validate the updated repository to discover any new publication points that appear in the repository and fetch their data.

As such, the command really is a shortcut for running `routinator vrps -f none`.

**--complete**
If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.
The RPKI Documentation

man
Displays the manual page, i.e., this page.

-o file, --output=file
If this option is provided, the manual page will be written to the given file instead of displaying it. Use -
to output the manual page to standard output.

13.8.5 Trust Anchor Locators

RPKI uses trust anchor locators, or TALs, to identify the location and public keys of the trusted root CA certificates. Routinator keeps these TALs in files in the TAL directory which can be set by the -t option. If the -b option is used instead, the TAL directory will be in the subdirectory tals under the directory specified in this option. The default location, if no options are used at all is $HOME/.rpki-cache/tals.

This directory can be created and populated with the TALs of the five Regional Internet Registries (RIRs) via the init command.

If the directory does exist, Routinator will use all files with an extension of .tal in this directory. This means that you can add and remove trust anchors by adding and removing files in this directory. If you add files, make sure they are in the format described by RFC 7730 or the upcoming RFC 8630.

13.8.6 Configuration File

Instead of providing all options on the command line, they can also be provided through a configuration file. Such a file can be selected through the -c option. If no configuration file is specified this way but a file named $HOME/.routinator.conf is present, this file is used.

The configuration file is a file in TOML format. In short, it consists of a sequence of key-value pairs, each on its own line. Strings are to be enclosed in double quotes. Lists can be given by enclosing a comma-separated list of values in square brackets.

The configuration file can contain the following entries. All path values are interpreted relative to the directory the configuration file is located in. All values can be overridden via the command line options.

repository-dir A string containing the path to the directory to store the local repository in. This entry is mandatory.
tal-dir A string containing the path to the directory that contains the Trust Anchor Locators. This entry is mandatory.
exceptions A list of strings, each containing the path to a file with local exceptions. If missing, no local exception files are used.
strict A boolean specifying whether strict validation should be employed. If missing, strict validation will not be used.
stale A string specifying the policy for dealing with stale objects.
    reject Consider all stale objects invalid rendering all material published by the CA issuing the stale object to be invalid including all material of any child CA.
    warn Consider stale objects to be valid but print a warning to the log.
    accept Quietly consider stale objects valid.
allow-dubious-hosts A boolean value that, if present and true, disables Routinator’s filtering of dubious host names in rsync and HTTPS URIs from RPKI data.
disable-rsync A boolean value that, if present and true, turns off the use of rsync.
rsync-command A string specifying the command to use for running rsync. The default is simply rsync.
rsync-args A list of strings containing the arguments to be passed to the rsync command. Each string is an argument of its own.

If this option is not provided, Routinator will try to find out if your rsync understands the \-contimeout option and, if so, will set it to 10 thus letting connection attempts time out after ten seconds. If your rsync is too old to support this option, no arguments are used.

rsync-timeout An integer value specifying the number seconds an rsync command is allowed to run before it is being terminated. The default if the value is missing is 300 seconds.

disable-rrdp A boolean value that, if present and true, turns off the use of RRDP.

rrdp-timeout An integer value that provides a timeout in seconds for all individual RRDP-related network operations, i.e., connects, reads, and writes. If the value is missing, a default timeout of 30 seconds will be used. Set the value to 0 to turn the timeout off.

rrdp-connect-timeout An integer value that, if present, sets a separate timeout in seconds for RRDP connect requests only.

rrdp-local-addr A string value that provides the local address to be used by RRDP connections.

rrdp-root-certs A list of strings each providing a path to a file containing a trust anchor certificate for HTTPS authentication of RRDP connections. In addition to the certificates provided via this option, the system’s own trust store is used.

rrdp-proxies A list of string each providing the URI for a proxy for outgoing RRDP connections. The proxies are tried in order for each request. HTTP and SOCKS5 proxies are supported.

dirty A boolean value which, if true, specifies that unused files and directories should not be deleted from the repository directory after each validation run. If left out, its value will be false and unused files will be deleted.

validation-threads An integer value specifying the number of threads to be used during validation of the repository. If this value is missing, the number of CPUs in the system is used.

log-level A string value specifying the maximum log level for which log messages should be emitted. The default is warn.

log A string specifying where to send log messages to. This can be one of the following values:

default Log messages will be sent to standard error if Routinator stays attached to the terminal or to syslog if it runs in daemon mode.

stderr Log messages will be sent to standard error.

syslog Log messages will be sent to syslog.

file Log messages will be sent to the file specified through the log-file configuration file entry.

