Master's Thesis Computing Science

Post-Quantum Cryptography for the RPKI

Temporary draft version to accompany blog on `sidnlabs.nl`*, to be replaced soon.*

Dirk Doesburg
s1040211

June 25, 2025

Supervisor SIDN Labs:
dr. ir. Moritz Müller
`moritz.muller@sidn.nl`

First assessor:
dr. ing. P.J.M. Van Aubel
`pol.vanaubel@ru.nl`

Second assessor:
dr. A.A. Westerbaan
`awesterb@cs.ru.nl`

Radboud University

Abstract

The Border Gateway Protocol is at the core of the internet, distributing routing information between networks. However, it was not designed with security in mind, enabling malicious networks to hijack traffic. The Resource Public Key Infrastructure (RPKI) has emerged as the primary tool to secure BGP routing, supporting mechanisms like Route Origin Validation (ROV) that make attacks more difficult, and are increasingly being adopted by network operators. However, the RPKI currently relies on RSA signatures, which are vulnerable to attacks by powerful quantum computers. While much research is done on post-quantum cryptography (PQC) and its application in protocols like TLS and DNSSEC, no such work has been done for the RPKI. This thesis lays the groundwork for a migration to post-quantum cryptography in the RPKI.

We show that the RPKI with insecure cryptography can be abused for severe attacks that are even more effective than original BGP attacks. This highlights the importance of migrating to post-quantum cryptography. We also find that the communication between CAs is an attractive target for quantum attackers, and that the RPKI relies on several related protocols that must be secured as well.

We then evaluate which post-quantum signatures can be a suitable replacement for RSA in the RPKI, primarily by proposing a method to estimate the performance impact of a given post-quantum scheme. A hybrid with Falcon-512 as post-quantum component emerges as a promising candidate, though alternatives can perform similarly.

Next, we introduce the novel *null scheme*: a method to avoid redundant signatures and public keys that are present in every signed object in the RPKI. This can reduce the size and verification time of the RPKI (e.g. reducing the median ROA size from 4354 to 2295 bytes when using a hybrid of Falcon-512 and RSA-2048), largely making up for the performance cost of switching to post-quantum signatures. This is particularly useful when introduced together with post-quantum signatures, sharing a single algorithm rollover.

Finally, we argue that the existing algorithm agility procedure (RFC6916) is impractical, and propose a simpler *mixed-tree* migration that (1) is not necessarily top-down, (2) requires no globally coordinated milestone days, and (3) uses simple key rollovers for individual CAs using the familiar RFC6489 key rollover procedure. In this approach, updated relying party software and trust anchors are distributed as soon as possible, while actual CA migrations can be delayed without problem.

Our proof-of-concept implementation in Krill and Routinator demonstrates the feasibility of this approach. We publish this implementation to provide a starting point for further research, including performance measurements and interoperability testing.

Acknowledgements

I was given the opportunity to perform my research with SIDN Labs. My heartfelt thanks go to the whole team for being so welcoming, and in particular, to Moritz, Lisa and Ralph. Our discussions and their feedback were invaluable to me. I would also like to thank my internal supervisor, Pol, for his guidance and taking the time to meet regularly. Finally, I'm grateful to Job Snijders and Tim Bruijnzeels for their expertise and discussing early versions of this work.

Contents

Chapter 1

Introduction

The internet's core routing infrastructure relies on the Border Gateway Protocol (BGP), a protocol fundamental to directing traffic between autonomous systems (ASes) across the global Internet. BGP enables these autonomous systems — networks operated by Internet Service Providers and other organizations — to exchange routing information. Despite its crucial role, BGP was not designed with security in mind and relies on "good faith" between AS operators. This jeopardizes the availability, integrity, and confidentiality of the Internet, both through human error (misconfiguration) and malicious actors that intentionally redirect traffic to be dropped, eavesdropped on, or manipulated.

The Resource Public Key Infrastructure (RPKI) has emerged as the primary tool to address these security concerns. The RPKI is a decentralized database that allows legitimate holders of internet resources (such as IP addresses) to make cryptographically verifiable statements about how routing should take place. These statements are in turn used to make secure routing decisions. The prime use case of the RPKI is Route Origin Validation (ROV), which is increasingly being adopted by network operators. Currently, it is configured to protect 56% of all IPv4 BGP originations against common misconfigurations and some attacks [50].[1] Other complementary techniques exist that offer further protection against more sophisticated attacks, including Autonomous System Provider Authorization (ASPA) and BGPsec (see section 2.2). While ASPA is approaching standardization and should soon begin to see deployment, the older BGPsec is not (yet) used in practice. This thesis focuses on the RPKI infrastructure itself—which underpins all these mechanisms. BGPsec in particular also uses different cryptographic signatures, outside the RPKI, but these are not in the scope of this thesis.

The RPKI currently uses RSA signatures [61, RFC7935] that are vulnerable to quantum attacks [66]. As quantum computing advances, the threat of quantum-enabled attacks on attempts to secure BGP routing is becoming increasingly real.

While the need for migration between cryptographic algorithms has been considered for the RPKI (for example in a published but controversial algorithm agility procedure: [RFC6916]), such transitions take time and coordination. Hence, it is necessary that the community starts investigating these transitions sooner rather than later. Taking into account the recent standardization of Post-Quantum Cryptography (PQC) algorithms by the National Institute of Standards and Technology (NIST) — which organized a process to evaluate and standardize quantum-resistant cryptographic algorithms — the time is right to start considering the adoption of PQC into the RPKI.

Introducing post-quantum cryptography in existing protocols can be challenging, due to both the performance impact of slower post-quantum algorithms and the difficulty of deploying changes. Previous efforts in transitioning cryptographic protocols to

[1] At the time of writing, 56% of IPv4 BGP prefix originations are ROV-Valid (this number is steadily increasing), meaning that networks *can* filter out some attacks with ROV that target such prefixes. Not all networks do so, but only a fraction of networks needs to perform ROV filtering for ROV to be effective.

quantum-resistant alternatives have been seen in domains such as TLS and DNSSEC [16, 49], but no significant work has been done for the RPKI.

Although RPKI ROV has not been fully adopted yet (so the current state of affairs is not yet as secure as one would hope), the potential impact of a quantum-enabled attack on RPKI is significant, similar to that of attacks on e.g. DNSSEC. Furthermore, BGP hijacking attacks in the past have typically involved well-resourced adversaries. Given that access to quantum computers is likely to be limited to sophisticated adversaries, the alignment between existing BGP threat actors and plausible early quantum adversaries makes quantum attacks especially relevant to the RPKI.

1.1 Contribution

In this thesis, we lay the groundwork for a transition to quantum-resistant cryptography in the RPKI. After presenting the necessary background information and briefly discussing related work in chapter 2, we cover the following topics:

- We first evaluate what threat quantum computing poses to the current RPKI. Chapter 3 shows what a quantum-enabled attack on the RPKI could look like, what impact it could have, and what parts of the RPKI are currently vulnerable.
- Next, we look for a suitable post-quantum replacement for RSA in chapter 4. We determine the requirements, model the expected performance impact, and apply this methodology to several candidate algorithms. Finally, we suggest standardizing multiple algorithms at the same time, which turns out to be potentially useful for a variety of reasons.
- As a special instance of using specialized algorithms for different use cases within the RPKI, chapter 5 defines a 'null scheme' that can replace a normal signature scheme in so-called *one-time-use* end-entity certificates. This proposal removes redundancy from every signed object in the RPKI, which can largely make up for the performance cost of switching to larger post-quantum signatures.
- The possible steps for a migration between signature algorithms are analyzed in chapter 6. While [RFC6916] already defines a procedure for algorithm agility, we consider it too complicated operationally. Therefore, we propose a simpler transition, closely based on a proposal coined by Brian Dickson ([19, 20]) during discussions about the drafts leading up to [RFC6916].
- Based on our proposed migration plan, we implement a proof of concept in chapter 7. We adapt the popular validator software *Routinator* and CA software *Krill* to support a post-quantum signature algorithm. Using this implementation, we then show that our transition procedure is feasible with minimal changes to existing software.

Finally, we reflect on our findings and provide actionable recommendations for the RPKI community in chapter 8.

Chapter 2

Preliminaries

2.1 Border Gateway Protocol

The internet can be viewed as a network of networks, where each network — called an Autonomous System (AS) — manages its internal routing independently. An AS represents a network under a single administrative control, such as an internet service provider, a large enterprise, or a content provider. Each AS is uniquely identified by a globally unique Autonomous System Number (ASN), often written as e.g. `AS64501`, which allows it to be distinctly recognized in global routing.

While internal routing within an AS can be handled by a variety of protocols, communication between different autonomous systems is orchestrated by the Border Gateway Protocol (BGP) version 4 [RFC4271]. In this thesis, we focus exclusively on *external BGP*—as opposed to *internal BGP*, one of multiple ways for routers *within* an AS to exchange information—and we simply refer to 'BGP'. BGP enables the exchange of routing information between these autonomous systems, allowing global internet connectivity.

BGP is a protocol that allows networks to exchange routing information in the form of *route announcements*, which describe paths to reach certain addresses. These announcements consist of:

- A *prefix*, which represents a range of IP addresses that can be reached through the route.
- An *AS path*, which is a sequence of autonomous systems that the route would traverse.

Prefixes are typically written as e.g. `a.b.c.0/24`, where the final part `/24` indicates the length of the prefix in bits. That is, the prefix `a.b.c.0/24` contains the 256 addresses from `a.b.c.0` up to `a.b.c.255`, and the prefix `a.b.0.0/16` contains the 65536 addresses from `a.b.0.0` up to `a.b.255.255`. So, prefixes are hierarchical, and one prefix can be *more specific* than another if it is longer. `a.b.c.0/24` is more specific (and a subset of) the shorter `a.b.0.0/16`.

BGP speakers use these route announcements to build a *routing table*, which is used to determine the preferred path to a destination. Each BGP speaker receives many announcements, and shares only the most preferred route for each prefix. Route preference is determined primarily by two factors: more specific prefixes are preferred, and among routes to the same prefix, those with the shortest AS path are chosen. When a most preferred route is found, a BGP speaker will share it with all of its neighbors, adding its own AS number to the AS path. This way, good routes are propagated throughout the internet, and sub-optimal routes are discarded. A toy example of BGP's operation is shown in fig. 2.1.

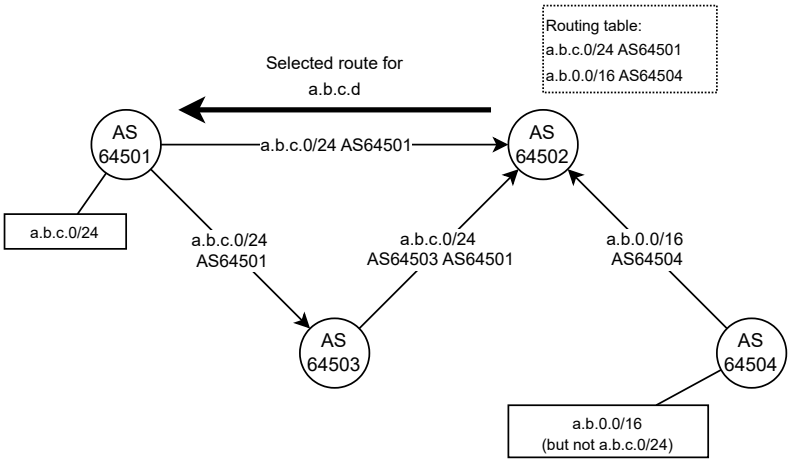


Figure 2.1: *An example of BGP operation from the perspective of AS64502, looking for a route to* `a.b.c.d`*. AS64502 receives 3 matching announcements. The one from AS64501 wins with prefix length 24 and only 1 hop. AS64503's announcement is discarded because it has more hops. AS64504's announcement is not discarded, but not used for* `a.b.c.d` *because a more specific route is known.*

2.2 BGP security

The Border Gateway Protocol fundamentally operates on the assumption that all autonomous systems are honest and trustworthy. There is no built-in mechanism to verify anything claimed in a BGP announcement, so an AS can lie by claiming to own (originate) a prefix that it does not, or by making up a fake AS path, that it never actually received from its neighbors. This makes it easy for traffic to be redirected in an undesirable way, both due to accidental misconfiguration and deliberate attacks.

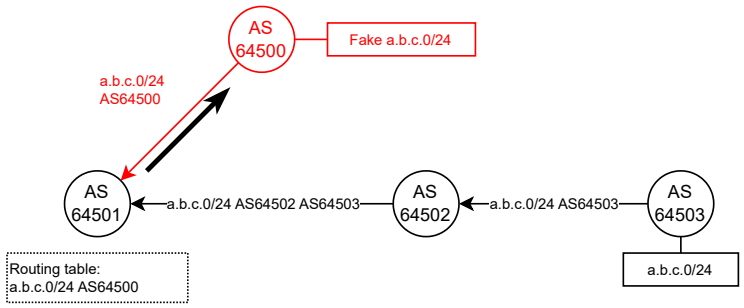
We show several examples, with AS numbers AS64500-64505 that are from a range reserved for documentation. The misbehaving network is always *AS64500*, and is highlighted in red and in italics for clarity.

Figure 2.2 shows two ways by which a malicious AS could hijack traffic. In 2.2a, *AS64500* announces `a.b.c.0/24` *AS64500*, lying about owning the prefix. Consequently, from the perspective of AS64501, *AS64500* has the shortest path at only a single hop. When AS64501 sends its traffic destined for e.g. `a.b.c.d` towards *AS64500*, *AS64500* can then choose to impersonate the legitimate destination, drop the traffic, or forward it to the legitimate destination while eavesdropping on it.

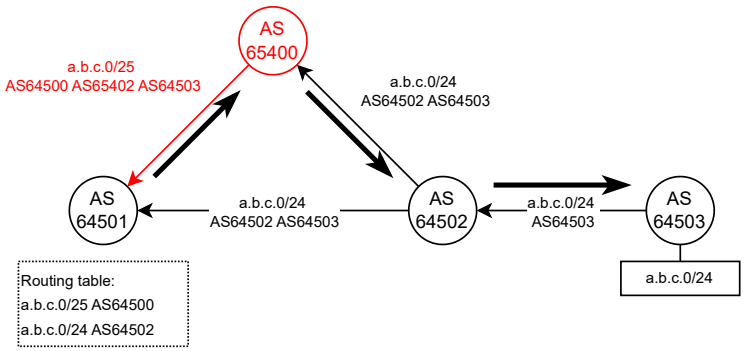
In 2.2b, a different approach is used where *AS64500* does not lie about the number of steps in the path, but about the length of the prefix. *AS64500* announces a more specific subset (`/25`) of the original prefix. In this case, the distance between the malicious AS and its victims (AS64501) does not matter: a more-specific route should be preferred regardless of its AS path length. On the other hand, a `/25` announcement is suspiciously specific, so it will often be ignored in practice, and the legitimate origin AS64503 can protect against this by announcing two `/25`'s instead of a single `/24`.

BGP hijacking is not merely a theoretical concern; it has repeatedly impacted internet operations with real-world consequences. Some notable incidents include:

- In 2008, Pakistan Telecom, in an attempt at censoring nationally, accidentally leaked announcements for YouTube's address space outside of Pakistan. These announcements were propagated by ASes worldwide. This caused global traffic for YouTube to be routed towards Pakistan, making YouTube unavailable from most of the world for several hours [53].



(a) *BGP hijack by illegitimately originating a prefix. This results in AS64500 having a shorter path from some other ASes, but not from everywhere.*



(b) *BGP hijack by announcing a more specific path to the legitimate origin. Here, the path length does not matter, as the more specific prefix should always be used.*

Figure 2.2: *Examples of BGP hijacks.*

- In 2018, attackers hijacked Amazon's Route 53 DNS service to redirect cryptocurrency users to a phishing site, stealing approximately $150 000 in cryptocurrency [67].

To mitigate these risks, many security mechanisms have been proposed that can make attacks harder to perform, and prevent misconfiguration from having widespread consequences. The Internet Routing Registry (IRR) is an early effort that established a distributed database where network operators can register their routing policies. While still widely used, IRR suffers from incomplete and sometimes inaccurate or outdated data.

Later, more robust cryptographically verifiable solutions based on the *Resource Public Key Infrastructure* (RPKI) have emerged. These include Route Origin Validation (ROV), which prevents unauthorized origin announcements, Autonomous System Provider Authorization (ASPA), which considers AS relationships in paths, and BGPsec, which provides comprehensive path validation.

These RPKI-based mechanisms are slowly but steadily being adopted by network operators, often in combination with filtering based on the IRR.

2.2.1 Resource Public Key Infrastructure

Each of the security mechanisms discussed here relies on the RPKI: a hierarchical system that allows an entity to verifiably assert that it is the legitimate holder of a set of IP addresses or an AS number [RFC6480].

The allocation of IP prefixes and ASNs is managed by the Internet Assigned Numbers Authority (IANA), which delegates these resources to five Regional Internet Registries

(RIRs) such as *RIPE NCC* for Europe and Russia. These RIRs in turn allocate resources—optionally through National or Local Internet Registries (NIRs or LIRs) or ISPs—to the final *resource holders*, such as ISPs or companies. The RPKI system is built on top of this existing hierarchy, consisting of certificates signed by a higher level authority (e.g. an RIR) on a public key of a lower level authority (e.g. an ISP), indicating the allocated resources.

Using a chain of these certificates, each resource holder can authoritatively make statements about the resources they hold, by publishing a signed object to the RPKI. The content of these signed objects depends on the specific security mechanism, such as ROAs or ASPAs discussed in the following sections. The validity of a signed object comes from an *end-entity* (EE) certificate: a certificate on a public key that is used only once to sign an RPKI object. The end-entity certificate is included with a signed object and serves to allow revocation of individual objects with existing revocation mechanisms, which would not be possible if signed objects were signed directly by a resource holder's CA keys.

2.2.1.1 Publication structure

The publication of objects in the RPKI happens through *repositories*. A repository is a location where RPKI objects of one or more resource holders are made accessible to Relying Parties (RPs). Each resource holder has a CA certificate that contains a URI pointing to its publication point. In practice, CAs often share a repository hosted by their parent — for example, an ISP might use publication infrastructure provided by their RIR rather than hosting their own. This is good for performance and saves resource holders the burden of hosting their own repository.

Relying parties (or *validators*) periodically download objects from all repositories using either `rsync` or the RPKI Repository Delta Protocol (RRDP) [RFC8182], maintaining a local cache of the entire RPKI. The validators use this cache to verify the chain of certificates from trust anchors to each signed object. We discuss the publication structure and its security implications in more detail in section 3.3.

2.2.2 Route Origin Validation

The first mechanism to partially secure BGP is Route Origin Validation (ROV), which provides a way to authorize an AS to *originate* a prefix. This is done by publishing a [RFC9582] Route Origin Authorization (ROA) object in the RPKI, which states that a certain prefix is allowed to be originated by a certain AS.

Such a ROA consists of:

- The ASN that is allowed to originate the prefix(es).
- One or more prefixes, which is the range of IP addresses that the ROA applies to.
- For each prefix, a *maxLength*, indicating the longest sub-prefix that the ASN is allowed to announce. This is often denoted as e.g. `-23` in `a.b.0.0/22-23`, meaning that announcements for `a.b.0.0/22`, `a.b.0.0/23`, and `a.b.2.0/23` are allowed, but `a.b.0.0/24` is not. If the *maxLength* is not larger than the legitimate announcements an AS makes, this mitigates attacks like in fig. 2.2b.[1]

When a BGP speaker receives a route announcement, it can check if the origin AS is allowed to originate the prefix by looking for a corresponding ROA in the RPKI. If there

[1] In that example, a ROA limiting the prefix length to 24 could prevent the attack, although *AS64500* could still announce a `/24` with a short path, to attract traffic from nearby victim ASes. ASes that are topologically further away from *AS64500* would still prefer the legitimate route, as honest victim ASes in between would make the path to *AS64500* longer than the best legitimate route.

exists a ROA with a matching prefix, *maxLength*, and ASN, the route announcement is considered *ROV-Valid*.[2] Otherwise, there might be a ROA for the prefix (or a less specific prefix that contains it) but that does not authorize the ASN or prefix length. In that case, the announcement is considered *ROV-Invalid*, and should probably not be used [RFC6483, RFC6811]. If there is no matching ROA at all, the route is *ROV-NotFound*. The practice of preferring ROV-Valid announcements over ROV-NotFound and ROV-NotFound over ROV-Invalid ones is what actually provides security. The fallback to ROV-NotFound is an attractive feature:

- It enables gradual adoption: ROV filtering affects only announcements for prefixes whose holders have opted in by publishing ROAs. Others are ROV-NotFound and are not affected.
- If (part of) the RPKI becomes unavailable for some reason, at first validators will keep functioning with the latest known state of the RPKI. Only once ROAs start to expire, do announcements start to fall back to ROV-NotFound. This is a safe failure mode, as BGP routing can continue normally with the last known state of the RPKI, or later with everything becoming ROV-NotFound. Unavailability of the RPKI should not make any destinations unroutable that could be reached before the outage.[3]

Furthermore, the barrier for adoption as resource holder is low, as it is easy to create and publish ROAs, and resource holders can benefit from ROV filtering even if they don't perform the filtering themselves: ROV is effective even if only a small fraction of autonomous systems do it, especially if these are well-connected ones, such as transit providers. For example, when a transit provider performs ROV filtering, a customer that is only connected to the provider will inherit the provider's filtering automatically [42].

2.2.2.1 Security provided by ROV

ROV effectively prevents unauthorized origin announcements such as fig. 2.2a: an AS cannot successfully claim to originate a prefix that is protected by a ROA for a different AS. Such announcements would be ROV-Invalid and filtered by ROV-performing ASes. By setting appropriate `maxLength` values as recommended in [RFC9319], ROV can also prevent more-specific prefix hijacks as in fig. 2.2b.

However, ROV only validates the origin AS, not the entire AS path. While this is very effective in preventing accidental misconfiguration from propagating throughout the internet, it does not prevent some ways of performing a prefix hijack. In particular, for a prefix `a.b.c.0/24` that can only be originated by AS64503, the malicious *AS64500* can simply announce a route with AS path *AS64500* `AS64503` to hijack traffic. This is an ROV-Valid announcement, as AS64503 is allowed to announce the prefix, despite *AS64500* not actually having a direct connection to AS64503. Where a false origination without ROV would trick all ASes that are closer to *AS64500* than AS64503, this *prepending* attack tricks only ASes that are closer to *AS64500* than to the *direct neighbors* of AS64503: announcing a path of 2 hops passing ROV is less effective than direct origination that works only without ROV. Back in 2010, [23] showed that with full ROV adoption, 13.6% of attacker-victim AS pairs are vulnerable to the prepending attack with 2-hop announcements, compared to 34.2% that are vulnerable to 1-hop announcements without ROV.[4] Figure 2.3 shows how the prepending attack propagates. There, *AS64500* acts as if there is a direct link from *AS64500* to AS64501,

[2]To avoid confusion with other notions of validity, we consistently use 'ROV-Valid', 'ROV-Invalid' and 'ROV-NotFound' to indicate the specific meaning of 'Valid' etc. in [RFC6811].

[3]There are edge cases when ROAs covering a prefix exist in multiple different CAs or repositories. See section 2.2.2.2.

[4]This likelihood differs depending on how well-connected attacker and victim are. Central ASes have a much higher chance of success as attacker, as they are closer to many victims. Similarly, they are less likely to be victims, as they are often close to the legitimate origin.

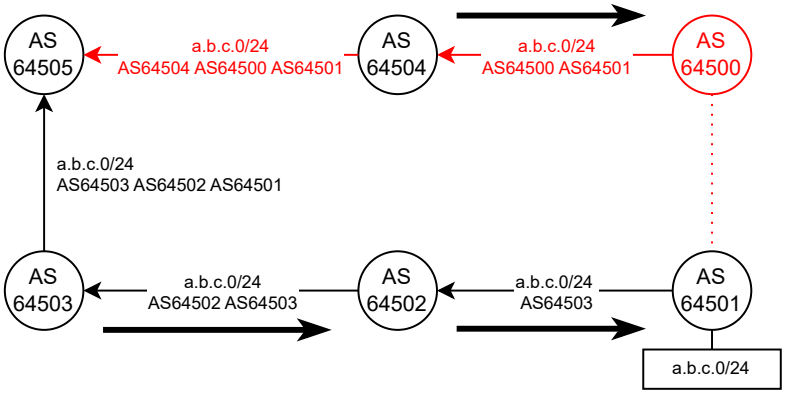


Figure 2.3: *Propagation of a hijack where AS64500 announces a path AS64500 AS64503 to pass ROV. Bold arrows indicate the path used from each AS. The path picked by AS64505 is a matter of local preference between the two 3-hop paths it receives.*

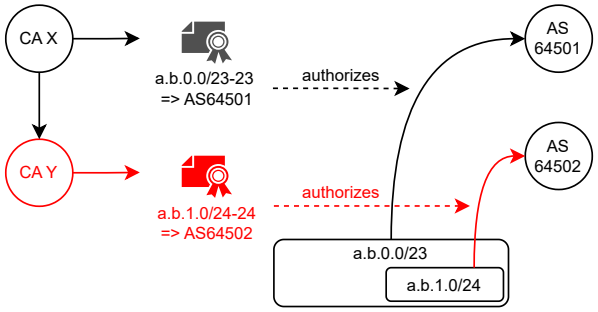


Figure 2.4: *Example of the risk of multiple CAs making ROAs that overlap. If the child CA Y (highlighted in red) becomes unavailable while parent X remains, RPs may consider only the parent's aggregate ROA, invalidating the child's more-specific announcement from AS64502.*

even if there is not. *AS64500* can attract the traffic from AS64504 and possibly AS64505.

2.2.2.2 Risk when ancestor CAs cover a prefix

If one CA (Y) has a prefix and an AS, and publishes a ROA `a.b.1.0/24-24 => AS64502`, and an ancestor (X) also publishes a ROA authorizing a *covering* prefix only to another AS, this introduces a risk. If the child CA's ROAs about a prefix disappear, while another CA's ROA covering the same prefix remains, that could cause announcements for the more-specific prefix to become ROV-Invalid. An example of this issue is shown in fig. 2.4. There, parent CA X publishes a ROA `a.b.0.0/23-23 => AS64501`, and a subordinate CA Y publishes a ROA authorizing a different ASN to originate a more-specific prefix: `a.b.1.0/24-24 => AS64502`. When CA X's products become unavailable (due to expiration or revocation of the ROA or the CA's resource certificate, or a publication server outage), an RP may consider only the ROA `a.b.1.0/24-24 => AS64502`, making the announcement of `a.b.1.0/24` ROV-Invalid, while it was ROV-Valid originally. If there were no other covering ROA (or both became unavailable *together*, such as when they're both in the same repository that suffers an outage), ROV would instead 'fail-open', changing from ROV-Valid to ROV-NotFound.

Accordingly, it is good practice to be careful with creating overlapping ROAs in

multiple CAs. This is often not a big risk when resource holders use a 'hosted CA' offering provided by their RIR. In that case, it is very unlikely that only one of the CAs covering a prefix will expire or become unavailable. However, if two different publication servers are used, or if the child CA's resource certificate expires, a problem can occur that makes legitimate announcements ROV-Invalid.

The relying party software Routinator can detect so-called 'unsafe VRPs': ROA payloads for prefixes where validation has failed for a part of the RPKI that *might* contain other ROAs for the prefix.[5] This can be used to ignore the covering ROAs in such a case, or to warn the RP operator about them.

2.2.2.3 Adoption

The use of ROV can be quantified in two ways: the portion of the IP address space or announced prefixes that is covered by ROAs, and the portion of networks that actually perform ROV filtering.

Coverage of ROAs has steadily increased since its introduction. At the time of writing, 56% of announced IPv4 prefixes[6] are ROV-Valid.

The practice of filtering out ROV-Invalid announcements is less common and harder to measure. [42] and [34] provide an in-depth analysis of the adoption of ROV filtering. Broadly speaking, few networks *perform* ROV filtering, but many networks benefit from it nonetheless.

2.2.3 AS Provider Authorization

Autonomous System Provider Authorization is a new mechanism that is being developed to address the security issues that remain despite ROV. While ROV ensures that prefixes are originated by authorized ASes, it cannot prevent manipulation of the rest of the AS path, such as the prepending attack described above.

ASPA provides a way for autonomous systems to explicitly authorize their *providers*: the ASes that are expected to provide transit for the AS. In BGP, an AS often has contractual peering relationships with other ASes it is connected with, such as a *Customer-Provider* relation where the customer pays the provider for transit, or a lateral peering relation where both ASes exchange traffic for one another. In the context of ASPA, the *provider* ('Provider+' in [3]) is an AS that is expected to be the next or previous hop of the customer in an AS path, including both transit providers and lateral peers. When two peers both participate in ASPA, they would attest to each other as providers.

This information can be used to check whether AS paths in a BGP announcement are plausible: if a path contains a step from *AS64500* to AS64501, and AS64501 has authorized some ASes but not *AS64500* as provider, the path is likely fraudulent. Additionally, ASPA can ensure that a path is *valley-free*. A valid path must consist of:

- An *up-ramp*, starting from the origin AS, going up through attested Customer-Provider relationships where the announcement comes from the customer to the provider.
- Then, optionally some hops for which no ASPA objects exist.
- Finally, a *down-ramp*, going through attested Provider-Customer relationships with the announcement going from provider to customer.

Figure 2.5 shows an overview of a valid path. Each of the three parts can potentially be empty. If a path is valid, that means that:

[5] `https://routinator.docs.nlnetlabs.nl/en/v0.14.2/unsafe-vrps.html`
[6] This makes op roughly 50% of the total announced IPv4 address space.

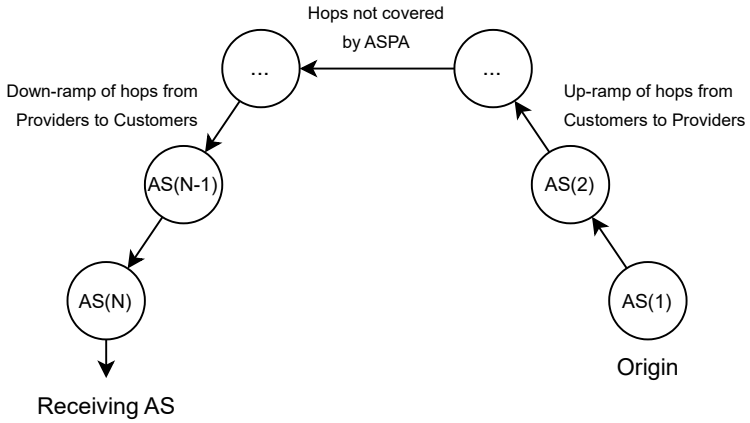


Figure 2.5: *ASPA verification overview.*

- If a customer AS has created an ASPA, there must not be a hop from the customer AS to any AS that is not a provider in the customer's ASPA objects.
- An announcement cannot go from a provider down to a customer (not a lateral peer), and then back up to another provider (that is again not a lateral peer). This is called *valley-free*: a customer can not take traffic from one provider and send it to another provider. Customers can only get traffic from their providers if that traffic is on its way to the customer or something below the customer, as part of the down-ramp.

2.2.3.1 Adoption

The ASPA mechanism has been under development since 2018 and is currently approaching standardization by the IETF's SIDROPS working group. Multiple implementations of ASPA exist, including in widely used RPKI CA software Krill[7] and the relying party software Routinator[8], as well as the pilot environment of the *RIPE NCC*. Because ASPA, like ROV, has a low barrier of entry for publishing and can provide security benefits even with limited adoption, it is likely that it will start getting adopted steadily as soon as it is standardized.

2.2.4 BGPsec

BGPsec [RFC8205] complements ROV, extending protection to the entire AS path. It is an in-band extension to BGP that adds signatures for the complete AS path, where each AS signs not only its own ASN but also the next AS in the path. This prevents any form of path manipulation attack, such as that in fig. 2.3.

However, BGPsec faces significant deployment challenges. It requires substantial computational resources for real-time signature generation and verification. Unlike ROV and ASPA, which provide incremental security benefits during partial deployment, BGPsec's security guarantees largely depend on widespread adoption. Consequently, BGPsec has seen negligible deployment in practice.

