dRR: A Decentralized, Scalable, and Auditable Architecture for RPKI Repository

Yingying Su*‡, Dan Li*†, Li Chen†, Qi Li*†, and Sitong Ling*
*Tsinghua University, †Zhongguancun Laboratory, ‡BNRist
{su-yy19, lingst21}@mails.tsinghua.edu.cn, lichen@zgclab.edu.cn, {tolidan, qli01}@tsinghua.edu.cn

***Abstract*—Although Resource Public Key Infrastructure (RPKI) is critical for securing inter-domain routing, we find that its key component, the RPKI Repository, is under studied. We conduct the first data-driven analysis of the existing RPKI Repository infrastructure, including a survey of worldwide AS administrators and a large-scale measurement of the existing RPKI Repository. Based on the findings of our study, we identify three key problems. Firstly, misbehaving RPKI authorities can easily manipulate RPKI objects, and Internet Number Resources holders (INRs holders) and Relying Parties (RPs) can neither prevent malicious behaviors of misbehaving authorities nor hold them accountable. Secondly, RPKI Repository is sensitive to failures: An attack or downtime of any repository Publication Point (PP) will prevent RPs from obtaining complete RPKI object views. Finally, we identify scalability issues with the current RPKI Repository, which are expected to worsen with the further deployment of Route Origin Authorization (ROA).**

To address these problems, we propose dRR, an architecture that enhances the security, robustness, and scalability of the RPKI Repository while being compatible with standard RPKI. By introducing two new entities: Certificate Servers (CSs) and Monitors, dRR forms a decentralized federation of CSs, which enables the RPKI Repository to proactively defend against malicious behavior from authorities and to tolerate PPs' failures. dRR is also scalable for future large-scale deployment. We present the design of dRR in detail and implement a prototype of dRR on a global Internet testbed spanning 15 countries. Experimental results show that, although new security features are introduced, dRR only incurs negligible latency for certificate issuance and revocation. The throughput of certificate updates achieved by dRR is 450 times higher than the current maximum RPKI certificate update frequency.

I. Introduction

RPKI [38] is promoted by IETF [29] to secure inter-domain routing against IP prefix hijackings, *i.e.*, Autonomous Systems (ASes) mis-originate IP prefixes that they do not own to illegitimately attract traffic [8], [45]. After a decade of deployment, RPKI has indeed demonstrated its effectiveness in protecting the Internet.[1] As shown in Figure 1, to establish a trustworthy mapping between ASes and prefixes, RPKI arranges the Certificate Authorities (CAs) in a hierarchy that mirrors IP address allocation, and five RIRs are trust anchors.

[1]On March 28, 2022, RPKI successfully protected Twitter (AS13414) from prefix hijacking by Russian ISP JSC RTComm.RU (AS8342) [17].

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24368
www.ndss-symposium.org

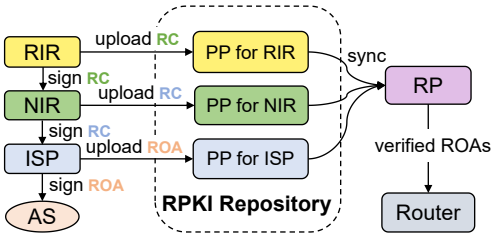


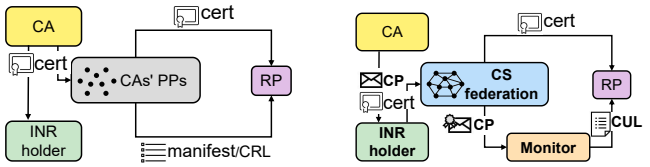
Fig. 1: RPKI architecture. RIR, NIR, and ISP represent the Regional Internet Registry, the National Internet Registry, and the Internet Service Provider, respectively.

Each CA holds a Resource Certificate (RC) issued by its parent authority, which records its allocated INRs. CAs can issue subordinate RCs to reallocate their resources or issue ROAs to authorize ASes to originate specific IP prefixes. Each CA will upload the objects it signs to the repository Publication Point (PP) it operates. These PPs collectively form the global RPKI Repository. Relying Parties (RPs), as entities that request RPKI data to make BGP routing decisions, periodically fetch and validate RPKI objects, and then use the verified ROAs to guide routers to perform Route Origin Validation (ROV) [11].[2]

Although the industry and the research community have been continuously strengthening RPKI [23], [52], [13], [19], [39], we find that a key component of RPKI remains neglected: the RPKI Repository. In this work, we conduct the first data-driven security analysis of the RPKI Repository, and identify three key problems that affect the integrity and accuracy of the stored RPKI objects and hinder future large-scale RPKI deployment.

- **Problem 1 (P1): Unilateral reliance on authority.** We observe that RPKI authorities (CA and its corresponding repository PP manager) have significant unilateral power, which poses three issues:
  - **P1.1** Since the current RPKI Repository is not tamper-resistant, authorities can unilaterally undermine any RPKI objects without consent from subordinate INR holders. Malicious or breached authorities can perform arbitrary operations on INR holders' certificates (*e.g.*, deletion, corruption, modification, or revocation), while they can also compromise RPs (*e.g.*, show incomplete or inaccurate RPKI object views to RPs).
  - **P1.2** INR holders and RPs can only rely on RPKI authorities without the ability to verify that their security requirements are being met. INR holders cannot know if their certificates are securely stored in PPs and can be publicly seen and fetched by all RPs; RPs cannot

[2]We have summarized all abbreviations used in dRR in Appendix B



(a) the current RPKI Repository. (b) dRR

Fig. 2: Comparison of the RPKI Repository and dRR

verify the integrity and accuracy of the RPKI objects they synchronize from PPs.

P1.3 The history of RPKI objects is not easily auditable. A complete longitudinal view of RPKI objects is necessary to defend against malicious manipulation of the RPKI Repository and to resolve conflicts between authorities and INR holders. Although RPs can periodically fetch RPKI objects to keep track of their historical versions, it is costly, without any guarantee of completeness. We believe auditability is best provided by the RPKI Repository, which naturally tracks all RPKI objects.

- **Problem 2 (P2): Every RPKI object is a singleton in RPKI Repository.** Although the current RPKI Repository is globally distributed, each CA only stores the RPKI objects it issues in the unique PP it runs. If any PP fails (malfunctions or is attacked), RPs will not be able to fetch the RPKI objects issued by its CA, and the integrity of the global RPKI object view cannot be guaranteed. Additionally, the singleton nature of RPKI objects introduces unwanted interdependence between the accessibility of a CA's PP and the reachability of the AS that the PP locates. Since RPKI ensures the routing security and reachability of ASes, it is fair to expect PPs to remain accessible during any incident when corresponding ASes become unreachable.
- **Problem 3 (P3): The current RPKI Repository is costly in RP refreshing and lacks scalability.** Refreshing the local cache of each RP involves traversing all PPs to fetch the updated RPKI objects. In the past five years, the number of PPs has grown more than 12 times and is expected to increase dramatically with the further deployment of ROA. The growth in the number of PPs increases the cost of RP refreshes, which may prevent RPs from obtaining routing information in a timely manner, and also prohibits the release of use cases such as temporary ROAs to enable ISPs to perform tactical traffic engineering to ease congestion. Furthermore, the low barriers to running PPs and joining RPKI Repository will introduce unforeseen risks to RPs, as some PPs may emerge with unexpected intentions.

Existing work [23], [52], [39] has been devoted to addressing some of the above problems. However, they cannot comprehensively address our concerns but partially mitigate the threats from authorities. We present a detailed introduction of these works in § VII. Furthermore, Google's Certificate Transparency (CT) [21] is a successful industry practice for web PKI security. However, web PKI and RPKI face entirely different threats: in webPKI, malicious CAs will issue fake certificates; whereas, in RPKI, malicious CAs may unilaterally compromise legitimate certificates. Since RPKI certificate signing and management are under the control of CAs, the core idea of CT, which requires CAs to log issued certificates, cannot proactively defend against RPKI certificate compromise. Therefore, in this paper, we propose a new architecture, dRR (decentralized RPKI Repository), based on the above observations to enhance the security, robustness, and scalability of the current RPKI Repository.

dRR is an extension and upgrade of the existing RPKI Repository to address these problems in a comprehensive way. As shown in Figure 2, dRR can incrementally replace the RPKI Repository with two new entities: Certificate Servers (CSs) and Monitors. With the core idea of "separating distribution from signing", dRR's differentiating design choice is that, instead of binding a PP to a CA, dRR's CSs form a distributed federation. All CSs are equal in the CS federation, and the federation, as a whole, can partially or completely replace the current PPs to host RPKI objects. INR holders can proactively upload their certificates to the CSs they trust. And any certificate issuance and revocation will be publicized in the CS federation in the form of Certificate Policies (CPs, defined in § IV-A2). Each CS will proactively push the publicized CPs to its connected dRR Monitors. Based on the CPs, Monitors can then provide verifiable Certificate Update Lists (CULs) to RPs, and also provide INR holders with services to check the authenticity and validity of specific certificates.

dRR can solve or at least mitigate the problems we identified:

- dRR's CS federation and Monitors collectively solve **P1**. INR holders can freely choose the CSs they trust to provide certificate hosting services for them. The specially defined CPs can provide proof that the relevant CAs and INR holders have confirmed the issuance and revocation of the certificates. Thus, dRR can prevent malicious manipulation of certificates by RPKI authorities (**P1.1**). For **P1.2**, Monitors provide RPs with verifiable CULs based on the maintained *M-Tree*, which is a domain-adapted Merkel Hash Tree for dRR. It allows RPs to check the integrity and accuracy of the obtained RPKI object views. With M-Tree, Monitors can also help INR holders verify the authenticity and validity of specific certificates. Finally, the chronologically recorded CPs by the CS federation can provide a widely recognized longitudinal view of RPKI objects for auditing (**P1.3**).
- dRR breaks the coupling between CAs and PPs, and solves **P2** by allowing INR holders to store certificates in multiple trusted CSs. This guarantees a truly distributed RPKI Repository, and the failure of one CS will not affect the integrity of RPKI object views.
- Regarding **P3**, the access mechanism of the CS federation effectively restricts the number of CSs that RPs need to access. It not only enhances the scalability of the RPKI Repository, but also mitigates the unforeseen risks caused by casually joined PPs.