The default if this value is missing is, unsurprisingly, default.

log-file A string value containing the path to a file to which log messages will be appended if the log configuration value is set to file. In this case, the value is mandatory.

syslog-facility A string value specifying the syslog facility to use for logging to syslog. The default value if this entry is missing is daemon.

rtr-listen An array of string values each providing the address and port which the RTR daemon should listen on in TCP mode. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.

http-listen An array of string values each providing the address and port which the HTTP service should listen on. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.
The RPKI Documentation

listen-systemd  The RTR TCP listening socket will be acquired from systemd via socket activation. Use this option together with systemd’s socket units to allow Routinator running as a regular user to bind to the default RTR port 323.

refresh  An integer value specifying the number of seconds Routinator should wait between consecutive validation runs in server mode. The next validation run will happen earlier, if objects expire earlier. The default is 600 seconds.

timeout  An integer value specifying the number of seconds an RTR client is requested to wait after it failed to receive a data set. The default is 600 seconds.

expire  An integer value specifying the number of seconds an RTR client is requested to use a data set if it cannot get an update before throwing it away and continuing with no data at all. The default is 7200 seconds if it cannot get an update before throwing it away and continuing with no data at all. The default is 7200 seconds.

history-size  An integer value specifying how many change sets Routinator should keep in RTR server mode. The default is 10.

pid-file  A string value containing a path pointing to the PID file to be used in daemon mode.

working-dir  A string value containing a path to the working directory for the daemon process.

chroot  A string value containing the path any daemon process should use as its root directory.

user  A string value containing the user name a daemon process should run as.

group  A string value containing the group name a daemon process should run as.

tal-label  An array containing arrays of two string values mapping the name of a TAL file (without the path but including the extension) as given by the first string to the name of the TAL to be included where the TAL is referenced in output as given by the second string.

If the options missing or if a TAL isn’t mentioned in the option, Routinator will construct a name for the TAL by using its file name (without the path) and dropping the extension.

13.8.7 HTTP Service

Routinator can provide an HTTP service allowing to fetch the Validated ROA Payload in various formats. The service does not support HTTPS and should only be used within the local network.

The service only supports GET requests with the following paths:

/metrics  Returns a set of monitoring metrics in the format used by Prometheus.

/status  Returns the current status of the Routinator instance. This is similar to the output of the /metrics endpoint but in a more human friendly format.

/version  Returns the version of the Routinator instance.

/api/v1/validity/as-number/prefix  Returns a JSON object describing whether the route announcement given by its origin AS number and address prefix is RPKI valid, invalid, or not found. The returned object is compatible with that provided by the RIPE NCC RPKI Validator. For more information, see https://ripe.net/support/documentation/developer-documentation/rpki-validator-api

/validity?asn=as-number&prefix=prefix  Same as above but with a more form-friendly calling convention.

In addition, the current set of VRPs is available for each output format at a path with the same name as the output format. E.g., the CSV output is available at /csv.

These paths accept filter expressions to limit the VRPs returned in the form of a query string. The field filter-asn can be used to filter for ASNs and the field filter-prefix can be used to filter for prefixes. The fields can be repeated multiple times.

This works in the same way as the options of the same name to the `vrps` command.

### 13.8.8 Relaxed Validation

The documents defining RPKI include a number of very strict rules regarding the formatting of the objects published in the RPKI repository. However, because PRKI reuses existing technology, real-world applications produce objects that do not follow these strict requirements.

As a consequence, a significant portion of the RPKI repository is actually invalid if the rules are followed. We therefore introduce two validation modes: strict and relaxed. Strict mode rejects any object that does not pass all checks laid out by the relevant RFCs. Relaxed mode ignores a number of these checks.

This memo documents the violations we encountered and are dealing with in relaxed validation mode.

**Resource Certificates (RFC 6487)** Resource certificates are defined as a profile on the more general Internet PKI certificates defined in RFC 5280.