Due to this limited adoption, and BGPsec's reliance on the RPKI, we do not consider BGPsec in this thesis. Migrating BGPsec, which uses its own signature algorithm, to post-quantum algorithms is challenging but left to future work. As BGPsec also relies on the RPKI, securing the RPKI against quantum attacks is a prerequisite for doing the same for BGPsec.

[7] `https://github.com/NLnetLabs/krill/`
[8] `https://github.com/NLnetLabs/routinator/`

2.3 Post-quantum cryptography

In the RPKI, cryptographic algorithms are used that are expected to not be secure against an attacker with a quantum computer. In particular, [RFC7935] (the successor of [RFC6485]) specifies that the RPKI currently allows only RSA signatures [61] with SHA-256 as digest algorithm.[9] The security of RSA is based on the hardness of factoring large numbers, which is known to be broken by Shor's algorithm [66] when a sufficiently large quantum computer is available. While no quantum computer currently exists that can break traditional cryptography, the development of quantum computers is progressing rapidly, and it is expected that they will be able to break RSA and other traditional cryptographic algorithms, be it in several years or several decades [48].

Post-Quantum Cryptography (PQC) aims to develop cryptographic algorithms that remain secure against both classical and quantum computers. These algorithms are based on various mathematical problems that are believed to be hard even for quantum computers [13, 4]. The National Institute of Standards and Technology (NIST) initiated a competition for PQC in 2016 to develop and standardize post-quantum cryptographic algorithms.[10] Authors were invited to submit their algorithms, which were then evaluated by NIST and the cryptographic community. In 2022, three post-quantum signature algorithms (and a key-establishment mechanism, which is not relevant to the RPKI) were selected for standardization: *Dilithium* [24], *Falcon* [26], and *SPHINCS+* [5], that are now named ML-DSA, FN-DSA and SLH-DSA respectively. Falcon's final specification as FN-DSA has not been published yet, so it is still called Falcon here.

Additionally, a second round of competition is ongoing to select additional algorithms, that are based on different mathematical problems than the selected algorithms from the first round (both ML-DSA and Falcon are *lattice-based* schemes), or that have a performance advantage over the selected algorithms. The final selection of additional algorithms will not take place until at least 2026 or 2027.

Beyond the *stateless* algorithms mentioned above, *stateful* signature schemes like XMSS [RFC8391] offer strong security guarantees but require careful state management to prevent key reuse, making them less suitable for many applications.

2.3.1 Challenges

Significant challenges exist when introducing post-quantum cryptography, that can make it hard to deploy them in practice.

- Post-quantum signature schemes typically have much larger signatures and/or public keys. This can hurt performance, or might even not fit in strict size limits imposed by an application.
- Similarly, many algorithms are computationally slower than traditional algorithms at key generation, signing, or verifying.
- Some post-quantum algorithms are relatively new and not as well-understood as traditional algorithms. They have not received as much scrutiny from the cryptographic community as traditional algorithms have, making it harder to trust in their security.
- Replacing the algorithms used in protocols can be hard, sometimes requiring complicated protocol changes that can take years to be widely adopted.

[9] As an exception, BGPsec Router Certificates use elliptic curve cryptography (which is also vulnerable to a quantum computer) [RFC8608], but we consider this out of scope as these do not influence security of the rest of the RPKI, and BGPsec is not (yet) widely deployed.

[10] `https://www.nist.gov/pqcrypto`

- Finally, efficient, secure, and certified implementations of post-quantum algorithms are not yet widely available. This will improve over time, but it is a challenge for early adopters, especially in sectors where certification of cryptographic components is mandatory. It is not directly clear whether certification requirements apply to RPKI operators.

2.4 Related work

To our knowledge, this thesis is the first work on post-quantum cryptography for the RPKI. Yet, it does not exist in a vacuum. Much related work has been done on post-quantum cryptography and its applications, algorithm agility in the RPKI, and RPKI measurements.

2.4.1 Post-quantum cryptography in related technology

The challenges of deploying post-quantum cryptography are not unique to the RPKI. Significant research and standardization efforts are being made on *building blocks* (X.509, algorithms, composites, etc.), that are direct prerequisites for the migration of many systems, including the RPKI. There is also much work on various *applications* of cryptography, such as TLS (with key exchange and web PKI certificates) and DNSSEC.

Applications of post-quantum cryptography can be categorized into two groups:

- Much work is primarily concerned with confidentiality, where there is a real risk of "store-now-decrypt-later" attacks. In such attacks, adversaries collect encrypted data today, to decrypt later once sufficiently powerful quantum computers become available. This is an urgent threat that has driven rapid deployment of post-quantum key establishment mechanisms (KEMs) in protocols like TLS, with major browsers already supporting post-quantum algorithms for session keys.
- For *authentication and integrity*, the threat is less immediate since attackers need real-time access to quantum computers to forge signatures.

For TLS, many researchers have evaluated the performance impact of using post-quantum KEMs, focusing on confidentiality, rather than on authentication. Still, some authors have investigated post-quantum certificates (for authentication) in TLS specifically [69], and [68] covers the combination of post-quantum key exchange and certificates.

Another common technology that is important for the RPKI is DNSSEC [30], where confidentiality is not a concern. Here, the urgency of the migration is lower (being immune to store-now-decrypt-later attacks), but strict size limits (large signatures require expensive query retries over TCP) make for an interesting challenge. For DNSSEC, some authors evaluate drop-in replacements for current signatures [49], while others propose alternatives that significantly change the protocol [27].

Our work on the RPKI is another instance of an effort to make an existing application quantum-safe. It shares similar goals and challenges with other applications, but the RPKI also depends on the quantum-safety of related protocols, as those are, in turn, building blocks of the RPKI.

2.4.2 Algorithm migration in the RPKI

Although post-quantum cryptography for the RPKI had not previously been studied, there have been efforts on the broader topic of algorithm agility. When the RPKI was being designed, the IETF was already aware of the need to have a mechanism for

algorithm agility. This was implemented by [RFC6916]. The community has expressed concerns about this mechanism both before it was finalized, and more recently [21, 10, 9].

A proposal [19, 20, 22] put forward during review of [RFC6916] forms the foundation of our suggested strategy in section 6.2. Apart from that, we are not aware of other alternative migration strategies that have been worked out in detail, but there has been some work on related topics.

2.4.2.1 Elliptic-curve signatures

After the IETF 117 meeting in 2023, there was a conversation on the SIDROPS mailing list about introducing ECDSA or EdDSA signatures [71]. This was mainly motivated by their smaller sizes. As an added benefit, doing a migration already could help prepare for a second migration towards PQC some years later.

The focus of these plans was on introducing elliptic-curve cryptography (ECC) for the one-time key pairs[11] in end-entity certificates only, not for CAs. This avoids a large part of the complexity of a full algorithm migration as CAs needn't replace their resource certificates.

The details of the migration this proposal would require were not discussed. However, the plan was explicitly to move towards a state where mixed certificates are used (ECC in end-entity certificates and RSA elsewhere). This is incompatible with [RFC6916] and does require RPs to be updated, even if CAs' certificates remain unchanged.

Following the mailing list conversation, Job Snijders implemented experimental support for validating ECDSA certificates (in a mixed-certificates setting) in `rpki-client`.[12] That feature has not been used in practice since.

2.4.2.2 TAL and BPKI TA replacement

While not directly related to *algorithm* rollover, the ability to do *trust anchor* rollovers — for both [RFC6490] TALs and [RFC8183] BPKI TAs — is instrumental for algorithm rollovers. TALs are often hard-coded in relying party software, and as we will see in section 6.4.2, BPKI TAs are also hard to replace in practice. There have been efforts to make both of these rollovers easier.

Since the RPKI's inception, no TALs have been introduced that are not backward compatible with the original TALs. Four of the RIRs have each always had a single TA key pair, and APNIC used to have five separate TALs[13]. During 2017, APNIC retired four of its TALs, keeping only the one for resources it got from IANA [2]. So, while some TALs were removed or had minor changes (such as the addition of RRDP publication points), no real TAL key rollover has ever been performed in the real world.

To better enable rollover of TALs, a *Trust Anchor Key* object was introduced in [RFC9691], which provides an in-band mechanism to schedule a TAL replacement. This has not been implemented in common RPKI software yet but is a promising method.

The need for BPKI TA replacements was highlighted at IETF 115 [10], and two ideas for a potential in-band replacement protocol were written down in [11] and [46].

2.4.3 Measurements

Measurements of the RPKI can be divided into three categories identified by [64].

[11] See section 4.6.3 and chapter 5.

[12] `https://github.com/openbsd/src/commit/ec1cc732eea452b2c8e9f1282111d9cc0104e4b6`

[13] One for resources it got from IANA, and one for resources transferred from each of the other RIRs.

ROA measurements are concerned with monitoring the practice of creating ROAs. This is relatively straightforward, by simply observing the content of the RPKI. The coverage can then be combined with observations of BGP updates to determine the fraction of BGP originations that are ROV-Valid or ROV-Unknown, giving an indication of how many prefixes *might* be protected by ROV. Famously, [50] is a real-time dashboard that presents various statistics on ROA coverage over time.

ROV measurements try to determine how many, and which, ASes perform ROV filtering. This is more difficult, as it is not directly visible, and ROV filtering by one AS can have an umbrella effect on adjacent ASes. Various techniques are proposed and compared in [60, 31, 63, 42]. ROV measurement methodologies can also be a useful tool for monitoring during an algorithm migration, for example in the methods we suggest in section 6.2.2 and appendix B.

RPKI resilience is about the robustness of components of the RPKI. Among others, measurements are done on DNS resolvers used by RPs and the use of DNSSEC in [30]. Research on the security of RP implementations began with [32]. [45] provides an overview of more recent work on RPKI resilience. Measurements in [38, 40] also try to identify which software is used by RPs: another useful monitoring tool for algorithm migrations.

A fourth category can be added, that is particularly relevant for selecting post-quantum algorithms for use in the RPKI.

RPKI performance measurements are concerned with the efficiency of the RPKI, and how it can be improved. These insights are relevant both to estimate the impact when post-quantum signatures are introduced and to take steps to improve the performance, potentially compensating for the cost of post-quantum schemes. Early measurements were done on `rsync` downloads [37]. More recent measurements (on RRDP as well as other steps from the creation of a ROA to propagation in BGP) include [25, 65, 1]. Several strategies to optimize downloading and validation in RPs are proposed in [58], and [65, 74, 72] suggest improvements on the CA side to reduce the RPKI size. Our chapter 5 fits among these suggestions to improve performance.

Chapter 3

Quantum threat to routing security

We start by investigating what threat quantum computing poses to the security of BGP routing. Throughout this chapter, we assume an attacker that, in contrast to the threat model considered in the design of the RPKI ecosystem, is capable of breaking traditional public-key cryptography.

From the perspective of a relying party (abstracting away the complexities involved in setting up the RPKI and fetching from it), the resulting worst-case attacker model is one with the following capabilities:

Capability 1 *Forge any kind of signature used in the RPKI. This includes CA certificates, EE certificates (and hence ROAs), as well as Certificate Revocation Lists (CRLs) and RPKI Manifests.*

This capability follows directly from the assumption that the attacker can break the cryptographic algorithms used in the RPKI (i.e. RSA). Additionally, in section 3.3.1 we see that even if the RPKI certificate chains use only quantum-safe algorithms, some other components of the RPKI ecosystem also need to be protected to ensure that an attacker cannot obtain a valid certificate through other means than forgery.

Capability 2 *Cause (forged) objects to be published at a repository, or otherwise end up being validated by an RPKI Relying Party.*

This capability does not necessarily result from broken cryptography. However, there are many avenues to achieve it. In section 3.3.4 we show that this capability is realistic.

These two capabilities encompass every 'adverse action' described in [RFC8211], which lists actions that can be performed in certain scenarios but does not further evaluate the impact of these actions.

Assuming this attacker model, we show attacks that are possible on ROV and ASPA themselves, limiting the protection they provide, and more importantly, we also show that using ROV while under this threat model allows for more severe attacks that would not be possible on BGP without ROV. Next, we consider the components of the RPKI ecosystem, identifying additional parts (on top of the RPKI's certificates themselves) that need to be secured against a quantum-enabled attacker to be secure.

3.1 Attacks on ROV

The attacks that are possible on ROV depend on the situation of a prefix prior to an attack: is a prefix currently covered by a valid ROA or not? We consider these scenarios separately and show from each scenario what an attacker can achieve.

3.1.1 Forging a ROA for a protected prefix

Currently, most IP addresses are in a prefix for which a ROA exists. Legitimate announcements for this prefix are ROV-Valid, and ROV-performing routers should drop any announcement that is ROV-Invalid. For this scenario, our attacker is clearly able to get rid of the protection offered by ROV, by forging another ROA for (part of) the prefix, for example authorizing the attacker's ASN and with a high `maxLength`. Next, the attacker can announce a very specific route to a part of the prefix, which will be ROV-Valid according to the forged ROA. Routers are now likely to accept this illegitimate announcement, as it has a longer prefix than any legitimate announcements, and is also ROV-Valid and hence probably not considered suspicious. So, for a prefix that is covered by a ROA, simply inserting a single forged ROA is enough to make a hijack attack likely to succeed. In particular, because a ROV-Valid announcement is likely to be considered more trustworthy than an equally specific ROV-NotFound announcement, the attacker's capabilities make their hijack more likely to succeed than a traditional hijack attempt on a prefix that is not even covered by a ROA.

3.1.2 Forging a ROA for an unprotected prefix

An even worse situation arises when a prefix is currently not covered by a ROA. The RPKI currently does not include a ROA for the prefix, such that any BGP announcement for a route in the prefix is ROV-NotFound, and most BGP routers will accept these announcements if they do not have e.g. a suspiciously long prefix. Without our attacker model, ROV simply offers no protection here because the prefix is not covered by a ROA. However, under our attacker model, the attacker can once again forge a ROA for the prefix that authorizes the attacker's ASN. When *only* the ROA chosen by the attacker exists, any *legitimate* BGP announcement for a route to the prefix will become ROV-Invalid, whereas the attacker's *illegitimate* announcements will be ROV-Valid. This makes it very likely that a router performing ROV will accept only the attacker's announcements, and drop any legitimate announcements. Here, it might also not be necessary to announce a more specific route than the legitimate announcements, as the legitimate announcements are likely to not be considered at all.

Mitigation Of course, a very simple mitigation exists to get from this scenario to the slightly less severe one in section 3.1.1: simply create a ROA for the prefix. This is a simple action that any AS should perform for their prefixes. However, the next attack shows that with some more effort, an attacker can also prevent such a legitimate ROA from being processed by a relying party.

3.1.3 Using revocation

The above attacks use only the publishing of a forged ROA. However, our attacker can similarly forge a CRL that revokes legitimate ROAs. Using this, any existing ROA can be revoked, degrading legitimate BGP announcements that were ROV-Valid to become ROV-NotFound and — once the attacker also publishes a forged ROA — ROV-Invalid, as shown in fig. 3.1. This attack can be done at any level in the RPKI hierarchy (including the trust anchors), so using only a few forged CRLs, the ROAs for large parts of the internet can be revoked.

Apart from publishing forged CRLs, which requires Capability 2, there are alternative ways to achieve the same result:

- Simply showing that the attacker can *create* a forged CRL or any other signature (even if they cannot publish it in a repository) compels the CA to revoke their certificate (section 4.9.1 of [RFC6484]). Hence, an attacker with only Capability 1 can still disable the protection offered by ROV.

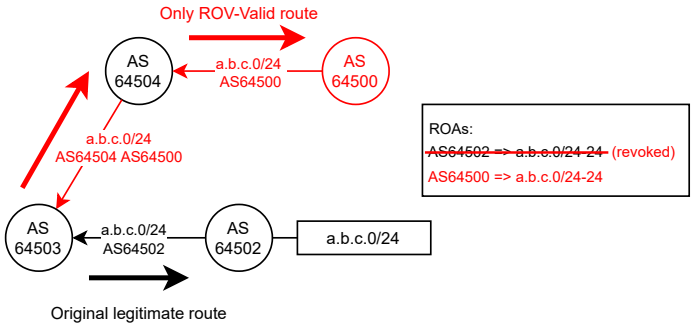


Figure 3.1: *An example of using a forged ROA and revoking a legitimate ROA to enhance a hijack. Due to the original ROA being revoked, the legitimate announcement is ROV-Invalid. Without revoking it, the attack would work on AS64504 but not on AS64503, as AS64503 is closer using the original route.*

- Instead of publishing a forged CRL, the attacker *may* cause the same effect by manipulating a manifest, although it is a matter of local policy what to do when there is a mismatch between a manifest and the other contents of a repository [RFC6486].
- The attacker can also simply DoS a repository, such that relying parties cannot download the objects in it, or, if the attacker is impersonating a repository already in order to achieve Capability 2 (see section 3.3.4), the attacker can simply prevent certain objects from being downloaded.

In summary, it is clear that there are currently many ways for a quantum-enabled attacker to disable the RPKI and protection of ROV as a whole. Due to the possibility of making legitimate routes ROV-Invalid, trust in ROV could be abused to perform attacks on BGP with a higher chance of success than without ROV being used at all. Using ROV with its current cryptography under a quantum-enabled threat model can result in significant disruption and makes attacks on BGP routing more likely to succeed than without ROV. Hence, it is very important to migrate the RPKI to use quantum-safe algorithms before a quantum-threat arises.

3.2 Attacks on ASPA

ASPA can be used on top of ROV for path validation. While of course, our attacker could modify ASPAs to *allow* any malicious route, bypassing the validation that ASPA provides, the manipulation of ASPAs can—like we have seen for ROV—have even more severe consequences.

By modifying or creating ASPAs for a victim AS, the attacker can easily make the victim unreachable, or reroute all traffic to the victim through the attacker's AS. Just as we've seen for ROV, the situation where ASPA validation is used, but an attacker can manipulate the ASPA objects in the RPKI, is worse than the situation where ASPA is not used at all.

3.2.1 Making an AS unreachable with an AS0 ASPA

The attacker can forge a so-called *AS0 ASPA* for a victim AS. Such an ASPA authorizes only AS0, which does not exist, as a provider, making any route to or through the victim AS ASPA-Invalid. This is a straightforward way to make a victim unreachable. When applied on an important AS such as a transit provider, or even several of

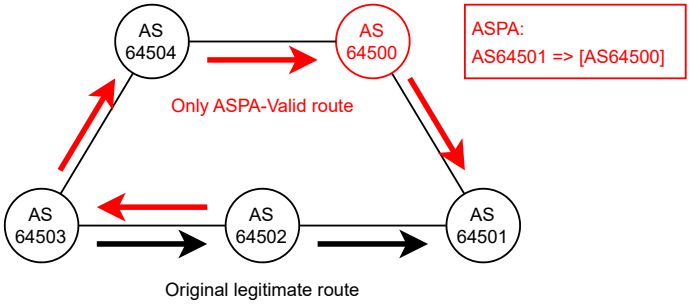


Figure 3.2: *An example of using a forged ASPA to enhance a hijack. Due to the ASPA, traffic is attracted even from ASes that have a much shorter route to the victim. Also note that this attack would pass ROV, and can pass or be adapted to pass BGPsec as well.*

them at the same time, the impact of this attack can be enormous—possibly even causing fragmentation of the internet, where large parts of the internet cannot reach other parts. Of course, a large-scale attack would quickly be noticed and can be mitigated by operators, but even a short period of such large-scale unreachability can be disastrous.

3.2.2 Enhancing a hijack with a forged ASPA

The above approach of invalidating legitimate routes can be combined with the strategy we have seen using ROV: invalidating legitimate routes to make a hijacking announcement more likely to be accepted. Using ASPA, this is very simple. Considering the malicious AS64500 who wants to eavesdrop, modify, or block all traffic to AS64501. The quantum-enabled attacker can simply publish an ASPA `AS64501 => [AS64500]`, which means that *only* routes through AS64500 as last hop will be allowed. Then AS64500 only needs to announce a prefix with AS path *AS64500 AS64501*, and all traffic to AS64501 will be rerouted through AS64500 (even from direct neighbors of AS64501 if they perform ASPA validation). An example of this attack is shown in fig. 3.2.

The attacks using ASPA seem to be even more straightforward than those using ROV, and the potential impact is even bigger because ASPA attacks can influence not only specific prefixes but also any route *through* a victim AS, making transit providers and internet exchanges an attractive target for large-scale attacks.

Performing the above attacks consists of either creating and publishing an ASPA for a victim AS that did not have an ASPA yet, or modifying an existing ASPA. Between these two, the former might be easier to perform, as there's no need to *also* take down the old ASPA. So, again just as with ROAs, ASes that already publish ASPAs are slightly harder to attack. *Adding* an attested provider can always be done by publishing an additional ASPA object, although it should be unusual for an AS to have multiple ASPA objects, and only adding a provider only works to bypass ASPA validation, not for the stronger attacks above.

Many variations of these attacks are also possible, including:

- Instead of an AS0 ASPA, an ASPA authorizing only non-adjacent ASes has the same effect.
- Instead of hijacking to eavesdrop or drop traffic, ASPAs could be used to route lots of traffic (e.g. from transit providers) through a small multi-homed AS that does not have the capacity to process the traffic, as DoS attack on that AS.

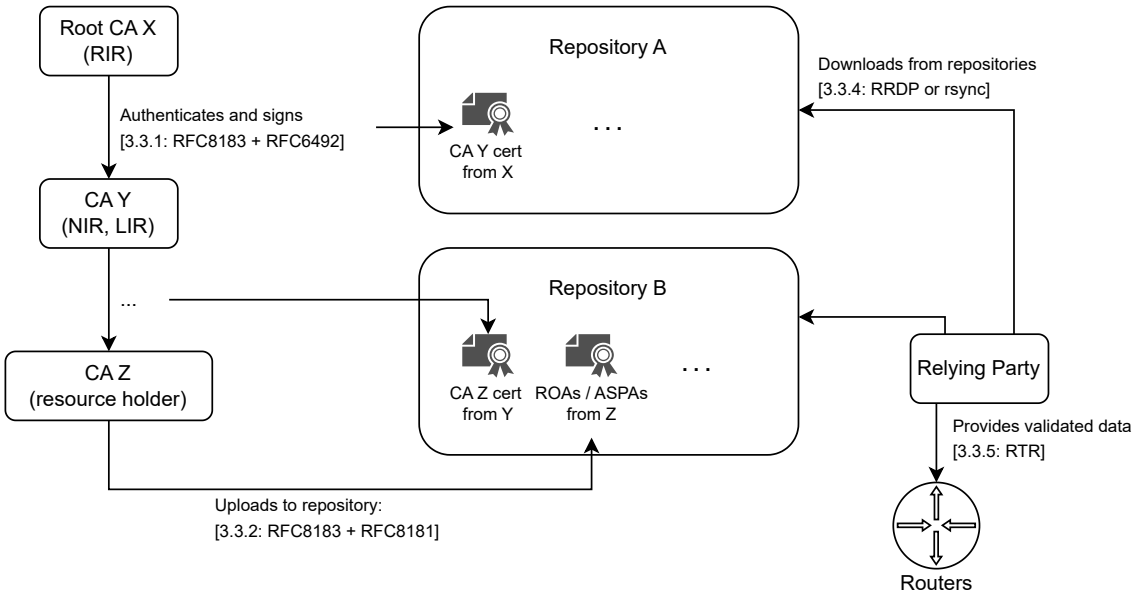


Figure 3.3: *Roles in the RPKI. Interactions are labeled with our corresponding subsections and related protocols.*

- Instead of enhancing a hijack with an ASPA from the victim AS attesting to the attacker AS, such a method can also be used at other steps along an existing path to the victim AS.

3.3 Setup and distribution

The certificates that are part of the RPKI itself are *only* meant to provide integrity of the objects in the RPKI from the phase of creating those objects, up to their validation by a Relying Party. However, the RPKI ecosystem consists of multiple components and phases that influence its security, such as setting up the relationships between parties, and while distributing data towards relying parties.

There is a reliance on several protocols to ensure availability (i.e. that Relying Parties can retrieve *all* and *up-to-date* RPKI objects), and the integrity *before* creating RPKI objects, and the integrity *after* validating them.

Figure 3.3 shows the roles in the RPKI and their relations. In this section, we discuss each relation in chronological order, starting with the setup of CA certificates, through publishing RPKI objects, fetching them for validation, and finally exposing validated data to routers. In each subsection, we show how the relation is involved in the security of the RPKI.

3.3.1 Parent to child CA

The relation between a Parent CA (e.g. an RIR) and a Child CA (e.g. an ISP) takes place using the [RFC6492] protocol, and out-of-band establishment of trusted public keys for both parties.

Public keys for this relation can (but do not have to) be exchanged using [RFC8183], which is a simple message format to exchange trust anchors between two parties, over an out-of-band authenticated channel. This trust anchor is, somewhat confusingly, called a *Business PKI* (BPKI) trust anchor, being potentially at the top of a hierarchy of multiple keys that can be used to sign [RFC6492] messages. In practice, the

BPKI TA is often used directly, and the term 'Business' in BPKI is an unfortunate choice.

Measures to ensure authentication and integrity of out-of-band channel are deliberately left unspecified, but in practice, this is often (for example, when setting up a child CA under the *RIPE NCC* RIR) done by uploading and downloading messages on a web interface. In light of a quantum threat, it is important — and the parent CA's responsibility — to ensure the security of this key exchange. For example, it is clearly crucial that the RIPE NCC's web interface is secured using quantum-safe TLS algorithms.

After setting up these public keys, further communication between parent and child is covered by [RFC6492]. This consists of messages that are signed using keys that are trusted based on an exchange as described above.[1] [RFC6492] specifies that the signature on its signed objects must be made using the algorithms specified in [RFC6485]. That is, the signatures on [RFC6492] messages use the same cryptography that is used for objects in the RPKI themselves. Currently, this is RSA, which we consider vulnerable to quantum attacks. Hence, when migrating the RPKI to quantum-safe algorithms, the [RFC6492] communication must be updated as well.

If the cryptography in [RFC6492] is broken, an attacker can impersonate a child CA to a parent CA, to get or revoke certificates for any resource held by the child CA. This would be a straightforward attack that needs *only* the capability to forge a signature on an [RFC6492] message, without needing to cause forged objects to be published in a repository that is not run by the attacker (Capability 2). The attacker can forge a request for a new certificate, with both the subject key and the publication point chosen by the attacker. So, using a single forged message, the attacker can take over a CA, allowing them to make and publish objects at will.

Thus, the channel to a parent CA may be the single most attractive target for an attacker. The only limitation is that not all resources are held by subordinate CAs that communicate with the parent using [RFC6492] with RSA signatures: resources held by users of an RIR's *hosted RPKI* service as discussed in section 3.3.3 are not vulnerable to this attack.

3.3.2 CA to repository

Every RPKI CA can choose their own repository to publish to, which can be, but does not have to be, run by the resource holders themselves. When a CA and the corresponding repository are run by the same entity—including when a *hosted RPKI* service (section 3.3.3) is used—communication between the two roles could be internal to a single piece of software, or take place on a private network that an attacker cannot access. Nonetheless, many CAs use an external repository, as hosting a repository comes with the operational burden of ensuring high availability, and using centralized repositories is beneficial for the downloading performance of relying parties.

In general, the communication channel (public keys) between a CA and repository is bootstrapped the same way as those between parent and child CAs: typically using [RFC8183], relying on unspecified measures for authentication and integrity. When such public keys have been set up, the CA can instruct the repository to publish or withdraw objects, by sending signed messages over plain HTTP or optionally HTTPS. The signatures [RFC8181] messages are identical to those in [RFC6492] described above in section 3.3.1, so likewise, this communication should be updated to use quantum-safe algorithms. On the other hand, the impact of compromise of this channel is different from that of the parent-child channel: it allows an attacker to withdraw and publish objects, but not create them. For our quantum-enabled attacker,

[1] For example, the keys used to sign [RFC6492] messages can include a certificate chain up to the (so-called *Business PKI*) trust anchors exchanged using [RFC8183].

this means that as long as this channel does not use quantum-safe signatures, it is an obvious avenue to get capability (b). Additionally, if the RPKI itself were to be migrated to use quantum-safe algorithms but [RFC8181] were not, a quantum-enabled attacker could still perform denial-of-service attacks by withdrawing objects, which can cause BGP routing failures.

3.3.3 Hosted RPKI

In some cases, an RIR or NIR offers its members a *hosted RPKI* service. That is, the registry runs a CA on behalf of the user, taking care of all cryptographic operations, as well as publication to a central repository. This allows resource holders to create and publish ROAs and possibly ASPA objects very easily through a web interface, making adoption more attractive. These services are used by the vast majority of resource holders.[2]

Using a hosted RPKI means that several roles are combined into one entity: the RIR that offers it combines the CA and repository roles, and instead of using [RFC6492, RFC8181] for communication between these roles, there is a single channel (such as a web interface) where users can manage their resources.[3] Security of this system then depends on the RIR's systems and the authentication and integrity of the user-facing interface. Hence, much like the bootstrapping of public keys between parent and child CAs, it is necessary for internet registries that offer a hosted RPKI service to ensure that their web interface is secured using quantum-safe TLS algorithms.

3.3.4 Repository to RP

Once RPKI objects are published in a repository, relying parties need to download them for validation. Originally, this was done using `rsync` [RFC6480]: each repository had, and still must have, a public, read-only `rsync` location. In this setting, relying parties periodically download all objects from all repositories over a clear text TCP connection.

More recently, the RPKI Repository Delta Protocol (RRDP) has been introduced. This solves scaling and security issues in the old `rsync` alternative: RRDP takes place over HTTPS, which allows a repository to scale horizontally and provides integrity and authentication of the repository. With the introduction of RRDP, it is more difficult to perform Denial-of-Service (DoS) attacks on the RPKI by attacking distribution points, and due to the use of HTTPS, to impersonate an RRDP publication point. However, if one manages to take down an RRDP publication point, relying parties will typically[4] fall back to the corresponding `rsync` publication point, which in turn is easy to attack. Hence, performing DoS or becoming Man-in-the-Middle (MitM) between a relying party and a repository is possible using a downgrade attack to the unauthenticated `rsync` protocol. In particular, it is realistic to assume that a quantum-capable attacker—who is clearly advanced and well-resourced—can pull off a relatively simple DoS and optionally MitM attack, and hence, has Capability 2.

[2] For example, at the time of writing, RIPE's main repository has over 20 000 users of their hosted CA service, and fewer than 300 resource certificates pointing to another repository. The latter even include so-called *hybrid CAs*, that perform the CA role themselves but still use a repository provided by RIPE.

[3] Note that even in a *hosted RPKI* service, there might be communication between CA and repository components where sections 3.3.1 and 3.3.2 apply, due to separation of concerns within such a service.

[4] This decision is controversial. [38] measured that few RPs actually downgraded to `rsync` if an advertised RRDP publication point was unavailable in 2020. However, soon after that, Routinator was changed to fall back: see `https://blog.nlnetlabs.nl/why-routinator-doesnt-fall-back-to-rsync/`.

3.3.5 RP to router

Finally, there is a step between the validating software and the actual router,[5] using the RPKI-to-Router (RTR) protocol [RFC6810, RFC8210]. This final link relies strictly on transport security, and hence should take place either over a cryptographically secured channel or on a trusted network; see sections 9 and 14 of [RFC8210]. If unprotected transport is used, the operator should ensure that even a quantum-enabled attacker cannot compromise the trusted network; necessary measures for this depend on the specific situation and are an operator's responsibility. For transport over one of the protected protocols in [RFC8210], changes might be necessary to make them quantum-resistant. We consider the security of these transport protocols out of scope for this thesis, but some may be quantum-safe already, while others need changes.

3.3.6 Reliance on other technology

Across the steps involved in setting up and distributing RPKI objects, we have seen that security relies on core components of the web infrastructure. While we do not provide a complete analysis here, we highlight two examples of such dependencies:

HTTPS The integrity provided by HTTPS and authentication using the Web PKI is crucial in hosted RPKI services, the setup of relations between CAs, and the downloading of RPKI objects over RRDP.