We implement a prototype of dRR and evaluate its performance on the Internet using a 100-node testbed that spans 15 countries. We deploy our distributed CSs primarily based on the geographic location of the current PPs, and use the Hotstuff protocol [56] to achieve the consensus of the CS federation on CPs. Our evaluation shows that the new security features of dRR introduce minimal overhead. With the expanding scale of CSs, the CS federation incurs only a latency of less than 2 seconds for the certificate issuance and revocation. The Monitor can fetch the newly confirmed CPs and refresh the M-Tree in 0.5 seconds. The experimental results show

that the additional closed-loop time from CAs' signing to RPs' synchronizing of certificates is no more than 3 seconds. Additionally, we evaluate the CS federation's throughput for handling certificate updates, which shows 450 times higher than the current frequency of RPKI certificate renewals.

In summary, we make the following contributions:

(1) We conduct a data-driven RPKI threat analysis, which involved a worldwide survey to understand the industry's concerns about RPKI, as well as the first large-scale measurement on the RPKI Repository. Based on these key insights, we propose the essential properties that a secure, robust, and scalable RPKI Repository should possess.

(2) We present dRR, an RPKI-compatible architecture to replace the current vulnerable RPKI Repository. To the best of our knowledge, dRR is the first solution designed to proactively defend against malicious behavior by RPKI authorities from the perspective of RPKI Repository.

(3) We break the current model that each RPKI object is a singleton in RPKI Repository and achieve the truly distributed storage of RPKI data. We also effectively solve the scalability problems caused by RPs to establish connections with a large number of PPs so that RPKI can better cope with large-scale deployment in the future.

(4) We implement a prototype of dRR and evaluate the performance of dRR on a global testbed with 100 nodes. We also demonstrate the effectiveness of our system.

II. Background and Threat Analysis

In this section, we overview the RPKI architecture and then perform the first data-driven RPKI threats analysis.

A. RPKI Overview

RPKI follows a PKI-based [53] architecture and the certificates are organized in a hierarchical structure. At the top of the hierarchy are the trust anchors. The five widely recognized trust anchors of RPKI are AFRINIC [1], APNIC [4], ARIN [5], LACNIC [36], and RIPE NCC [49]. As shown in Figure 3, each RIR holds a root RC that encompasses all INRs. CAs in RPKI can issue subordinate RCs that bind a subset of their INRs to the public keys of the sub-CAs (*e.g.*, APNIC assigning prefixes `27.111.32.0/19` and `43.224.168.0/22` to Indonesia) or ROAs (leaf nodes) that authorize ASes to originate specific IP prefixes (*e.g.*, PT.Inet.Global.Indo authorizing `AS24532` to announce prefixes `27.111.39.0/19-24`). Entities that possess RCs are CAs (such as APNIC, Indonesia, and PT.Inet.Global.Indo); entities holding ROAs or RCs are INR holders (such as Indonesia, PT.Inet.Global.Indo, `AS45101`, and `AS24532`, with their parent CAs being the certificate issuers).

RPKI Repository. Each CA operates a publicly accessible PP [26] to host the RPKI objects it has issued. The URL of the PP is recorded in the `Subject Information Access` (SIA) field in its RC. As shown in Figure 3, Indonesia's PP stores the RCs and ROAs it has signed for subordinate INR holders, a Certificate Revocation List (CRL) [14] that records the revoked RPKI objects, and a manifest [7] that records all valid objects in the PP. RPs will ignore the RPKI objects that are present in PPs but not listed in the manifest or have been listed in the CRL. Hereby, each CA's PP exclusively stores the RPKI objects issued by the respective CA, while collectively, all PPs form the global RPKI Repository that maintains a comprehensive view of the RPKI objects.

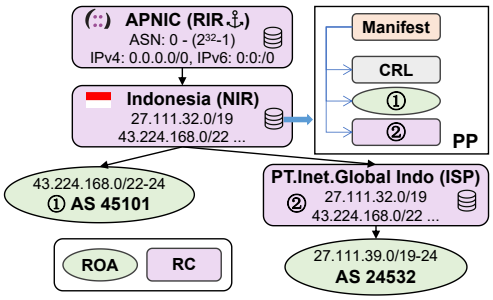


Fig. 3: Example of RPKI hierarchical structure.

Route origin validation. RPs utilize RPKI Repository Delta Protocol (RRDP) [10] or Rsync protocol to periodically traverse all PPs and fetch RPKI data to their local caches. After synchronization, RPs follow the hierarchical structure, starting from the top of the certificate tree, to verify all RCs and ROAs [28][3]. Then, they use the valid ROAs to generate a *prefix-to-AS* mapping table and install the table to border routers which then can use it to verify incoming BGP updates. IETF recommends that border routers should discard all invalid BGP updates and allow valid and unknown ones [11], [44].

B. Data-driven RPKI Threat Analysis

We conduct a data-driven threat analysis for RPKI Repository, and summarize three problems, as listed in § I. We identify **P1** and **P3** from a worldwide survey to understand industry concerns about malicious RPKI authority and running PPs. We received valid responses from 68 AS administrators (ROA adopters), and their feedback confirmed the urgency of addressing these problems. Appendix D shows the details of our survey. For **P2**, we conduct the first large-scale measurement on the RPKI Repository and reveal the current deployment status of PPs.

1) Malicious RPKI Authorities: RPKI authorities control the signing and management of RPKI objects in their PPs. Therefore, they have the ability to unilaterally undermine the INR authorization to subordinate INR holders by maliciously manipulating RPKI objects, especially RCs and ROAs. For example, Indonesia's revocation of RC② will lead to PT.Inet.Global Indo losing the resource authorization for its prefixes, consequently invalidating the ROA for `AS24532`. Compromised signed objects would prevent RPs from obtaining authentic and complete RPKI data, thereby affecting the accuracy of ROV. The malicious behaviors of the RPKI authority that we are concerned about include:

- **Revocation:** listing RCs or ROAs in the CRL. Revoked objects will be considered invalid by RPs.
- **Deletion:** deleting RCs or ROAs from the PP. Malicious deletions will prevent RPs from obtaining these objects.
- **Corruption:** corrupting RCs or ROAs. The corrupted objects will fail the verification and be rejected by RPs.
- **Modification:** modifying RCs and ROAs. Malicious modifications often involve a reduction in the authorized INRs or changes to the authorized entities.

Any malicious actions against the will of INR holders may undermine the authority of RPKI data. Therefore, the prefix-to-AS mappings in the affected ROAs will be compromised, resulting in relevant valid or unknown BGP updates becoming

[3]Currently, there are several open-source relying party software [35], [54], [12], [41], [34], [50] available to assist the RP in automating the certificate synchronization and validation processes.

ROV invalid and then being rejected by border routers (**P1.1**). However, INR holders and RPs rely heavily on the RPKI authorities, making it difficult for them to promptly perceive that their will has been compromised (**P1.2**). Since RPKI lacks a trustworthy historical RPKI object record, it is also challenging, if not impossible, to hold authorities accountable even if attacks are detected after the fact (**P1.3**).

In our survey, 44.1% of the AS administrators explicitly expressed concerns about malicious RPKI authorities (11.5% of them chose "not sure"). Two administrators provided additional feedback, indicating that they consider the threat from the RPKI authorities to be the most serious problem. One of them said they had lost all their ROAs due to administrative/human reasons. It can be seen that threats from RPKI authorities are also widely recognized in the industry.

2) Vulnerability of RPKI Repository: We then analyze the vulnerability of the PP based on the measurement of the current RPKI repository, which will lead to **P2**.

To analyze the RPKI Repository, we obtained a complete RPKI snapshot on April 1, 2023. There are 36,976 RCs and 122,251 ROAs. For each RC, we extract the SIA field that records the URL of its HTTPS-based RRDP file. We find that there are currently 61 independent PPs[4]. We use 20,000 globally distributed DNS resolvers to resolve their domain names, with the aim of finding all IP address records for each PP. Then, we use the routing data maintained by RIPE NCC[48] to obtain the AS where each PP is located. Appendix C shows our detailed measurement results.

Our analysis shows that although RRDP is designed to leverage CDN infrastructure for resilient service, we find that only 8 out of 61 PPs are hosted in CDNs, with 7 being hosted on Cloudflare `AS13335` and 1 on Amazon `AS16509`. We also find that out of 61 PPs, 58 of them are hosted in a single AS. This means that the availability of these PPs is highly dependent on the reachability of a single AS. Worse still, among the PPs whose corresponding ASes have deployed ROAs, we have identified that 14 of them carry the ROA of the ASes in which they are located. Once the PP goes down and the RPs cannot fetch its ROAs, the route of the AS that the PP locates in may be downgraded by ROV adopters. Then, even if the PP is restored, those ROV adopters still cannot access the PP, in which case the access of the PP depends on the reachability of the AS it locates in, while the reachability of the AS also depends on the access of the PP.

Real-world incidents of PP breakdowns do occur at times. On 6 April 2020 [42], the PP maintained by RIPE NCC suffered a sudden increase in connection to service, resulting in it appearing as down to many RPs for 7 hours (RIPE's services are now hosted on CDNs). On 15 May 2020, the Japan-operated PP was out of service for 10 hours due to hardware failure [30], and between 26 Jan 2022 and 2 Feb 2022, due to full disk space, all ROAs in its PP again became invalid [31]. The current storage mode of RPKI objects implies that a single point of failure in PPs may hinder RPs from obtaining complete RPKI views. Since a truly distributed and highly reliable RPKI Repository is crucial for RPKI security, we expect that RPKI Repository does not rely on the availability of a single PP or even the reachability of a single AS.

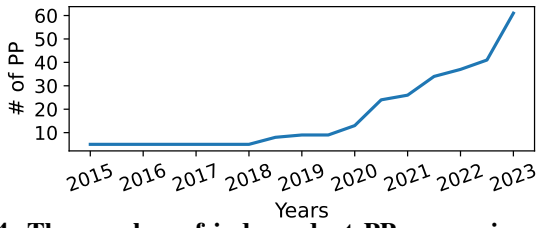


Fig. 4: The number of independent PPs over nine years.

3) Scalability of RPKI Repository: Finally, we analyze the scalability of the RPKI Repository from the perspective of the growth in the number of repository PPs (**P3**).

As shown in Figure 4, RPKI Repository has grown from 5 PPs run by five RIRs to more than 60 PPs. On the one hand, with a deeper understanding of RPKI, ROA deployers prefer to adopt delegated RPKI to flexibly control their RPKI objects. [5] On the other hand, RPKI Repository is designed without a strict admission mechanism, allowing entities to register as delegated CAs and operate PPs with a small fee and identity verification[2], [3], [6], [37], [46]. Therefore, some PPs for unwanted purposes (measurement, attack experiments, etc.) are gradually emerging. Since RPs are required to check the updates of all PPs when refreshing their local caches, the increasing number of PPs will result in higher refresh costs. Even worse, a malicious CA can create a large number of descendant RCs and operate numerous PPs to make RPs endlessly retrieve PPs, thus exhausting and paralyzing RPs. We find that Koen van Hove has demonstrated the feasibility of this attack by manipulating his PP (parent.rov.koenvanhove.nl) [55].