**Subject and Issuer** The RFC restricts the type used for CommonName attributes to PrintableString, allowing only a subset of ASCII characters, while RFC 5280 allows a number of additional string types. At least one CA produces resource certificates with Utf8Strings.

In relaxed mode, we will only check that the general structure of the issuer and subject fields are correct and allow any number and types of attributes. This seems justified since RPKI explicitly does not use these fields.

**Signed Objects (RFC 6488)** Signed objects are defined as a profile on CMS messages defined in RFC 5652.

**DER Encoding** RFC 6488 demands all signed objects to be DER encoded while the more general CMS format allows any BER encoding – DER is a stricter subset of the more general BER. At least one CA does indeed produce BER encoded signed objects.

In relaxed mode, we will allow BER encoding.

Note that this isn’t just nit-picking. In BER encoding, octet strings can be broken up into a sequence of sub-strings. Since those strings are in some places used to carry encoded content themselves, such an encoding does make parsing significantly more difficult. At least one CA does produce such broken-up strings.

### 13.8.9 Signals

**SIGUSR1: Reload TALs and restart validation** When receiving SIGUSR1, Routinator will attempt to reload the TALs and, if that succeeds, restart validation. If loading the TALs fails, Routinator will exit.

### 13.8.10 Exit Status

Upon success, the exit status 0 is returned. If any fatal error happens, the exit status will be 1. Some commands provide a `--complete` option which will cause the exit status to be 2 if any of the rsync commands to update the repository fail.
This is the user handbook of the RTRlib. It provides guidance on how to use the library for development and gives an overview of some command line tools that are based on RTRlib. Further information can be found on the RTRlib website\textsuperscript{1} and its source code repository on Github\textsuperscript{2}.

14.1 About

14.1.1 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.

14.1.2 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).

\textsuperscript{1} Project website – https://rtrlib.realmv6.org
\textsuperscript{2} Source code on Github – https://github.com/rtrlib/rtrlib
14.1.3 License

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

14.1.4 Supported RFCs

The current version implements RFC 6811 and RFC 8210.

14.1.5 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.

14.2 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.

14.2.1 Apple macOS

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

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

14.2.2 Archlinux

For Archlinux we maintain two PKGBUILDs in the Archlinux User Repository, rtrlib and rtrlib-git. 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

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

(continues on next page)

---

1 Homebrew – http://brew.sh
2 https://aur.archlinux.org/packages/rtrlib/
3 https://aur.archlinux.org/packages/rtrlib-git/
### 14.2.3 Debian

RTRlib is part of the official Debian package repository since Buster\(^4\) and can be installed using \texttt{apt}. The following packages are available:

- \texttt{librtr0} includes the basis library.
- \texttt{librtr0-dev} includes header files etc. for developers.
- \texttt{rtr-tools} includes basic command line tools based on RTRlib.
- \texttt{librtr0-dbgsym} includes debugging symbols.
- \texttt{librtr-doc} includes offline documentation.

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

\texttt{apt install librtr0 librtr-dev}

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

\texttt{apt install librtr0 rtr-tools}

### 14.2.4 Gentoo

The FRR routing project maintains a gentoo overlay\(^5\) that contains an \texttt{ebuild} for the RTRlib. First, setup \texttt{layman}\(^6\), then install rtrlib with the following commands:

\begin{verbatim}
# If this doe not work try layman -f
layman -a frr-gentoo
emerge rtrlib
\end{verbatim}

### 14.2.5 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:

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

\(^4\) 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
The RPKI Documentation

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

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\(^10\).

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>
```

---

\(^7\) 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
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
```