DNS(SEC) For availability, the DNS is crucial for virtually every part of the RPKI ecosystem. It is involved for example in locating repositories from relying party software, and could be an avenue to DoS the RPKI or become MitM. [30] shows that DNS is a weakness for the resilience of many relying parties and that a significant number of relying parties do not properly perform DNSSEC validation. Improvement in that regard, as well as upgrading DNSSEC to use quantum-safe algorithms, is necessary to protect the RPKI from DoS attacks and quantum-enabled DNS spoofing.

This is not an exhaustive inventory of all related technologies and protocols that the RPKI ecosystem depends on. For example, there are multiple transport options listed in [RFC8210] that need to provide integrity and authentication, and while those same properties are required from HTTPS connections for RRDP, confidentiality can also be essential in password-based authentication to RIRs' web interfaces. A comprehensive inventorization of all related protocols and the specific security properties that the RPKI requires from each of them is left for future work.

3.4 Conclusions

We have seen severe attacks, that can hurt routing security beyond simply not enjoying the protection of ROV and ASPA:

- ROV with broken cryptography can be abused to successfully perform hijacks, that would not succeed if there was no ROV at all. This is even easier with ASPA.
- There is also the possibility to (temporarily) make large portions of the internet unreachable, by making many announcements invalid.

As a consequence, when it is known that these attacks are realistically feasible (perhaps if they occur for the first time in practice), operators are left with no choice but to

[5] There may be a caching layer between validators and relying parties, but such a cache would normally connect to both validators and routers over the same RTR protocol. See for instance RTRTR: `https://github.com/NLnetLabs/rtrtr`.

disable use of the RPKI, falling back to a world without ROV and ASPA. This means that the attacks we have described can probably be performed at most once, and have effect only temporarily until operators manually intervene. However, it also shows that migrating to post-quantum algorithms is strictly necessary to ensure that the RPKI can be used safely. With broken cryptography, the RPKI is not just ineffective, but a liability.

Furthermore, we have found in section 3.3.1 that the most attractive target for a quantum attacker is the communication between a parent CA and a child CA, using [RFC6492], and usually secured based on a trust anchor established through [RFC8183]. This channel is an easy target because it uses a long-lived key without a revocation system, and does not require the capability to inject objects in repositories. This channel would be a good first thing to migrate, also because it is a local matter between CAs, invisible to RPs. The same goes for the channel between CA and repository, although it is a less attractive target. Of course, the certificate chains in the RPKI itself also need to be updated.

Additionally, there are several related protocols that need to be quantum-resistant too. Most important are the web interfaces of internet registries, such as those of hosted CA offerings, and for setting up [RFC8183] trust anchors between CAs. This includes the TLS layer, but also user authentication. There's also a general reliance on security of protocols like DNSSEC and HTTPS, for instance to avoid DoS attacks on repositories.

Chapter 4

Finding a suitable post-quantum algorithm

We have seen that the use of RSA prescribed in [RFC7935] will need to be replaced to prevent the serious quantum-enabled attacks from chapter 3. By replacing RSA or adding a new algorithm, both the chain of trust down to an RPKI signed object, and the communication between RPKI CAs and repositories (as discussed in sections 3.3.1 and 3.3.2) — when authenticated using [RFC8183] — can be secured against quantum attacks.

In this chapter, we will determine the requirements for a new algorithm that can replace RSA in this context, and evaluate several candidate algorithms against these requirements.

Several factors determine how suited an algorithm is for use in the RPKI. Considering that post-quantum algorithms usually have larger keys and signatures, and slower signing and verification times (or at least some of these properties) compared to RSA, the main considerations are (1) that a replacement obviously needs to be secure, and (2) that it should minimally impact the performance of the RPKI as a whole.

Throughout this chapter, we assume that the current signature algorithm is replaced with a single post-quantum alternative, and that this alternative is adopted in every object. This is because a migration procedure has been standardized that *requires* that only one signature algorithm is allowed. Consequently, a state of partial adoption is not possible (after the migration). However, in chapter 6 we present an alternative migration procedure, that also supports having multiple allowed signature algorithms at the same time. While we recommend the use of this alternative migration, and indeed making multiple algorithms available, we do not consider it in this chapter to keep analysis of the performance impact simple.

4.1 Security

A post-quantum signature algorithm must primarily be secure, both against traditional attackers and against quantum adversaries. The NIST PQC candidates each have a target security level, ranging from level 1 ('harder to break than AES-128') up to 5 ('harder to break than AES-256'). The security level is a measure of the number of operations that are needed to break the algorithm, and is a good way to compare the security of different algorithms. Most algorithms come with several parameter sets, trading larger keys/signatures and slower signing/verification for higher target security levels. This means that a decision will need to be made on the security level that is required or desired for the RPKI.

4.1.1 Minimum level

A minimum security level could be based on the traditional security offered in the current situation by RSA-2048. This provides 112 bits of traditional security and is not recommended by NIST after 2030 [47]. NIST level 1 corresponds to at least 128 bits of security in the traditional sense, so it should be significantly more secure than RSA-2048. So, even level 1 candidates are good enough as a replacement for RSA-2048.

4.1.2 Future-proof choice

While a level 1 candidate may suffice for the coming years, there are good reasons to opt for a higher level. Changing the signature algorithm used in the RPKI is not straightforward: only a single algorithm is currently allowed, and as every relying party needs to read every object, significant coordination is needed to make a change. In contrast to for example TLS (where client and server can negotiate a cipher suite they both support), all parties in the RPKI need to support the same algorithm(s) at once. So, changing the algorithm suite *again* a short time after the first change is not desirable. This makes it wise to make a very conservative choice and pick an algorithm that will likely last for many years if other constraints allow it.

Another argument in favor of a conservative choice is that, again due to the need for coordination in the RPKI, it is necessary that all parties involved agree on the choice of algorithm. If some people do not trust a proposed choice, it will be hard to get the consensus needed for a smooth transition.

On the other hand, NIST level 1 already offers more traditional security than RSA-2048, so it should remain sufficiently secure for many years to come. It is perhaps more likely that in the coming years a second migration is required due to a weakness in a particular scheme, than due to security level 1 being considered too weak, so picking a higher level is probably not necessary. Moreover, using a scheme with NIST target level 3 or higher could come with some complications. The RPKI uses SHA-256 in many cases, which could become the weakest link, such that it also needs to be replaced. This is not a problem, but adds some complexity and increases some files' sizes. Because it would complicate our analysis and comparison of performance, we do not consider it in this thesis.

4.1.3 Maturity

Apart from the NIST target level of a candidate, there are differences in the confidence the cryptographic community has in the security of each algorithm. Some schemes are well-understood, and people are confident that they indeed provide the security claimed in their NIST level. Others are based on newer ideas, and need more years of cryptanalysis to be trusted. Using a well-understood algorithm can be a good way to reduce the risk of having to replace the algorithm again soon if a weakness is found. This, in turn, would make consensus in the RPKI community easier to achieve.

Furthermore, the algorithms that are already standardized and deployed in practice (ML-DSA, SLH-DSA, and soon FN-DSA) will have a head start towards getting production-ready implementations, and support by Hardware Security Modules (HSMs) that are sometimes used by CAs. Standardization of other algorithms will take several years, making the 3 selected options more attractive on the short term.

4.1.4 Hybrids

Another option to mitigate the fact that some candidate signature algorithms are not yet well-understood is to use a hybrid scheme, combining a post-quantum signature with a traditional one. This way, even if the post-quantum scheme is broken, an

attacker still also needs to break the traditional scheme. Doing this guarantees that after migration, the RPKI will still be at least as secure as it is now. For most post-quantum signatures, using a hybrid is imperative, at least for the coming years, as their security is not understood enough. However, hash-based schemes, that can be proven secure assuming only the security of an underlying hash function, are an exception. So, while for all other schemes we need to account for some computational and size overhead for a hybrid scheme, hash-based signatures could be used without a hybrid.

In conclusion, NIST target security level 1 suffices in principle, as it is already an improvement from RSA-2048 even in terms of traditional security. Of course, higher levels would be nice to prevent having to upgrade again in the future. Maturity of an algorithm is also important, so we prefer a well-understood scheme. A hybrid scheme is probably necessary, which implies some overhead on top of the already larger keys and signatures of post-quantum algorithms.

4.2 RPKI latency

The performance of the RPKI as a whole is primarily reflected by the time it takes for a change made by a resource holder to be picked up by relying parties globally. We call this the *RPKI latency*. Current validators like Routinator periodically download the latest changes (or sometimes the full contents, if there is no data cached), and validate them. The latency is then determined by the validator's polling interval, the time to actually perform downloading and then to verify the objects. The polling interval is configurable (often set to 10 minutes) [38] and not affected by the signature algorithm, but the downloading and verification times are strongly related to the signature algorithm that is used.

In [65], the downloading and verification delays are analyzed in a variety of settings. It is found that there are large differences in performance between repositories, validators, and regions. While it is important to realize that there are so many factors involved in the RPKI latency, what we need is a *relative* measure of the performance when using a specific algorithm. We present two methodologies to estimate the downloading and verification times, which can later be applied to any candidate algorithm.

4.2.1 Downloading

Since post-quantum signatures are usually larger than those for RSA, the size of RPKI objects, and hence the time needed to download them will increase when using a post-quantum algorithm. To make a well-informed decision for an algorithm, we need to estimate the performance impact of the new algorithm on the downloading time.

Normally, a validator downloads a few changes at a time, but sometimes a full download of a repository is necessary, either because the validator lost its cache, or because the repository starts a new RRDP session. Full downloads (from a single repository or all repositories at once) will be delayed by larger signatures and keys, much more than the more common small updates (*deltas*), so they are the focus of our analysis. Furthermore, most of the RPKI data is normally collected using RRDP. While `rsync` is still mandatory to support, it is typically used for only a small fraction of the data, and it is likely to be phased out even more in the future. So, we try to predict the delay for downloading the full RPKI using RRDP. For downloads over `rsync`, it still holds that larger files will take longer to download, but it's even harder to accurately predict the delay this causes. Measuring or estimating the impact on `rsync` downloads, as well as deltas, is left for future work.

Fresh RRDP downloads happen by first fetching a *notification* file from every repository, which includes the location of a *snapshot* file. Then, RPs download the snapshot, which is an XML file with all current RPKI files base-64 encoded in it. So, for a full download of a repository (not an update using delta files), the downloading consists of 2 files per repository. Out of these, the snapshot file is the only one whose size depends on the signature algorithm.

Downloading a snapshot from a repository takes (1) a delay to establish a connection, (2) a delay for requesting the files, and for the server to start responding, (3) the time for downloading the notification file, and finally (4) the actual transfer time for the snapshot. Roughly speaking, there's a part (1, 2, and 3) that does not change with the size of the RPKI, and a part (4) that takes time proportional to the size of the content. In a simple formula, one might model the downloading time as:

$$t_{download} = t_{const} + \frac{s}{\mathrm{b}}$$

where t_{const} is the constant part of the delay (including downloading the notification file that does not grow with the signature and key sizes), s is the size of the RPKI, and b is a measure for the transfer rate at which the actual RPKI content is downloaded.[1]

Since only s depends on the signature algorithm, the delay caused by a new signature algorithm is proportional to the difference in s between the new and old algorithms. To then obtain the change in delay caused by a certain choice for an algorithm, we need to know:

- the (change in) the total size of the RPKI, i.e. s for current and new algorithms;
- how large the size-dependent fraction of the downloading time with RSA is, i.e. $\frac{s}{b}$ for RSA.

4.2.1.1 RPKI size

The objects that make up s can be either RPKI Resource Certificates ([RFC6487]) or CRLs, or RPKI Signed Objects ([RFC6488]). The majority of objects are Signed Objects, which contain one public key, and two signatures each. Resource Certificates consist of one public key and one signature, and the corresponding CRLs have one signature each. Overall, there must then be twice as many signatures as public keys in the RPKI. This is confirmed in table 4.1. Hence, assuming that the structure and content of the RPKI remain similar, the size of the RPKI changes proportionally to:

$$s_{pk} + 2s_{sig}$$

where s_{pk} and s_{sig} are the sizes of the public key and signature of a new algorithm, respectively. This formula is a good way to compare the combination of algorithms' signature and public key sizes for the RPKI.

Next, we can also predict precisely the total size s of the RPKI for a given algorithm, based on a recent snapshot of the RPKI. We count the size and number of files per type in the RIPE NCC RPKI archive [14] snapshot from the first of February 2025.[2] This gives the total size (corresponding to s in the formula above) of the current RPKI, and the number of signatures and public keys in it. These numbers are shown in table 4.1. Indeed, the table reflects the expected $2:1$ ratio between signatures and

[1] For simplicity, we ignore overhead for the RRDP protocol, such as base-64 encoding, included hashes over the files, but also possible savings by compression at the HTTP level. Most of these factors do not change much with the algorithm: the number of files and hence hashes, URLs and XML elements do not change. Compression is already unlikely to do very much on the file *contents* (instead affecting mainly the more redundant XML and base-64 encoding of the RRDP layer), as they consist largely of cryptographic material that does not compress well.

[2] We count in the `rta/unvalidated` directories for each TA. The `validated` directory contains duplicates of a subset of these files, that pass validation.

public keys. We can use the number of signatures and public keys later on to predict the growth from an algorithm's signature and public key size.

Table 4.1: *Number of files, signatures and public keys, and total size per type in the 2025-02-01 RPKI archive snapshot.*

File type	Files	*sig*/file	*pk*/file	Signatures	Public keys	Total size (bytes)
Res. cert.	45 135	1	1	45 135	45 135	70 545 531
CRL	47 197	1	0	47 197	0	35 175 003
Manifest	47 208	2	1	96 416	47 208	122 831 055
ROA	300 845	2	1	601 690	300 845	609 042 128
ASPA	239	2	1	478	239	436 733
Total	440 863			788 916	393 427	838 030 450

4.2.1.2 Bandwidth: worst-case

Knowing the size of the RPKI, we next need to translate changes in size to changes in downloading time. This is much harder to predict, as it depends very much on individual repositories and relying parties:

- How and where is the repository hosted? Using a global CDN, or from a single location?
- How is the bandwidth and latency between a specific validator and repository?

When using RRDP, the downloading is aggregated in one large snapshot file, instead of separate requests for each RPKI object. This means that there is relatively little time spent on establishing connections and sending requests. Nevertheless, since not only establishing connections, but also fetching notification files and other constant delays are involved, we expect that in our formula $t_{download} = t_{const} + \frac{s}{\mathrm{b}}$, the constant part t_{const} is not negligible.

Since the downloading time depends so much on specifics of a validator, there are no good statistics available. Still, [25] report (without much explanation) that the downloading of the RPKI takes 4 minutes. This seems consistent with the duration of downloading a fresh copy of the RPKI on a laptop with a decent internet connection, but makes no distinction between the time of actually downloading data ($\frac{s}{b}$) versus the constant delays (t_{const}).

In the absence of more precise numbers, the best we can do is to derive a strict *upper bound* on the part of the downloading time that depends on the RPKI size. While we expect t_{const} to be significant, we can assume for a 'worst-case' estimation that $t_{const} \approx 0$, and then:

$$4 \text{ minutes} \approx \frac{s}{b}$$

Filling in the total size $s = 838$ MB from section 4.2.1.1, we get a 'worst-case' bandwidth of $b = \frac{1632 \text{ MB}}{240 \text{ s}} = 3.5$ MB/s.

Notably, this is a very conservative estimate. As t_{const} is actually significant, the true average bandwidth must be higher, or in other words, the size-dependent downloading time $\frac{s}{b}$ must be shorter. So, when calculating with $\frac{s}{b} = 240$ s we are overestimating the delay caused by the signature algorithm, making any conclusions based on this number conservative and safe.[3]

[3]Furthermore, the delays for a complete download of the RPKI occur infrequently in practice. Typically, only a few changes are downloaded at a time, but still, it is desirable that complete downloads don't take excessively long.

4.2.1.3 Bandwidth: measurement

To get a better estimate of the bandwidth for normal validators, we perform our own measurements, without assuming $t_{const} \approx 0$.

We perform measurements of $\frac{s}{b}$ directly, using an instrumented variant of Routinator. This modified validator logs every successful request for an RRDP snapshot, with the total duration for that request and the total size of the objects included in it. With this, we have performed full downloads of the RPKI 10 times, running on a machine in a data center to simulate a realistic deployment of a validator. Our finding is that the average time spent downloading RRDP snapshots is only $\frac{s}{b} = 14.5$ s. Much more time is spent on downloading notification files (29.5 s) and waiting for timeout of unsuccessful requests (for a total duration of around 5 minutes). More details on our methodology and results can be found in appendix A.

Our measured 14.5 s is from a machine with a good bandwidth to the internet of roughly 2 Gbps. This can be realistic for some validators that are running in data centers, but there may also be many validators with a lower bandwidth. So, our 14.5 seconds is a slight underestimation for many validators. The real cost for downloading will generally be between the 14.5 s and the 240 s, but closer to the former than the latter.

When comparing candidates in table 4.3, we provide estimates of the downloading time using both the assumption of $\frac{s}{b} = 240$ s and $\frac{s}{b} = 14.5$ s.

4.2.2 Validation

After downloading new data, the validator needs to parse the objects and perform validation, including verifying the signatures.

The time to validate all objects again consists of a part that does not change depending on the signature algorithm (parsing and most validation steps), and a part that is roughly proportional to the cost of verifying a signature. So here, the difference in latency for validation, assuming no other changes to the RPKI content, is simply proportional to the difference in signature verification time between the old and new algorithm.

To get an idea of how the validation time compares to the downloading time, we again need to know how long it current;y takes. In the current RPKI, validation time is not problematic. Where [25] observe that the downloading contributes 4 minutes of latency, the processing time adds only 1 more minute. However, this statistic says nothing about the fraction inside these durations that is affected by the signature algorithm, so we attempt to estimate this.

4.2.2.1 Signature verification time

We perform a small experiment to measure the CPU time that is currently spent on validating RSA signatures in Routinator.

- We build and install Routinator v0.14.1, on a laptop with Ubuntu 24.04, 32 GB RAM and Intel i7-1255U.
- We first download a full copy of the RPKI, using `routinator --fresh update`. This ensures that no downloading is necessary in the next steps.
- Repeating 30 times, we measure the time it takes to validate the downloaded RPKI copy:

```
/usr/bin/time -f "%e, %U, %S" -a -o verify_time.csv \
routinator --validation-threads=1 -q \
    vrps --noupdate -f none
```

This performs validation of the full RPKI, with only one thread, and without producing any output.

- Next, we modify Routinator to use a dummy signature verification that immediately returns `Ok(())`[4] and install this. Using this, we can measure how much time is spent parsing and processing objects, without the actual signature verification.
- Again, we measure the validation time 30 times.

Table 4.2: *User CPU time spent on validation with and without signature verification (10 executions per scenario).*

Scenario	user time (s)	std. dev. (s)
With verification	18.21	0.74
Without verification	5.21	0.23

So, as can be seen from table 4.2, the RSA signature verification on general-purpose hardware takes about 13 CPU seconds, out of the total 18.2 seconds per validation run with RSA.[5]

4.3 Signing and key generation

Key generation and signing speed are not very important in the RPKI, as it happens in the background, and much less frequently than verification. We can distinguish between three situations where signing and key generation happen:

- Key generation of a CA's key pairs happens only incidentally, involving only a single key at once.
- Signing using a CA's key pair is infrequent,[6] but can sometimes happen thousands of times in a short time frame. It is hard to parallelize (with more CPU cores) when CAs employ physical Hardware Security Modules (HSMs).
- Generating and signing with one-time key pairs happens as often as signing objects with a CA's key pair (see chapter 5). However, as these private keys needn't be stored in an HSM, it is easier to parallelize this operation. In chapter 5, we also propose a way to avoid the need to make these signatures at all.

So, key generation is not much of a concern, but signing speed could be, particularly in the following situations where many signatures are made in a batch.

4.3.1 Re-issuance during key rollover

Signing many objects happens in the most extreme case in a [RFC6489] key rollover. During a rollover, a CA must reissue its subordinate certificates and Signed Objects with a new key pair. For the very largest manifests to date, that means re-signing up to around 50 000 objects. [RFC6489] prescribes that normally, the CA should have a staging period of *at least* 24 hours. Even for such an extremely large CA, there is normally ample time to reissue everything even if signing takes up to 1 second.[7]

[4] A small change in `PublicKeyFormat::verify` at `https://github.com/NLnetLabs/rpki-rs/blob/v0.18.5/src/crypto/keys.rs#L138`.

[5] Our raw data is at `https://github.com/SIDN/pqc-rpki/tree/main/algorithm-selection/verification-measurements`, or can be requested from the author or Radboud University.

[6] This also goes for signing messages to publication servers and parent / child CAs.

[7] At 1 signature per second, signing all objects with the CA's key takes 14 hours. The additional key generation and signature per object, not counted in the 14 hours, is not problematic, as it concerns one-time key pairs, that can be parallelized easier, for example without using an HSM. Alternatively, for such large CAs, a staging period of a few days can also be used.

4.3.2 Changes to aggregate ROAs

Another situation in which it may be necessary to perform many signatures in a short time frame is when many new Signed Objects have to be made. For example, consider a very large CA that performs ROA aggregation, so there are n ROAs (for n ASes) that each include at least 2 prefixes, including a single prefix p that is authorized for every AS. If the CA loses holdership of p (it is removed from the CA's resource certificate), the CA will need to quickly reissue every ROA to not include p, to avoid all ROAs being invalidated through fate-sharing described in [RFC9455].

It is unclear how many signatures and in what time frame this could need to be done in practice. Normally, such changes to a resource certificate would not be unexpected, and the subordinate CA can prepare for it in advance, but it is a possible situation, in which it is very important that a CA can resolve it quickly.

To be safe, considering that several CAs currently have multiple thousands of objects, we can assume that it might be necessary to sign thousands of objects in a short time frame. Then, having a signing time of significantly less than a second per signature is a good idea.

Most post-quantum signature candidates have signing and key generation speeds that are nowhere near the 1-second mark on general-purpose hardware, and for these, any difference in signing speed is inconsequential. Very slow signing is no problem for most CAs and for day-to-day operation of the biggest CAs, but may either complicate key rollovers, handling changes to the CA's resource certificates, or require potentially complicated hardware upgrades for the largest CAs. Hence, a suitable signature scheme should have a signing time of not more than a fraction of a second, to avoid operational issues for large CAs.

4.4 Candidate algorithms

In the previous sections, we have found the following requirements:

Security NIST target level 1 suffices. However, a conservative choice would be good to avoid needing another algorithm migration too soon. Regardless of target level, maturity is an important factor. For most schemes, a hybrid with a traditional scheme is necessary.

Latency The next most important factor is performance for relying parties. Sections 4.2.1 and 4.2.2 propose a method we can use to estimate the impact each algorithm has on the delay for a full download and validation run.

Signing While precise signing speed has little impact on the RPKI, it is desirable to be able to easily make several signatures per second. Otherwise, this can lead to operational concerns for large CAs that incidentally need to make many signatures quickly.

Now, we can evaluate several promising post-quantum signature algorithms and parameter sets against these requirements. Since every candidate has a sufficient claimed security level, we start off estimating performance impact, and after that, discuss other aspects per candidate.

4.4.1 Latency

Using the estimated effects on the RPKI latency from sections 4.2.1 and 4.2.2.1, we present in table 4.3 an overview of several promising post-quantum signature algorithms and parameter sets. The table assumes full adoption for all signed objects

and certificates, and does not account for overhead for a hybrid with a traditional scheme.

We include the current RSA-2048 for reference, and Ed25519 as another comparison that has been considered as a candidate replacement algorithm in the RPKI for its small keys and signatures. Next, ML-DSA and SLH-DSA have been standardized by NIST, and Falcon is expected to be standardized very soon. SQIsign, MAYO, HAWK, FAEST and SNOVA are interesting candidates for NIST's call for additional signature schemes. We include only algorithms and parameter sets that do not have extremely large keys or signatures or slow verification, as there are many such candidates that are definitely not suitable for the RPKI. All the numbers are based on the overview from [78]. For many candidates, these are still subject to change, and the benchmarks are taken from each scheme's submission document and might not be entirely accurate or comparable. New attacks that break the schemes may also be found, especially for the candidates for NIST's call for additional signatures.

Table 4.3: *Characteristics of several post-quantum signature algorithms. The downloading time estimates are based on a baseline of 240 or 14.5 seconds, corresponding to the worst-case from literature (4.2.1.2) and our more realistic measurement (4.2.1.3) respectively. The ratio between these two columns is constant by definition. For verification time, we use a baseline of 13 CPU seconds for RSA verification (4.2.2).*

			Sizes (bytes)			Estimated times (seconds)		
Algorithm	Param.	NIST level	*pk*	*sig*	Total RPKI size	Download (worst-case)	Download (measured)	Verification CPU time
RSA	2048	-	272	256	**838 MB**	**240.0**	**14.5**	**13.0**
EdDSA	Ed25519	-	32	64	592 MB	169.6	10.2	37.6
ML-DSA	44	2	1 312	2 420	3.0 GB	846.1	51.1	34.2
	65	3	1 952	3 309	3.9 GB	1 119.1	67.6	51.8
	87	5	2 592	4 627	5.2 GB	1 489.0	90.0	80.9
Falcon	512	1	897	666	1.4 GB	403.1	24.4	23.4
	1024	5	1 793	1 280	2.2 GB	642.7	38.8	46.4
SLH-DSA	SHAKE-128s	1	32	7 856	6.7 GB	1 930.1	116.6	1 376.3
	SHAKE-128f	1	32	17 088	14.0 GB	4 015.9	242.6	3 729.5
SQIsign	I	1	65	148	671 MB	192.3	11.6	1 473.3
	III	3	97	224	744 MB	213.1	12.9	5 373.3
	V	5	129	292	810 MB	232.0	14.0	10 313.3
MAYO	1	1	1 420	454	1.4 GB	414.1	25.0	44.3
	2	1	4 912	186	2.6 GB	747.0	45.1	16.3
	3	3	2 986	681	2.2 GB	641.8	38.8	100.5
	5	5	5 554	964	3.5 GB	995.1	60.1	246.7
HAWK	512	1	1 024	555	1.4 GB	392.3	23.7	42.8
	1024	5	2 440	1 221	2.5 GB	702.3	42.4	87.5
FAEST	128s	1	32	4 506	4.1 GB	1 173.2	70.9	2 826.2
	128f	1	32	5 924	5.2 GB	1 493.6	90.2	408.2
	EM-128s	1	32	3 906	3.6 GB	1 037.6	62.7	2 137.2
	EM-128f	1	32	5 060	4.5 GB	1 298.3	78.4	321.5
SNOVA	(24, 5, 4)	1	1 016	248	1.1 GB	322.0	19.5	47.3
	(25, 8, 3)	1	2 320	165	1.6 GB	450.2	27.2	63.2

Our data and calculations can be found on `https://github.com/SIDN/pqc-rpki/tree/main/algorithm-selection/candidates`, and on request from the author and Radboud University.

Table 4.3 shows that there is a wide range in both signature and public key sizes, and verification speed. A trade-off needs to be made between the increase in downloading and verification time, as well as some other properties of the candidate algorithms. While it may seem logical to just pick the algorithm with the smallest sum of downloading and verification time (for one of the downloading estimates), this is not necessarily the best choice. Several limitations apply to the estimates in the table:

- One of the two downloading time estimates is based on the *upper bound* where downloading time is proportional to a duration of 240 seconds. This is certainly an overestimation.
- The other downloading time is based on the 14.5 seconds we measured in appendix A. This is a more realistic estimate, but from a very good internet connection. The majority of validators will have a somewhat slower connection.
- Furthermore, in both columns we display the estimate of only the $\frac{s}{b}$ time. There is also some constant time that is not affected by the signature algorithm. Especially the downloading estimate based on a 14.5 s baseline is *not* the total time it takes to download the RPKI, which would be roughly 4 or 5 minutes more.
- Similarly, the verification delay is about single-core performance. In practice, verification can be parallelized heavily, although that does come at some cost.

Furthermore, the numbers assume that the structure of the RPKI and the computational resources of validators remain the same. In practice however, some optimization is possible for both of these aspects.

4.4.1.1 Changes to RPKI structure

Currently, some CAs perform *ROA aggregation*: combining many small ROAs (that concern e.g. a single prefix) for an AS into a single larger ROA with all prefixes for that AS. This practice can greatly reduce the size of the RPKI, because the per-object overhead of a ROA is very large compared to the content: a single-prefix ROA can be over 1.5 KB, despite containing only a few bytes of useful data (the prefix and the max length). Still, [RFC9455] generally recommends against this practice, because it leads to the "shared fate", where if a single aggregate ROA becomes invalid (e.g. it expires, or one of the prefixes is removed from the CAs resource certificate), every prefix in the aggregate ROA is no longer considered by relying parties.

Most RIRs do aggregate ROAs in their hosted RPKI service. This does not actually produce the risks discussed in [RFC9455], because the users indicate only which announcements they want to allow, and the actual ROAs are made automatically. Expiry and revocation are safely handled without manual intervention.

One example of a CA that did not aggregate ROAs led to an incident on the 30th of January 2025. A large CA had published over 50 000 objects. This is not only inefficient in terms of repository bandwidth, but because the CA's manifest includes the name and hash of each object, it also tripped manifest file size limits in some old replying party software [73]. [65] and [74] show that, especially under the ARIN trust anchor, there is much room for optimization through ROA aggregation.

If a new algorithm with larger public keys and signatures causes the overhead per signed object to increase, this could motivate large CAs to (despite [RFC9455]) start performing ROA aggregation, reducing the number of files, which could compensate for the per-file overhead.

4.4.1.2 Scaling validators

The CPU time measured in section 4.2.2.1 is using Routinator with a single thread. However, validation can be parallelized across multiple cores. Since Relying Party software normally runs on general purpose hardware (not on actual routers), it would be quite easy for an operator to e.g. switch from a small (virtual) machine to one with many more cores. Where a current validator would easily support a 10-minute refresh interval on a 2-core machine, if signature verification is a few times slower, scaling up to a 4 or 8-core machine is a simple and cheap way to compensate.[8]

4.4.1.3 Caching in validators

Another measure to speed up validation is by caching signature verification results. Most validators verify every signature once per validation run. However, most signatures do not change between runs, so the result of a verification could be cached and reused.[9] [58] shows that this can greatly reduce the CPU time spent on validation. A downside is that this adds some complexity to validator implementations, but this becomes more worthwhile if the signature verification is costly. Like the downloading time, the full verification cost also applies when a validator has no cache.

4.4.1.4 Conclusion on latency

We have seen that the downloading times in table 4.3 could be improved a bit if more CAs decide to perform ROA aggregation. However, for the downloading time, *scaling* is not easy. There are limits to the bandwidth of a repository and a validator that cannot be stretched much.

The verification speed is more flexible, as there is a lot of room for optimization through caching and parallelism, but this comes at the cost of more complex implementation, and perhaps computational resources.

Based solely on the impact on latency, we can conclude that Falcon in particular is an excellent option, featuring small keys and efficient verification. HAWK, MAYO, and SNOVA perform similarly, and ML-DSA can be a viable alternative as well. The verification times for each of these do not demand aggressive optimization. The downloading times are likely no problem for well-connected validators. Even in our worst-case estimate, the sizes are manageable considering the infrequency of full downloads, but the clear differences in the worst case make Falcon-512, HAWK-512, MAYO-1 and SNOVA-(24, 5, 4) stand out.

SLH-DSA and FAEST are poor options from a performance perspective. While in our realistic estimate, their downloading time can be handled, the sizes impose a big load on repositories and validators with more limited bandwidth. The verification times are also slow enough to strictly require optimization through caching and still make the initial verification take excessively long. In the best case, there is significant cost both in terms of bandwidth, CPU time, and implementation complexity, that together makes them poor options compared to the alternatives. In the worst case, they are simply unusable.

SQIsign similarly has inconveniently slow verification. The very short signatures are appealing, but even if verification results are cached, the small size may not make up for the expensive initial verification cost.

[8] In Routinator v0.14.1, verification is parallelized with a CA as unit of work, which does not scale perfectly: each CA's objects are processed by a single thread, instead of each individual object. Hence, the runtime is bounded by the CPU time needed for the biggest CA. Changing this to per-object parallelism would improve the scaling over more than a few cores, but this is simply not necessary with RSA signatures. For a slower signature algorithm, it could be beneficial.