For **P3**, we asked AS administrators using hosted RPKI about their willingness to adopt delegated RPKI in the future. 45.3% of them say they have plans to run their own PPs for flexible certificate control. The result shows that delegated RPKI will emerge as a trend, and the number of PPs will inevitably increase. The growth in the number of PPs not only threatens the scalability of RPKI but also brings unexpected risks to RPs. Since each PP connects to all RPs in the world, we urge that the Internet community pays more attention to this scalability issue.

C. Threat Model

The main participants in dRR include CAs, INR holders, RPs, and CSs. dRR focuses on solving the problems we defined in § II-B, including malicious RPKI authorities with excessive unilateral power (**P1**), and the vulnerability (**P2**) and scalability (**P3**) issues of the RPKI Repository.

In dRR, honest INR holders expect their certificates to be securely stored in the repository and accessible to all RPs; honest RPs hope to obtain trustworthy RPKI object views and timely perceive RPKI data updates. For a malicious RPKI authority, it can unilaterally manipulate its issued RPKI certificates, using cryptographic (revocation or modification) or non-cryptographic (deletion or corruption) means, to diminish the set of resources associated with the victim INR holders. It

[4]Since all PPs now support RRDP and serve RPs through RRDP files, our analysis of the RPKI Repository focuses primarily on RRDP. SIA fields of RCs held by the same organization will point to the same PP. Therefore, the number of PPs is much lower than that of RCs.

[5]RIRs offer hosted RPKI [9] to help entities quickly deploy RPKI. For hosted CAs, RIRs allow them to apply for RCs. The difference is that RIRs will help them issue ROAs, and the RC and ROAs will be stored in RIRs' PPs. CAs deploying delegated RPKI will sign RPKI objects by themselves and run their own PPs to store objects.

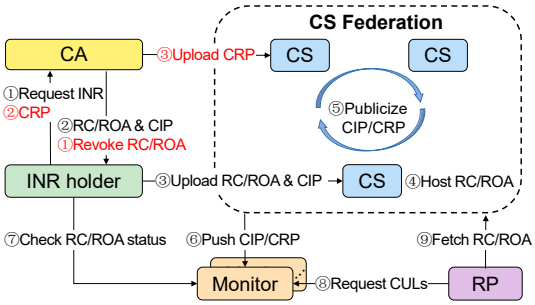


Fig. 5: The dRR architecture and the workflow.

can also show incomplete and inaccurate RPKI views to RPs. Malicious actions taken by authorities undermine the goal of INR holders and RPs. In dRR, malicious INR holders may abuse IP prefixes to launch Internet attacks or attempt to falsely claim ownership of certificates that belong to others. Malicious CSs may delete or corrupt the certificates they host or disrupt the CS federation's efforts to maintain a widely recognized RPKI object global ledger. An attacker may compromise the private keys of INR holders and CSs. Note that other attacks that are unrelated to the exploitation of weakness in the RPKI Repository architecture, such as DDoS, are beyond the scope of our paper.

III. Architectural Overview

A. Design Goals

Our design goals of dRR are as follows: First, dRR should balance the disproportionate power between the RPKI authorities and INR holders. It should actively defend against malicious behavior by RPKI authorities, ensure the secure storage of INR holders' certificates, and provide trustworthy RPKI object views to RPs. Second, dRR should implement a truly distributed RPKI Repository. Any single point of failure should not affect the integrity of the data obtained by RPs. Third, dRR should be sufficiently scalable to accommodate the future deployment of RPKI, especially the delegated RPKI. dRR should *not* affect the existing RPKI certificate issuance and validation, and should be compatible with RPKI while supporting incremental deployment.

B. dRR Overview

Towards these goals, we aim to design a new RPKI Repository, to serve as a decentralized, scalable, and auditable intermediary between the certificate issuance system and RPs. As a key component connecting the certificate supply side and the certificate demand side, it is important to ensure the robustness and security of the RPKI Repository. For current RPKI, binding the Repository to CAs not only grants significant power to RPKI authorities and enables them to engage in unilateral malicious actions towards their issued certificates, but also affects the scalability of RPKI. Therefore, dRR defends threats to RPKI by constructing an enhanced RPKI Repository that is independent of the certificate supply side. As shown in Figure 5, dRR serves as a new RPKI Repository with two entities: the *CS federation*, which is formed by *Certificate Servers* (CSs) and can be seen as upgraded PPs, and the *Monitors*.

CS Federation. CS federation inherits the responsibility for hosting RCs and ROAs for INR holders. Compared to traditional PPs, the CS has two main improvements: (1) they are independent of CAs, all CSs are equal and together form the CS federation, and (2) while helping INR holders host the certificates, the CS will also publicize the fingerprint and operation (issuance or revocation) of these certificates they host among CS members. Specifically, INR holders are no longer required to store certificates in the PP controlled by the parent CA, but they can freely choose multiple CSs they trust to provide services for them. Along with the new certificate, the CA will also provide a Certificate Issuance Policy (CIP), which can be used to prove the authenticity of the certificate to the INR holder. The CIP will then be publicized among the CS federation to show that it is widely recognized (Step 1 to Step 5 in Figure 5, marked in black). When a certificate is to be revoked, the CA that signed the certificate should publish a Certificate Revocation Policy (CRP) to the federation. It can be used to prove all affected INR holders have agreed on the revocation (Step 1 to Step 3, marked in red). After the CS federation confirms the CPs, the corresponding certificates can be successfully issued or revoked.

Monitors. The Monitor has two responsibilities: (1) provide RPs with verifiable Certificate Update Lists (CULs) to allow them to verify the authenticity and integrity of the RPKI object view, and (2) provide INR holders with services to check the authenticity and validity of specific certificates. Specifically, whenever the CS federation confirms a batch of CPs, each CS will proactively push the newly added CPs to the Monitors it serves so that Monitors can update their locally maintained M-Trees (Step 6). RPs no longer traverse all PPs to check updates but fetch verifiable CULs from the Monitor they trust. Then, based on CULs, RPs synchronize the newly added certificates and delete the revoked ones to complete the local cache refresh (Step 8 to Step 9). INR holders can also ask the Monitor whether the certificate they are concerned about has been protected by dRR, and the Monitor returns the result and the proof to INR holders (Step 7).

In dRR, the certificate hosting service provided by the CS federation meets the needs of INR holders to actively control their certificates, while also ensuring that RPs will not fail to fetch certificates due to the failure of a single CS. Additionally, the admission mechanism of the federation also mitigates the scalability issues caused by the arbitrary growth of PPs. Furthermore, with Monitors, RPs can easily verify the integrity of the obtained RPKI views. dRR replaces the RPKI Repository with the more powerful CS federation, which is the key to achieving the expected properties of RPKI Repository.

IV. Design Details

In this section, we describe the main components of dRR in detail. We start by introducing the CS federation, including CS operations and certificate policies. Next, we present the M-Tree data structure maintained by Monitors and how they serve RPs and INR holders. Finally, we discuss the incremental deployment mechanism for dRR.

A. CS Federation

CS federation consists of multiple CSs. The fundamental design principles of CSs are as follows: (1) unlike the PP which is bound to a specific CA, the CS is independent of the CA. All CSs are equal and together form a CS federation; (2) entities that want to operate a CS in the federation must pass security and reliability verification by existing CS members; (3) INR holders can freely choose any CSs they trust to provide

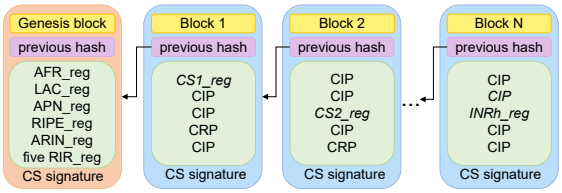


Fig. 6: The global ledger maintained by the CS federation. Genesis block contains the RIR registration messages, and subsequent blocks contain CS and INR holder registration messages (*CS_reg* and *INRh_reg*), CIPs, and CRPs.

certificate hosting services for them. These principles ensure the freedom of trust for INR holders as well as the security and reliability of the CS federation. We envision five RIRs to begin the deployment of dRR, forming the initial CS federation. Other eligible entities can join the federation as CSs with the approval of five RIRs and existing CS members. All members of the CS federation collaborate to ensure secure certificate storage and enable public auditability of certificate operations. To achieve that, we design a series of CS operations that each CS should support.

1) CS Operations: CS operations include registration of CSs and INR holders, hosting certificates, and publicizing certificate policies. In dRR, new CS members and INR holders who deploy dRR must complete identity registration in the federation first. All the registration information and CPs will be uploaded to the CS federation. With the consensus of the existing CS members, the operations will be appended to the tamper-resistant global ledger maintained by the federation. The ledger provides a complete longitudinal view of RPKI objects for effective auditing. In dRR, CS members can use any security-proven consensus protocol [22], [57], [56], [40], [43] to achieve consensus on these operations.

CS Registration. The CS registration is conducted with CS Registration Message (the format is shown in Table I), which consists of the following fields: `CS_ID`, the registration type `T_R`, the latest public key `P_K` and its corresponding signature `CS_SIG`, and the signature with the previous key `Pre_SIG` (optional). To initialize the federation, we expect five RIRs to register and join the federation first. They should also register a shared identity for joint decision-making. For the shared identity, the `CS_ID` will be *All-RIRs* and the `P_K` will be jointly held by all RIRs in the form of threshold signatures[51]. As shown in Figure 6, the initialization registration message of the five RIRs will be written into the genesis block of the ledger when the system is initialized.

For an entity that wants to run a CS, its initial registration message (*I_R*) must be submitted to the federation by five RIRs using their jointly held identity. If a CS wants to change its `P_K`, the update message (*U_R*) can be submitted by itself, with the requirement that a signature signed with the previous private key should be attached to `Pre_SIG` field. For a CS who wants to leave the federation, its deregistration message (*D_R*) should also be submitted by five RIRs. After existing CS members reach a consensus, these messages will be written into the global ledger (shown in Figure 6), and the corresponding entity will either officially run a CS, leave the CS federation, or complete updating the `P_K`.