### 14.3 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`.

#### 14.3.1 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 14.1 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 14.1: Output of the rtrclient tool.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Prefix Length</th>
<th>ASN</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 89.185.224.0</td>
<td>19 - 19</td>
<td>24971</td>
</tr>
<tr>
<td>+ 180.234.81.0</td>
<td>24 - 24</td>
<td>45951</td>
</tr>
<tr>
<td>+ 37.32.128.0</td>
<td>17 - 17</td>
<td>197121</td>
</tr>
<tr>
<td>+ 161.234.0.0</td>
<td>16 - 24</td>
<td>6306</td>
</tr>
<tr>
<td>+ 85.187.243.0</td>
<td>24 - 24</td>
<td>29694</td>
</tr>
<tr>
<td>+ 2a02:5d8::</td>
<td>32 - 32</td>
<td>8596</td>
</tr>
<tr>
<td>+ 2a03:2260::</td>
<td>30 - 30</td>
<td>201701</td>
</tr>
<tr>
<td>+ 2001:13c7:6f08::</td>
<td>48 - 48</td>
<td>27814</td>
</tr>
<tr>
<td>+ 2a07:7cc3::</td>
<td>32 - 32</td>
<td>61232</td>
</tr>
<tr>
<td>+ 2a05:b480:fc00::</td>
<td>48 - 48</td>
<td>39126</td>
</tr>
</tbody>
</table>

\(^{11}\) API reference – https://rtrlib.realmv6.org/doxygen/latest
14.3.2 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 14.2 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 14.2: Output of rpki-rov showing validation results of multiple prefixes.
A fully-featured RPKI relying party software, written by the RIPE NCC in Java. This application allows operators to
download and validate the global RPKI data set for use in their BGP decision making process and router configuration.
The project consists of two separate deployable units called the RPKI Validator and RPKI-RTR Server.

15.1 Installation

RIPE NCC provides a total of four options for installations:

- **CentOS (rpm)**
- **Debian (deb)**
- **Generic build (tgz)**
- **Docker image**

15.1.1 CentOS

We have set up a repository with CentOS 7 RPMs for Prod builds. You can add the repository to your system as follows:

```bash
˓→centos7/ripencc-rpki-prod.repo
```

You might have to install ‘yum-utils’ first:

```bash
sudo yum install yum-utils
```

Install the RPKI Validator:

```bash
sudo yum install rpki-validator
```
The RPKI Documentation

Install the RPKI-RTR Server:

```
sudo yum install rpki-rtr-server
```

Run and enable the services:

```
sudo systemctl enable rpki-validator-3
sudo systemctl start rpki-validator-3

sudo systemctl enable rpki-rtr-server
sudo systemctl start rpki-rtr-server
```

To monitor the logs:

```
sudo journalctl -f -u rpki-validator-3
sudo journalctl -f -u rpki-rtr-server
```

The RPKI Validator 3.1 will be running on http://localhost:8080/
The RPKI-RTR Server will be running on http://localhost:8081/
You can also explore the API at http://localhost:8080/swagger-ui.html

15.1.2 Debian

The Debian packages for the RPKI Validator and RPKI-RTR Server can be found at: https://ftp.ripe.net/ripe/tools/rpki/validator3/prod/deb/

Download the suitable package and proceed with the installation:

Install the RPKI Validator:

```
sudo apt install ./rpki-validator-3-latest.deb
```

Install the RPKI-RTR Server:

```
sudo apt install ./rpki-rtr-server-latest.deb
```

Run and enable the services:

```
sudo systemctl enable rpki-validator-3
sudo systemctl start rpki-validator-3

sudo systemctl enable rpki-rtr-server
sudo systemctl start rpki-rtr-server
```

To monitor the logs:

```
sudo journalctl -f -u rpki-validator-3
sudo journalctl -f -u rpki-rtr-server
```

The RPKI Validator 3.1 will be running on http://localhost:8080/
The RPKI-RTR Server will be running on http://localhost:8081/
You can also explore the API at http://localhost:8080/swagger-ui.html
15.1.3 Generic build

You can find generic production builds at: https://ftp.ripe.net/tools/rpki/validator3/prod/generic/ Download the suitable package and unpack it.

To run the RPKI Validator generic build:

```
./rpki-validator-3.sh
```

To run the RPKI-RTR generic build:

```
./rpki-rtr-server.sh
```

The RPKI Validator 3.1 will be running on http://localhost:8080/
The RPKI-RTR Server will be running on http://localhost:8081/
You can also explore the API at http://localhost:8080/swagger-ui.html

15.1.4 Docker

To run the Centos/RPM based image with systemd:

```
docker pull ripecc/rpki-validator-3-docker:latest
docker run --privileged --name rpkival -p 8080:8080 -d ripecc/rpki-validator-3-
˓→docker:latest
```

To run the generic alpine based image:

```
docker pull ripecc/rpki-validator-3-docker:alpine
docker run --name validator-3-alpine -p 8080:8080 -d ripecc/rpki-validator-3-
˓→docker:alpine
```

The RPKI Validator 3.1 will be running on: http://localhost:8080/
More info can be found at https://hub.docker.com/r/ripencc/rpki-validator-3-docker

15.1.5 Extra TALs

By default, the Validator will have Trust Anchor Locators (TALs) installed for AFRINIC, APNIC, LACNIC, RIPE NCC, but not ARIN.