[9] That is, for each verified signature, the boolean cryptographic verification result can be stored. During later validation runs, only timestamps and revocation need to be checked, not the costlier cryptographic verification.

4.4.2 Other requirements

Now that we have an idea of what algorithms are performant enough for relying parties, we can evaluate the candidates on other aspects.

4.4.2.1 Signing speed

Table 4.4 indicates an estimate signing time per signature, based on the reported CPU cycles from [78] and a 2.5 GHz CPU. Note that this is a somewhat imprecise comparison, as the benchmarks were reported from a variety of benchmarking platforms.

Put simply, most algorithms easily sign fast enough, but just as we have seen for the verification performance, SLH-DSA and to a lesser degree SQIsign sign too slowly to be practical. For SLH-DSA in particular, the SHAKE-128f parameter set offers much faster signing than SHAKE-128s, but much larger signatures and slower verification, that make it unsuitable from the perspective of a validator.

4.4.2.2 Maturity

The performance aspects leave us with a few acceptable candidate algorithms and parameter sets. We now discuss the trust in their security for each option individually.

Table 4.4: *Estimated signing speed on a 2.5 GHz CPU.*

Algorithm	Param.	NIST level	Signing time (ms)
RSA	2048	-	10.800
EdDSA	Ed25519	-	0.017
ML-DSA	44	2	0.133
	65	3	0.212
	87	5	0.257
Falcon	512	1	0.404
	1024	5	0.821
SLH-DSA	SHAKE-128s	1	1 873.028
	SHAKE-128f	1	95.918
SQIsign	I	1	40.640
	III	3	123.680
	V	5	203.000
MAYO	1	1	0.188
	2	1	0.114
	3	3	0.407
	5	5	0.955
HAWK	512	1	0.034
	1024	5	0.072
FAEST	128s	1	5.115
	128f	1	0.689
	EM-128s	1	3.761
	EM-128f	1	0.562
SNOVA	(24, 5, 4)	1	0.123
	(25, 8, 3)	1	0.148

Falcon Having been selected for standardization, Falcon is an option that has been studied well, and the standardized version will likely have good implementations available soon, even in hardware security modules. Like ML-DSA and HAWK, Falcon is a *lattice-based* scheme. A downside of Falcon used to be that it required floating-point arithmetic for signing, which was hard to make constant-time. However, efficient constant-time implementations have been developed [56, 26], and signing in the RPKI happens offline anyway, so timing attacks are not a big concern. Falcon does not offer an intermediate-level (e.g. 3) parameter set, but the level 5 parameter set is not very expensive.

HAWK HAWK is another lattice-based scheme, that is very similar to Falcon. However, it does not use any floating point arithmetic, and depends on a different hardness assumption [8]. HAWK is a candidate for NIST's call for additional signature schemes, and it is not clear whether it will be standardized. That makes Falcon more appealing from a maturity perspective. Like Falcon, HAWK has no intermediate-level parameter set but a reasonably cheap level 5 option.

MAYO MAYO is a *multivariate* scheme, not based on lattice problems. It is also a candidate for NIST's call for additional signature schemes, which primarily aims to standardize an alternative to lattice-based signature schemes. This makes MAYO a good alternative in case lattice-based schemes are broken. However, attacks have been found against multivariate schemes with specific parameter sets, including one that affects the Round 2 submission of MAYO-2 [59].

SNOVA SNOVA is another multivariate scheme. That has also been affected by several attacks. This underlines that the security of multivariate schemes is actively being researched and not understood enough yet.

ML-DSA Finally, ML-DSA is the only suitable scheme that is already standardized by NIST. This makes it easy to implement right now, but compared to Falcon, this head start will likely not last long. ML-DSA does offer an intermediate security level, and the smallest parameter set satisfies target level 2, but even the smallest parameter set is more expensive for the RPKI than Falcon-1024.

Unfortunately, SLH-DSA and FAEST, as well as stateful signature schemes XMSS and LMS [12, RFC8554][10], have impractically large signatures and expensive verification. Their security reduces to well-known underlying hash functions (or AES in case of FAEST), which makes them a secure option. However, their performance cost is too high to be viable for the RPKI.

4.4.3 Conclusion

Considering all factors together, we find that Falcon is the most suitable candidate, having already been selected for standardization, and offering the best performance. The Falcon-512 parameter set is a good option, having minimal performance impact. Falcon-1024 would also work well, but NIST level 1 suffices, already offering an increase in traditional security compared to RSA-2048 (from 112 bits to at least 128). If a higher security level is chosen, hash functions may need to be replaced as well.

If lattice-based schemes turn out to be insecure, MAYO or SNOVA could provide an alternative. Hash-based schemes and FAEST are not practical, due to their sizes, verification cost, and slow signing.

Throughout the rest of this thesis, we will focus primarily on using Falcon-512, but using any of the alternatives would work very similarly.

[10] Keeping state for stateful signatures — which in many cases makes stateful schemes impractical — is not an insurmountable problem in the RPKI. Nonetheless, the signature sizes make them poor candidates.

4.5 Hybridization

Since the hash-based signatures have turned out ill-suited, the RPKI will have to use a post-quantum scheme whose security can not yet be confidently relied upon. To mitigate this risk, a hybrid scheme can be used, combining a post-quantum scheme with a traditional one. This guarantees that the combination is at least secure against a traditional adversary, even if the post-quantum scheme is broken.

Several approaches to hybridization are possible, with slight differences in cost and security. The security properties of various methods of combining schemes are discussed in detail in [7].

Apart from the obvious requirement that forging a signature of a hybrid scheme should be at least as hard as forging a signature for the strongest component, another interesting property is *non-separability*. Simply put, given a hybrid signature, it should not be possible to deconstruct this into a valid signature for one of the components. This property can be further divided into *strong* and *weak* non-separability, with strong non-separability (SNS) meaning that it is impossible to create a valid signature at all, and weak non-separability (WNS) allowing the creation of a valid signature, as long as it can be seen that the signature was constructed from a hybrid signature.

Many different constructions — known as *combiners* — can be used to turn 2 separate signature schemes into a hybrid. We can describe a hybrid signing algorithm as Sign_H, constructed from two components Sign_1 and Sign_2.

The simplest example is *concatenation*, which provides no non-separability by itself:

$$\mathsf{Sign}_\mathsf{H}(m) = \mathsf{Sign}_1(m) \| \mathsf{Sign}_2(m)$$

Additionally, [7] describes several alternatives that provide non-separability. While various options exist, the most practical option for us is to simply use the concatenation approach, hash the message, and add a label to the message before signing, for domain separation. The domain separation here provides weak non-separability, but primarily, separability is offered by the simple requirement that keys are not used for both hybrids and individual signatures:

$$\mathsf{Sign}_\mathsf{H}(m) = \mathsf{Sign}_1(\mathit{label} \| m) \| \mathsf{Sign}_2(\mathit{label} \| m)$$

Such an approach — details about hashing the message and encoding of signatures and keys are still being discussed — is proposed in [55]. This draft aims to standardize hybrid signatures using ML-DSA, but in a way that easily extends to other post-quantum components once they get standardized.[11]

A resulting hybrid algorithm can be treated as any single signature scheme: the signatures and public keys behave just as those for other (single) algorithms. The hybrid algorithms have their own algorithm identifiers that are independent of the component algorithms, so they can be used as a drop-in replacement without changes to protocols *other than allowing the new algorithm*. The structure of objects does not need to change.

4.5.1 Other constructions

Many other constructions are possible, that have their own advantages, but are less practical for the RPKI.

Among others, there are two reasons to use a different hybrid construction.

[11] At the time of writing, [55] seems close to being finalized, perhaps soon after the upcoming IETF 123 in July 2025, together with related drafts about use of ML-DSA and hybrids for ML-KEM.

Backward compatibility A construction like that in [55] is not backward compatible, in the sense that verifiers need to understand the hybrid algorithm. [7] attempts to use different constructions that allow existing software to verify only the traditional component signature, ignoring the post-quantum component. However, the RPKI standards impose a strict structure[12] that makes changes to its specification and implementations necessary even for such attempts at backward compatibility. Opting for a non-backward compatible construction, that presents as a single atomic signature is a practical choice that keeps changes to the specification minimal, only changing the set of allowed algorithms.

Performance For some specific combinations of component schemes, [6] shows methods to combine signatures such that the total size is (slightly) less than that of two separate signatures. This can reduce the overhead of employing a hybrid scheme, but comes at the cost of requiring modified versions of the component algorithms.

For the RPKI, the simple concatenation with a label for domain separation is a good choice that is easy to implement. It is also the only approach that is on the way to being standardized (at least for ML-DSA).

4.5.2 Cost

When using a simple concatenation, the size of the composite signature is essentially the sum of the sizes of the component signatures, possibly with a few bytes of overhead for encoding. The same applies to public keys, and the verification and signing time also consists of simply verifying/creating both halves of the signature.

Based on [55], the sensible traditional schemes to match with Falcon-512 or the alternative NIST level 1 post-quantum components are Ed25519, RSA-2048 or RSA-3072. With only 112 bits of security, RSA-2048 will be somewhat weak in a couple of years. NIST does not recommend its use after 2030 [47], so considering the long time a migration will take, it seems sensible to use a 128-bits traditional component. On the other hand, we are proposing a hybrid, with the traditional component only being a contingency, so perhaps the traditional component need not be strong.

Using the same metrics as in table 4.3, the predicted costs of using several hybrid pairs are shown in table 4.5. Again, numbers are based on [78]. The sizes and verification time for RSA-3072 are added by us.[13]

The trade-off between traditional components comes down to more CPU time for verification (for Ed25519) or slightly longer downloading time. If cryptographic verification is cached (4.4.1.3), Ed25519 is a good option. Otherwise, the increase in verification CPU time (that if not cached, is repeated for every validation run) may be more costly than the size difference between Ed25519 and RSA-2048. When comparing Ed25519 to RSA-3072, it is less clear which option is better for validators.

Overall, we recommend that — if RP software maintainers agree to implement the caching mechanism from section 4.4.1.3 — the hybrid of Falcon-512 with Ed25519 is the best option. If verification caching is not implemented, there is no pronounced winner.

[12] For example, an RPKI Signed Object must contain exactly one `SignerInfo` element [RFC6488]. Having two separate `SignerInfo` elements for two components of a hybrid is not currently allowed.

[13] The sizes are easily checked. Verification times are set as 2.2 times that of RSA-2048, based on the difference between RSA-2048 and RSA-3072 in a simple local benchmark with `openssl speed -seconds 120 rsa`. Like all of the speed benchmarks from [78], they should be taken with a grain of salt.

4.6 Multiple algorithms

So far, we have found that a hybrid with Falcon-512 is a good choice for a post-quantum signature scheme in the RPKI. Our analysis of the performance impact was based on assuming complete adoption, and allowing a single algorithm throughout the RPKI.

Forgoing [RFC6916] (see chapter 6), it is also possible to have multiple algorithms in the RPKI at the same time. There could be mixed use *during* a transition, multiple algorithms can be chosen from freely even *after* a transition, or different algorithms can be used for different use cases. Allowing the use of multiple algorithms can be beneficial for several reasons.

4.6.1 Mixed certificates during migration

In chapter 6, we conclude that a migration using *mixed certificates* is the most practical way to transition to post-quantum signatures. In such a migration, we allow certificates where the subject is using a different algorithm than the issuer. So, during

Table 4.5: *Estimated downloading and verification times for hybrid schemes. The top rows, without hybridization, are directly from table 4.3 for comparison. The cost of each component can simply be added together; for instance, the difference between downloading for plain Falcon-512 and Falcon-512 + Ed25519 is the same as the difference between ML-DSA-44 and ML-DSA-44 + Ed25519.*

			Estimated times (s)		
Traditional component	Post-quantum component	Total RPKI size	Download (240 s)	Download (14.5 s)	Verification CPU time
RSA-2048	-	**838 MB**	**240.0**	**14.5**	**13.0**
-	Falcon-512	1.4 GB	403.1	24.4	23.4
	HAWK-512	1.4 GB	392.3	23.7	42.8
	MAYO-1	1.4 GB	414.1	25.0	44.3
	SNOVA-(24, 5, 4)	1.1 GB	322.0	19.5	47.3
	ML-DSA-44	3.0 GB	846.1	51.1	34.2
RSA-2048	Falcon-512	1.7 GB	491.5	29.7	36.4
	HAWK-512	1.7 GB	480.8	29.0	55.8
	MAYO-1	1.8 GB	502.6	30.4	57.3
	SNOVA-(24, 5, 4)	1.4 GB	410.5	24.8	60.3
	ML-DSA-44	3.3 GB	934.6	56.5	47.2
RSA-3072	Falcon-512	1.9 GB	534.9	32.3	52.0
	HAWK-512	1.8 GB	524.1	31.7	71.4
	MAYO-1	1.9 GB	545.9	33.0	72.9
	SNOVA-(24, 5, 4)	1.6 GB	453.8	27.4	75.9
	ML-DSA-44	3.4 GB	977.9	59.1	62.8
Ed25519	Falcon-512	1.5 GB	421.1	25.4	61.0
	HAWK-512	1.4 GB	410.3	24.8	80.4
	MAYO-1	1.5 GB	432.1	26.1	81.8
	SNOVA-(24, 5, 4)	1.2 GB	340.1	20.5	84.9
	ML-DSA-44	3.0 GB	864.2	52.2	71.8

the migration, multiple algorithms are allowed to be used at the same time, until at some point an old algorithm is phased out.

This approach makes migrating easier than the alternative ([RFC6916]) where entirely disjoint RPKI trees are constructed for the old and new algorithms. It allows an alternative transition that need not be globally coordinated or top-down.

4.6.2 Fallback algorithms

Two different algorithms B and C could be specified as mandatory-to-implement as relying party, even if CAs are expected to prefer one of them.[14]

In case one of the two algorithms is found to be vulnerable, already having another algorithm that will be accepted by RPs enables CAs to quickly switch to the other algorithm. This way, requiring RPs to already support an additional algorithm provides a safety net. Switching to the second algorithm is much faster without awaiting IETF response and the lengthy process of updating all relying parties.

For the transition to post-quantum signatures, such a fallback can be particularly useful. Various hardness assumptions are used for different candidate algorithms. While our recommendation from section 4.4.3 is to use the *lattice-based* Falcon, it's conceivable that lattice-based cryptography turns out to be insecure. In that case, it would be very valuable to already have support for a fallback algorithm that is not based on lattice problems, such as MAYO, even if it is less attractive in terms of performance.

4.6.3 Specialized algorithms per use case

The single algorithm currently specified in [RFC7935] is used for a variety of purposes. The primary purpose (in the RPKI content) is performance-critical, impacting sizes and verification time of hundreds of thousands of objects. Other use cases have different requirements and could benefit from different algorithms.

4.6.3.1 RFC8183 trust anchors

A clearly different use case is for authentication of communication between CAs, and from CAs to repositories. For this communication, [RFC8183] is used to establish a trust anchor. Messages between parties are signed using this TA.

This communication is not performance-critical, as it involves only two parties, and a low frequency of messages. The trust anchors are also very long-lived. So, for this use case, it could be considered to use a more secure algorithm but less performant algorithm than for the rest of the RPKI. For example, it might be sensible to use something like Falcon-1024 + ECDSA P-521, or even SLH-DSA for the BPKI TA. While hash-based signatures would be acceptable in terms of performance, and preferable from a security perspective, a trade-off should be made to keep the set of algorithms that need to be implemented small. That may make Falcon-1024 more attractive when Falcon-512 is already used for the rest of the RPKI.

4.6.3.2 Stronger keys for trust-anchors

A very similar case can be made for the trust anchors in the RPKI themselves. As these are hard to update,[15] it is sensible to use strong crypto there. Performance is not much of a concern as there are only a handful of TA key pairs, each validated only once per validation run. As the RIRs actually use a few key pairs (e.g. an *online* CA

[14] One algorithm may be more efficient or otherwise more attractive than the other, or the algorithms profile could even recommend CAs to use a particular single algorithm. The benefit of having two algorithms comes only from RPs — not CAs — being prepared to accept both.

[15] The TA public keys are in TALs that are often shipped together with RP implementations.

as the only subordinate under the *offline* TA) for security, a stronger algorithm could also be applied to one more layer of the hierarchy.

4.6.3.3 Short-lived objects

Finally, the opposite could be done for short-lived key pairs. In particular, signed objects in the RPKI often contain a one-time-use EE certificate. This is a certificate for a key pair that is used to sign only one object.

For short-lived objects such as Manifests, the algorithm of the one-time key pair embedded in the signed object could be chosen to be short and not quantum-resistant, assuming that a quantum attacker needs considerable time to break traditional cryptography.

While this approach can help performance, it is not without risk, and still makes use of a superfluous, ephemeral key pair. In chapter 5 we propose an alternative that removes this redundant public key and second signature, while still retaining the possibility to make use of the X.509 revocation system.

An instance of this idea was suggested in the conversation following [71], using elliptic-curve cryptography for EE certificates. Here, the different signatures are used only in EE certificates, not because they are weaker (though they are indeed not quantum-resistant) but instead to avoid having to migrate CAs' resource certificates.

Chapter 5

Redundancy in RPKI objects

In section 4.6 we showed that there are reasons to support multiple signature algorithms at the same time. One of the reasons for this is to allow using specialized algorithms for specific use cases, to improve security where performance is not critical, or to improve performance where possible.

A special case where it might make sense to use a different signature algorithm than in the rest of the RPKI is in signed objects with so-called 'one-time-use' end-entity certificates. All RPKI signed objects ([RFC6488]) contain an EE certificate that certifies a key pair used to sign the object itself. This key pair is almost always used only once, which enables revocation of individual objects through the EE certificate for its one-time key. This is wasteful.

We suggested in 4.6.3.3 that for short-lived EE certificates, a shorter, faster, but weaker signature scheme could be used than for the rest of the RPKI. In this chapter, we define a different method to get rid of the overhead of including a second signature and a public key in each signed object. We introduce a "null scheme" that can replace the signature algorithm specifically in one-time-use EE certificates. This is both more secure and more efficient than using a weaker signature scheme.

5.1 One-time-use EE certificates

Before introducing an efficient alternative for the signatures on signed objects, let us examine the current situation and its cost.

5.1.1 One-time-use for revocation

The purpose of using one-time-use EE certificates, instead of directly signing objects with the CA's key, is to allow for revocation of individual objects, using the existing X.509 CRL mechanism [RFC5280].

Revocation is required to prevent replaying old objects, while still supporting long lifetimes of each object. Making objects inherently short-lived is no alternative: validators would need to frequently fetch numerous new objects, and objects expire earlier if a repository is unavailable. That has major performance implications and makes temporary unavailability of a repository problematic.[1]

The use of EE certificates for signed objects is defined in [RFC6487] and [RFC6488]:

[1] Unavailability of a repository currently only means that *changes* are not getting distributed. Long-lived objects remain valid, so the latest state of the RPKI is used. If objects were short-lived, after a few hours of unavailability, the latest state of the RPKI would expire instead, disabling the protection it offered. That would make denial-of-service attacks on repositories an attractive way to bypass ROV filtering.

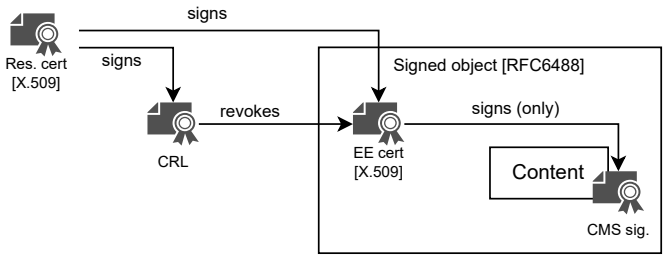


Figure 5.1: *Structure of a signed object. A signed object contains an EE certificate issued by a CA. The EE certificate's subject is a key used only once to sign a CMS* signed-data *structure with the object's content. The CA's CRL can revoke the signed object by revoking the one-time-use EE certificate.*

> The private key associated with an EE certificate is used to sign a single RPKI signed object, i.e., the EE certificate is used to validate only one object. The EE certificate is embedded in the object as part of a Cryptographic Message Syntax (CMS) signed-data structure [RFC6488]. Because of the one-to-one relationship between the EE certificate and the signed object, revocation of the certificate effectively revokes the corresponding signed object.
>
> [...]
>
> EE certificates used to validate only one instance of a signed object, and are not used thereafter or in any other validation context, are termed "one-time-use" EE certificates.

Each signed object includes exactly one EE certificate, that certifies the key pair used to sign the CMS *signed-data* structure that contains the actual content. For almost all signed objects, this is a one-time-use certificate, as that enables using the X.509 revocation mechanism to prevent replay attacks when objects are withdrawn.[2] An overview of the structure of a signed object is shown in fig. 5.1.

In principle, an EE certificate could be 'one-time-use', yet still have a subject key that is not used only once. The wording from [RFC6488] above can be misleading in this regard, as *keys* must in fact be used only once, not just *certificates*:

- For object m_1, a key pair (pk_1, sk_1) is generated and an EE certificate c_1 is created. m_1 is signed using sk_1, and c_1 is attached to m_1 to allow validation.
- For another object m_2, the same key pair (pk_1, sk_1) is reused, but a new EE certificate c_2 is created. m_2 is signed using sk_1, and c_2 is attached to m_2 to allow validation.

In this hypothetical scenario, each EE certificate is used only once, but the corresponding private key is reused. In reality, this should never occur: it allows for replay once one of the objects is revoked.

- The issuer revokes m_1 by adding a CRL entry for c_1.
- An attacker can now replay m_1, by attaching c_2 to it. The signature from sk_1 on m_1 still validates using c_2, and c_2 has not been revoked. This is possible because the signature on m_1 does not cover the attached certificate: c_1 can be replaced by c_2 without invalidating the signature.

Consequently, for revocation of individual signed objects to be effective, it is necessary that every signed object has a one-time-use key pair, not only a one-time-use EE

[2] An exception is for Manifests ([RFC6486]), where 'sequential-use' EE certificates are also allowed. See section 3 of [RFC6486]. However, one-time-use certificates are allowed and common, and with our proposal are more efficient.

certificate. In the specification cited above, *"The private key associated with an EE certificate is used to sign a single RPKI signed object"* is a stronger statement than *"The EE certificate is used to validate only one object"*. Throughout the RPKI, "one-time-use EE certificates" should be understood as one-time-use key pairs, not just certificates.

5.1.2 Cost

A result of the approach of including EE certificates in each signed object is that signed objects are larger than they need to be.

Currently, a signed object contains:

- The object content itself.
- A signature over the object content, using the one-time-use private key associated with the EE certificate.
- The one-time-use public key inside the EE certificate.
- A signature from the issuer over the one-time-use public keys and some EE certificate attributes, also as part of the EE certificate.
- And some metadata and encoding overhead, both in the CMS structure and the X.509 EE certificate.

This includes two signatures and one public key, of which one signature and the public key are only present to fit the revocation system.

In the current RPKI, we find a median size of ROAs of 2125 bytes.[3] Out of this, $2 \cdot 256 + 272 = 784$ bytes are used for the cryptographic material: over a third of the total size.

When a post-quantum algorithm is introduced, signatures and public keys will grow even larger. For example, assuming the same median-sized ROA, and the hybrid Falcon-512 + RSA-2048, the size of the ROA becomes roughly 4354 bytes, including 3031 bytes of cryptographic material. So, the overhead of cryptographic material may increase to almost three-quarters of a ROA.

Therefore, it would be very attractive if the overhead can be reduced from two signatures and a public key, to as close as possible to only the one signature that is really needed to sign the object.

5.2 Removing the one-time-use signature

5.2.1 Requirements of the one-time key pair

The crucial observation that enables us to avoid including a large signature and public key in each object, is that the requirements of the one-time-use key pair are much more relaxed than those of general digital signatures.

When a private key is guaranteed to be used only once, a *One-Time Signature* (OTS) scheme can be used. The most well-known instantiation is Lamport OTS [41], which is based on a one-way function. This construction is considered quantum-safe, and variants of it are used in the construction of hash-based post-quantum signatures like XMSS and SLH-DSA [12, 5].

For typical digital signatures, the EUF-CMA or the stronger SUF-CMA security model is used. Here, an attacker is allowed to adaptively query a signing oracle many times, with the goal to present a forged message-signature pair (m', σ') where either m' was

[3] This is the median size of ROAs from the snapshot used for table 4.1.

not queried before (EUF-CMA), or the pair (m', σ') was never returned from a query (SUF-CMA).

The corresponding model for OTS schemes is similar but allows only one query to the signing oracle. The attacker can query a signature for any message m, getting σ in response. The attacker must then present a pair (m', σ') where $m' \neq m$ (the one-time equivalent of EUF-CMA), or where (m', σ') is not (m, σ) (the one-time equivalent of SUF-CMA).

When a private key is used to sign only one object and then discarded, the one-time model is sufficient as there is no way for the attacker to query a signature for any other message.

5.2.1.1 Signed data does not depend on the public key

Using an OTS scheme for the one-time-use key pairs is reasonable, but does not do much for performance as hash-based OTS schemes still yield large signatures or public keys. We can do even better, because OTS schemes still satisfy a fundamental requirement that we don't need. For both normal and one-time signatures, the public key can be made *before* the message to be signed is known. This is not needed in our peculiar case.

In our use case, the public key is included in the EE certificate. This EE certificate gets signed using the CA's private key, and is then attached to the signed object in the `certificates` field of the CMS signed-data structure.

The signature made *using* the one-time key, is over a digest of the `signedAttrs` field of the CMS signed-data structure, which in turn contains, among other things, a digest of the actual object content. Importantly, the `certificates` field is *not* an input the signature algorithm [RFC6488, RFC5652]. So, for our particular case, the one-time public key does not need to be known before the message is signed.

In our case, key generation happens at the same time as the signing. Also, the signature in the CMS signed-data structure is on the content, but not the attached certificate [RFC5652]. A signed object can be viewed as

$$\big(cert, data, \mathsf{Sign}_{sk_{ots}}(data)\big)$$

where *cert* is the EE certificate:

$$cert = \big(pk_{ots}, attrs, \mathsf{Sign}_{sk_{ca}}(pk_{ots}, attrs)\big)$$

Here, (sk_{ots}, pk_{ots}) is the ephemeral one-time-use key pair, $attrs$ are the attributes included on the EE certificate. The signature $\mathsf{Sign}_{sk_{ca}}(pk_{ots}, attrs)$ is the signature from the CA over the public key and attributes.

Because the input to the signature *using* the one-time key does not depend on the one *cert* or specifically pk_{ots}, the public key can be generated based on the message it signs.

5.2.2 Null scheme

We define a more limited combination of the classic Sign and KeyGen algorithms, that meets the relaxed requirements for our use case: SignOnce, taking a message as input, returning a tuple (pk, σ) as output. So, there is no private key, and the public key is generated together with the signature.[4] While an implementation of SignOnce is not sufficient for a general (one-time) signature scheme, it is all we need.

[4] For a normal signature scheme with algorithms KeyGen, Sign, SignOnce is simply KeyGen followed by Sign.

There is a very simple algorithm that satisfies our unusual requirements. Using an underlying hash function H, we define our public key to be the hash of the message, and the signature to always be null. Here, H should be collision-resistant. Second pre-image resistance would suffice if we assume that the attacker cannot cause a CA to sign a message of their choice.

$$\mathsf{SignOnce}(m) \stackrel{\text{def}}{=} (H(m), \mathsf{null})$$

Verification is simply checking that the message hashes to the public key:

$$\mathsf{Verify}_{pk}(m, \sigma) \stackrel{\text{def}}{=} (\sigma = \mathsf{null} \wedge H(m) = pk)$$

A signed object using our null scheme would look like $(cert, data, \mathsf{null})$ where

$$cert = \Big(H(data), attrs, \mathsf{Sign}_{sk_{ca}}\big(H(data), attrs\big)\Big)$$

The one-time public key in $cert$ is $H(data)$, and the CMS signature is null.

Despite this being a useless signature scheme in general, it is secure. Given a public key:

- It is impossible to find a *different* signature for the same message, as there exists only one unique signature.
- Finding a different message for which a signature can be forged reduces to finding a collision in the underlying hash function.

For a trivial proof, consider an attacker who has two distinct valid pairs (m, σ) and (m', σ') for the same public key pk. As the space of possible signatures contains only null, it must be that $m \neq m'$. As both pairs are valid, $H(m_1) = H(m_2) = pk$, so the attacker has a collision in H. If we do not allow the attacker to choose m_1 and hence pk, but instead give the attacker m_1 and pk,[5] finding m_2 breaks not only the collision resistance, but also the second pre-image resistance of H.

Note that signed objects already depend on collision or second pre-image resistance of a hash function. The signature in the signed-data structure is actually made over a digest of the content, not directly over the content itself. This makes using our null scheme *at least* as secure as the current reliance on both a normal signature scheme, *and* a hash function. The collision an attacker needs to break our scheme directly also breaks the hash used as input to the current proper signature scheme.

5.2.3 Use in signed objects

Our null scheme can be used as a drop-in replacement for a normal signature algorithm in the signed-data structure of a signed object. Changing the RPKI to use our approach involves *only* an algorithm rollover. By treating our scheme as any other signature scheme, the existing structure with an EE certificate that can be revoked can be maintained. It is not necessary to implement a new revocation system, and [RFC6488] does not need to be updated.

5.2.3.1 Implementation details

While we have defined our $\mathsf{SignOnce}$ algorithm to return a hash of its input for easy analysis, its implementation when used to sign CMS signed-data can be done differently.

The process of signing a CMS signed-data structure for RPKI signed objects is as follows:

[5] This corresponds to finding a new signed object for one of the *existing* non-revoked EE certificates in the RPKI.

- The content is hashed using a digest algorithm. The result is in the `message-digest` attribute in the `signedAttrs` field of the CMS structure.[6]
- The `signedAttrs` field (which includes the message digest) is encoded and hashed once more [RFC5652].
- The resulting digest over `signedAttrs` is the input to the signature algorithm.

So, regardless of what signature algorithm is used, there is *also* a digest algorithm used to construct the input to the signature algorithm.

Our proposal is to replace the signature algorithm, but we already know that it only gets digests as input to be 'signed'. Therefore, the actual algorithm to use can be

$$\mathsf{SignOnce}(m) \stackrel{\text{def}}{=} (m, \mathsf{null})$$

where m is always the digest of the `signedAttrs` field. This avoids doing one more hash operation and means that we do not need multiple distinct instantiations of our null scheme for different digest algorithms.

5.2.4 Cost reduction

The cost of our alternative depends on the underlying hash function. For RSA, as well as for the NIST level 1 or 2 post-quantum algorithms we suggest, this would be SHA-256.

The size of the null scheme's public keys is the same as the size of the hash output: 32 bytes for SHA-256. The signature is empty, so an octet string of 0 bytes.

When compared to using RSA, the null scheme saves $272 + 256 - 32 = 496$ bytes per signed object, leaving $256 + 32 = 288$ bytes of cryptographic material. This reduces the size of the median ROA from 2125 bytes to 1629 bytes. On the whole RPKI from table 4.1, 172 MB could be saved, out of 838 MB in total.

When compared to using Falcon-512 + RSA-2048, the null scheme saves $1169 + 922 - 32 = 2059$ bytes per signed object, leaving $922 + 32 = 954$ bytes of cryptographic material. The median ROA shrinks from 4354 to 2295 bytes, almost compensating for algorithm rollover. On the whole RPKI, 717 MB of the 1.7 GB total from table 4.5 could be saved. This amounts to 82% of the increase in total size when switching from RSA-2048 (838 MB from table 4.1) to Falcon-512 + RSA-2048. For even bigger signature algorithms, the savings are even larger.

Besides the reductions in size, our null scheme also simplifies the cryptographic verification. As we do not even need to introduce another call to the hash function (see section 5.2.3.1), we entirely avoid 1 of the 2 signature verifications per signed object, or roughly 35% of the total number of signature verifications. For the Falcon-512 + RSA-2048 hybrid, this leads to a verification CPU time of 23.7 seconds, as opposed to the 36.4 seconds from table 4.5 without the null scheme.

Overall, the null scheme can make the RPKI substantially more efficient, which can be paired with the introduction of post-quantum signatures to nearly compensate for the cost increase of the new signature algorithm.