The eligibility of entities operating CS must be verified by all RIRs and existing CS members. To ensure the security of certificates and the stability of serving RPs, it is important that CS operators possess a robust, secure, and reliable service infrastructure, such as CDNs. Additionally, CS members should also have a good reputation and be diversified in affiliations. For example, state-run institutions and large ISPs (*e.g.,*Akamai, Amazon, Cloudflare, *etc.*) are suitable candidates. The strict eligibility requirements not only guarantee the quality of services provided by the CS federation but also address the issue of uncontrolled expansion of CS members.

TABLE I: CS Registration Message.

Field Name	Content
`CS_ID`	ID of the CS. This field uniquely identifies a CS.
`T_R`	The type of registration, *e.g.,* initializing registration (*I_R*), update registration (*U_R*), or deregistration (*D_R*).
`P_K`	The latest public key (P_K) of the CS whose private key is used for consensus on CPs and CS/INR holder registrations.
`CS_SIG`	The CS's signature on current registration signed with the `P_K`'s corresponding private key.
`Pre_SIG`	When the `T_R` field is *U_R*, this field will contain the CS's signature on current registration signed with previous P_K's corresponding private key.

TABLE II: INR holder Registration Message.

Field Name	Content
`INR_holder_ID`	ID of the INR holder.
`T_R`	The type of registration, *e.g.,* initializing registration (*I_R*), update registration (*U_R*, update `P_K` or `Trusted_CSs`), or deregistration (*D_R*).
`P_K`	The latest public key (P_K) of the INR holder whose private key is used to sign the CPs.
`Trusted_CSs`	IDs of all CSs trusted by the INR holder. It can be represented as a sequence: [CS_1_ID, CS_2_ID, ... ,CS_n_ID].
`INR_holder_SIG`	The signature on current registration signed with the `P_K`'s corresponding private key.
`Pre_SIG`	When the `T_R` field is *U_R* and the `P_K` is changed, this field will contain the signature on current registration signed with the previous P_K's corresponding private key.

INR holder Registration. Each INR holder deploying dRR should register its identity in the CS federation. The registration message must contain the fields listed in Table II.

The `INR_holder_ID` field descriptively represents a unique entity, containing identity-distinguishable information such as the name of the organization that owns the certificates (*e.g.*, Google). Each INR holder will host its certificates in the CSs it trusts, which are listed in the `Trusted_CSs` field. The `INR_holder_SIG` field contains its signature on the current registration, which is signed with the `P_K`'s corresponding private key. The `T_R` field indicates the type of registration: initialization (*I_R*), update (*U_R*), or deregistration (*D_R*). Update registration messages allows INR holder to update their keys and `Trusted_CSs` lists. For updating the key, the `P_K` field will contain a new public key and a signature signed with the previous `P_K`'s corresponding private key must be attached to `Pre_SIG` field.

After confirming the identity, one of the CSs trusted by the INR holder will submit the registration message to the CS federation. As shown in Figure 6, after the CS members verify the signature, the INR holder registration message will be recorded in the global ledger.

Hosting Certificates and Publicizing CPs. INR holders are free to choose their trusted CSs to host their RCs and ROAs. The INR holder should also choose one of the CSs hosting the certificates to publicize the corresponding certificate policies to the CS federation. As shown in Figure 6, every CP that has been verified by the CS members will be recorded in the

TABLE III: Fields in Certificate Issuance Policy (CIP).

Field Name	Content
VERSION	The version of the current CIP.
ISSUER	INR_holder_ID of the certificate issuer (*i.e.*, CA).
SUBJECT	INR_holder_ID of the certificate owner.
CERT	The hash of the protected certificate.
CERT_T	The type of the protected certificate, *RC* or *ROA*.
ISSUER_RC	The hash of the protected certificate's parent RC.
VALIDITY	The validity period of this certificate. It is a tuple: (*notBefore*, *notAfter*), and must be the same as the validity period in the certificate.
CS_SET	IDs of the CSs hosting this certificate. It is represented as a sequence: [CS_1_ID, CS_2_ID, ... , CS_n_ID].
CIP_HASH	The hash of this CIP.
CIP_SIG	The issuer's signature on this CIP.

TABLE IV: Fields in Certificate Revocation Policy (CRP).

Field Name	Content
VERSION	The version of the current CRP.
R_M	Method of revoking the certificate: *self* or *rir*.
CRP_ISSUER	The issuer of this CRP.
CERT_SET	The hash list of the revoked certificates. It is represented as a sequence: [$CERT_1$, $CERT_2$, ... ,$CERT_n$].
CRP_HASH	The hash of this CRP.
CRP_SIG	The CRP_ISSUER's signature on this CRP.

global ledger in chronological order.

2) Certificate Policies: dRR has two certificate policies, CIP and CRP, which represent the certificate issuance and revocation, respectively. The CIP contains the necessary information to prove that this specified certificate has been recognized by both the CA and the INR holder. It can serve as proof of the real existence of the certificate. The CRP contains information that proves the mutual agreement of the CA and all affected INR holders in revoking the certificate. It can prevent CAs from unilaterally revoking the certificates.

Certificate Issuance Policy. When a CA issues RCs and ROAs for subordinate INR holders, it also provides them with CIPs to show the authorization of these certificates. CIPs provided by the CA (ISSUER) to the INR holder (SUBJECT) contain the fields listed in Table III. The ISSUER and SUBJECT fields must match the INR holder ID in their registration messages, and the CERT field uniquely identifies the certificate protected by this CIP. For an INR holder, its certificates can be hosted in different CS sets. For more important ones, the INR holder may prefer to host them in more CSs to enhance robustness. Note that CS_SET in CIPs must be a subset of the Trusted_CSs in INR holders' registration messages.

Certificate Revocation Policy. When revoking an RC or ROA, the CA that signed the object needs to obtain a CRP from downstream INR holders to confirm that the revocation has obtained the consent of all affected parties. The CRP contains the fields listed in Table IV. dRR also allows five RIRs to jointly sign CRPs for mandatory certificate revocation. For example, if INR holders exploit IP prefixes for malicious activity, it may be necessary to consider forcibly reclaiming their resources. In this case, the R_M field, which indicates the revocation method, will be recorded as *rir* and the CRP_ISSUER will be *All-RIRs*. When a CA performs certificate revocation, it must ensure that the certificate in the CERT_SET has no descendant certificates or that the certificate and its descendant certificates belong to the same INR holder. It means that the revocation of an RC must recursively start from the lowest-level descendant certificate to ensure that all affected INR holders have agreed on the revocation. But for mandatory revocation, regardless of the status of descendant certificates, the revocation will be enforced, and all certificates will be invalidated.

TABLE V: The table shows the total number of RCs (RC_num) and the cumulative distribution of the number of CRPs that need to be issued to revoke RCs of five RIRs. For example, for LACNIC, the revocation of 91.59% RCs (4,529) requires at most two CRs to be issued. The statistic is generated from the RPKI snapshot on April 1, 2023.

RIR	1 CRP	2 CRPs	3 CRPs	4 CRPs	5 CRPs	RC_num
AFRINIC	21.88%	91.97%	96.41%	98.12%	98.46%	585
LACNIC	46.11%	91.59%	97.04%	97.31%	98.65%	4945
APNIC	29.23%	80.89%	92.22%	96.16%	97.64%	8047
RIPE NCC	24.39%	78.82%	90.47%	94.48%	96.16%	19821
ARIN	22.05%	67.92%	84.82%	91.48%	94.41%	3578
RIRs	28.07%	80.12%	91.31%	95.16%	96.74%	36976

We count the number of descendant certificates of each RC to estimate the number of CRPs that need to be issued for RC revocation. The result is shown in Table V, which indicates that 90% of RC revocations requires only less than 3 CRPs to be issued. It should be noted that the actual number of CRPs that need to be issued will be less than our estimation because the revoked RC and its descendant certificates may belong to the same owner, and in this case, only one CRP is required.

Certificate Policy Validation. CPs provided by INR holders during the certificate issuance and revocation will be submitted to the CS federation and validated by CS members. If the CP is verified and recognized by the majority of CS members, it will be permanently recorded in the global ledger.

Algorithm 1: CIP Validation

CIP: dict{key=field, value=values}, the CIP that needs to be verified.
INRh_reg: dict{key=INR holder ID, value=registration msg}, all INR holders' registration messages.
CIPs: dict{key=cert_hash, value=CIP info}, all valid CIPs.
CRPs: hash list of all revoked certificates.
Function VALIDATION(*CIP, INRh_reg, CIPs, CRPs*):
  if CIP[VERSION] $\neq 1$ **then**
    return $false$
  if CIP[CERT] $\notin CRPs$ *and* CIP[CERT] $\in CIPs.keys$ **then**
    return $false$
  if CIP[ISSUER] $\notin INRh\_reg.keys$ *or* CIP[SUBJECT] $\notin INRh\_reg.keys$ **then**
    return $false$
  if CIP[CS_SET] $\not\subset INRh\_reg[CIP[SUBJECT]][Trusted\_CSs)]$ **then**
    return $false$
  if CIP[ISSUER_RC] $\in CRPs$ *or* CIP[ISSUER_RC] $\notin CIPs.keys$ **then**
    return $false$
  CIP_{rc} = $CIPs[CIP$[ISSUER]] // Get the CIP of the ISSUER_RC
  if CIP_{rc}[SUBJECT] $\neq CIP$[ISSUER] **then**
    return $false$
  if CIP_{rc}[CERT_T] $\neq RC$ **then**
    return $false$
  if !($CheckValidity(CIP_{rc}, CIP)$) **then**
    return $false$
  KEY = $INRh\_reg[CIP$[ISSUER]][P_K]
  if !($CheckSignature(CIP$, KEY)) **then**
    return $false$
  return [B_NUM,CIP_HASH,CERT];

Algorithm 1 shows the pseudocode of the CIP validation process. The algorithm starts by verifying the certificate status and the identities of the ISSUER and the SUBJECT. Then, it will verify whether the CS_SET in the CIP is a subset of the Trusted_CSs in the SUBJECT registration message. Next, the validity of the ISSUER_RC and whether it is truly owned by the ISSUER will be verified. Since ROAs are leaf nodes and