You can download the ARIN TAL at https://www.arin.net/resources/manage/rpki/tal/
Any of the formats will work, but the “RIPE NCC RPKI Validator format” will ensure that the TAL will have a friendly name like “ARIN”.

You can use the following script to upload it:

```
./upload-tal.sh arin-ripevalidator.tal http://localhost:8080/
```

The script should be in the root folder if you unpacked the generic build, or in /usr/bin if you installed it using RPM/Debian package.

Alternatively, you can put extra TAL files to the preconfigured-tals directory of the RPKI Validator installation. This directory is scanned on the start and all the parseable TALs are picked up for validation. For the RPM/Debian package installation the directory is /var/lib/rpki-validator-3/preconfigured-tals/.
15.2 RPKI Validator

Set up to run as a daemon, and has the following features:

- Supports all current RPKI objects: certificates, manifests, CRLs, ROAs, router certificates, and ghostbuster records
- Supports the RRDP delta protocol
- Supports caching RPKI data in case a repository is unavailable
- Uses an asynchronous strategy to retrieve (often delegated) repositories, so that unavailable repositories do not block validation
- Features an API
- Has a full UI
- Supports exceptions through local filters and assertions

15.3 RPKI-RTR Server

A separate daemon that implements RPKI to the Router protocol (RTR), allowing validated prefix origin data to be delivered to routers. The RPKI-RTR Server is set up as a separate daemon because not everyone needs to run it. Far more importantly, a separate daemon allows you to start multiple instances for redundancy.

For more information, check the release notes. You can also contribute to the project on GitHub.

15.4 System Requirements

You will need a UNIX-like system with OpenJDK 8 or higher and rsync. You will also need at least 1.5GB of RAM available on your server (2GB in total if you also run the RPKI-RTR server). One (virtual) CPU should be enough. The repository objects are stored in a file-based database, rather than in memory, for which we recommend at least 10GB of available disk space.
CHAPTER 16

Index

• genindex
Symbols
-accept-arin-rpa
  command line option, 145
-allow-dubious-hosts
  command line option, 143
-chroot=path
  command line option, 148
-complete
  command line option, 146–148
-decline-arin-rpa
  command line option, 145
-dirty
  command line option, 144
-disable-rrdp
  command line option, 144
-disable-rsync
  command line option, 144
-expire=seconds
  command line option, 148
-group=group-name
  command line option, 148
-history=count
  command line option, 148
-http=addr:port
  command line option, 147
-listen-systemd
  command line option, 147
-logfile=path
  command line option, 145
-pid-file=path
  command line option, 148
-refresh=seconds
  command line option, 147
-retry=seconds
  command line option, 148
-rrdp-connect-timeout=seconds
  command line option, 144
-rrdp-local-addr=addr
  command line option, 144
-rrdp-proxy=uri
  command line option, 144
-rrdp-root-cert=path
  command line option, 144
-rrdp-timeout=seconds
  command line option, 144
-rsync-command=command
  command line option, 144
-rsync-timeout=seconds
  command line option, 144
-rtr=addr:port
  command line option, 147
-stale=policy
  command line option, 143
-strict
  command line option, 143
-syslog
  command line option, 144
-syslog-facility=facility
  command line option, 145
-user=user-name
  command line option, 148
-validation-threads=count
  command line option, 144
-working-dir=path
  command line option, 148
-V, -version
  command line option, 145
-a asn, -asn=asn
  command line option, 147
-a asn, -filter-asn=asn
  command line option, 147
-b dir, -base-dir=dir
  command line option, 146
-c path, -config=path
  command line option, 143
-d, -detach
  command line option, 147
-f format, -format=format
  command line option, 145