5.2.5 Conclusions

The null scheme idea is a drop-in replacement that saves almost one signature and a public key in size and a signature verification in computation per signed object. This is a substantial improvement over the current approach, with benefits that become more pronounced when larger and slower signature algorithms are used.

[6] While optional for general signed-data structures, this field is mandatory in [RFC6488] objects.

The security of the null scheme matches the current dual-signature approach, as both ultimately depend on the collision and second pre-image resistance of the underlying hash function, as well as security of the signature algorithm used to sign the EE certificate. The null scheme simply removes a redundant signature while maintaining exactly the same security properties.

Our proposal is a simple drop-in replacement that can nearly compensate for the size increase induced by switching to post-quantum algorithms. The precise savings depend on the specific post-quantum algorithm chosen. Similarly, it can reduce the number of signature verifications by a third, all without affecting security. Hence, we recommend introducing it together with post-quantum algorithms.

Chapter 6

Algorithm migration process

In the previous chapters, we have found that migrating the RPKI to a post-quantum signature algorithm is necessary, and found a suitable choice for that algorithm: a hybrid with Falcon-512 as post-quantum component, as well as some alternatives. We have also argued that allowing multiple algorithms at the same time can be beneficial. Considering the end goal of using post-quantum algorithms, the next question is: how do we get there? In this chapter we, discuss this transition.

6.1 RFC6916

In the early days of the RPKI, the Secure Inter-Domain Routing (SIDR) working group of the IETF defined an algorithm agility procedure in [RFC6916]. The core principle of this approach is that "mixed" certificates are prohibited. That is, a resource certificate with an algorithm A subject is to be signed only with an algorithm A signature, and an algorithm B subject only with an algorithm B signature, implying a top-down migration process. Such a migration is necessarily towards a single new algorithm; migrating towards a state where multiple algorithms can be chosen from by CAs is not supported.

The procedure starts by publishing an update to [RFC7935] defining the new algorithm B, as well as a new timeline document, that defines 5 global milestone dates. These dates are:

```
CA Ready Algorithm B Date:  After this date, all non-leaf CAs MUST be
            ready to process a request from a child CA to issue a
            certificate under the Algorithm Suite B.  All CAs
            publishing an [RFC6490] Trust Anchor Locator (TAL) for
            Algorithm Suite A MUST also publish the correspondent TAL
            for Algorithm Suite B.

CA Go Algorithm B Date:  After this date, all CAs MUST have reissued
            all their signed product sets under Algorithm Suite B.

RP Ready Algorithm B Date:  After this date, all RPs MUST be prepared
            to process signed material issued under Algorithm Suite
            B.

Twilight Date:  After this date, a CA MAY cease issuing signed
            products under Algorithm Suite A.  Also, after this date,
            an RP MAY cease to validate signed materials issued under
            Algorithm Suite A.

End-Of-Life (EOL) Date:  After this date, Algorithm Suite A MUST be
            deprecated using the process in Section 10, and all
            Algorithm Suite A TALs MUST be removed from their
            publication points.
```

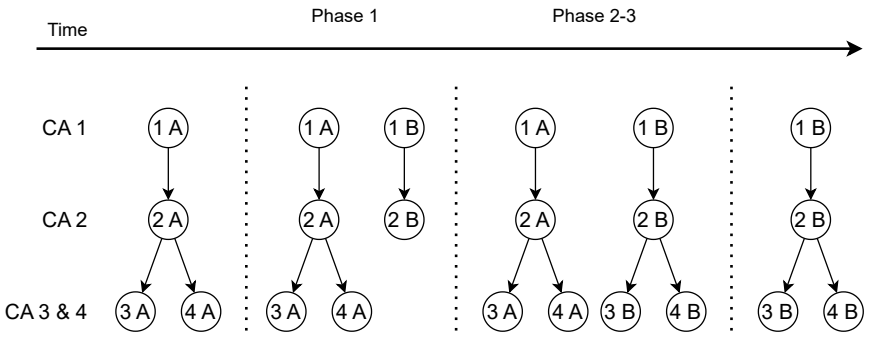


Figure 6.1: *Timeline of RPKI trees throughout an [RFC6916] algorithm roll. The nodes represent resource certificates: "1 A" is a resource certificate for the algorithm A public key of CA 1, and the edges indicate issuing certificates to children. During phase 1, a duplicate tree is constructed top-down. When this is done, RPs can switch to validating the B tree, and finally, the A tree can be removed.*

In "Phase 1", between *CA Ready Alg. B date* to *CA Go Alg. B Date*, every CA must issue copies of all its products under both the old and new algorithms. That is, by *CA Go Alg. B Date*, there should be two separate RPKI trees — one using only the old algorithm A and the other using only new algorithm B — that contain identical meaning. The creation of the algorithm B tree takes place in a top-down fashion: a CA can only get its algorithm B certificate once its parent has also obtained an algorithm B certificate.

The distinct copies of the RPKI are supposed to be maintained (kept synchronized) throughout "Phase 2" and "Phase 3" at least until the *Twilight Date*. By that time, every relying party must have been updated to accept the Algorithm B tree, and CAs may stop maintaining the Algorithm A tree. An example of this is shown in fig. 6.1.

We believe this procedure to be operationally infeasible, due to the extensive coordination required, the need to maintain two parallel trees, and the exclusion of allowing multiple algorithms. Several software implementers and RIR operators have expressed similar concerns, and stated that they would prefer an alternative that uses mixed certificates [9, 10]. Furthermore, there has been a proposal to introduce EdDSA or ECDSA signatures, particularly in a mixed setting [71]. This resulted in experimental ECDSA support in `rpki-client`, that allows mixed certificates.[1]

6.1.1 Coordination

The first problem is the need for global coordination. The 5 dates, expected to take place over the course of several years, are to be published in a *Best Current Practice* RFC. This means that they need to be planned far in advance, and are hard to postpone if the need arises. It is unusual or even unprecedented to plan future dates in an RFC.[2]

Additionally, the milestone dates are strict deadlines that every CA and RP must meet. If a significant number of CAs or RPs is not ready by a milestone, the milestone needs to be postponed. Considering the fact that several repositories are still only available over `rsync`, and that some CAs still publish expired objects, it seems inevitable that a

[1] `https://github.com/openbsd/src/commit/ec1cc732eea452b2c8e9f1282111d9cc0104e4b6`

[2] An exception is the legacy [57] that contained future dates. However, this predates the formal internet standards process, and the milestone dates were relatively nearby.

handful of CAs and relying parties will not be ready in time, regardless of how much time is given.

Finally, the top-down process to create an algorithm B tree is a limitation that can cause delays.

- If one CA operator wants to migrate at some time, but its parent CA has not migrated yet, the operator must wait until the parent has migrated. Delays on any level in the hierarchy cascade down, so the total duration can be needlessly long. Issues at low-level CAs can also become apparent very late in the process, making it plausible that the milestone dates need to be postponed at the last moment. Considering that the timeline and any changes to it are to be published as standards-track RFCs, rescheduling is at best a lengthy process.
- The top-down process also makes it hard to experiment with the new algorithm. Since an algorithm B certificate can only be issued under an algorithm B CA, a low-level CA cannot do real-world experimentation on a small scale: real-world experimentation can only commence when the algorithm B tree is under construction or complete.

Overall, an approach with multiple global milestone dates is risky, the top-down process could make the transition slower than necessary, and makes early experimentation impossible.

6.1.2 Parallel trees

Another drawback of the procedure is the use of disjunct RPKI trees that must be kept synchronized. The impact of this is twofold.

- Most importantly, it is a complicated task for CAs to perform. Current RPKI CA software makes it easy to act as a CA, but the software and standards ([RFC6492]) do not currently support the synchronization of separate trees. Keeping two trees synchronized requires significant implementation effort from CAs, and likely even a new provisioning protocol [10, 9]. Having to do so for several months or years is a significant burden. The complicated changes to standards and their implementations take time, and can also make operators reluctant to start migrating, delaying the transition.
- Second, the existence of disjunct trees has performance impact. While in chapter 4 we have assumed a complete transition to a new algorithm, the [RFC6916] procedure involves a significant time (years) during which both trees are available. RPs do not *need* to process both trees, but it is reasonable to assume that some RPs will attempt to do so, to monitor for any differences in the content of the trees. The procedure says:

  > `[...] it is RECOMMENDED that Suite B capable RPs fetch and validate Suite B products sets during Phase 2. If an RP encounters validation problems with the Suite B products, it SHOULD revert to using Suite A products. RPs that are Suite B capable MAY fetch both product sets and compare the results (e.g., ROA outputs) for testing.`
  >
  > `In Phase 3, all RPs MUST be Suite B capable and MUST fetch Suite B product sets. If an RP encounters problems with Suite B product sets, it can revert to Suite A products.`

  For the duration of at least phases 2 and 3 (likely several years), this means that the size of the RPKI for publishers is doubled in terms of the number of files. Whereas our size and time estimations in section 4.4.1 concern a single algorithm B tree, during the transition, publishers and some RPs will need to work with the sum of the current (algorithm A) tree and the new (algorithm B) tree.

6.1.2.1 Estimating performance cost

In chapter 4 we have estimated performance cost of various algorithms in a single tree. The temporary cost of working with duplicated trees is a bit harder to predict.

Assuming our proposed Falcon-512 + RSA-2048 hybrid as algorithm B, that temporarily leads to a total *size* of 2.6 GB, instead of 1.7 GB for the new tree alone. But this does not translate directly to the downloading and verification times.

Unlike in our estimations for a single tree, the two separate trees will have separate publication points, leading to differences in the fraction of the spent time that does not depend on the signature algorithm: there are more repositories, not only larger downloads per repository.

We can predict a signature verification CPU time of 36.4 s + 13.0 s = 49.4 s for the combination of the two trees, but this does not include the algorithm-independent fraction of the total processing time. We assumed in chapter 4 that only the signatures would be replaced, and the number of files and their content would remain exactly the same. With a duplicate tree, this assumption does not hold, and the actual cost of keeping the old tree around is more than just the + 13.0 seconds of CPU time.

Likewise, the added cost of downloading will be considerably more than what one would expect based solely on the total size. Not only the total data transfer increases, but also the number of requests, which we considered constant in 4.2.1.

At any rate, there is a significant performance penalty for some RPs, and increased cost for hosting repositories.

6.1.3 Design

The choice for a top-down migration without mixed certificates was based on several fundamental assumptions, and some findings that follow from these assumptions.

6.1.3.1 Only one current algorithm

The algorithm agility procedure fundamentally assumes that after migrating, there should be only one allowed algorithm. That is, when migrating, it is not only necessary that some CAs start using a new algorithm, but also that *all* CAs stop using the old algorithm. So, the procedure treats introducing a new algorithm and deprecating the old one as inseparable. In 4.6 and 6.3.1 we discuss this, and show that a state where multiple algorithms are allowed could be beneficial. Migration to such a state cannot be done with the procedure in [RFC6916].

6.1.3.2 RPs update last

Relying parties are expected to be the last to change, only after all CAs have switched to provide products under the new algorithm. Thus, it would be necessary to provide a full algorithm A RPKI tree for a long time, until every RP has changed to support algorithm B.

Indeed, it stands to reason that there will be RPs that take a long time before updating, and CA operators may be more actively involved in the RPKI, such that they can be expected to update their software sooner. After all, there are simply far fewer parties actually acting as CA (not using a hosted CA service) than there are RPs.

On the other hand, while this was not known when the algorithm agility procedure was designed, we now know that nearly all RPs use one of a handful of RP software implementations [38, 40]. Thus, updating RPs may not be as difficult as was assumed, consisting of minor changes to these few implementations, and waiting for the operators to perform a routine software update to a version that includes these changes.

Conversely, we can nowadays also observe that there are CAs that are not actively maintained: several repositories are frequently unreachable or contain no valid objects [72]. These CAs could delay the migration process, although those that do not have valid products anyway are not a good reason to postpone a planned milestone date.

6.1.3.3 Exponential growth

Based on the above assumptions, it was observed that allowing mixed certificates (alg. A parent signing alg. B child or vice versa) could lead to exponential growth of the RPKI, in particular since RPs would update last, requiring an all-A RPKI tree to be maintained for a long time.

Consider CAs X, Y and Z. If mixed certificates are allowed, each CA could have a key for both algorithms A and B. Then, X could sign certificates for both keys of Y, using both keys of X. Similarly, Y could sign certificates for both keys of Z, and using all 4 of the certificates from X for Y. This way, there can be exponentially many certificates in the RPKI. Hence, the idea of a mixed tree was rejected altogether [36, 35].

Note however, that there is no reason for CAs to request or issue most of these options:

- Validators that understand only algorithm A can only accept all-A certification paths. So, CAs could request or issue A certificates on A keys for compatibility with old RPs.
- Other than that, B should be more secure, so CAs should prefer to request or issue B certificates on B keys wherever possible.

So, there is no incentive for any CA to have more than 2 different certificates: one from an A issuer on an A key (for backward compatibility), and the other from a B issuer on a B key, or from an A issuer on a B key if the parent does not have a B certificate yet. Hence, exponential growth seems unlikely in practice.

6.1.3.4 Bottom-up impossible?

There was also the idea that a CA could not unilaterally decide to change its certificate to a new algorithm, because the parent CA needs to be able to verify the proof of possession of the new algorithm key. Indeed, the parent needs to *verify* an algorithm B signature, but it is not necessary that it itself has an algorithm B certificate. To support a bottom-up migration, it is only needed that a parent updates its software to handle algorithm B verification, which is a simple local change with no risk of invalidating the CA's published products.

6.1.3.5 Alternative

The fundamental assumptions underlying the procedure did not go unchallenged. In particular, in response to a draft [28], Brian Dickson questioned the idea that only a single algorithm should be allowed, and that a globally coordinated top-down migration is inevitable [21]. During a lengthy, heated discussion on the mailing list, he suggested, among other things, that introducing algorithm B could be separate from deprecating algorithm A. Dickson proposed a simpler alternative, that does not require global coordination and a top-down transition [19, 20].

In the end, it appears that the fundamental disagreement between Dickson and the RFC's authors was not resolved. However, the alternative did not get much traction, and after 2 years with small textual changes and no other considerable objections on the mailing lists, the draft was published [43, 44].

In sections 6.2 and 6.3, we work out Dickson's alternative, and argue that, considering the current state of the RPKI, it seems to be more practical than [RFC6916].

6.2 Mixed-tree approach

In the previous section, we have seen the standardized procedure for algorithm agility, that works top-down, without allowing mixed certificates. Now, we propose an alternative, based closely on Dickson's proposal [19, 20]. In contrast to [RFC6916], this alternative:

- treats introducing new algorithm B and deprecating old algorithm A as separate processes;[3]
- allows mixed certificates, where an algorithm A parent CA can sign a B subordinate, and vice versa;
- thus enables a laissez-faire approach, where CAs can individually decide to switch to another algorithm.

The core principle of this approach is the fundamental assumption that, before widespread migration of CAs to use new algorithm B, all relying parties are capable of validating algorithm B signatures. In section 6.3.2 we further examine this assumption. Consequently, it is permissible that during the transition, there is no all-A RPKI tree. That is, CAs need not maintain an algorithm A key issued by an algorithm A parent. This, in turn, avoids the theorized 'exponential growth' from section 6.1.3.3, or the need to maintain two separate copies of the RPKI.

In our proposal, CAs can unilaterally decide to switch between two algorithms, essentially with a normal [RFC6489] key roll. This naturally also makes it possible to allow CAs to choose from multiple algorithms as discussed in section 4.6.

Since introducing a new algorithm and removing an old one can be seen as separate, we describe three parts of the migration process separately. First, we show from a high level how introducing a new algorithm works. Then, we propose measuring methods to track support in validators. Next, we show precisely how a single CA can migrate between algorithms, and finally, we discuss deprecation of an old algorithm.

6.2.1 Introducing algorithm B

When a new algorithm is to be introduced, three phases can be distinguished, although no milestone dates need to be defined in advance. Examples of part of the RPKI throughout a mixed-tree transition are shown in fig. 6.2.

Phase 0 This is the steady state before the introduction of the new algorithm. Towards the end of this phase, the new algorithm (or set of algorithms) is agreed upon. Drafts for a new algorithm document are being written, but nothing is published yet. Still, there could already be early experimentation in the real world by having a test CA using A issue a certificate to another subordinate test CA using B. RP software maintainers may already publish updates with support for the new algorithm.

Phase 1 Phase 1 starts with the publication of the new algorithm document that obsoletes [RFC7935] and defines the new algorithm B. This document allows the use of algorithm A as well as B, optionally with a recommendation to issue only with B. In particular, it *requires* relying parties to accept both algorithms.

During phase 1, RPs need to update to accept algorithm B signatures. Furthermore:

[3] Thus, it supports a steady-state where multiple algorithms can be used, rather than assuming the goal that only one algorithm is allowed while not in a [RFC6916] transition.

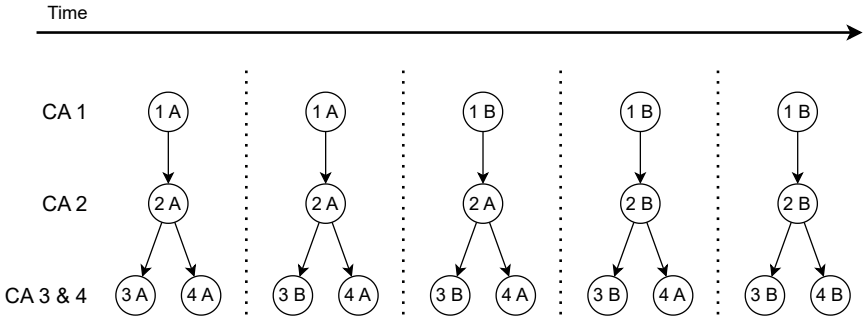


Figure 6.2: *Timeline of an RPKI tree throughout our transition with mixed certificates. A leaf CA can be used early on for experimentation (e.g. '3 B' being the first B CA). During phase 2, all CAs can transition in any order. The final all-B tree may be reached freely, or only after officially deprecating algorithm A.*

- CAs *may* update to be able to verify algorithm B signatures. Pairs of CAs may start using algorithm B for their bilateral communication (using [RFC8183]), and parents can become capable of verifying algorithm B proofs of possession, that already makes them capable of issuing A certificates to B subordinates. All of these changes have strictly local impact. No B products need to be published for this.
- Ideally, the RIRs already publish new TALs for algorithm B trust anchor certificates, even though these are not yet usable (the certificate they point to is not used to issue anything yet and the current A TALs remain in use). This allows the new set of TALs to be included in RP software updates.

Real-world experimentation can take place using a 'leaf' CA. A normal CA[4] signs a certificate for a testing CA that has a B key. This enables testing and monitoring the readiness of RPs for the new algorithm. In section 6.2.2 we present several methods of measuring whether RPs are indeed ready to validate B products. These measurements are crucial to determine when the next phase can start.

Phase 2 When enough RPs are known to accept algorithm B, CAs can widely start switching to algorithm B. We propose a wide range of methods to monitor for this in section 6.2.2. Each CA can individually decide whether and when to switch.

The algorithm rollover for an individual CA is detailed in section 6.2.3. The only requirement for a subordinate CA to switch to algorithm B is that its parent CA is capable of verifying algorithm B proofs of possession; not that the parent already has a B key. Readiness of parent CAs to issue for B subordinates can be communicated out-of-band.[5]

From the start of phase 2, (virtually) all RPs can process B, so there is no reason for a CA to have both A and B certificates and duplicate signed objects. Thus, there is no exponential growth or parallel tree maintenance. Parent CAs can also enforce this by issuing only a single certificate[6] for each subordinate. Similarly, — although this might be unjustifiable — parents could enforce that their subordinates use a particular algorithm, for example by refusing to renew A certificates after a certain date.

Note that, in contrast to [RFC6916], it is not necessary to coordinate a specific date

[4]The parent CA must be prepared to verify a B proof of possession, but can still have only an A key.

[5]For example, through their Certification Practice Statement (CPS).

[6]An exception is needed during [RFC6489] key rollovers.

when phase 2 starts. At some point — potentially already before phase 1 — some experimental leaf CAs will be created with algorithm B, whose products are initially rejected by most RPs. Over time, RPs will start accepting these products, until the experiments we recommend in 6.2.2 show that almost all RPs accept B. The first few migrations of production CAs mark the start of phase 2, and after the first few CAs switch without issues, the rest can follow suit safely. No timeline needs to be imposed, and the transition can occur naturally by decisions of individual CAs based on real-world measurements.

There is also no definite end for phase 2. While the share of algorithm B certificates in the RPKI may grow from a small fraction of experimental certificates to the majority, A certificates can keep being issued indefinitely. It is conceivable that phase 2 is soon followed by deprecation of algorithm A, or theoretically a third algorithm could first be introduced, making phase 2 overlap with the phase 0 of another transition.

6.2.2 Measuring RP readiness

For our mixed-tree transition, it is crucially important that RPs are ready to validate new algorithm B signatures, before production CAs migrate. We propose several complementary methods for measuring RP readiness for the new algorithm. Together, these methods can be used to confidently measure the adoption of new RP software that accepts algorithm B, and thus inform operators when it is safe to migrate to the new algorithm. More detail about each of these methods is given in appendix B.

Monitoring RP software from a repository We can count the versions of RP software that are in use by logging requests to an RRDP repository. This shows what number of validators run updated versions, but not what those RPs are used for, and whether they actually verify B products.[7]

Measuring reachability of B repository We can count how many RPs download from a repository only pointed to using a B certificate, versus those that reach another repository. This is a more reliable indicator.[8]

Measuring use of a new TAL We can measure whether a new Trust Anchor Locator (potentially with a new algorithm) is being used by individual RPs, by logging requests that fetch the TA certificate. This is useful both during a 'normal' algorithm A to A TA key rollover, and an algorithm migration, as discussed in 6.2.3.2.

Measuring effect on routing Finally, we can measure the actual effect that a migration would have in real-world routing, by adapting methodologies that are currently being used to measure ROV adoption in general. Here, we can also combine looking at (1) route propagation with BGP collectors [29, 31], (2) testing reachability from inside networks with probes [60, 31, 63], and (3) testing reachability from real users with advertisement-based measurements [34]. Together, this gives a thorough overview of the real-world impact of a migration.

While each experiment requires significant setup, they share many components. We recommend performing a combination of these methods. By overlapping their experimental setups, the additional effort for performing more than one of these experiments is minimal, but the combination of their results provides high confidence in the readiness of RPs. Only when the combined measurements show widespread acceptance of algorithm B among RPs that are used for filtering, the first large production CAs can safely transition.

[7] For example, outdated software might be run by researchers, instead of being used for ROV filtering, and some validators might be reused for filtering in many networks at once. Also, despite a recent version of software being reported, an RP might have disabled algorithm B or not been configured for new B TALs.

[8] This can also measure adoption by `rsync`-only RPs, and measure actual acceptance of B instead of software versions.

6.2.3 Single CA algorithm roll

During phase 2 — and for experimental purposes also before then — CAs can unilaterally migrate to a new algorithm. This is possible by simply following the existing procedure for key rollovers [RFC6489], just with the old key using algorithm A, and the newly created key using algorithm B.

Technically, key rollover consists of creating a 'NEW CA', reissuing all products of the 'CURRENT CA' and finally placing the NEW CA products in the exact same locations as those of the CURRENT CA. The NEW and CURRENT CA represent key pairs: they are only logically different, not organizationally. In line with [RFC6489], the steps are:

1. "Generate a new key pair for the NEW CA." For the algorithm roll in particular, the new key pair uses algorithm B.
2. "Generate a certificate request with this key pair, and pass the request to the CA that issued the CURRENT CA certificate." If the parent CA allows using algorithm B, this results in the publication of the NEW CA's certificate.[9] The NEW CA has no published products yet, and the CURRENT CA is still in use.
3. "Publish the NEW CA's CRL and manifest." The CRL is initially empty, and the manifest indicates the CRL as the only published object.
4. "The NEW CA enters a Staging Period." This is a period of at least 24 hours, during which the NEW CA reissues all products of the CURRENT CA. Copies of every issued certificate and signed object are created, with identical content but a different issuer. For any subordinate CA, a new resource certificate is made, pointing to the same subordinate publication point, but now signed with algorithm B.
5. "Upon expiration of the Staging Period, the NEW CA MUST publish the signed products that have been reissued under the NEW CA, replacing the corresponding products issued under the CURRENT CA at the NEW CA's repository publication point." When this has been done, the CURRENT CA becomes OLD and the NEW CA becomes CURRENT. Relying parties can validate the algorithm B products.
6. "Generate a certificate revocation request for the OLD CA certificate and submit it to the issuer of that certificate."

An overview of the migration for a single CA is shown in fig. 6.3.

6.2.3.1 Aborting

If there is, despite the measurements recommended in 6.2.2, any concern that some RPs might not validate B products yet, there is a possibility to quickly abort the rollover before executing step 6. Consider a scenario where some RPs do not accept B. This can lead to problems only after step 5, when the old products are replaced with algorithm B counterparts.

Some time after step 5, operators can notice that RPs fail to validate the algorithm B equivalents of the withdrawn products. If problems are detected, the OLD CA can quickly be reinstated, by reverting step 5. The OLD CA's products, that have not been revoked yet, can be published again, in their original location, without needing to reissue them, so, just like performing step 5, reverting it can also be done in a matter of minutes.

Although [RFC6489] recommends that step 5 be executed over the span of not more than a few minutes, and should be seen as atomic by RPs, it is also possible to perform step 5 in multiple smaller parts. The CA can replace a subset of the

[9] The NEW CA certificate has an algorithm B subject, and can have either an A or B issuer.

products first, monitor for problems, and only proceed with the rest when no issues are detected.[10]

The option to postpone step 6 and revert step 5 — or even do step 5 in multiple stages — can be particularly useful for the first CAs that transition in phase 2. After several CAs migrate successfully, the risk diminishes, allowing others to complete step 6 more confidently.

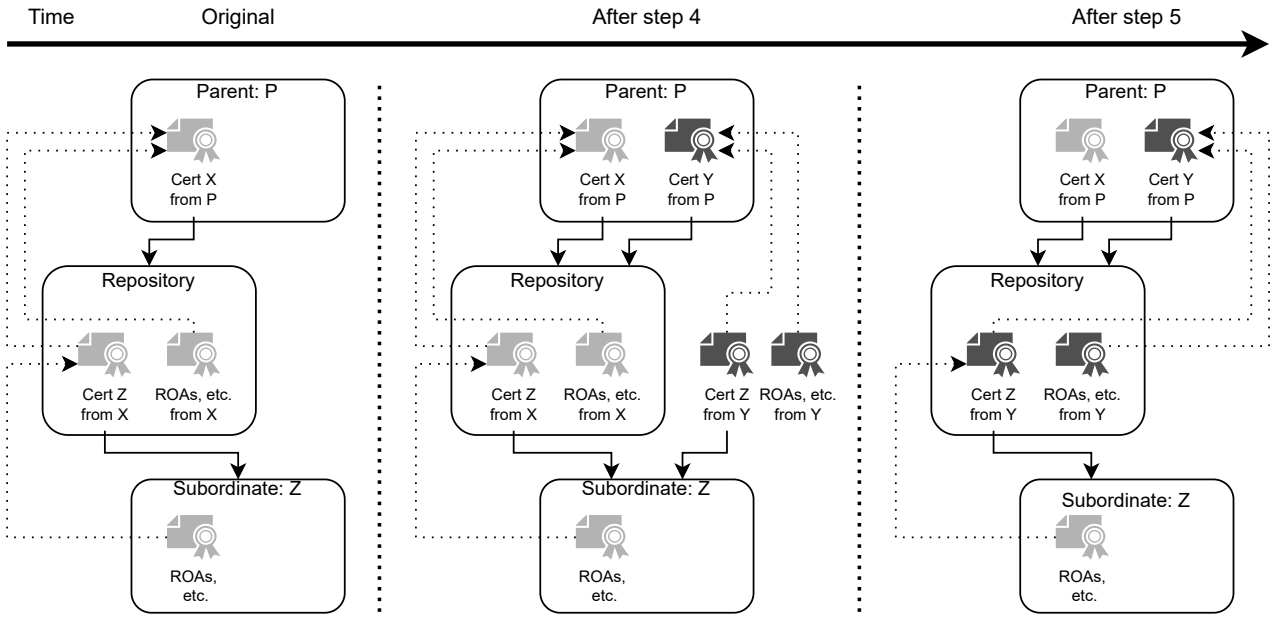


Figure 6.3: *Key rollover for a single CA, from key pair X to Y. Light icons indicate the old products and dark icons indicate new products. The dotted arrows indicate the Authority Information Access (AIA) extensions in each object, pointing to the issuer's certificate. During step 4, the parent P has published a new certificate with subject Y (an algorithm B key pair) pointing to the same publication point as the old certificate. At step 5, the new objects, prepared (but not published) during step 4, are published at the exact location of their old counterparts, so the AIA extensions in Z's products point to the new certificate for Z, without Z's involvement. CRLs and manifests are not shown for simplicity; from step 3 up to and including 5, there are actually two CRLs and manifests in the single, shared publication point.*

6.2.3.2 Root CAs

An important special case of a key rollover is that of a trust anchor. TAs have no parent CA, so requesting a new certificate is not possible. Instead, the TA operator generates a new key pair, creates a new self-signed (TA) resource certificate, and publishes a new [RFC8630] Trust Anchor Locator (TAL) that points to the new certificate. While TALs have so far only been distributed out-of-band, [RFC9691] introduces an in-band method to announce a successor for a TA key, in a Trust Anchor Key (TAK) object. This supports distributing a TAL without RP operators' involvement, yet it does not significantly change the process needed for a TA key rollover. The process for a TA key rollover is visualized in fig. 6.4.

[10] Place a few of the new *B* objects in the places of their *A* counterparts. Then add these to the NEW manifest and remove them from the CURRENT manifest. Later, replace the rest, add everything to the NEW manifest, and empty the CURRENT manifest.

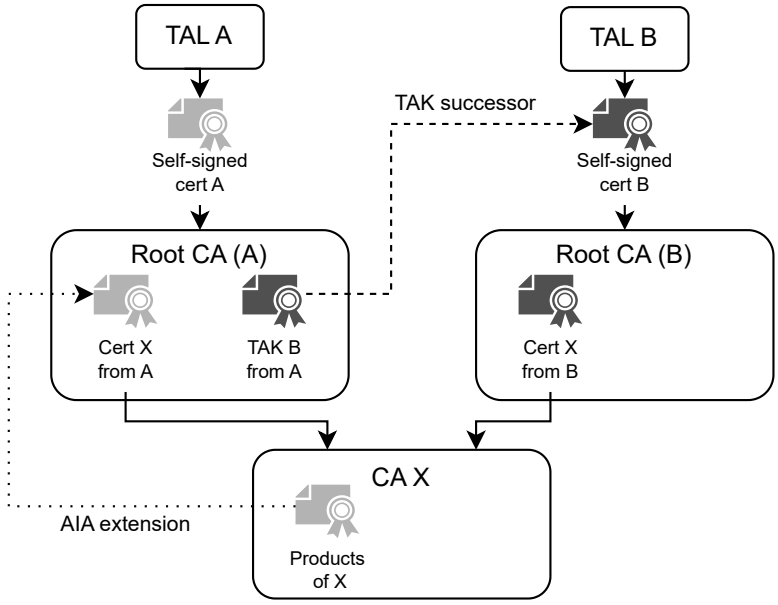


Figure 6.4: *Overview of a TA key rollover, from key pair A to B. The dark icons represent the newly created objects. The TAK object is specific to [RFC9691], and can be left if every RP manually adds TAL B after receiving it out-of-band.*

Using the new certificate, all products issued by the old TA instance need to be reissued under the new instance. However, unlike for normal CAs, the reissued products cannot replace their old counterparts, as initially, RPs will not have the new TAL yet. Thus, the new products are to be published at a separate location.

Note carefully that this means that, until the direct subordinates update the AIA extensions in their signed products, the subordinates' products will have an AIA that only matches the certification path to the old TA.

The profile for resource certificates ([RFC6487]) requires that the AIA extension is present, but does not explicitly specify how the extension should be validated. The relevant step of validation is:

> The certificate contains all fields that MUST be present, as defined by this specification, and contains values for selected fields that are defined as allowable values by this specification.