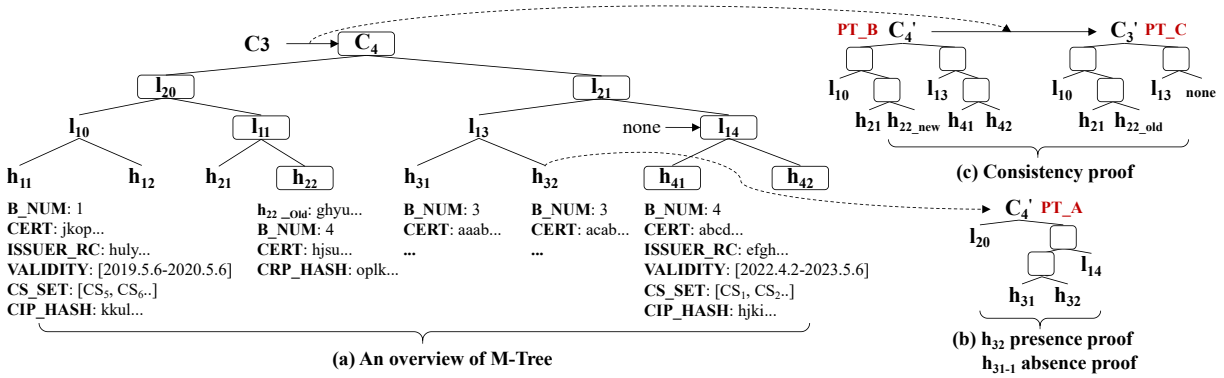


Fig. 7: This picture shows the process of inserting CIPs and CRPs in block 4. Sub-Figure(a) shows the overview of M-Tree, and $\rightarrow$ indicates the change from the C_3 version M-Tree to the C_4 version M-Tree. Nodes with boxes in Sub-Figure(a) show the nodes that have changed. Sub-Figure(b) shows the pruned tree that proves the presence of entry h_{32} and the absence of entry h_{31_1}. Sub-Figure(c) shows the consistency proof of the changes from the C_3 version M-Tree to the C_4 version M-Tree.

sub-certificates cannot be issued by ROAs, when the algorithm detects that the type of `ISSUER_RC` is *ROA*, it will return an error. Finally, the algorithm will verify the CIP's validity and the `ISSUER`'s signature on this CIP. Any verification failure will result in the CIP not being accepted by the CS federation. After CS members reach a consensus, the CS uploading the CIP will return the metadata of this CIP to the INR holder, including the block height (`B_NUM`) of the CIP, the hash of the CIP (`CIP_HASH`), and the hash of the certificate (`CERT`) protected by dRR. The INR holder can use the metadata to confirm the status of its certificate.

We have presented pseudocode for the validity verification process CheckValidity(), the signature verification process CheckSignature(), and the CRP verification algorithms in Appendix A.

B. Monitor

In this section, we introduce the Monitor's core data structure, M-Tree. Then, we describe how Monitors serve INR holders and RPs based on M-Trees.

1) Verifiable Data Structures Maintained by Monitors: The Monitor in dRR accepts queries from INR holders about the existence and status of a specific certificate and provides verifiable CULs to RPs. Therefore, the data structure maintained by the Monitor must be able to provide three proofs:

- **Proof of presence.** Given an element, proof of the existence of the element can be given. It can be used to provide the INR holder with proof of the existence of a specific certificate.
- **Proof of absence.** Given an element, proof of the non-existence of the element can be given. It can be used to provide the INR holder with proof of the non-existence of a specific certificate.
- **Proof of consistency.** Given two versions of the data structure, it can be proven that modifications between log revisions are trustworthy and that the log does not contain non-existing elements or exclude existing ones. It can be used to provide RPs with credible proof of the certificate changes (CULs) between two-time points.

Typically, MHTs [15] are binary trees that contain entries in leaves, while the intermediate nodes and the root of the tree contain the hash of concatenated child nodes. When a new entry is inserted chronologically, a *commitment* (root hash of the tree) of the current tree version will be generated. An MHT can efficiently insert a new entry with $\lceil \log_2 L \rceil$ time overhead, and it can also provide proofs of presence and consistency with size $\lceil \log_2 L \rceil$, while L is the number of leaves[6]. The classical MHT can satisfy our need to provide proof of presence and consistency, but it cannot provide proof of the absence of an entry. We improve the MHT in combination with our specific scenario so that it can also provide proof of absence.

M-Tree. We call the data structure maintained by the Monitor an M-tree. M-Tree is an MHT with leaf nodes containing CP information. We make three design choices to make MHT meet our needs: (1) M-Tree will generate a *commitment* after inserting a block's CIPs and CRPs; (2) the newly added CIPs in one block will be appended into the M-Tree according to the lexicographical order of their certificate hashes; (3) M-tree records the newly added CRP by modifying the CIP entry of the certificate to be revoked in the CRP. Figure 7 gives an overview of the M-Tree, showing the process of inserting two CIPs and one CRP of *block 4* to the M-Tree and the corresponding commitment changes (C_3 to C_4).

Inserting CIPs. As shown in Figure 7(a), h_{41} and h_{42} are new entries added for two CIPs in *block 4*. Each entry contains the necessary content for RPs to obtain the certificate, including the block number `B_NUM` of the CIP and the `ISSUER_RC`, `CERT`, `VALIDITY`, `CS_SET`, and `CIP_HASH` fields in the CIP. h_{41} and h_{42} are inserted in the lexicographic order of their certificate hash values. M-Tree requires $\lceil \log_2 L \rceil$ overhead to insert a new CIP, and the new entry format is as follows:

$$h_{41} = HASH(\texttt{B\_NUM} \,||\, \texttt{ISSUER\_RC} \,||\, \texttt{CERT} \,||\, \texttt{VALIDITY} \,||\, \texttt{CS\_SET} \,||\, \texttt{CIP\_HASH})$$

Updating CRPs. The CRP in *block 4* is the revocation of the h_{22} certificate in *block 2*. The M-Tree will not create a new entry for the CRP but will update h_{22}. The new entry h_{22_new} will include the old h_{22_old}, the block number `B_NUM` of the CRP, and the `CERT` and the `CRP_HASH` fields in the CRP. M-tree also needs $\lceil \log_2 L \rceil$ time overhead to complete the updating of a CRP. The updated entry format is as follows:

[6]We denote the proof size in terms of the number of hashes that need to be computed to verify the proof.

$h_{22\_new} = HASH(h_{22\_old} \,||\, \texttt{B\_NUM} \,||\, \texttt{CERT} \,||\, \texttt{CRP\_HASH})$

Commitments. When the Monitor completes the update of the CIPs and CRPs in one block, it will generate a *commitment* of the current M_Tree version. Each Monitor must publish the metadata of its M-tree historical versions, *i.e.*, the block number and the corresponding *commitment*. The publicly available metadata can be used to verify the consistency of the *commitment* provided by the Monitors. To do this, each Monitor should maintain a *Commitment Update File* that can be accessed via HTTPS. Its specific format can be as follows:

```
<commits, xmlns = "https://.../monitor",
B_NUM = 11 >
    <B_NUM = 11, commit = "abcd...">
    <B_NUM = 10, commit = "bkdk...">
    ...
    <B_NUM = 1,  commit = "klod...">
</commits>
```

2) Serving INR holders: The INR holder may be concerned about whether its CIP has been publicized in the CS federation and the corresponding certificate has been protected by dRR. If so, what is the certificate's status, *i.e.*, whether it has been marked as expired or revoked? The INR holder should provide the tuple $\langle$B_NUM,CERT$\rangle$, which records the block number of the CIP and the hash of the certificate to the Monitor, and the Monitor returns the result along with the proof.

Certificate existence proof. As shown in Figure 7(b), when the INR holder asks whether the certificate in h_{32} $\langle \texttt{B\_NUM} = 3, \texttt{CERT} = acab... \rangle$ exists, the Monitor will return a pruned tree that contains the entry of h_{32} and the hash of h_{31}, intermediate nodes I_{20} and I_{14} to the INR holder. With the pruned tree, the INR holder can easily reconstruct the *commitment* of $C_4^{'}$. Then the INR holder can access the *Commitment Update Files* provided by other Monitors to check the authenticity of $C_4^{'}$. After verification, the INR holder can confirm that its certificate has been protected by dRR.

Certificate absence proof. Supposing that the INR holder is concerned about the existence of the certificate $\langle \texttt{B\_NUM} = 3, \texttt{CERT} = abab... \rangle$. If the certificate exists, it must be between h_{31} and h_{32} according to the lexicographic order. Obviously, it does not exist. Then, as shown in Figure 7 (b), the Monitor will return the pruned tree that contains the entry of h_{31} and h_{32}, and the intermediate nodes of I_{20} and I_{14} to the INR holder. Based on the pruned tree, after confirming that there is no problem with the reconstructed commitment, the INR holder can know that the certificate does not exist.

The service provided by Monitors allows INR holders to monitor the status of their certificates, thereby enabling them to deal with any anomalies that may arise in time.

3) Serving RPs: Since RPs need to synchronize newly added certificates and remove revoked ones periodically, the Monitor can provide RPs with verifiable CULs. Then, the RPs delete revoked certificates and synchronize the new certificates from the designated CS based on CULs to complete their local cache refreshment. To obtain the latest CUL, RPs should submit a tuple $\langle$B_NUM,C$\rangle$, which records the last synchronized block number and the corresponding commitment to the Monitor, and the Monitor feeds back the updated entry along with the integrality proof to RPs.

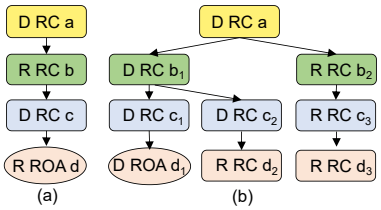


Fig. 8: The certificate chains in a semi-deployed state of dRR. Mark *D* means that the certificate is dRR-protected, while Mark R is not.

Certificate update list. As shown in Figure 7 (c), the RP has completed the synchronization of the first three blocks and now needs to synchronize the *block 4* incrementally. The RP submits $\langle \texttt{B\_NUM} = 3, \texttt{C} = C_3 \rangle$ to the Monitor, and the Monitor will return a pruned tree (*PT_B*) that contains the CUL (the entry of h_{22_new}, h_{41} and h_{42}) and the proof (the hash of h_{21}, the intermediate nodes of I_{10} and I_{13}) to the RP. Based on the pruned tree, the RP needs to perform two verification steps:

1. Verify that $C_4^{'}$ reconstructed from *PT_B* is trusted.
2. Verify that *PT_C* can be deduced from *PT_B*, which can prove that $C_4^{'}$ evolved from C_3. The consistency proof can prove that the updated entry from C_3 to $C_4^{'}$ has only h_{22_new}, h_{41}, and h_{42}, thus proving the integrity of the CUL. To do this, the RP can perform a version rollback of M-Tree, *i.e.*, replace h_{22_new} with h_{22_old} in h_{22_new}'s entry, and delete h_{41} and h_{42} (replace with none) to reconstruct the pruned tree of $C_3^{'}$. Finally, the RPs check whether the reconstructed $C_3^{'}$ is equal to C_3.

After the above verification, the RPs can confirm the integrity and consistency of the CUL provided by the Monitor. Then, RPs can delete revoked certificates and fetch new ones according to the newly added CRPs and CIPs. A newly added certificate may be stored in several CSs; RPs can request any of them to complete the certificate synchronization.

Initializing a new Monitor. We suggest that national/regional institutions or large ISPs can operate Monitors to serve their responsible INR holders and RPs. In the initial phase of dRR, due to the limited number of historical blocks, an entity that wants to run a Monitor can obtain the historical global ledger for building its local M-Tree. After dRR has been running for some time, subsequent new Monitors can download the M-tree from any existing Monitor and compare the M-tree root hash with the commitments provided by other Monitors to confirm data integrity. This approach can avoid the overhead of traversing the historical global ledger while ensuring the integrity of the M-Tree. Then, the Monitor can request newly added blocks to dynamically update its M-Tree.

C. Incremental Deployment

Like traditional RPKI, the incremental deployment of dRR also follows the top-down deployment model. If a certificate wants to be protected by dRR, its parent RC must already be protected by dRR. Two main considerations make it necessary: security and compatibility. As shown in Figure 8 (a), $RC\ c$ is protected by dRR, but $RC\ b$ is not. If $RC\ b$ is maliciously deleted or revoked by the owner of $RC\ a$, the security of $RC\ c$ cannot be guaranteed. Furthermore, when the owner of $RC\ a$ wants to revoke $RC\ b$, the unilateral revocation mechanism of RPKI will conflict with the requirement of dRR that the revocation must be approved by the affected INR holder. Therefore,

as shown in Figure 8 (b), the certificates protected by dRR naturally form unbroken chains. The chain is not necessarily complete (just like the chain of $RC\ a \rightarrow RC\ b_1 \rightarrow RC\ c_2$, without $RC\ d_2$), but it must be from the root certificate of RIR to the lowest dRR deployer.

1) Certificate Management: When a CA has deployed dRR and can sign CIPs for its subordinate INR holders, INR holders are free to choose whether to deploy dRR or not. Therefore, a CA can retire its PP only if all sub-certificates have been protected by dRR. Otherwise, the CA should keep running the PP to serve subordinate INR holders.

For INR holders that do not deploy dRR, their certificate management remains unchanged. Certificates are still stored in the PP of the parent CA and can be unilaterally manipulated by the parent CA.

For INR holders that have deployed dRR, their certificates will not be stored in PPs, and the manifest in the parent CAs' PP will not record their certificates. Revocation of their certificates must be done in the CS federation. As shown in Figure 8 (b), if the owner of $RC\ a$ wants to revoke $RC\ b_1$ which is protected by dRR, it must first complete the recursive revocation of $RC\ c_1$, $RC\ c_2$, and $ROA\ d_1$. The recursive revocation of $RC\ b_1$ will not cause additional harm to $RC\ d_2$, because RPKI allows the unilateral revocation of $RC\ b_1$, which will also invalidate $RC\ d_2$. In contrast, the deployment of dRR by $RC\ c_2$ may indirectly benefit $RC\ d_2$ as it avoids $RC\ d_2$ from being threatened by higher-level CAs (the owners of $RC\ b1$ and $RC\ a$).

2) Certificate Synchronization: When dRR is partially deployed, the RP should complete two synchronization phases in each round of local cache refresh: (1) complete the refresh of dRR-protected certificates according to CULs. Specifically, RPs should fetch the newly added certificates and build certificate chains according to the `ISSUER_RC` field that records the parent RCs. Then, the RPs delete the revoked certificate recorded in the `CERT` field of the CRPs; (2) complete the refresh of RPKI objects (RCs and ROAs without dRR deployed, CRLs, and manifests) through Rsync or RRDP. The RP must traverse all PPs to fetch certificates without dRR deployed. Furthermore, when CUL and CRL conflict, *i.e.*, the hash of a dRR-protected certificate is listed in the CRL of its parent CA, the RP should refuse to revoke this certificate. Both initial synchronization and incremental synchronization of RPs need to fetch all dRR-certificate and other RPKI objects.

V. dRR Analysis

In this section, we give the analysis of dRR, including the dRR property analysis and the security analysis.

A. Properties of dRR

We summarize four core properties of dRR and elaborate on the key insights that lead to these core design decisions. We also discuss the rationale and effectiveness of each property.

Decentralization. dRR addresses the issue of disproportionate division of power between RPKI authorities and INR holders. By decoupling certificate issuance and management, dRR aims to achieve a fair balance of rights between CAs and subordinate INR holders. It empowers INR holders with proactive control over their certificate management, including storage and revocation. This proactive control ensures that INR holders are not solely reliant on RPKI authorities, preventing potential abuse or unilateral actions that could compromise the authenticity of their certificates, such as deletion, corruption, modification, or unauthorized revocation.

Trust flexibility. dRR ensures the freedom of trust for both INR holders and RPs. INR holders are empowered to freely choose trusted CSs to provide certificate hosting services for them, thus preventing their certificates from being compromised by RPKI authorities. Furthermore, dRR enables RPs to choose trusted Monitors to provide them with verifiable CULs. This empowers RPs to verify the integrity and accuracy of the RPKI object views they obtained, which is very important to routing security.

Public auditability. All certificates protected by dRR are publicly auditable. dRR naturally fulfills the need for auditability by letting CSs form a federation. Each CS proactively uploads the fingerprints of the certificates it hosts to the federation so that a widely recognized global ledger of all RPKI certificates and certificate operations can be maintained. Based on this ledger, dRR ensures that all certificates' status can be verified and that the historical RPKI certificates can be audited. Ensuring the transparency of all certificates is important to prevent certificates from being arbitrarily manipulated and hold accountability after the fact.

Robustness and Security. We design dRR to be robust and secure. dRR selects competent candidates to act as CSs to store all RPKI objects. It can significantly enhance the robustness and effectively ensure stable service for tens of thousands of RPs. The truly distributed storage architecture mitigates the risk of a single point of failure in the RPKI Repository. Furthermore, decoupling the repositories from CAs addresses the problem of unlimited expansion of PPs. The federation's access mechanisms further enhance the scalability of the RPKI Repository while significantly reducing the potential for malicious behavior by CS members.

With these core properties, dRR is a more robust, secure, and scalable RPKI Repository, and can proactively defend against attacks from malicious RPKI authorities and better cope with future RPKI deployment.

B. Security Analysis

We discuss the potential attacks to dRR according to the threat model in § II-C, and show how dRR defends against them. We consider the threat from external attackers, especially for the compromise of the private keys. Since the main participants in dRR are INR holders and CSs, we also discuss the threats from both.

Compromised private keys. For an INR holder, the key for signing certificates (RC_K) or signing CPs (CP_K) may be compromised: (1) only RC_K is compromised. For the INR holder acting as a CA, if dRR protect all its sub-certificates and it no longer maintains a PP, the CA will not be threatened. The attacker cannot maliciously sign sub-certificates with valid CIPs nor revoke CA's dRR-protected sub-certificates. If the CA still runs a PP that the attacker can access, the attacker may revoke or sign sub-certificates, which the current RPKI also faces; (2) only CP_K is compromised. Suppose the attacker can also access the INR holder's trusted CS. In this case, the attacker may sign CRPs to maliciously revoke the INR holder's ROAs or those RCs with no sub-certificates and resign the registration message to change CP_K. But the attacker cannot

sign a valid CIP for resource authorization; (3) both keys are compromised. In this extreme case, the attacker can issue malicious sub-certificates, sign CRPs to perform revocations, or change the CP_K. Once CP_Ks are compromised, we expect INR holders to update the `P_K` in the registration message as soon as possible. They should also immediately roll back the CPs to revoke the maliciously signed certificates and recover the original valid ones. In general, dRR is more resilient to key compromise than current RPKI, because for RPKI, when RC_K is compromised, the attacker can issue and revoke certificates at will.

CSs's private keys may also be compromised, the attacker may attempt to upload fake CPs to consume the CS federation or disrupt the consensus process to compromise global ledger's integrity. Since fake CPs cannot be verified by other CSs, the attacker cannot affect the ledger's integrity. Furthermore, if a CS is found to submit a large number of fake CPs, the CS federation can prohibit it from submitting CPs (*i.e.*, ignore all its fake CPs). Once a CS's private key is compromised, the CS can apply to RIRs to have the old registration information deregistered and submit a new CS registration with a new key.

Malicious INR holders. A malicious INR holder may abuse IP prefixes to launch Internet attacks. In this case, dRR's design allows RIRs to enforce the revocation of their IP prefix usage rights. A malicious INR holder may also want to consume the CS federation or claim fake ownership of certificates belonging to others. For consuming the CS federation, a malicious INR holder may upload fake certificates and CPs to exhaust the CS federation (*e.g.*, uploading broken certificates). dRR expects the services provided by CSs to operate in a paid mode, so an attacker with a finite budget will be unwise to exhaust the federation. Furthermore, dRR expects CSs trusted by the INR holder to verify the holder's certificates and CPs in advance. If a CS is found to help malicious INR holders upload fake CPs, it will face the risk of being kicked out of the federation. For claiming fake ownership of certificates, the malicious IRN holder will not succeed, because they cannot provide corresponding valid CIPs that record the signature of issuers.

Malicious CSs. A malicious CS may deliberately not submit the INR holders' CIPs or CRPs to the CS federation. In this case, INR holders can easily confirm whether the CS has provided services for them by asking the Monitor for their certificates' status. A malicious CS may also delete or corrupt the certificates it hosts or submit many fake CPs to consume CS federation. Since dRR allows INR holders to freely choose multiple CSs to host their certificates, RPs can always synchronize with their certificates at one honest CS. Furthermore, if a CS is found to act maliciously, the CS federation can limit the number of CPs it can submit, and five RIRs can also initiate a CS deregistration operation to kick it out of the federation. We envision the penalties for CSs to be agreed upon by the CS federation. A malicious CS may also disrupt the consensus process to affect the global ledger's integrity. For the *Hotstuff* consensus protocol chosen by dRR, as long as the number of malicious nodes does not exceed one-third of the total nodes, the malicious nodes will be isolated and their behaviors will not affect the global ledger's integrity (details on this topic can be found in [56]).