As long as the old TA instance is still in use, the old AIA value is indeed an allowable value. However, when validating top-down from the new CA instance, this requirement is ambiguous and historically caused confusion [62, 76].

[RFC9691], which defines the TAK object, explicitly states that the outdated AIA extensions are to be ignored.

> Finally, note that the publication locations of CA certificates for delegations to child CAs under each key pair will be different; therefore, the Authority Information Access 'id-ad-caIssuers' values (Section 4.8.7 of [RFC6487]) on certificates issued by the child CAs may not be as expected when performing top-down validation, depending on the TA public key that is used. However, these values are not critical to top-down validation, so RPs performing such validation MUST NOT reject a certificate simply because this value is not as expected.

Although [RFC9691] is not implemented in common RP software yet, it appears that current common RP software does not reject a certificate whose AIA points to an unexpected location.[11] Furthermore, even if an unexpected AIA value leads to rejection of a certificate, RPs that have only the old TAL will observe a correct AIA value until subordinates update it, and RPs that have both TALs will observe a correct AIA from one starting point, and an incorrect one from the other, still leading to successful validation under one of the TALs, so even rejecting on unexpected AIA values need not be a problem. Still, a TA key rollover has not yet been executed in practice, so care should be taken when performing one to ensure that the temporarily inaccurate AIA extensions do not cause problems.

While TA key rollover is by itself not trivial, and unprecedented, it will definitely need to happen at some point (even if the algorithm does not change). The transition to a new algorithm in a rollover does not make any difference to the process, so while TA rollover by itself is challenging, we believe that it does not pose any additional challenges when combined with a transition to a new algorithm.

6.2.4 Deprecating algorithm A

Once a new algorithm B (or even several new algorithms) has been adopted, an old algorithm A can be deprecated. There is a spectrum of options that can be chosen depending on the urgency of deprecation. Note that, in contrast to [RFC6916], deprecating A is not required to start enjoying the protection from algorithm B.

It is possible to use wording in the initial document that obsoletes [RFC7935] to give directions to CAs, but also to later publish another document that further deprecates algorithm A.[12]

B is equal to A. In the initial replacement of [RFC7935], two algorithms can be presented without a preference: CAs may choose either one. With this approach, deprecation of A is not enforced. While B might become ubiquitous by the choices of CAs, it is still allowed to keep using A forever.

Recommend migrating The initial document can recommend the use of algorithm B over A when generating new keys, or state that CAs *should* migrate.

Introduce B as mandatory. The initial document can mandate using algorithm B as soon as possible. For example, it can state that CAs must not create new A keys, and should perform a key roll to B keys as soon as their parent supports it. It can even suggest that parents may refuse to renew certificates for A subordinates, giving them leverage to enforce adoption.

This spectrum of wording in the first replacement of the algorithms document can guide adoption of B, but must keep verification of A products mandatory at least during the transition. For example, "RPs MUST validate both algorithm A and B signatures", regardless of whether making new A products is still allowed. An initial document could already allow removing that support once it is no longer needed in practice: "RPs MUST validate both algorithm A and B signatures, *until A is no longer used*", but to avoid ambiguity about when ending support for A is acceptable, it is also possible to postpone this to a next iteration of the algorithms document. In a later version (that might initiate another algorithm introduction, or only clean up A) all references to A can be removed, or validation of it can be made optional.

[11] For FORT validator v1.6.6 see `https://github.com/NICMx/FORT-validator/blob/1.6.6/src/object/certificate.c#L1873-L1882`. For Routinator v0.14.2 see `https://github.com/NLnetLabs/rpki-rs/blob/v0.18.5/src/repository/cert.rs#L955-L964`. For rpki-client see `https://github.com/openbsd/src/blob/d48ea33/usr.sbin/rpki-client/cert.c#L1048-L1051`.

[12] While we discuss introduction of B and deprecation of A, our proposed procedures also work with multiple algorithms, such as introducing both B and C, and deprecating only A or even both A and C.

6.3 Comparison

Our proposed method differs from [RFC6916] in two fundamental assumptions.

- Introducing and deprecating an algorithm can be separate. Our procedure does not make the assumption that the RPKI will always allow only one algorithm.
- We require that all RPs be first to update, instead of all CAs.

The former is not a restriction, but offers more flexibility. The latter can be restricting, though we believe it to be a reasonable requirement. We evaluate the implications of the two different underlying assumptions in order, and then compare the resulting characteristics of the two procedures.

6.3.1 Separation of introducing and deprecating

The [RFC6916] procedure fundamentally couples the introduction of algorithm B with the deprecation of algorithm A. Foregoing this coupling does not impose any restrictions on the possible migrations: doing a transition from a single algorithm A to a single B is simply a sequence of first introducing B and later deprecating A.

On the contrary, the coupling in [RFC6916] rejects the possibility that multiple algorithms might intentionally be allowed. There can be merit in having multiple mandatory-to-implement algorithms for relying parties.

- In section 4.6.2 we discussed that having a fallback algorithm, based on a different hardness assumption than the main algorithm, is a good idea. If one algorithm is broken, RPs are already prepared to accept products signed with the other algorithm, so that CAs can quickly switch. If this is done, RIRs should also publish (fallback) TALs for the fallback algorithm, despite not actually publishing anything with it yet.
- As suggested by Dickson [22], it is — given the long duration of a migration — conceivable that the proportion of time when only one algorithm is allowed may be less than the vast majority. While this makes the temporary operational complexity and performance overhead of [RFC6916]'s parallel trees unattractive, there is also a further risk: it might be necessary to start an algorithm migration even before the first one is complete.[13] In [RFC6916], the options are then to wait for the first migration to complete, to abort and restart the first migration, or to use *three* parallel trees.

  Our proposal handles this gracefully. By separating introduction from deprecation, it is naturally possible to introduce a third algorithm C before deprecation of A has finished.

Without assuming that only one algorithm should be allowed, using mixed certificates is natural.[14]

6.3.2 RPs update first

The most significant difference between our approach and [RFC6916] is that we assume that RPs update before CAs. This is a consequence of the previous difference. To avoid the foreseen exponential — or more likely linear in the number of algorithms — growth[15] that led to [RFC6916], we drop compatibility with old validators slightly

[13] For example, removing hybridization once post-quantum signatures are trusted enough, increasing the security parameters, or replacing one hybrid with another due to the post-quantum component being broken.

[14] With multiple algorithms and no mixed certificates, separate trees would be needed for every algorithm.

[15] Each CA creates duplicate objects for each possible combination of algorithms (exponential), or more sensibly only *A*-*A*, *B*-*B* (linear), to retain compatibility with old RPs. See 6.1.3.3.

earlier.

The requirement that all RPs must support algorithm B before production CAs start migrating seems stringent, but we believe it is both feasible and preferable given the current RPKI ecosystem.

Essentially, it seems that updating RPs is easier than updating CAs for multiple reasons.

Few RP implementations Currently, only three well-known, supported RP software implementations are available. From measurements at an RRDP repository (as in appendix B.2) run by NLnet Labs [39, 40], we see that roughly 4000 relying parties use one of these three implementations. About 100 are using the discontinued RIPE Validator 3[16] and OctoRPKI[17]. So, it is easy to coordinate rolling out updates to the few implementations, and to wait for them to be adopted.

Adoption of such new versions will take a long time: discontinued implementations are still in use, and a significant fraction of RPs is running outdated versions of the supported implementations [39, 77].

Job Snijders suggests that one reason for this could be that for many operating systems, only old versions of the software are available in the package manager [70]. So, improving the availability of new versions in package managers is one way to help speed up adoption. Given the relatively low number of validators, efforts to raise awareness among operators, for example through mailing lists of network operator groups, could also help promote new versions.

These steps would be wise anyway, as some used versions are still vulnerable to attacks presented in [32] and recently discovered vulnerabilities in `rsync` [17, 18].

CA updates are harder The CA side of things is more complex, as there are big differences between CAs in terms of size, whether they have subordinates, use of HSMs, and so on. Consequently, many CAs have made their own implementations, of which they are the sole user. This makes migrating CAs organizationally complex, requiring considerable manpower for changes to proprietary software, configuration, and possibly investment in new HSMs. In particular in the [RFC6916] migration, rather complicated software is necessary to keep the two trees synchronized.

Compared to the minimal cost of accepting a simple update of open-source validator software, the cost for CAs is much higher. In turn, this makes it likely that they take a long time to update, especially for [RFC6916] where a top-down order is required.

In contrast, our technique is both much simpler for CAs, and does not impose the top-down order. This allows CAs to migrate in their own time, without having to wait for their parents, and their subordinates having to wait for them.

CAs have no incentive Apart from the technical and organizational complexity for CAs, they also have no incentive to migrate before RPs do. When an RP starts offering algorithm B products, this initially offers no security benefit, yet comes at significant cost. With [RFC6916], the security benefit comes only after the *Twilight Date*, when all CAs have switched for a long time, *and* all RPs must have started accepting algorithm B products. Thus, there is no incentive for early adoption by CAs. Dickson calls this a *"stick"-only* approach, as opposed to the *"carrot" and "stick"* when mixed certificates are used.

[16] `https://github.com/RIPE-NCC/rpki-validator-3/`
[17] `https://github.com/cloudflare/cfrpki/`

Together, this causes us to believe that a transition where RPs update first is feasible, and could be faster than one where CAs update first top-down.

6.3.3 Resulting aspects

The two different assumptions lead to a number of differences in the resulting procedures. In most aspects, mixed certificates offer an advantage over parallel trees.

Easier for CA Our proposal should be simpler to implement for CAs. CA software doesn't need to handle parallel trees, only a slight variation on the more common and simple key rollover procedure. We demonstrate this in chapter 7 through a proof-of-concept in Krill. The required changes are minimal, and algorithm rollover works with Krill's existing [RFC6489] support.

No performance cost for parallel trees Similarly, parallel trees have more operational cost: performance overhead for RPs, as well as increased hosting cost for repositories. While this cost is not permanent (only during migration), we have seen that migrations can take a long time.

Risky for first CAs One potential downside of our approach is that there is some risk to the first non-experimental CAs that migrate, as they will drop their A products at once.

The gravity of this risk depends on the content that a CA publishes. Often, only a single CA publishes objects about a particular resource. In that case, ROV (ASPA works similarly) has a fail-open quality, where unavailability of ROAs causes legitimate announcements to change from ROV-Valid to ROV-NotFound, and not ROV-Invalid (see section 2.2.2.2). However, there can be cases where another CA has algorithm A products for the same prefix, such that RPs that understand only algorithm A. Then, replacing the A products with B products can cause legitimate announcements to change from ROV-Valid to ROV-Invalid from the perspective of outdated validators, potentially making a prefix unreachable.

The risk when using a mixed-tree strategy, because:

- our extensive monitoring methods in section 6.2.2 can be used to confidently measure whether RPs will accept algorithm B;
- a CA can inspect the RPKI to see whether any other CAs impose this risk for its resources;
- as a contingency plan, CAs can abort their migration in minutes, as described in 6.2.3.1.

A similar risk applies to the *Twilight Date* in [RFC6916], where it is also assumed that all RPs have updated before disabling the A tree. Some of the monitoring methods in 6.2.2, could similarly be applied to [RFC6916] to monitor whether the *RP Ready Alg. B Date* deadline has been met in time. Still, establishing milestone dates far in advance discourages decision-making based on real-time insights, as postponing milestones is a lengthy process.

Overall, both procedures have some risk for the first CAs when they remove their A products, but also offer ways to mitigate this risk.

Laissez-faire The [RFC6916] procedure requires global coordination, and a top-down order. It is difficult to achieve consensus about particular milestone dates years ahead of them. To our knowledge, imposing future dates in a BCP RFC is unprecedented, and it seems unattractive for operators to be bound to such dates. Instead, our laissez-faire approach allows CAs to determine their own timelines individually. This can make the actual transition easier and faster, and

it may avoid hesitation of operators that could slow down the phase leading up to the transition.

Early experimentation The mixed-tree approach allows for early experimentation in the real RPKI. A leaf CA can be used that makes B products, and updated RPs can already validate them. This makes it possible to start the transition long before actual standardization, as is e.g. also being done with ASPA objects. This way, operational experience can guide standardization, giving more confidence in the changes before they are finalized.

Taking such pre-standardization experiments even further, it is even possible and attractive to update all RPs as soon as possible. RP implementations could be changed to accept algorithm B *right now*. This possibility can then be kept dormant for a long time, without CAs switching yet (except for a few leaf CAs for monitoring). This involves no operational cost to CAs for maintaining parallel trees, and gives lots of time for RPs to adopt updated software. Since the RPKI only needs to provide integrity (no risk of *store-now-decrypt-later* attacks), this can be used to prepare for a migration long in advance, with no risk or cost. At the last minute, when the quantum threat becomes real, the actual migration of CAs (to a less performant algorithm, and with some risk) can be done very quickly as RPs were already updated long before.

So, the mixed-tree transition enables a timeline with very early updates to RPs at no risk or cost, while postponing the more risky and costly migration of CAs until it is really necessary. In [RFC6916], this is not possible because (1) the timeline needs to be coordinated far ahead of time, (2) CAs are required to move first, and (3) there is increased operational cost throughout the transition.

Early protection Finally, there is a difference in when the protection of algorithm B takes effect. In [RFC6916], the protection applies once relying parties stop using A products; this is roughly by the *Twilight Date*.[18] At that point, *all* CAs have been producing B products for a long time, and RPs also must have updated some time ago. Until this time, while B products are available and RPs can use them, the remaining support for A makes the RPKI vulnerable to a downgrade attack. In short, [RFC6916] offers the new level of protection after at least two prerequisites have been met:

(a) all CAs provide B products,

(b) all RPs have updated to accept (only) them.

The protection applies at the same time for every resource (prefix, ASN).

On the other hand, the mixed-tree alternative starts providing protection for each CAs resources individually. The prerequisites for one holder's resources to enjoy algorithm B security are:

(a) the particular CA and its ancestors[19] have switched to B,

(b) all RPs have been updated to accept B (as well as A products under B roots).

These requirements are strictly weaker:

- Both approaches require all RPs to update before the protection takes effect.

[18] Before the *Twilight Date*, RPs *must* use B, but *can* revert to A products. Afterwards, using A is not recommended.

[19] Since each TA has (or could have, through forgery) a root certificate with *all* resources (the `/0` prefixes and all ASNs), it is also necessary that all 5 TALs have been replaced. Alternatively, [75] could offer a way to limit the scope of RSA trust anchors when some RIRs are still using RSA.

- In the mixed-tree transition, a CA can achieve the protection for their resources long before the last CA has switched, as long as its parent has done the same.

In particular, users of the few hosted CA offerings — the vast majority of resource holders — can be protected early, while giving delegated CAs as much time as they want to update.

6.4 Recommendations

We have now evaluated post-quantum candidate algorithms and their performance impact, as well as possible migration procedures. In this section, we integrate our findings into a set of actionable recommendations for the RPKI community, categorized into the following topics:

- The selection of algorithms to introduce and the migration of the RPKI itself, which have been the main focus of this thesis.
- Replacing the [RFC8183] BPKI trust anchors, which is an important first step to transition ahead of the rest of the RPKI.
- The need to update various related protocols on which the RPKI relies.
- Standardization of the new algorithms and the migration approach.

The following sections elaborate on these topics.

6.4.1 Migration and set of algorithms

For the migration of RPKI's main use of signatures, we recommend adopting the mixed-tree approach for two decisive reasons:

1. We believe the mixed-tree approach to enable a much simpler migration than [RFC6916]. It is more attractive in particular for CAs, which makes it likely that the community can start adopting it soon.
2. The mixed tree supports migrating to a state where multiple algorithms are allowed. This is necessary for the recommendations below, that offer operational advantages during the migration, and improved security.

With this kind of transition, we then suggest the following.

Introduce two post-quantum options As discussed in section 4.6.2, it would be wise to introduce not one but two post-quantum algorithms. One should be the overall winner: Falcon-512 in a hybrid. The other, while not intended to be used in practice, should be a fallback option that is based on a different hardness assumption. With the lattice-based Falcon as main replacement, a fallback could be the multivariate MAYO, or even less performant options may be acceptable as the fallback is only meant as a contingency plan. The addition of a fallback algorithm may be postponed, for example waiting for NIST's standardization of a non-lattice-based schemes while already introducing Falcon.

Introducing more than two schemes is also possible. Using many different algorithms is inconvenient for implementers, but adding stronger parameter sets for future-proofing (e.g. Falcon-1024 with a corresponding traditional component and longer digests) seems sensible.

Implement our null scheme To mitigate the cost of introducing bigger signatures and public keys, the *null scheme* we define in chapter 5 should be implemented. This can almost make up for the increased sizes of post-quantum algorithms. Introducing the null scheme at the same time as the post-quantum algorithm(s)

is convenient, requiring RPs to update only once. Furthermore, acceptance of the null scheme is somewhat harder to monitor than that of a whole new algorithm, because it applies only to [RFC6488] objects, not to resource certificates, so measuring through access to repositories (appendix B.2) is not possible. Thus, it is safer to introduce the null scheme together with the new algorithm(s) than introducing it separately.

Update (only) RPs very soon We expect widespread adoption of new RP software versions to take a very long time. Thus, the earlier new versions with support for the new algorithms are available, the better.

Despite our *null scheme*, switching to a new algorithm involves significant cost for CAs. On the other hand, for RPs, the cost of *accepting* a new algorithm is negligible. So, it is practical to first roll out new RP versions and wait for their adoption. The capability of accepting a new algorithm should then be kept dormant for a long time. This gives CAs ample time to prepare their migration, without suffering the performance cost of the new algorithm.

The actual algorithm rollovers of individual CAs (except for some early adopters for experimentation) can then be postponed until the last minute, when the quantum threat becomes credible.

Updates to validators can be rolled out as soon as standardization of post-quantum algorithms is complete. Validators do not depend on availability of HSMs that can sign with the new algorithms, which can take more time after a signature algorithm's specification is finalized.

Additionally, validators can be updated *before* an update to [RFC7935] is finalized. For example, `rpki-client` has been updated with experimental support for ECDSA (allowing mixed certificates), which is not specified in [RFC7935].

Publish future TALs soon One important detail for the above idea of updating RPs first and postponing migration of CAs for a long time, is that we do need RIRs to already create and publish TALs for the new algorithms. This way, they can already be incorporated in the new RP versions, despite the TALs not being used for any published products yet.

Perform monitoring experiments Finally, multiple of the experiments from section 6.2.2 should be performed to keep track of the adoption of updated validator software. This, too, can be done ahead of standardization.

These are all relatively simple steps that can be taken in the upcoming years, without needing global coordination, and without introducing risk or performance impact. Once the quantum threat becomes real, CAs will be able to quickly switch to the post-quantum replacement. At that time, RPs will already support the new algorithm and have updated TALs, such that the switch can be fast. Protection against quantum-enabled attacks takes effect for individual CAs as soon as they switch, and RP operators have disabled the old TALs.

6.4.2 Replacing RFC8183 trust anchors

A different but no less important migration is needed for the channels between CAs and their subordinates, as well as CAs' repositories, typically authenticated with a [RFC8183] BPKI. This channel is a particularly attractive target for a quantum attacker (see section 3.3.1).

Migrating this channel to use post-quantum signatures could theoretically involve only the two parties that use it. The channel is invisible to RPs and all other CAs. As such, it can and should be done much sooner than switching the signatures used in the RPKI itself.

As we've seen in 4.6.3.1, it can be considered to use a different, stronger scheme for it, because performance is not as important. Replacing them again is (currently) not practical: it could in principle be done by *manually* redoing a [RFC8183] exchange.

6.4.2.1 Current implementation

In practice, there is poor support for replacing BPKI trust anchors. [RFC8183] does not specifically provide for repeating the out-of-band TA setup, although simply repeating the procedure could be expected to work. A problem is that actual software (that is, at least Krill) makes the assumption that each CA has a *single* so-called ID cert, that it uses for authentication towards *all* other parties. With this approach, regenerating this ID cert[20] directly breaks the channel with every communication partner. This means that not only a CA's channels to parents and repositories need to be re-established quickly, but this also needs to be done by all the CA's subordinates.

To support a *local* BPKI migration, a CA should at least be able to maintain previous ID certs, for example keeping a mapping from the peer's 'handle' to the key that was shared with it. Then, two parties can locally update their BPKI TA either by truly re-running the [RFC8183] procedure, or by some other means to be defined in the future.

At the IETF 115 meeting, Tim Bruijnzeels already brought up the lack of a way to replace BPKI TAs [10]. He also drafted an idea for a successor to [RFC8181] (communication between CA and publication server) that includes a way to update BPKI TAs between CAs and publication servers without using out-of-band communication [11]. Similarly, another draft ([46]) adds new messages to both [RFC6492] and [RFC8181] at the same time. Even without changing protocols, software can at least change to enable manually updating TAs independently per communication partner.

As no practical implementations for BPKI TA rollovers exist yet, we recommend this as a relatively straightforward action item for implementers and the community to pick up. The BPKI is an easy target for quantum attackers, but should also be easy to protect due to its local nature.

While there is no good support for BPKI TA replacement right now — and we strongly recommend implementing that — preparing CA software to default to post-quantum algorithms for *new* relations is already helpful before a good TA replacement procedure is available. This way, operators of new CAs don't have to repeat the setup later on. As soon as a way to update BPKI TAs in place is available, CAs should be encouraged to quickly upgrade their existing BPKI TAs, because it can make quantum-enabled attacks much harder, even when both involved CAs are still using traditional cryptography for their resource certificates.

6.4.3 Other technology

Of course, CAs and repositories also need to adopt post-quantum algorithms for TLS and DNSSEC to prevent attacks on their hosted RPKI offerings, transport between repositories and validators, and the out-of-band communication used in [RFC8183]. In particular, while our work is concerned only with signatures (that are not at risk of *store-now-decrypt-later* attacks), confidentiality of TLS traffic is important to protect password-based authentication of RIRs' web interfaces. This is urgent, as attackers could capture and store traffic to such interfaces right now, and decrypt it years later to obtain credentials. Multifactor authentication can also help in this regard.

[20] There is an API endpoint that triggers this, but was implemented primarily for testing, not for actual use: `https://github.com/NLnetLabs/krill/blob/v0.14.5/src/daemon/ca/manager.rs#L500-L510`.

6.4.4 Standardization of mixed-tree migration

Finally, both the selection of new algorithms and the migration procedure will need to be documented somehow.

For the changes to the algorithms that are used, a new version of [RFC7935] will clearly need to be created. The first steps of our recommended actions can be done ahead of standardization. First, consensus should be reached among RIR operators and software maintainers, about the algorithms to be introduced. Then, a new algorithms profile can be drafted and implemented in validator software. It does not seem necessary to await the full standardization process before implementation: successful real-world deployment is excellent evidence to support an internet draft.

Replacing the algorithms profile [RFC7935] is the only required change in standardization to perform a mixed-tree migration. The new algorithms profile can simply specify that both RSA and the new algorithm(s) are allowed; the transition does not require any further global coordination. However, it seems wise to also document explicitly that [RFC6916] is no longer the recommended procedure for algorithm migration. To make this clear, another (BCP) RFC can be created that describes the mixed-tree migration procedure, obsoleting [RFC6916]. This new algorithm agility procedure would be much more flexible than [RFC6916], allowing introduction and deprecation of algorithms independently, and without imposing that a coordinated timeline is laid out in advance. Hence, standardizing the algorithm agility procedure is not a prerequisite for *performing* it. Consequently, the community can decide to either

(a) first document the mixed-tree procedure, replacing [RFC6916], before starting the first migration by updating the algorithms profile, or

(b) perform the migration first by updating the algorithms profile, and later (when a migration has successfully been performed) document the mixed-tree procedure that is then already proven effective.

The latter option seems more attractive, allowing the migration to post-quantum schemes to start early, and documenting only a procedure that is already known to work. This aligns with the SIDR and SIDROPS working groups' tradition of requiring implementation experience before finalizing a standard. It also avoids the risk of standardizing a procedure that is — like [RFC6916] — later deemed inadequate.

Overall, we suggest to first reach consensus on introducing new algorithms through a mixed-tree approach, and to document this in a new algorithms profile to replace [RFC7935]. Documenting the mixed-tree approach for repeated use in the (far) future can wait until the first migration is proven successful, but if the community disagrees, the opposite order is also possible.

Chapter 7

Implementation

To demonstrate the feasibility of our proposed mixed-tree migration approach, we implement a proof of concept (PoC) in commonly used RPKI software: Routinator and Krill. We also publish the source code, as a starting point for further experimentation, such as performance measurements and interoperability testing.

Our implementation serves as evidence that (1) acceptance of post-quantum signatures in a mixed-tree setting can easily be integrated into existing RPKI software, and (2) the mixed-tree migration approach as outlined in chapter 6 works in practice.

7.1 Design

We have implemented a post-quantum signature scheme for the mixed-tree approach in Routinator and Krill. These two software packages are widely used in the RPKI ecosystem, and rely on a shared underlying Rust crate `rpki`, which is responsible for the core functionality of the RPKI.

Choosing the combination of Routinator and Krill is not only a natural choice because of their popularity, but also allowed us to leverage the shared crate to avoid duplication: in the end, no changes at all were needed in Routinator, only in Krill and the shared crate.

7.1.1 No hybrid yet

While we propose in chapter 4 to use a hybrid signature scheme with Falcon-512, our implementation uses Falcon-512 directly. The reasoning for this is twofold.

First, details of the hybridization of post-quantum signatures are still under discussion in the IETF. Precise formats of signatures and public keys are not quite finalized yet.

Second, we chose to use `liboqs-rs` for our implementation of post-quantum signature schemes, as it provides a straightforward Rust interface to the `liboqs` C library. `liboqs-rs` does not support hybrid signatures. An alternative would be to use `oqsprovider`, an extension to OpenSSL, that supports a selection of hybrid signatures. However, it is more difficult to package and use in Rust.

For a real implementation, we recommend using hybrid signatures through OpenSSL once native support becomes available, or alternatively, using `oqsprovider` in the interim. This remains as future work, as it wasn't essential for demonstrating the viability of the mixed-tree approach.

7.1.2 Changes

The main steps to support validation of mixed certificates with Falcon-512 were:

- Adding variants to the `PublicKeyFormat` and `SigningAlgorithm` enums in the `rpki` crate to support Falcon-512.

- Changing the `RpkiSignatureAlgorithm` struct (representing the RSA signature algorithm) to an enum that also has a variant for Falcon-512.
- Adding implementations of (de)serialization and verification of Falcon-512 signatures and public keys.

These changes were straightforward, as the `rpki` crate was designed to already neatly represent signatures and public keys of various algorithms. With these minimal changes in place, Routinator could be built without any further modifications, and it was able to validate certificates signed with Falcon-512.

Slightly more involved was the implementation of *creating* Falcon-512 products. The `rpki` crate includes a `Signer` trait that handles creating signatures. There are several implementations of it, to support hardware security modules and software signing with OpenSSL. To make use of `liboqs-rs`, we opted to create a separate `OQSSigner` implementation of the `Signer` trait, particularly for creating post-quantum signatures. Krill uses a configuration file where multiple signer instances can be defined, with one being preferred for new key pairs, but alternatives being available to allow using old key pairs they contain. As we introduce a new signer type for post-quantum signatures, this configuration can be used to switch between the old and new algorithm, while still keeping the RSA keys available.

With our new `OQSSigner`, different configurations can be used throughout a migration to set the supported and preferred algorithms. First, our updated Krill can run with a configuration that only uses OpenSSL, to maintain current RSA signing.

```
default_signer = "OpenSSL signer"
one_off_signer = "OpenSSL signer"

[[signers]]
type = "OpenSSL"
name = "OpenSSL signer"
```

A second signer can be added to prefer post-quantum signatures, while still allowing use of existing OpenSSL keys. This does not trigger a rollover yet, but means that newly generated keys will be post-quantum.

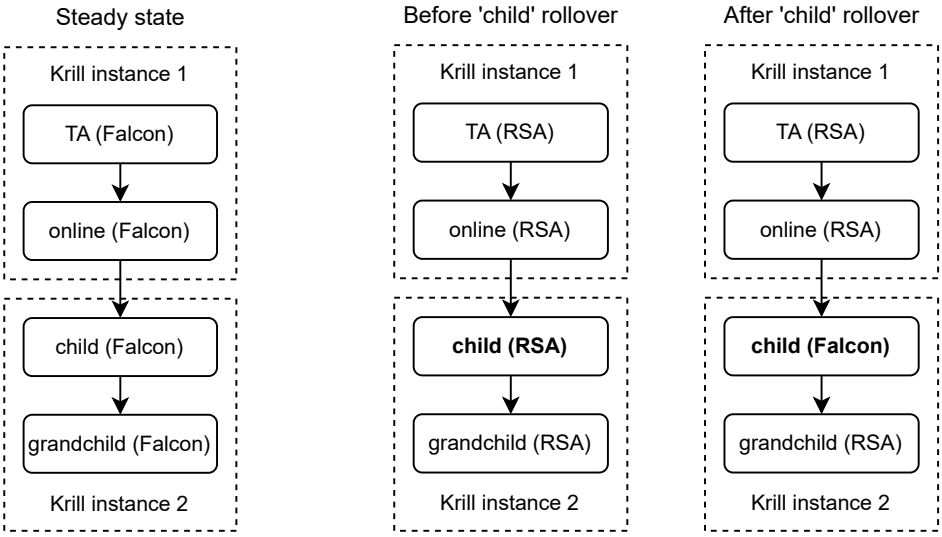
```
default_signer = "OQS signer"
one_off_signer = "OQS signer"

[[signers]]
type = "OpenSSL"
name = "OpenSSL signer"

[[signers]]
type = "OQS"
name = "OQS signer"
```

If in this state a new CA is created in the Krill instance, it will use a post-quantum key. Any new ROAs will use a post-quantum one-time-use key in the ROA's EE certificate. An actual rollover as described in section 6.2.3 can easily be triggered with the `krillc rollover` CLI. Even switching back, while keeping keys available, is possible by changing the `default_signer` and `one_off_signer` configurations, and doing a rollover again.

The source code of our modified Routinator and Krill is available at `https://github.com/SIDN/pqc-routinator/` and `https://github.com/SIDN/pqc-krill/`, and on request from the author and Radboud University.

7.2 Evaluation

Using our modified versions of Routinator and Krill, we demonstrate that the migration we proposed in chapter 6 works in practice, and is convenient. We have created a set of testing scripts that automate the process of setting up an RPKI testbed, including a trust anchor and several CAs in Krill, and Routinator to validate the CAs' products. Both the scripts and their output are available at `https://github.com/SIDN/pqc-rpki/`, as well as on request from the author and Radboud University.



(a) *RPKI tree during the steady state test, with all CAs using Falcon-512.*

(b) *RPKI tree during the single CA rollover test. Initially, each CA has an RSA key pair. The* child *CA then performs a key rollover to a Falcon-512 key pair, while the others remain unchanged.*

Figure 7.1: *RPKI trees used in our tests.*

7.2.1 Steady state

The first step to demonstrate that our implementation works is to comprehensively test a steady state where the RPKI fully uses post-quantum signatures. We do this by creating a complete RPKI environment consisting of:

- A trust anchor
- An *online* CA under the TA with its own publication point
- A *child* and *grandchild* CA, sharing another publisher

The trust anchor and *online* CA run in one Krill instance, and the *child* and *grandchild* run in another, so that the communication between Krill processes is also tested. Both Krill instances are configured with only our `OQSSigner`. As the environment is created from scratch with post-quantum algorithms, the [RFC8183] BPKI trust anchors are also created with Falcon. The structure used in this test is shown in fig. 7.1a.

Each CA gets some internet resources and publishes some ROAs. We then use Routinator, configured with the trust anchor's TAL, to check that the implementation creates and validates post-quantum products as expected. Indeed, Routinator is able to validate the ROAs, and by inspecting the publication points the precise format and sizes of objects can be inspected.

7.2.2 Single CA rollover

Next, we want to see an actual migration from RSA to post-quantum signatures. For this, we create a similar environment as in the steady state test, but now initially using RSA signatures only. Then, we perform a single CA algorithm rollover at the *child* CA, as described in section 6.2.3, using the `krillc rollover` commands that normally trigger a [RFC6489] rollover. Figure 7.1b shows the RPKI tree before and after the rollover in this test.