As discussed above, dRR may face several threats, but dRR can resist most of them, and the potential damage from these attacks is mostly limited and recoverable.

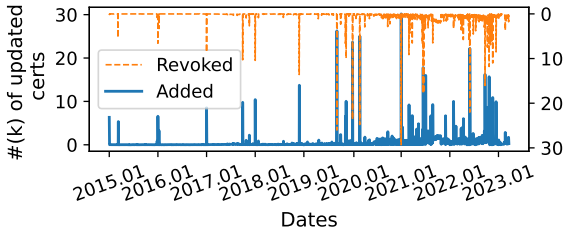


Fig. 9: The number of added and revoked certificates per day from Jan 1, 2015 to Apr 1, 2023.

VI. Evaluating dRR on a Global Testbed

In this section, we implement our prototype of dRR, and evaluate it on a 100-node global testbed.

A. Implementation

We describe the implementation of the CS federation and Monitor and then introduce the real testbed we built for dRR.

CS federation. We use the *Hotstuff* protocol, which has an open-source implementation [25] to accomplish consensus among CS members. Hotstuff [56] is a leader-based Byzantine Fault-Tolerant (BFT) replication protocol for the partially synchronous model. By rotating leaders, this protocol only needs to send a linear number of messages to reach consensus. While providing high throughput for processing operations, it never sacrifices decentralization. The security of Hotstuff is also well-proven (see the paper [56] for more details). We believe Hotstuff is suitable for CS members to achieve consensus on operations, especially on CIP and CRP.

We use *Protocol Buffers 3.16* [20] to define our message format according to the field in the CS and INR holder registration, CIPs, and CRPs. The hash value in our CPs and registration message is done with the sha-256 algorithm, and the signature algorithm is implemented with sha-256 RSA. Then, we follow the detailed design in § IV-A to implement the CS with 512 lines of Golang. Each CS supports uploading and verifying CS operations and CPs, and will proactively push the CPs confirmed by the CS federation to Monitors it serves. In addition, each CS runs the Hotstuff protocol and participates in the CS federation. Among multiple Hotstuff versions, we choose the *Chained Hotstuff*, and the leader in each consensus round is selected in a round-robin way.

Monitor. For each of our Monitors, we implement the M-Tree data structure using 328 lines of Python. The hash value calculation of M-Trees is based on sha-256 algorithm.

Deployment on a Global Testbed. We build an 100-node testbed that spans 15 countries or regions[7] to fully evaluate dRR. Among them, 50 nodes are CS servers, and the remaining are Monitor servers. For different CS federation scales, we evenly distribute the CS nodes among all our regions. The geographical distribution of the Monitor nodes also follows that of the CSs. Each of our CSs and Monitors runs on a Ubuntu 18.04 server with 4 cores and 8GB RAM. We allocate a peak bandwidth of 200Mbps to each CS node, and 100Mbps to each Monitor node.

[7]Regions and countries include Australia (7), India(7), China(7), Hongkong,China(7), Singapore(7), Japan(7), Germany(7), Britain(7), Silicon Valley,USA(7), Virginia,USA(7), Indonesia(6), Malaysia(6), Dubai,UAE(6), Philippines(6), and Thailand(6). All on Alibaba Cloud.

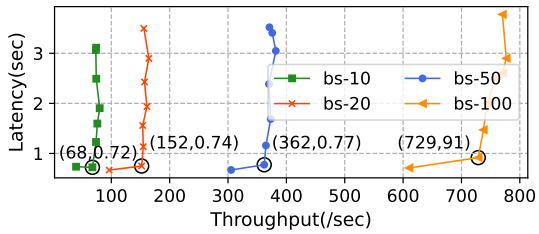


Fig. 10: The throughput and average latency under different batch sizes. Data in the circle represents the maximum throughput and the corresponding average latency.

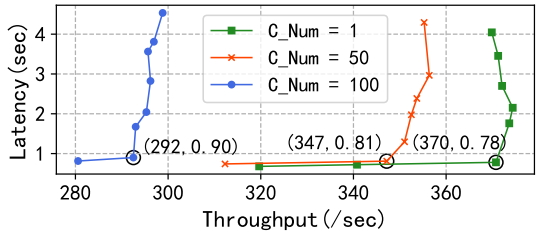


Fig. 11: The throughput and average latency under different numbers of revoked certificates in one CRP.

B. Evaluation

In the following, we start by setting up a baseline performance of the current RPKI Repository by collecting a public data set. With this baseline, we proceed to evaluate the performance of both the CS federation and the Monitors.

Current certificate update frequency. We can estimate the required performance of the RPKI Repository by the frequency of current RPKI certificate renewals. To obtain this statistic, we download daily verified RCs and ROAs maintained by RIPE NCC [47] and evaluate the number of daily updated certificates over the past eight years. We generate a hash snapshot for the daily RPKI certificates and obtain the number of newly added and revoked certificates by calculating the new and missing hash elements every day. Figure 9 shows our results. We observe that the number of newly added and revoked certificates per day has increased over time, especially after 2020, which is closely related to the rising deployment rate of RPKI. On average, the number of newly added and revoked certificates per day is below 5,000, but the peak also reaches 60,000. The cause of this peak, we suspect, is the key rollover performed by RPKI CAs.

Evaluating CS federation. We focus on two performance metrics: the *throughput* of the CS federation and the *latency* of a CP from submission to confirmation. We first perform extensive experiments to obtain appropriate parameters for dRR. We then evaluate the scalability of dRR using the selected parameters.

Since the throughput of Hotstuff is closely related to the number of CPs (batch size) that can be contained in one block, we test the throughput and latency of our system with the batch size of 10, 20, 50, and 100, respectively. Specifically, given the batch size, we vary the CIP sending rate for each CS until the system is saturated. Each experimental data is the average result of 20 identical experiments. It can be seen from Figure 10 that as the batch size increases, the throughput also increases. For each given batch size, when the CIP sending rate exceeds the maximum throughput, the latency will increase significantly, because some CIPs may wait in the queue to be processed; and the throughput increases little or even decreases. Balancing between throughput and latency, we fix the batch size at 50 in the following experiments.

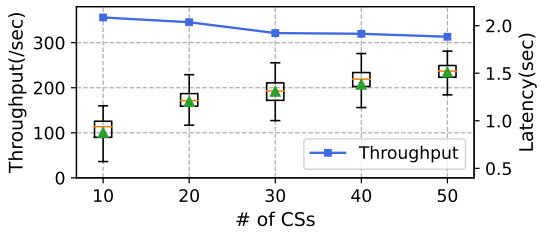


Fig. 12: The maximum throughput and the corresponding latency distribution of the system under different CS scales. Candlesticks show the maximum and minimum latency and the average latency (green triangle).

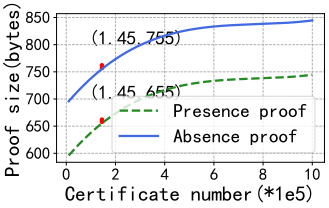


Fig. 13: The size of proofs provided to INR holders under different certificate scales. The red dot indicates the current certificate scale.

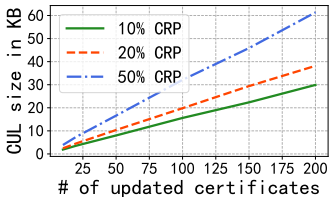


Fig. 14: The size of CULs provided by Monitors to RPs under different number of updated certificate and CRP ratios.

The size of CRP also affects the throughput and latency. To understand its impact, we adjust the number of revoked certificates in each CRP's `CERT_SET` field to evaluate the performance of the federation in processing CRPs. We fix the batch size to 50 and evaluate the throughput and latency when the certificate number is 1, 50, and 100, respectively. As shown in Figure 11, we observe that the throughput drops by 6% when the certificate number is less than 50 but drops by 20% when the number rises to 100. Considering the system should maintain high throughput, we expect to limit the number of revoked certificates in each CRP to 50.

Finally, with the parameters for dRR chosen, we expand the number of CSs to evaluate the system's scalability. Under different numbers of CS, we fix the batch size to 50 and let CSs randomly send CIPs and CRPs to measure the system's maximum throughput and the corresponding latency distribution. As shown in Figure 12, with the expansion of the scale of the CS federation, the system's throughput decreases, and the latency increases, which follows our expectation. We would like to highlight that when the number of CSs reaches 50, the throughput can exceed 310 per second, which means 26.78 million CPs can be processed per day. This number is 450 times higher than the current maximum RPKI certificate update frequency in Figure 9. At such high throughput, we observe that the CP can still be confirmed within 2 seconds.

Evaluating Monitor. We examine two performance metrics: *the size of proofs and CULs* provided by Monitors and *the latency* for Monitors to get a new block and update the corresponding M-Trees.

Figure 13 shows the proof size provided by Monitors to INR holders. The proof size grows logarithmically with the number of total certificates. At the current RPKI certificate

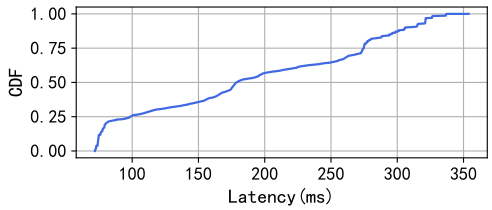


Fig. 15: The latency distribution of Monitors.

scale, both presence and absence proof sizes are within 1 KB. We also evaluate the size of the verifiable CUL provided by the Monitor to RPs under the different number of updated certificate and CRP ratios (10%, 20%, and 50%). As shown in Figure 14, the size of the CUL is positively correlated with the number of updated certificates. For dRR, the average size of the verifiable update information for one certificate is between 100 B and 300B. For current RPKI, both manifest and CRP are about 3KB in size, and more than 97% of CAs currently have less than ten certificates in their PPs, which means that the average size of one certificate's update information will occupy 300B to 3KB. Therefore, the size of CULs provided by dRR will not consume more bandwidth than the manifests and CRLs provided by CAs.

In dRR, each CS will proactively push the new block to the Monitors it serves so that Monitors can extract the CIPs and CRPs in the block and update their M-Trees. We let a CS server in Silicon Valley serve 50 Monitors distributed in 15 countries (regions). We keep the CS federation running and let the CS continuously pushes 10,000 new blocks to 50 Monitors. Figure 15 shows the latency distribution (from sending the block by the CS to the Monitor completing the M-Tree update about this block) of 500,000 blocks received by 50 Monitors. It can be seen that when the CS federation confirms a new block, the Monitor can complete the update of its M-Tree for this block within 500 milliseconds (ms), which means that the latency caused by the Monitor is less than 500 ms.