While this test migrates only one CA, it demonstrates every combination of mixed certificates. Initially, there is an all-RSA tree, but after migrating, we have:

- The *online* CA using RSA under an RSA trust anchor.
- The *child* CA using Falcon, under an RSA issuer (an A issuer signs a B subordinate).
- The *grandchild* CA still using RSA, under a Falcon issuer (a B issuer signs an A subordinate).

Each CA publishes objects, and we take snapshots of both the publication points and Routinator's validation result at every step. This way, we can see that, as expected, the *grandchild* products remain unchanged and valid throughout the process: the migration is indeed a strictly local operation for the *child* and *online* CAs.

This test confirms that the single CA rollover in our mixed-tree migration is convenient for CA operators. They can simply update configuration (or the defaults could be changed by the software maintainers over time), after which the rollover can be performed with the already existing key rollover commands. Once RPs support the new algorithm, CA operators can simply try to do a rollover whenever it suits them, without needing to worry about global coordination or creating a synchronized parallel tree for [RFC6916].

Do note that in this test, the [RFC8183] BPKI trust anchors for communication between the *online* and *child* CAs are not updated to Falcon. In section 6.4.2 we found that current software poorly supports replacing BPKI trust anchors in place. This is an important topic for future work. On the other hand, our steady state test does establish post-quantum BPKI TAs from the start: the difficulty is in replacing existing TAs without breaking communication channels. Establishing these channels from scratch is easy.

7.2.3 Conclusions

Our proof-of-concept demonstrates that integrating post-quantum signatures into the RPKI ecosystem via mixed certificates is not only feasible but requires minimal changes to existing software. The modifications to support Falcon-512 in the shared `rpki` crate were straightforward, with no changes needed in Routinator itself. Unilateral algorithm rollovers by individual CAs demonstrably work, with existing [RFC6489] key rollover commands. This supports our assertion that the mixed-tree migration approach is practical.

While our proof of concept implements Falcon-512 directly rather than in a hybrid scheme, it establishes that the underlying approach — regardless of the choice for a specific algorithm — is sound. The extension to hybrid signatures and other validator implementations would be a natural next step towards real-world deployment, along with addressing the challenge of updating BPKI trust anchors in place, and employing our null scheme from chapter 5 in one-time-use certificates.

Chapter 8

Discussion and conclusions

This thesis lays the groundwork for the transition to post-quantum cryptography in the RPKI. We have explored the implications of quantum computing for the RPKI, evaluated which post-quantum cryptographic solutions are suitable, and explored a practical migration strategy.

As this is the first work on PQC for the RPKI, many areas have not been fully explored. Instead, we deliberately cover a wide range of relevant topics, yielding many suggestions for future work, rather than exploring a narrow aspect in detail. This enables a comprehensive understanding of the steps needed to make the RPKI quantum-safe.

In this chapter, we review our findings from the previous chapters and the recommendations we make based on them. We outline the limitations of our work, that directly lead to many avenues for future research. This discussion is organized into four main sections: first reviewing our evaluation of the quantum threat (8.1), then the selection of post-quantum algorithms, including our null scheme (8.2), the approach to performing a migration (8.3), and the implementation we made (8.4). We discuss the further research, standardization, and implementation efforts that stem from each topic in their own sections. Finally, we summarize our contributions in 8.5.

8.1 Evaluating the quantum threat

As established in chapter 3, the advent of quantum computing poses a severe threat to the RPKI. Our analysis shows how an attacker with quantum capabilities can not only undermine the protections offered by the RPKI, but actually exploit reliance on the RPKI beyond simply bypassing it. In the presence of a quantum attacker, the RPKI enables novel attacks on BGP that are not possible without the RPKI. An attacker could use these to disrupt routing on a global scale, or perform effective targeted attacks. Consequently, a quantum-vulnerable RPKI is unacceptable as soon as the quantum threat becomes credible.

Finding 1 *Using the RPKI with broken cryptography is more dangerous than using BGP with no RPKI at all.*

This first finding may come as a surprise. Intuitively, and for many other applications, the use of weak cryptography would be preferable over no cryptography at all. For example, for internet browsing, privacy is better preserved using TLS with quantum-vulnerable encryption, than using no encryption at all. For the RPKI, however, this is not the case. This highlights the importance of preparing the RPKI *before* it becomes vulnerable, rather than viewing the transition as an optional 'upgrade' that can be postponed.

While the finding sounds concerning, its impact is limited in practice. If a quantum attack on the RPKI were to become feasible or is demonstrated, operators would simply disable the RPKI, falling back to plain BGP to avoid more severe consequences.

Finding 2 *Communication between CAs is an attractive target for a quantum-enabled attacker.*

Our analysis in section 3.3.1 reveals that the communication channel between parent and child CAs is likely the most attractive target for a quantum-enabled attacker. This communication currently relies on the same RSA-based cryptography as the RPKI certificates themselves. It presents an attractive target, as only a single forgery is needed to compromise a CA's internet resources. Other avenues require the attacker to not only create forgeries, but also to inject them into the RPKI somehow. We've shown the latter to be realistically feasible, but it takes more effort and could be less effective than compromising the parent-child channel.

This is a useful insight: the [RFC6492] channel is a component whose migration can be prioritized. Not only is it an easy attack vector, but as shown in section 6.4.2, it also turns out to be a relatively easy component to migrate, being invisible to relying parties. So, its migration can and should be prioritized: while a single update to the specification of algorithms to be used can cover changes to both [RFC8183] and the RPKI itself, *deployment* of the new algorithm should start with [RFC8183] TAs.

Finally, we identified that there is a reliance on several other underlying technologies, such as TLS and DNSSEC, that secure (1) web interfaces of hosted CA services and for performing [RFC8183] setup, (2) RRDP and `rsync` downloads by RPs, and (3) some transport layers that can be chosen for the transfer of validated data from validators to BGP routers.

8.1.1 Future work

Further research is needed on two of these topics. Both are essential to make the RPKI quantum-safe: there is no point in securing the RPKI itself when related protocols form an Achilles heel.

- It turned out (in section 6.4.2) that, while it is theoretically possible to manually update BPKI TAs (e.g. after redoing a [RFC8183] exchange), this is poorly supported in practice. To address this, in-band support for updating a BPKI TA should be developed. [11, 10] and [46] are a good starting point for this.
- Some underlying protocols, such as TLS and the web PKI need to become quantum-resistant. There is already ongoing work on this, but, protection of confidentiality is (understandably) prioritized over authentication. The RPKI, however, relies mostly on authentication and integrity.

  Future research should inventorize precisely which underlying protocols are used in the RPKI, and what security properties they need to provide. Then, implementers can ensure that the necessary measures are taken to use the protocols in a quantum-safe manner. The transport layers in the RTR protocol (section 3.3.5) are an example where several underlying protocols can be chosen. Future work could identify any choices that are quantum-safe by definition or can be used in a quantum-safe manner; operators and implementers can then choose a safe option.

8.2 Algorithm selection

Chapter 4 provides an evaluation of which post-quantum signature algorithms to use in the RPKI. In 4.2, we develop a methodology to estimate the performance impact of a particular post-quantum candidate on the downloading and verification performance of an RPKI validator.

Using this methodology, various candidates can systematically be compared with each other, and with the current RSA-based RPKI. An important limitation is that the

estimated durations should only be interpreted as relative numbers, useful for mutual comparison — not as absolute numbers. They are based on:

- (rough) estimates of the duration of a *full* download over RRDP,
- full adoption of a single post-quantum scheme, and
- no changes to the content of the RPKI.

As such, the numbers are not representative of costs one might expect in the real world, where partial downloads are commonplace, and where large regional differences exist between RPs, and between repositories with and without CDNs [65].

The performance benchmarks of post-quantum signature implementations are not precise either, and are subject to change as implementations mature and become optimized.

Despite these limitations, the *relative* performance differences identified in our study provide a sound basis for algorithm selection. Using our methodology, we find that:

Finding 3 *A hybrid with Falcon appears to be a good replacement for RSA in the RPKI.*

There are several alternatives (MAYO, HAWK, or even ML-DSA) that could also be considered. We found no compelling reason to prefer any specific traditional component for use in a hybrid. The trade-off between size and verification speed for EdDSA (or ECDSA) has no clear winner due to the expected differences in bandwidth of RPs, and possibilities to optimize validation through caching.

Next, we found several reasons to introduce not one, but multiple post-quantum signature algorithms in the RPKI:

Finding 4 *Other signature schemes can additionally be introduced as fallback and for specialized use cases.*

First, introducing a fallback algorithm based on a different hardness assumption than the main algorithm provides a valuable safety net. With the lattice-based Falcon as primary candidate, a fallback could, for instance, be the multivariate MAYO. If lattice-based cryptography is unexpectedly broken, relying parties would already be prepared to accept products signed with the fallback algorithm, enabling certificate authorities to quickly switch to the fallback.

Second, we identified specialized use cases within the RPKI where different algorithms could be beneficial. For hard-to-update components such as Trust Anchor Locators and BPKI trust anchors, that are not performance-critical, a strong algorithm like Falcon-1024 or even SLH-DSA could be justified.

Both of these ideas do have the downside of increasing complexity, requiring more algorithms to be implemented in software, and to be available in the HSMs used by CAs. Additionally, standardization of Falcon-512 should be coming relatively soon, while alternatives based on different hardness assumptions are in a much earlier stage. This means that introducing a fallback algorithm could delay the transition to PQC. The community will need to weigh the benefits and drawbacks of these options.

Chapter 5 introduced another instance of a specialized algorithm:

Finding 5 *Our 'null scheme' can compensate for much of the performance cost of post-quantum signatures.*

The null scheme is a novel approach that addresses the overhead introduced by the current use of one-time-use EE certificates in RPKI signed objects. We propose to leverage the unique requirements of these certificates to eliminate their overhead. Not only is the private key used only once, but the message to be signed can also be known *before* creating the key pair. These requirements are even weaker than those for normal one-time signatures (like [41]), allowing us to replace the one-time public key and signature with a single message digest. This maintains the exact same security as the current approach, while dramatically reducing both size and computational overhead. The null scheme can be introduced as if it were a normal signature algorithm (using an algorithm rollover) without requiring any changes to the structure of signed objects.

The practical benefits are substantial: the performance savings are already useful in the RSA-based RPKI, but as signature sizes and verification times increase with a migration to post-quantum signatures, the null scheme's benefits become greater. When the introduction of post-quantum signatures is combined with the null scheme, the performance impact of post-quantum signatures can be nearly eliminated by the reduction from the null scheme. The precise savings depend on the signature algorithm that is used.

The null scheme is beneficial by itself, but we highly recommend adopting the null scheme simultaneously with the introduction of post-quantum algorithms. This minimizes the performance impact of post-quantum signatures, and allows the two algorithm introductions to be rolled out together, rather than performing two separate algorithm rollovers in quick succession.

8.2.1 Future work

Several areas for future research could address the limitations of our current methodology of measuring and predicting the performance impact of signature schemes in the RPKI. These include:

- Predicting the performance impact on `rsync` transfers and RRDP delta updates (rather than full RRDP downloads), which constitute most of the routine downloads in practice.
- Establishing a more accurate method for predicting the downloading bandwidth, taking into account differences between repositories and the locations of RPs.

  For instance, our measurements from section 4.2.1.3 could be repeated from multiple vantage points, and could possibly be combined with statistics on RPs to get an idea on the distribution of RPs' bandwidths.
- Benchmarking the signature verification performance impact more precisely, for instance by measuring validation time using a prototype implementation on a fake post-quantum RPKI snapshot that is structurally equivalent to a snapshot of the real RPKI.

  Our implementations in Routinator and Krill could be used for this, but doing the same with different relying party implementations is also valuable.
- Measuring the number of files and bytes that can be saved by performing (more) ROA aggregation, as discussed in section 4.4.1.1. This could help offset the increased overhead per object from post-quantum signatures.
- Exploring the implications of adopting a post-quantum scheme with a high NIST target level (5), such as needing to use larger hash sizes in some places.

These topics can help estimate more accurately what performance impact a migration to post-quantum signatures will have, and guide the final choice of algorithm, including a traditional component for hybrid signatures.

Additionally, further research could be done to inform the decision on introducing additional signature schemes for the BPKI, TALs, or as fallback. This can include investigating the performance impact in these specific use cases and looking out for developments in the standardization process of additional post-quantum signatures.

8.3 Migration strategy

In chapter 6, we analyzed the transition process to post-quantum cryptography in the RPKI. We showed that the standardized algorithm agility procedure in [RFC6916] is operationally impractical due to its requirement for global coordination, maintenance of parallel RPKI trees, and top-down migration order. Instead, we proposed a mixed-tree migration approach, closely based on Dickson's original proposal [19, 20], that allows for a more flexible and practical transition.

Finding 6 *The migration strategy from [RFC6916] is impractical; a 'mixed-tree' migration should be used instead.*

Our mixed-tree approach fundamentally differs from [RFC6916]. It allows the introduction of multiple signature algorithms that can be used interchangeably, and does not impose a globally coordinated timeline. A very important requirement for this migration is that relying parties must be updated to support the new algorithms before CAs start migrating: in contrast to [RFC6916], the mixed-tree strategy does not maintain an RPKI tree with the old algorithm for backward compatibility.

Individual CAs can migrate when they want, using the familiar [RFC6489] key rollover procedure. We have proven in section 7.2.2 that this works in practice. Summarized very briefly (more detail is in section 6.4), we suggest that the following steps should be taken.

1. **Publish future TALs**: RIRs should create and publish TALs for new post-quantum trust anchors early, allowing them to be included in RP software updates, long before they are used in practice.
2. **Update RP software**: Roll out validator software supporting post-quantum algorithms (and the null scheme from chapter 5) as soon as possible, also long before CAs start migrating. This gives plenty of time for RPs to update.
3. **Migrate BPKI trust anchors**: As soon as possible, migrate [RFC8183] trust anchors. These TAs for [RFC6492] and [RFC8181] communication do not need RPs to be updated first.
4. **Set up monitoring experiments**: Use the measurement techniques in section 6.2.2 and appendix B to track RP readiness for the new algorithms.

Work on these steps can start as soon as, and partially even before, the necessary changes to the RPKI specifications are finalized. When these steps have been performed, it is time to wait for RPs to update. Only after a long time, when enough RPs are ready and/or the quantum threat becomes credible, can CAs start actually migrating. The essence of the strategy is this:

Finding 7 *Updated RP software and TALs should be rolled out as soon as possible; actually migrating CAs' certificates can wait.*

Thus, it is sensible to initiate discussion in the SIDROPS working group very soon. The earlier plans are made and implemented, the longer time there is for RPs to update, which makes the migration less risky.

8.3.1 Future work

The most important next step towards being able to actually perform a migration is to reach consensus on it. This requires discussion in the community, which can be initiated constructively by writing down an initial draft that describes a mixed-tree migration (perhaps with a selection of algorithms to introduce, including our null scheme) in detail. This forms a clear starting point for discussion, that can be iterated on until a consensus is reached.

For the smaller step of enabling BPKI TA replacement, this approach of writing a draft to initiate discussion was already employed by Tim Bruijnzeels [10, 11]. Resuming the conversation on that topic is another good next step.

8.4 Implementation and testing

In chapter 7, we demonstrated the feasibility of our mixed-tree migration approach through a proof-of-concept implementation in Routinator and Krill. Our implementation successfully showed that post-quantum signatures can be integrated into existing RPKI software with minimal changes, and that individual CAs can perform algorithm rollovers using the proven [RFC6489] key rollover procedure.

Our implementation was limited in scope, serving primarily to validate the core migration concept. While this may not seem like only a small contribution, showing working code is considered to be an important prerequisite for the adoption of new ideas in the RPKI community.

8.4.1 Future work

Our implementation also provides a starting point for further development and experimentation.

Hybrid signatures Our proof-of-concept used Falcon-512 directly rather than the hybrid schemes we recommend in chapter 4. Hybrid signatures should be implemented, either through oqsprovider or natively through OpenSSL once available. Using hybrid signatures is unlikely to behave much differently from using a single post-quantum signature algorithm, but clearly, an implementation should be made that uses whichever (hybrid) algorithm will actually be chosen.

Null scheme The null scheme proposed in chapter 5 should be implemented and tested, to verify the theoretical performance benefit we predicted. In Krill and Routinator, this should be a straightforward change, but it did not fit our timeline.

Aborting key rollovers Similarly, the possibility for aborting an [RFC6489] key rollover, as discussed in section 6.2.3.1, can be implemented and tried out.

Interoperability Perhaps the most important reason to experiment with post-quantum signatures, the null scheme, and the migration process, is to ensure interoperability between different implementations. Our implementation was in a validator and a CA, that share most of their code in `rpki-rs`.

To do actual interoperability testing, at least one additional implementation is needed, for example in `rpki-client` and ideally also in other CA software.

Performance benchmarking Using our implementation, as well as any others in other software, performance testing should be conducted.

This includes both performing measurements of signature verification cost, as already suggested in section 8.2.1, and verifying the benefits of the null scheme.

BPKI migration mechanisms As stated in section 6.4.2, current software poorly supports updating BPKI trust anchors in place. While future work should also introduce an in-band mechanism for updating BPKI TAs, there is also already the possibility to improve support for out-of-band, manual updates. Software like Krill should be able to maintain multiple ID certificates for itself, such that it can use a new one towards recipients that are aware of it, while keeping the old one for recipients that are have not yet replaced the TA. This can at least enable gradual manual key rollovers, instead of requiring all subordinates of a CA to update their known trust anchors at the same time. Additionally, it should be possible to configure that post-quantum algorithms should be used for the BPKI, while keeping RSA for the RPKI itself.

Real-world experiments Finally, it would be interesting to (already) set up an experimental CA implementation in the real RPKI, as a leaf CA. This can enable interoperability testing with validators and other CAs, and at the same time be used as part of the monitoring experiments from section 6.2.2 and appendix B.

Some of these steps can serve as excellent evidence in support of drafts that precisely specify both algorithm selection and migration strategy.

8.5 Conclusions

This thesis presents the first work on post-quantum cryptography for the RPKI, establishing the foundation for making this critical internet infrastructure quantum-safe. Our research provides a broad view of the challenges and solutions needed to protect the RPKI against the emerging quantum threat.

We have demonstrated that the RPKI enables severe attacks in the presence of quantum attackers, that make the RPKI dangerous to use when it is realistically vulnerable. Upgrading the RPKI to use post-quantum cryptography — *before* these attacks become feasible — is therefore essential to ensure routing security in the future.

Next, we have presented a methodology for comparing the performance impact that can be expected from different post-quantum signature schemes in the RPKI. Using this methodology, a hybrid approach with Falcon-512 and a traditional component emerges as a good candidate to replace RSA in the RPKI. Other candidates are also viable, and in general, the performance impact of post-quantum signatures appears manageable.

In particular, performance impact of post-quantum signatures can be limited by introducing optimizations such as (1) increased aggregation of ROAs, (2) caching the result of signature verifications, and (3) adopting our null scheme in RPKI signed objects.

The null scheme represents a novel construction that can significantly offset the performance overhead of post-quantum signatures by eliminating the cryptographic redundancy in one-time-use EE certificates, without affecting security at all. This scheme is a valuable contribution independently, but is particularly useful when combined with the migration to post-quantum signatures, sharing a single migration process, and offering performance benefits that become even more pronounced with larger post-quantum signatures.

We have also explored potential benefits of introducing multiple post-quantum algorithms rather than a single replacement for RSA. This includes the null scheme, but possibly also a post-quantum fallback algorithm to provide resilience against cryptographic breakthroughs, and using stronger algorithms for specialized components like the BPKI and TALs that are hard to update but not performance-critical. While these

ideas would increase implementation complexity, they can offer valuable flexibility and security benefits.

Finally, we have shown that the migration strategy from [RFC6916] is operationally impractical, and proposed instead a mixed-tree migration approach that allows for flexible, individual CA transitions using the proven key rollover procedure. In the proposed strategy, RP updates and TALs are distributed as soon as possible, while actual CA migrations can be delayed without problem.

Our proof-of-concept implementation demonstrates the feasibility of this approach. We publish this implementation to provide a starting point for further research, including performance measurements and interoperability testing.

The findings and recommendations presented in this thesis provide the RPKI community with the necessary groundwork to begin planning and implementing a transition to post-quantum cryptography. This process can start with the creation of early drafts that describe a possible selection of post-quantum algorithms and the migration steps to get there, which can then be discussed by the community.

Bibliography

[1] Emile Aben, Ties de Kock, and Agustín Formoso. "RPKI-flutter". In: *RIPE 89*. Recording: https://ripe89.ripe.net/archives/video/1453/. Oct. 2024. URL: https://ripe89.ripe.net/wp-content/uploads/presentations/76-rpki-flutter-ripe89-emileaben.pdf.

[2] APNIC. *Single Trust Anchor transition (completed 2018)*. URL: https://www.apnic.net/community/security/resource-certification/apnic-rpki-trust-anchor-locator/ (visited on 06/02/2025).

[3] Alexander Azimov et al. *BGP AS_PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects*. Internet-Draft draft-ietf-sidrops-aspa-verification-20. Work in Progress. Internet Engineering Task Force, Jan. 2025. 21 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-sidrops-aspa-verification/20/.

[4] D.J. Bernstein, J. Buchmann, and E. Dahmen, eds. *Post-quantum cryptography*. Germany: Springer, 2009. ISBN: 978-3-540-88701-0. DOI: 10.1007/978-3-540-88702-7.

[5] Daniel J. Bernstein et al. "The SPHINCS+ Signature Framework". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2129–2146. ISBN: 9781450367479. DOI: 10.1145/3319535.3363229. URL: https://doi.org/10.1145/3319535.3363229.

[6] Nina Bindel and Britta Hale. *A Note on Hybrid Signature Schemes*. Cryptology ePrint Archive, Paper 2023/423. 2023. URL: https://eprint.iacr.org/2023/423.

[7] Nina Bindel et al. "Transitioning to a Quantum-Resistant Public Key Infrastructure". In: *Post-Quantum Cryptography*. Ed. by Tanja Lange and Tsuyoshi Takagi. Cham: Springer International Publishing, 2017, pp. 384–405. ISBN: 978-3-319-59879-6.

[8] Joppe W. Bos et al. *HAWK*. version 1.1. Feb. 2025.

[9] Tim Bruijnzeels. Private communications. Apr. 2025.

[10] Tim Bruijnzeels. "Challenges and Lessons Learned in deploying RFC6492, 8181 and 8183". In: Recording: https://www.youtube.com/watch?v=lHV8OaY5cj0. IETF 115. London, United Kingdom, Nov. 2022. URL: https://datatracker.ietf.org/meeting/115/materials/slides-115-sidrops-challenges-and-lessons-learned-in-deploying-rfc6492-8181-and-8183-00.

[11] Tim Bruijnzeels. *RPKI Publication Protocol Version 2*. Internet-Draft draft-timbru-sidrops-rpki-publication-v2-00. Work in Progress. Internet Engineering Task Force, Oct. 2022. 23 pp. URL: https://datatracker.ietf.org/doc/draft-timbru-sidrops-rpki-publication-v2/00/.

[12] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. "XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions". In: *Post-Quantum Cryptography*. Ed. by Bo-Yin Yang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 117–129. ISBN: 978-3-642-25405-5.

[13] Johannes Buchmann et al. "Post-quantum signatures". In: *Cryptology ePrint Archive* (2004).

[14] Taejoong Chung et al. "RPKI is Coming of Age: A Longitudinal Study of RPKI Deployment and Invalid Route Origins". In: *Proceedings of the Internet Measurement Conference*. IMC '19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 406–419. ISBN: 9781450369480. DOI: 10.1145/3355369.3355596. URL: https://doi.org/10.1145/3355369.3355596.

[15] Cloudflare. *Is BGP safe yet?* URL: https://isbgpsafeyet.com/.

[16] Deirdre Connolly. *ML-KEM Post-Quantum Key Agreement for TLS 1.3*. Internet-Draft draft-ietf-tls-mlkem-00. Work in Progress. Internet Engineering Task Force, Apr. 2025. 11 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-mlkem/00/.

[17] *CVE-2024-12087*. 2024. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-12087.

[18] *CVE-2024-12088*. 2024. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-12088.

[19] Brian Dickson. *Re: [sidr] WGLC for draft-ietf-sidr-algorithm-agility-03*. IETF Mail Archive. Nov. 2011. URL: https://mailarchive.ietf.org/arch/msg/sidr/pglp5kVhih_Pf8wCmbDYolYIW44/.

[20] Brian Dickson. *Re: [sidr] WGLC for draft-ietf-sidr-algorithm-agility-03*. IETF Mail Archive. Nov. 2011. URL: https://mailarchive.ietf.org/arch/msg/sidr/AYN-U2X6_n36mtLfsDsROcADP2M/.

[21] Brian Dickson. *Re: [sidr] WGLC for draft-ietf-sidr-algorithm-agility-03*. IETF Mail Archive. Nov. 2011. URL: https://mailarchive.ietf.org/arch/msg/sidr/SyUIqpJ3BrOM8znvnJWQSkxzHz8/.

[22] Brian Dickson. *Re: [sidr] WGLC for draft-ietf-sidr-algorithm-agility-03*. IETF Mail Archive. Nov. 2011. URL: https://mailarchive.ietf.org/arch/msg/sidr/Is7oElFs4qeiuVUok_JPy3UI8fM/.

[23] Mark Dranse and David Murray. "BGP Route Origin Validation". In: *APNIC 30*. Presentation of blog post at https://labs.ripe.net/author/david_murray/bgp-route-origin-validation/. Gold Coast, Australia, Aug. 2010. URL: https://conference.apnic.net/30/pdf/bgp-route-validation-lightning.pdf.

[24] Léo Ducas et al. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (Feb. 2018), pp. 238–268. DOI: 10.13154/tches.v2018.i1.238-268. URL: https://tches.iacr.org/index.php/TCHES/article/view/839.

[25] Romain Fontugne et al. "RPKI Time-of-Flight: Tracking Delays in the Management, Control, and Data Planes". In: *Passive and Active Measurement*. Ed. by Anna Brunstrom, Marcel Flores, and Marco Fiore. Cham: Springer Nature Switzerland, 2023, pp. 429–457. ISBN: 978-3-031-28486-1.

[26] Pierre-Alain Fouque et al. *Falcon: Fast-Fourier lattice-based compact signatures over NTRU*. Specification v1.2. Oct. 2020.

[27] Andrew Fregly et al. *Stateless Hash-Based Signatures in Merkle Tree Ladder Mode (SLH-DSA-MTL) for DNSSEC*. Internet-Draft draft-fregly-dnsop-slh-dsa-mtl-dnssec-04. Work in Progress. Internet Engineering Task Force, Apr. 2025. 36 pp. URL: https://datatracker.ietf.org/doc/draft-fregly-dnsop-slh-dsa-mtl-dnssec/04/.

[28] Roque Gagliano, Stephen Kent, and Sean Turner. *Algorithm Agility Procedure for the Resource Public Key Infrastructure (RPKI)*. Internet-Draft draft-ietf-sidr-algorithm-agility-03. Work in Progress. Internet Engineering Task Force, Aug. 2011. URL: https://datatracker.ietf.org/doc/draft-ietf-sidr-algorithm-agility/04/.

[29] Yossi Gilad et al. *Are We There Yet? On RPKI's Deployment and Security*. Cryptology ePrint Archive, Paper 2016/1010. 2016. URL: https://eprint.iacr.org/2016/1010.

[30] Tomas Hlavacek et al. “Behind the Scenes of RPKI”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 1413–1426. ISBN: 9781450394505. DOI: 10.1145/3548606.3560645. URL: https://doi.org/10.1145/3548606.3560645.

[31] Tomas Hlavacek et al. “Practical Experience: Methodologies for Measuring Route Origin Validation”. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2018, pp. 634–641. DOI: 10.1109/DSN.2018.00070.

[32] Koen van Hove, Jeroen van der Ham-de Vos, and Roland van Rijswijk-Deij. “rpkiller: Threat Analysis of the BGP Resource Public Key Infrastructure”. In: *Digital Threats* 4.4 (Oct. 2023). DOI: 10.1145/3617182. URL: https://doi.org/10.1145/3617182.

[33] Geoff Huston. *How we measure: DNSSEC validation*. APNIC Blog. Oct. 2023. URL: https://blog.apnic.net/2023/10/31/how-we-measure-dnssec-validation/.

[34] Geoff Huston. *How we measure: RPKI ROA signing and Route Origination Validation*. APNIC Blog. Nov. 2023. URL: https://blog.apnic.net/2023/11/09/how-we-measure-rpki-roa-signing-and-route-origination-validation/.

[35] Steve Kent. “Algorithm Agility Procedure for RPKI: WGLC Issues & Fixes”. In: *IETF 82*. Taipei, Taiwan, Nov. 2011. URL: https://datatracker.ietf.org/meeting/82/materials/slides-82-sidr-5.

[36] Steve Kent, Roque Gagliano, and Sean Turner. “Algorithm Transition for the RPKI”. In: *IETF 79*. Beijing, China, Nov. 2010. URL: https://datatracker.ietf.org/meeting/79/materials/slides-79-sidr-4.

[37] Steve Kent and Kotikalapudi Sriram. “RPKI rsync Download Delay Modeling”. In: *IETF 86*. Orlando, Florida, USA, Mar. 2013. URL: https://datatracker.ietf.org/meeting/86/materials/slides-86-sidr-1.

[38] John Kristoff et al. “On Measuring RPKI Relying Parties”. In: *Proceedings of the ACM Internet Measurement Conference*. IMC ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 484–491. ISBN: 9781450381383. DOI: 10.1145/3419394.3423622. URL: https://doi.org/10.1145/3419394.3423622.

[39] NLnet Labs. Private communications. Mar. 2025.

[40] NLnet Labs. *NLnet Labs RPKI statistics*. URL: https://rov-measurements.nlnetlabs.net/stats/ (visited on 03/21/2025).

[41] Leslie Lamport. *Constructing Digital Signatures from a One Way Function*. Tech. rep. CSL-98. Oct. 1979. URL: https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/.

[42] Weitong Li et al. “RoVista: Measuring and Analyzing the Route Origin Validation (ROV) in RPKI”. In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. IMC ’23. Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 73–88. ISBN: 9798400703829. DOI: 10.1145/3618257.3624806. URL: https://doi.org/10.1145/3618257.3624806.

[43] Alexey Melnikov. *[sidr] draft-ietf-sidr-algorithm-agility submitted to AD for publication*. IETF Mail Archive. Nov. 2012. URL: https://mailarchive.ietf.org/arch/msg/sidr/i1tFaczvglAejudx3qwYBd3BoDs/.

[44] Alexey Melnikov. *Shepherd writeup for draft-ietf-sidr-algorithm-agility*. Nov. 2012. URL: https://datatracker.ietf.org/doc/draft-ietf-sidr-algorithm-agility/shepherdwriteup/.

[45] Donika Mirdita, Haya Schulmann, and Michael Waidner. *SoK: An Introspective Analysis of RPKI Security*. 2024. arXiv: 2408.12359 [cs.CR]. URL: https://arxiv.org/abs/2408.12359.

[46] Q Misell. *A BPKI key rollover protocol for the Resource Public Key Infrastructure (RPKI)*. Internet-Draft draft-misell-rpki-bpki-key-rollover-00. Work in Progress. Internet Engineering Task Force, July 2024. 6 pp. URL: https://datatracker.ietf.org/doc/draft-misell-rpki-bpki-key-rollover/00/.

[47] Dustin Moody et al. *Transition to post-quantum cryptography standards*. Tech. rep. National Institute of Standards and Technology, 2024. DOI: https://doi.org/10.6028/NIST.IR.8547.ipd.

[48] Michele Mosca and Marco Piani. *Quantum Threat Timeline Report 2023*. Tech. rep. Global Risk Institute, Dec. 2023. URL: https://globalriskinstitute.org/publication/2023-quantum-threat-timeline-report/.

[49] Moritz Müller et al. "Retrofitting post-quantum cryptography in internet protocols: a case study of DNSSEC". In: *SIGCOMM Comput. Commun. Rev.* 50.4 (Oct. 2020), pp. 49–57. ISSN: 0146-4833. DOI: 10.1145/3431832.3431838. URL: https://doi.org/10.1145/3431832.3431838.

[50] National Institute of Standards and Technology. *NIST RPKI Monitor*. URL: https://rpki-monitor.antd.nist.gov/ (visited on 06/02/2025).

[51] RIPE NCC. *RIPE Atlas*. URL: https://atlas.ripe.net/.

[52] RIPE NCC. *Routing Information Service*. URL: https://ris.ripe.net/.

[53] RIPE NCC. *YouTube Hijacking: A RIPE NCC RIS ase study*. URL: https://www.ripe.net/about-us/news/youtube-hijacking-a-ripe-ncc-ris-case-study/ (visited on 05/15/2025).

[54] University of Oregon. *RouteViews*. URL: https://www.routeviews.org/.

[55] Mike Ounsworth et al. *Composite ML-DSA for use in X.509 Public Key Infrastructure and CMS*. Internet-Draft draft-ietf-lamps-pq-composite-sigs-04. Work in Progress. Internet Engineering Task Force, Mar. 2025. 82 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/04/.

[56] Thomas Pornin. *New Efficient, Constant-Time Implementations of Falcon*. 2019.

[57] Jon Postel. *NCP/TCP transition plan*. RFC 801. Nov. 1981. DOI: 10.17487/RFC0801. URL: https://www.rfc-editor.org/info/rfc801.

[58] Mikhail Puzanov. "Low Latency RPKI Validation". In: *RIPE 88*. Recording: https://ripe88.ripe.net/archives/video/1370/. May 2024. URL: https://ripe88.ripe.net/wp-content/uploads/presentations/11-Low-latency-RPKI-validation.pdf.

[59] Lars Ran. *Wedges, oil, and vinegar – An analysis of UOV in characteristic 2*. Cryptology ePrint Archive, Paper 2025/1143. Presented at Eurocrypt 2025. 2025. URL: https://eprint.iacr.org/2025/1143.

[60] Andreas Reuter et al. "Towards a Rigorous Methodology for Measuring Adoption of RPKI Route Validation and Filtering". In: *SIGCOMM Comput. Commun. Rev.* 48.1 (Apr. 2018), pp. 19–27. ISSN: 0146-4833. DOI: 10.1145/3211852.3211856. URL: https://doi.org/10.1145/3211852.3211856.

[61] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[62] Austein Rob. *Re: [sidr] key rollover and algorithm migration*. IETF Mail Archive. June 2010. URL: https://mailarchive.ietf.org/arch/msg/sidr/7aaXNzQm5rD7rd5ucsaE-voK51U/.

[63] Nils Rodday et al. “Revisiting rpki route origin validation on the data plane”. In: *Proc. of Network Traffic Measurement and Analysis Conference (TMA), IFIP*. 2021.

[64] Nils Rodday et al. “The Resource Public Key Infrastructure (RPKI): A Survey on Measurements and Future Prospects”. In: *IEEE Transactions on Network and Service Management* 21.2 (2024), pp. 2353–2373. DOI: 10.1109/TNSM.2023.3327455.

[65] Khwaja Zubair Sediqi et al. “Syncing with RPKI: Exploring Causes of Delay in Relying Party Synchronization”. In: *RIPE 88*. Recording: https://ripe88.ripe.net/archives/video/1384/. May 2024. URL: https://ripe88.ripe.net/presentations/121-rpki_synchronization_delay_Zubair.pdf.

[66] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[67] Aftab Siddiqui. *What Happened? The Amazon Route 53 BGP Hijack to Take Over Ethereum Cryptocurrency Wallets*. Mutually Agreed Norms for Routing Security (MANRS). URL: https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/ (visited on 05/15/2025).

[68] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. “Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH”. In: *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’20. Barcelona, Spain: Association for Computing Machinery, 2020, pp. 149–156. ISBN: 9781450379489. DOI: 10.1145/3386367.3431305. URL: https://doi.org/10.1145/3386367.3431305.

[69] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. “Post-Quantum Authentication in TLS 1.3: A Performance Study”. In: *Network and Distributed Systems Security (NDSS) Symposium 2020*. San Diego, CA, USA, Feb. 2020. DOI: 10.14722/ndss.2020.24203. URL: https://www.ndss-symposium.org/ndss-paper/post-quantum-authentication-in-tls-1-3-a-performance-study/.

[70] Job Snijders. Private communications. Apr. 2025.

[71] Job Snijders. *[Sidrops] Signed Object signed with Ed25519 (RFC 8419 proof-of-concept)*. IETF Mail Archive. Sept. 2023. URL: https://mailarchive.ietf.org/arch/msg/sidrops/CG2BWxOa6Ly8F0huOULIBd4hGEc/.

[72] Job Snijders. *policy proposal: ”Automatic Revocation of Persistently Non-functional Delegated RPKI CAs”*. RIPE Routing Working Group Mail Archive. Feb. 2025. URL: https://mailman.ripe.net/archives/list/routing-wg@ripe.net/thread/USQUMNOE3L3UUD3JZVI6LH7VMDRPL7K4/.

[73] Job Snijders. *Re: ARIN RPKI Trust Anchor Issue*. NANOG mailing list. Jan. 2025. URL: https://lists.nanog.org/archives/list/nanog@lists.nanog.org/message/OSAQ7LZJZHILPGJD4VNXVGQ3U5OIGIN5/.

[74] Job Snijders. *RPKI’s 2024 Year in Review*. RIPE Labs. Jan. 2025. URL: https://labs.ripe.net/author/job_snijders/rpkis-2024-year-in-review/ (visited on 05/30/2025).

[75] Job Snijders and Theo Buehler. *Constraining RPKI Trust Anchors*. Internet-Draft draft-snijders-constraining-rpki-trust-anchors-08. Work in Progress. Internet Engineering Task Force, May 2025. 103 pp. URL: https://datatracker.ietf.org/doc/draft-snijders-constraining-rpki-trust-anchors/08/.

[76] Bruijnzeels Tim. *Re: [sidr] RPKI: Three questions regarding RFC 6487*. IETF Mail Archive. Jan. 2019. URL: https://mailarchive.ietf.org/arch/msg/sidr/4ycmff9jEU4VU9gGK5RyhZ7JYsQ/.

[77] Harrison Tom. *Re: [Sidrops] I-D Action: draft-ietf-sidrops-signed-tal-09.txt*. IETF Mail Archive. Apr. 2022. URL: https://mailarchive.ietf.org/arch/msg/sidrops/A8JqQUv7aT5ioj4UWtqsULnhqYM/.

[78] Thom Wiggers. *Post-Quantum signatures zoo*. PQShield. Data from https://github.com/PQShield/nist-sigs-zoo/tree/d525707/data/parametersets.csv. URL: https://pqshield.github.io/nist-sigs-zoo/ (visited on 06/02/2025).

[RFC4271] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. DOI: 10.17487/RFC4271. URL: https://www.rfc-editor.org/info/rfc4271.

[RFC5280] Sharon Boeyen et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. May 2008. DOI: 10.17487/RFC5280. URL: https://www.rfc-editor.org/info/rfc5280.

[RFC5652] Russ Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. Sept. 2009. DOI: 10.17487/RFC5652. URL: https://www.rfc-editor.org/info/rfc5652.

[RFC6480] Matt Lepinski and Stephen Kent. *An Infrastructure to Support Secure Internet Routing*. RFC 6480. Feb. 2012. DOI: 10.17487/RFC6480. URL: https://www.rfc-editor.org/info/rfc6480.

[RFC6483] Geoff Huston and George G. Michaelson. *Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs)*. RFC 6483. Feb. 2012. DOI: 10.17487/RFC6483. URL: https://www.rfc-editor.org/info/rfc6483.

[RFC6484] Derrick Kong et al. *Certificate Policy (CP) for the Resource Public Key Infrastructure (RPKI)*. RFC 6484. Feb. 2012. DOI: 10.17487/RFC6484. URL: https://www.rfc-editor.org/info/rfc6484.

[RFC6485] Geoff Huston. *The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)*. RFC 6485. Feb. 2012. DOI: 10.17487/RFC6485. URL: https://www.rfc-editor.org/info/rfc6485.

[RFC6486] Matt Lepinski et al. *Manifests for the Resource Public Key Infrastructure (RPKI)*. RFC 6486. Feb. 2012. DOI: 10.17487/RFC6486. URL: https://www.rfc-editor.org/info/rfc6486.

[RFC6487] Geoff Huston, Robert Loomans, and George G. Michaelson. *A Profile for X.509 PKIX Resource Certificates*. RFC 6487. Feb. 2012. DOI: 10.17487/RFC6487. URL: https://www.rfc-editor.org/info/rfc6487.

[RFC6488] Matt Lepinski, Andrew Chi, and Stephen Kent. *Signed Object Template for the Resource Public Key Infrastructure (RPKI)*. RFC 6488. Feb. 2012. DOI: 10.17487/RFC6488. URL: https://www.rfc-editor.org/info/rfc6488.

[RFC6489] Stephen Kent, Geoff Huston, and George G. Michaelson. *Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI)*. RFC 6489. Feb. 2012. DOI: 10.17487/RFC6489. URL: https://www.rfc-editor.org/info/rfc6489.

[RFC6490] Sam Weiler et al. *Resource Public Key Infrastructure (RPKI) Trust Anchor Locator*. RFC 6490. Feb. 2012. DOI: 10.17487/RFC6490. URL: https://www.rfc-editor.org/info/rfc6490.

[RFC6492] Byron Ellacott et al. *A Protocol for Provisioning Resource Certificates*. RFC 6492. Feb. 2012. DOI: 10.17487/RFC6492. URL: https://www.rfc-editor.org/info/rfc6492.

[RFC6810] Randy Bush and Rob Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810. Jan. 2013. DOI: 10.17487/RFC6810. URL: https://www.rfc-editor.org/info/rfc6810.

[RFC6811] Prodosh Mohapatra et al. *BGP Prefix Origin Validation*. RFC 6811. Jan. 2013. DOI: 10.17487/RFC6811. URL: https://www.rfc-editor.org/info/rfc6811.

[RFC6916] Roque Gagliano, Stephen Kent, and Sean Turner. *Algorithm Agility Procedure for the Resource Public Key Infrastructure (RPKI)*. RFC 6916. Apr. 2013. DOI: 10.17487/RFC6916. URL: https://www.rfc-editor.org/info/rfc6916.

[RFC7935] Geoff Huston and George G. Michaelson. *The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure*. RFC 7935. Aug. 2016. DOI: 10.17487/RFC7935. URL: https://www.rfc-editor.org/info/rfc7935.

[RFC8181] Sam Weiler, Anuja Sonalker, and Rob Austein. *A Publication Protocol for the Resource Public Key Infrastructure (RPKI)*. RFC 8181. July 2017. DOI: 10.17487/RFC8181. URL: https://www.rfc-editor.org/info/rfc8181.

[RFC8182] Tim Bruijnzeels et al. *The RPKI Repository Delta Protocol (RRDP)*. RFC 8182. July 2017. DOI: 10.17487/RFC8182. URL: https://www.rfc-editor.org/info/rfc8182.

[RFC8183] Rob Austein. *An Out-of-Band Setup Protocol for Resource Public Key Infrastructure (RPKI) Production Services*. RFC 8183. July 2017. DOI: 10.17487/RFC8183. URL: https://www.rfc-editor.org/info/rfc8183.

[RFC8205] Matt Lepinski and Kotikalapudi Sriram. *BGPsec Protocol Specification*. RFC 8205. Sept. 2017. DOI: 10.17487/RFC8205. URL: https://www.rfc-editor.org/info/rfc8205.

[RFC8210] Randy Bush and Rob Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*. RFC 8210. Sept. 2017. DOI: 10.17487/RFC8210. URL: https://www.rfc-editor.org/info/rfc8210.

[RFC8211] Stephen Kent and Di Ma. *Adverse Actions by a Certification Authority (CA) or Repository Manager in the Resource Public Key Infrastructure (RPKI)*. RFC 8211. Sept. 2017. DOI: 10.17487/RFC8211. URL: https://www.rfc-editor.org/info/rfc8211.

[RFC8391] Andreas Huelsing et al. *XMSS: eXtended Merkle Signature Scheme*. RFC 8391. May 2018. DOI: 10.17487/RFC8391. URL: https://www.rfc-editor.org/info/rfc8391.

[RFC8554] David McGrew, Michael Curcio, and Scott Fluhrer. *Leighton-Micali Hash-Based Signatures*. RFC 8554. Apr. 2019. DOI: 10.17487/RFC8554. URL: https://www.rfc-editor.org/info/rfc8554.

[RFC8608] Sean Turner and Oliver Borchert. *BGPsec Algorithms, Key Formats, and Signature Formats*. RFC 8608. June 2019. DOI: 10.17487/RFC8608. URL: https://www.rfc-editor.org/info/rfc8608.

[RFC8630] Geoff Huston et al. *Resource Public Key Infrastructure (RPKI) Trust Anchor Locator*. RFC 8630. Aug. 2019. DOI: 10.17487/RFC8630. URL: https://www.rfc-editor.org/info/rfc8630.

[RFC9319] Yossi Gilad et al. *The Use of maxLength in the Resource Public Key Infrastructure (RPKI)*. RFC 9319. Oct. 2022. DOI: 10.17487/RFC9319. URL: https://www.rfc-editor.org/info/rfc9319.

[RFC9455] Zhiwei Yan et al. *Avoiding Route Origin Authorizations (ROAs) Containing Multiple IP Prefixes*. RFC 9455. Aug. 2023. DOI: 10.17487/RFC9455. URL: https://www.rfc-editor.org/info/rfc9455.

[RFC9582] Job Snijders et al. *A Profile for Route Origin Authorizations (ROAs)*. RFC 9582. May 2024. DOI: 10.17487/RFC9582. URL: https://www.rfc-editor.org/info/rfc9582.

[RFC9691] Carlos M. Martínez et al. *A Profile for Resource Public Key Infrastructure (RPKI) Trust Anchor Keys (TAKs)*. RFC 9691. Dec. 2024. DOI: 10.17487/RFC9691. URL: https://www.rfc-editor.org/info/rfc9691.

Appendix A

Measuring RRDP bandwidth

In section 4.2.1.2, we used a *worst-case* estimate of the bandwidth b at which the RPKI is downloaded over RRDP. This estimate was based on the 4 minutes duration stated by [25], and assuming that t_{const} in the model $t_{download} = t_{const} + \frac{s}{b}$ is negligible. This provided useful, but very conservative numbers to evaluate post-quantum algorithms with. However, since we know t_{const} to be significant, it is useful to obtain a more realistic estimate, which is the purpose of section 4.2.1.3 and this appendix.

To better estimate b or $\frac{s}{b}$ we measure it directly, using an instrumented version of Routinator v0.14.1.

As announced in 4.2.1.3, we modify Routinator to log the total duration of every HTTPS request, and the total size of the RPKI objects (after base-64 decoding) embedded in the downloaded RRDP snapshot files. Then, we perform several measurements of downloading new snapshots, and use this to obtain a better estimate of the duration $\frac{s}{b}$: 14.5 seconds on average from a data center.

A.1 Method

We modify Routinator to log for each request the difference in time between the start of the request and the end of the response, as well as the requested URL. Additionally, while parsing RRDP snapshot files, we collect the total size of the RPKI objects in `<publish>` elements, after base-64 decoding. This corresponds to the actual file sizes of the objects, but excludes overhead due to the XML format and base-64 encoding.

After collecting these logs, we obtain the total time spent downloading snapshot files, which is $\frac{s}{b}$. The time spent downloading notification files is not part of this, as it does not change based on the key and signature sizes, and is chosen to be part of t_{const} instead. RRDP delta files are never downloaded in a fresh downloading run: they are only used to update an existing local cache. Our measuring of the RPKI sizes is not necessary to obtain $\frac{s}{b}$, but servers as a sanity check (for example including the total amount that was downloaded, which should be s), and gives insight on differences between repositories.

Our modified Routinator can be installed with:

```
cargo install \
    --git https://github.com/SIDN/pqc-routinator \
    --branch measuring-rrdp-bandwidth routinator --locked
```

Using this modified Routinator, we download a fresh copy of the RPKI 10 times. We use the `--disable-rsync` and `--rrdp-connect-timeout 30` flags to ensure that only

RRDP is used, and to avoid waiting for a long time for a few repositories that never respond.[1]

The following shell script was used to perform the measurements, and extract the relevant logs to JSON files.

```
for i in {1..10} ; do
    /usr/bin/time -f "%e, %U, %S" -a -o update_time.csv \
        routinator \
            -v --logfile "timing_${i}.log" \
            --fresh --disable-rsync --rrdp-connect-timeout 30 update;
    # Extract content sizes of RPKI objects in snapshots.
    cat timing_${i}.log \
        | grep "size: " \
        | jq -Rs '
            split("\n[") | [
                .[] | split(", ") | {
                    size: (.[0] | split("size: ") | .[1] | tonumber),
                    session: .[2],
                    serial: (.[3] | tonumber)
                }
            ]
        ' > snapshots_${i}.json;
    # Extract timing of HTTP requests.
    cat timing_${i}.log \
        | grep "timing: " \
        | jq -Rs '
            split("\n[") | [
                .[] | split(", ") | {
                    url: (.[0] | split("timing: ") | .[1]),
                    time: (.[1] | tonumber)
                }
            ]
        ' > timings_${i}.json;
    echo "Done with run $i.";
done
```

Then, we matched every snapshot size log entry (containing session ID, serial number and size) with the unique corresponding timing log entry (containing the URL and time taken to download in μs) by searching for the session ID and the term "snapshot" in the URLs. From this information (pairs of time and size for every successfully downloaded RRDP snapshot), we calculate statistics on the time spent downloading actual RPKI objects.

Measurement setup To simulate a typical relying party, we use a virtual machine in the SIDN Labs network (AS 215088), with roughly 2 Gbps bandwidth to the internet.

A.2 Results

By measuring 10 times from the SIDN Labs network, we get the results in table A.1. Our full data is available at `https://github.com/SIDN/pqc-rpki/`, and on request from the author and Radboud University.

[1] From inspecting the logs, we have confirmed that during our testing, the same repositories consistently failed to respond, so adding the timeout did not affect the measurements, that only trace successful HTTP requests.

We measure several things:

Snapshots is the mean of the sum of times for downloading individual RRDP snapshot XML files. This is the time that is really spent downloading RPKI content: in a fresh download, Routinator downloads everything using only snapshots, no delta files (and we have disabled `rsync`).

Notifications measures the time downloading RRDP notification files. These do not include any actual RPKI objects, so do not depend on the key and signature sizes. Hence, it is part of t_{const} and not $\frac{s}{b}$. Since snapshots are always downloaded *after* a notification file, the (constant) cost of the initial connection to a repository falls under this category: downloading the snapshot normally reuses the existing connection.

Total duration is the wall-clock time for the full invocation of Routinator, which depends heavily on the configured timeouts, and includes time spent parsing files, as well as time spent on unsuccessful HTTP requests, such as those for a few repositories that do not respond.

Out of these, the time for downloading snapshots is exactly the value for $\frac{s}{b}$ that we need. For all of the measurements, the total size of the downloaded RPKI objects is roughly 802 MB: slightly less than the 838 MB from table 4.1. This difference is expected, as we have disabled `rsync`, causing a few repositories are missing, and the measurements are from a different date (2025-03-04).

As expected, we find that $\frac{s}{b}$ is indeed only a small part of the total duration. From our machine, the size-dependent fraction of the downloading time was roughly 14.5 seconds: only about 6% of the 4 minutes used as worst-case estimate in section 4.2.1. That corresponds to an effective bandwidth of $\frac{802 \text{ MB}}{14.5 \text{ s}} = 55.3$ MB/s, as opposed to the 3.5 MB/s found there.[2]

Therefore, for many validators, it makes sense to assume much faster transfer of the actual RPKI content, although 14.5 seconds is on a machine with very good bandwidth, that might not be representative for many validators. Even assuming several times lower bandwidth, the time spent downloading is far below the 4 minutes used in section 4.2.1.

Table A.1: *Time spent downloading RRDP snapshots.*

Location	Measurement	Mean time (s)	Std. dev. (s)
SIDN Labs VM	Snapshots	14.5	1.5
	Notifications	29.5	5.6
	Total wall clock duration	323.9	13.7

[2]That is, the speed of downloading in terms of the RPKI objects' content size, not their base-64 encoded and possibly compressed form that actually goes over the network.

Appendix B

Measuring RP readiness

In this appendix, we suggest several procedures for measuring RP readiness to accept products using a new algorithm. These are important for the first production CAs to make an informed decision about when it is safe to migrate to the new algorithm B.

Some of these experiments are specific to our *mixed-tree* algorithm transition, based on the assumption that a testing CA can be created with algorithm B under an A production issuer; this is not possible in the approach of [RFC6916]. However, we also define procedures that can be used to monitor RP readiness during an [RFC6916] migration, and during a (normal, A-to-A) TA key rollover.

While each of these experiments individually takes considerable effort to set up, components can be shared between them.[1] This makes performing a combination of these experiments feasible, providing a very thorough view of the readiness of RPs to accept new algorithms.

B.1 Monitoring RP software from a repository

As recommended in [RFC8182], RP software implementations indicate their name and version in the User-Agent HTTP header, whenever they make RRDP requests to a repository. This information provides a straightforward way to monitor what software is used by RPs, since every RP normally accesses every repository periodically. Measuring this can be done by simply logging the User-Agent header and IP address of every request made to a repository.

This approach was used (measuring from a repository created specifically for this) in [38] to identify *which* validator implementations were used. Next, the method also appears (this time using APNIC's repository) in [77]. Similarly, NLnet Labs collect and report this information continuously since 2023.[2]

Logging User-Agent headers provides a complete view of the software that is active at any time, as long as it supports RRDP. There are two limitations, that both can be addressed:

- Not all validators use RRDP. Software that does not use RRDP does not show up here. It is still easy to measure this small fraction, by logging access to the `rsync` version of the measuring repository. This is also done in [38].
- The second most popular RP software, OpenBSD's rpki-client, does not report its version in the User-Agent header. This implementation currently accounts for roughly 20% of RPs. This can be addressed by simply updating rpki-client to report a version number.

[1] For example, logging User-Agent headers as in appendix B.1 can be done on the parent repository from B.2, and CAs created for B.2 can also be used to create the ROAs for B.4.

[2] Data is collected on the `rov-measurements.nlnetlabs.net` repository, and a basic analysis is shown on `https://rov-measurements.nlnetlabs.net/stats/`.

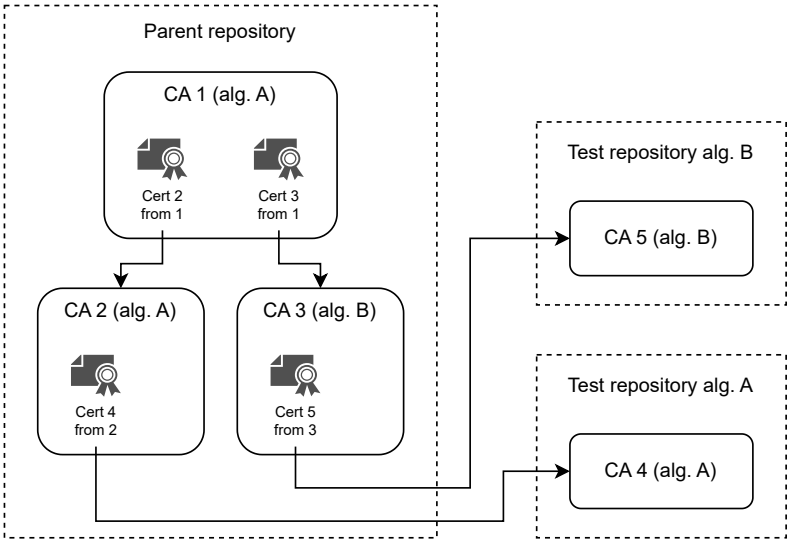


Figure B.1: *Example structure to measure RPs that reach repositories through a B certificate, versus only through A certificates. RPs access the 'Test repository alg. B' only if they validate the certificates for CAs 3 (with B subject) and 5 (with B issuer). Other CAs will reach only the 'Test repository alg. A' (which could be left out and replaced by monitoring access to the parent repository).*

For both limitations, the next method in B.2 provides more reliable results, just without the granularity of a precise version.

The User-Agent header is a simple and effective way to measure the versions of RP software that are in use. This kind of monitoring is already being done in practice and does not require e.g. introducing an experimental algorithm B CA.

B.2 Measuring reachability of B repository

Another way to measure directly how many RPs accept an algorithm B is by monitoring requests to a testing repository that is only advertised under an algorithm B resource certificate. RPs that do not understand algorithm B will reject the resource certificate that points to the testing repository, and hence never access it. The (number of) RPs that reach the testing repository can be compared to those that reach another (advertised through an all-A chain) repository, to count or identify the RPs that are not ready. An example of a structure of CAs and repositories that can be made to measure this is shown in fig. B.1.

This method is more reliable than the previous one. It measures the real acceptance of algorithm B by RPs, so it accounts for differences in configuration,[3] and validators that do not report accurate version information. It also covers `rsync`-only validators.

B.3 Measuring use of a new TAL

During a TA key rollover (see section 6.2.3.2), it is necessary to monitor whether all RPs are using the new TAL, before the old TAL can be removed, or even before the TA's direct subordinate certificates' (that are usually also under control of an RIR) Authority Information Access extension can safely be updated to refer to the new TA

[3]For example, support for a new algorithm could be behind an opt-in configuration option, such that the version number does not indicate whether the algorithm is actually accepted.

certificate.[4] Much like the previous method, this can be done by monitoring requests to the TA certificate linked in the new TAL, and comparing the requests to it with those to the old TA certificate. When no RPs that access the old TA certificate do not (also) access the new TA certificate, the new TAL is configured everywhere.

B.4 Measuring effect on routing

The previous methods all measure individual relying parties, and do so quite accurately. However, the number of RPs is only a proxy for what is actually relevant: the impact that migrating has on real-world routing. RPs that are not actually being used for route filtering are much less important than those that are. Accordingly, we also present two methods that measure propagation in actual routing.

A common technique to measure adoption of Route Origin Validation (ROV) is to create a few specific ROAs, and announce routes through BGP, such that some routes are ROV-Invalid, and others are ROV-Valid. Then, the propagation of an ROV-Valid announcement can be compared with an ROV-Invalid one to get a rough idea of how many ASes are performing ROV. This is used among others in [60, 31] and in several online testing tools, such as [15].

We can use a modification of this technique to measure how many ASes either do not perform ROV, or accept algorithm B, and compare it with the use of ROV accepting only A.

- First, we create a ROA using A for a prefix `a.b.0.0/23` for some AS X, and announce the `/23` prefix from that AS X.
- Next, we create a ROA using B for a more-specific prefix `a.b.0.0/24` for a different AS Y, and announce it from that AS Y. This announcement is ROV-Valid to an RP that accepts B, and more-specific that the covering announcement. Both networks that do not perform ROV, and those that perform ROV *and* accept B should accept this announcement. On the other hand, networks that perform ROV but do not accept B reject it as ROV-Invalid.
- To get a baseline of what announcements can be expected to be accepted, simply because networks *do not* perform ROV, we also announce the more-specific `a.b.1.0/24` from AS Y. This is ROV-Invalid to any RP, so should only be propagated by networks that do not perform ROV.

Of course, this method can be repeated for IPv6, with a `/47` prefix and the two contained `/48` prefixes. Many alternative setups are also possible, such as periodically creating and withdrawing the ROAs. The exact setup can be determined by implementers. An example is shown in fig. B.2. Using such a setup, the real-world effect can be measured in many ways.

B.4.1 Announcement propagation to BGP collectors

First, we can look directly at the propagation of different BGP announcements to see how many and which ASes accept the announcement. This approach — often categorized as a *control-plane* measurement — was first used in [29] in an uncontrolled or passive experiment, looking at propagation of ROV-Invalid announcements that occur naturally. It was also used in [31] for a controlled experiment where, as we propose, several ROAs and announcements are made to purposefully create some ROV-Invalid announcements. Then, BGP collectors such as *RouteViews* ([54]) and *RIPE RIS* ([52]) report from many vantage points what BGP announcements they

[4] Although [RFC6487] does not require it and [RFC9691] forbids it, there was confusion about whether the AIA's value should be checked during validation. Therefore, there is some risk in the temporarily inaccurate AIA value during TA key rollovers.

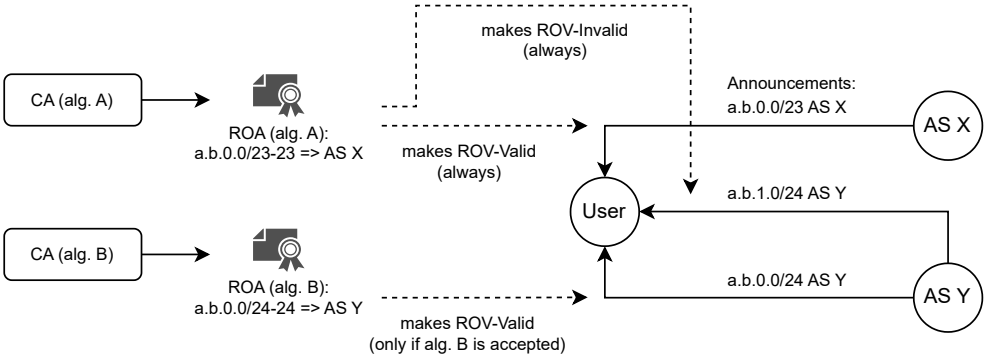


Figure B.2: *Example structure to measure propagation of algorithm B ROAs through propagation of announcements. Three announcements are made. One will be used only when ROV is performed and B is not accepted. One will only be used if ROV is not performed, and the last is preferred when ROV is performed and B is accepted. Use or propagation of these announcements can be measured from the 'user', which can be a BGP collector, looking glass, or a real user.*

see. This can be used to derive information about specific ASes and the internet as a whole. For example, the following can be derived:

(1) If the `a.b.0.0/24 ASY` announcement (that is ROV-Valid only when the RP accepts B) is seen just as much as the `a.b.1.0/24 ASY` (ROV-Valid to every RP), then most ROV-performing networks must be accepting B.

(2) For a particular AS Z, if a route (ending with) `a.b.1.0/24 ASZ ASY` is seen, but `a.b.0.0/24 ASZ ASY` is not, then AS Z is almost certainly performing ROV, but not accepting B.

More details about the various kinds of conclusions that can be drawn from BGP collectors with this kind of controlled experiment are given in [29, 31, 60], and applying such analysis to the specific case of measuring RP readiness is left to the implementers of the experiments we suggest.

B.4.2 Reachability from looking glasses

The BGP collectors used in the previous method do not have a complete view of the internet, as many ASes do not peer with them. For some ASes it allows reliably drawing conclusions, but many ASes are less visible.

An alternative to the *control-plane* measurement with BGP collectors is to do *data-plane* experiments. As suggested in [60] and used in [31, 63], one can use looking glasses like *RIPE Atlas* ([51]), that provide thousands of vantage points. From these, one can directly measure the actual routing behaviour from many locations.

By hosting two different web servers for `a.b.0.1`, in AS Y and AS X that return different content, we can check from a probe in AS Z whether the probe reaches the server in AS Y or AS X. Reaching X means that at least some AS prevented the `a.b.0.0/24 ASY` announcement from being used, so was performing ROV without accepting B. Alternatively (or additionally), `traceroute` can be used to get information about the entire path taken towards `a.b.0.1`, offering more insight into which ASes might have filtered the `a.b.0.0/24 ASY` origination or not. Again, the details of how to interpret such results are left to the implementers of the experiments, but can be based on the methods used in [31, 63].

B.4.3 Reachability from real users

To get even more insight in the real-world impact on reachability that switching to algorithm B would have, one can even use real users, as opposed to RIPE Atlas probes to try to reach `a.b.0.1`. This can be achieved with ad-based experiments, as performed by APNIC to measure adoption of DNSSEC validation and ROV [33, 34]. This provides endlessly many vantage points, and can be used to measure impact by the population of affected real-world users, instead of e.g. by the number of ASes. While this method gives the least detailed information of the range of methods we propose (only whether AS X or Y is reached), it does provide exactly the information that really matters: whether real users will be able to reach everything normally if a widespread algorithm migration were to happen.