Finally, we estimate *the additional closed-loop time overhead* introduced by dRR.

dRR only changes the certificate management, the certificate issuance on the CA side and the certificate validation on the RP side remain the current way. As shown in Figure 16, with dRR, CS federation needs to publish CPs after uploading certificates, which will cause a delay of no more than 2 seconds. Additionally, Monitors will introduce a delay of no more than 1 second for obtaining newly added CPs and updating their M-Trees. For RPs to refresh their local caches, we expect that dRR does not cause more latency. Because in dRR, RPs do not need to traverse all PPs to check the updating information but only need to obtain CULs from one Monitor and fetch RPKI data from CSs. Therefore, the additional closed-loop time overhead introduced by dRR will be no more than 3 seconds. Note that, currently, the bottleneck of the time overhead is the time required for signing certificates and the time interval for RPs to refresh local caches, which are highly variable and take about tens of minutes to several hours [18], [33], [9]. Furthermore, with dRR, the certificate update throughput will not decrease because the throughput achieved by dRR is 450 times higher than the current maximum update frequency. In summary, we believe that the additional overhead introduced by dRR is negligible.

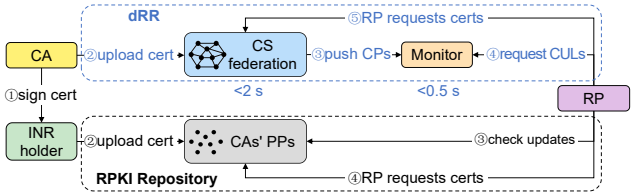


Fig. 16: The workflow of dRR and current RPKI which highlights the possible latency introduced by dRR.

VII. Discussion

Modification required for RPKI RFCs. As described above, dRR implements a new RPKI Repository through two entities, the CS federation and Monitor. Therefore, the RFCs that need to be modified primarily involve RFC 6481 [26], which defines the profile of the resource certificate repository structure, and RFC 8182 [10], which defines the RP synchronization protocol (RRDP). RFC 6481 needs to be updated to describe the architecture of the dRR CS federation. Similarly, RFC 8182 also needs to be updated to describe the Monitor and the format of CUL. Additionally, the section in RFC 8897 [16] (involving requirements for RP software) that describes how RPs fetch and cache RPKI objects also needs to be updated. Note that the other more than 40 RFCs related to RPKI's core mechanisms, such as certificate issuance and validation, remain unchanged. We believe that the changes required by dRR represent only a small portion of the overall RPKI design, while the security benefits introduced by it are significant.

The deployment of hosted RPKI. To support the early RPKI adopters in deploying ROAs, RIRs introduced hosted RPKI to assist them in signing and managing certificates. In dRR, entities that originally deploy hosted RPKI can still have RIRs assist them in signing sub-certificates and CPs, but their certificates can be hosted in the trusted CSs. Furthermore, we expect INR holders can entrust the sub-certificate or CP signing right to the CSs they trust, not just RIRs. As a result, dRR provides a non-single source of trust for ROA deployers. However, the hosted model is insecure, and the potential risks it may bring are similar to "Compromised private keys" (as described in § V-B). Therefore, we expect INR holders to take responsibility for managing their certificates and CPs.

Potential benefits of dRR. RPKI requires CAs to periodically refresh their private keys for signing sub-certificates to ensure certificate security [27]. Each key rollover of a CA means that all sub-certificates must be re-signed, which undoubtedly burdens RPs to refresh their local caches. In dRR, the issuance and revocation of certificates are naturally protected by CPs. Therefore, dRR does not require CAs to frequently perform key rollover but requires them to periodically refresh the key for signing CPs. Since CP key updating is completed by CS members and is nontransparent to RPs, it imposes no additional burden on RPs. Additionally, based on verifiable CULs, dRR can also resist mirror world attacks from RPKI authorities [23], which involves showing different views to different RPs and causing conflicting routing decisions. In current RPKI, RPKI authorities can also compromise RCs and ROAs by manipulating manifests or CRLs [32]. For example, they can hide their certificates by not recording certificate hash entries in the manifest. Since dRR no longer relies on manifests and CRLs to verify the certificate validity, dRR also provides defense against such attacks.

Limitations. dRR only protects the existing certificates and

does not address the issue of CAs unwilling to sign new certificates for INR holders or modify old certificates according to INR holders' wishes. dRR also does not actively prevent the CA from reallocating its resources. For this threat, INR holders can use the INR conflict detection algorithm [58] to verify whether there is a duplicate allocation of their obtained INRs. dRR does not focus on attacks against the Internet protocols on which RPKI depends, such as disabling RPKI by attacking the DNS system that RPKI relies on [24]. We believe these issues can be addressed by dedicated security mechanisms.

VIII. Related Work

Since the standardization of RPKI, both the academic and industrial communities have been dedicated to enhancing the security and scalability of RPKI.

Consent [23] and DRPKI [52] are schemes that aim to address the problem of malicious RPKI authorities. Consent [23] has developed a tool to detect changes in the RPKI object that can cause legitimate status changes in BGP updates. However, the detector can only detect attacks after the fact, and it is difficult to differentiate between legitimate operations by authorities and malicious behavior. Consent has also designed a new signed RPKI object *.dead*, which serves as confirmation that the INR holder consents to the RPKI authority revoking their RPKI objects. Although the new object can prevent malicious revocations by RPKI authorities, it cannot prevent them from maliciously deleting or corrupting certificates.

DRPKI means distributed RPKI. It proposes that each RPKI object can be collaboratively signed by five RIRs through threshold signature, thus limiting the unilateral power of a single authority. Since DRPKI gives all rights to RIRs and requires all RPKI objects to be signed by them, it breaks the current RPKI certificate issuance hierarchy and may further intensify the centralization of power in RPKI. Therefore, DRPKI is only applicable to hosted RPKI, and it is not compatible with the emerging trend of delegated RPKI. Additionally, RFC 8211 [32] and the paper [13] also analyze potential threats from RPKI authorities, but they do not propose any solutions.

Since Rsync imposes a significant CPU and memory resource overhead on server sides, RRDP is standardized by IETF to offload the computation burden on PP servers. This is accomplished by utilizing HTTPS-based Snapshots and Delta Files, thereby allowing RPKI Repository to serve more RPs and accommodate the further deployment of RPKI ROV. However, RRDP cannot solve the scalability problem caused by the increase in the number of PPs. As the number of PPs increases, it will have an impact on RPs' real-time awareness of global RPKI data changes.

In summary, with these efforts, RPKI can become more secure and scalable. However, these solutions have limitations and cannot fully solve the problems that we are concerned about. These limitations motivated the design of dRR.

IX. Conclusion

In this paper, we present a comprehensive data-driven analysis of threats to RPKI, which involves a worldwide survey and large-scale measurement. Our analysis uncovers three key problems that pose significant threats to RPKI. Based on these insights, we propose dRR, a new RPKI-compatible Repository architecture to balance the rights between RPKI authorities and INR holders while enhancing the robustness, security, and scalability of the current RPKI Repository. To achieve this, dRR introduce two new components: the CS federation and Monitors. We implement and evaluate a prototype of dRR on a global testbed in the real Internet environment. Our experiments show that the new security features of dRR introduce minimal overhead. Although dRR cannot solve all the threats to RPKI, we also see an urgent need to make the RPKI Repository more secure, robust, and scalable. dRR is a concrete first step towards this goal.

In the future, we will consider enabling Monitors to proactively push new CULs to RPs while retrieving newly added certificates from CSs and delivering them to RPs. In this way, PRs only need to establish a connection to a single entity and can avoid the periodic refresh of their local caches. This enables RPs to obtain RPKI update data more timely.

Acknowledgment

We thank our shepherd and the anonymous reviewers for their thoughtful comments. Li Chen is the corresponding author. This work is supported by the National Key R&D Program of China (2022YFB3104800) and in part supported by NSFC under Grant 62132011.

References

[1] AFRINIC, "African network information centre," https://afrinic.net/.

[2] "Afrinic membership," https://afrinic.net/membership.

[3] APNIC, "Apnic membership," 2023, https://www.apnic.net/get-ip/apnic-membership/.

[4] "Asia-pacific network information centre," 2023, https://www.apnic.net/.

[5] ARIN, "American registry for internet numbers," https://www.arin.net/.

[6] "Arin membership," 2023, https://www.arin.net/participate/oversight/membership/.

[7] R. Austein, G. Huston, S. Kent, and M. Lepinski, "Manifests for the resource public key infrastructure (rpki)," *RFC6486*, vol. 2, 2012.

[8] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 265–276, 2007.

[9] A. Band and the RPKI Community, "The rpki documentation," *RPKI Community*, 2022.

[10] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein, "The rpki repository delta protocol (rrdp)," Tech. Rep., 2017.

[11] R. Bush, "Origin validation operation based on the resource public key infrastructure (rpki)," *IETF RFC7115 (January 2014)*, 2014.

[12] cloudflare, "Octorpki," 2023, https://github.com/cloudflare/cfrpki.

[13] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg, "On the risk of misbehaving rpki authorities," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile," Tech. Rep., 2008.

[15] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *USENIX security symposium*, 2009, pp. 317–334.

[16] S. K. D. Ma, "Requirements for resource public key infrastructure (rpki) relying parties," *RFC8897*, 2020.

[17] T. Eyes, "Twitter outage analysis: March 28, 2022," 2022, https://www.thousandeyes.com/blog/twitter-outage-analysis-march-28-2022.

[18] R. Fontugne, A. Phokeer, C. Pelsser, K. Vermeulen, and R. Bush, "Rpki time-of-flight: Tracking delays in the management, control, and data planes," in *International Conference on Passive and Active Network Measurement*. Springer, 2023, pp. 429–457.

[19] Y. Gilad, O. Sagga, and S. Goldberg, "Maxlength considered harmful to the rpki," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 101–107.

[20] google, "Protocol buffers," 2022, https://developers.google.com/protocol-buffers.

[21] Google, "Working together to detect maliciously or mistakenly issued certificates," 2023, https://certificate.transparency.dev/.

[22] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 803–818.

[23] E. Heilman, D. Cooper, L. Reyzin, and S. Goldberg, "From the consent of the routed: Improving the transparency of the rpki," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 51–62.

[24] T. Hlavacek, P. Jeitner, D. Mirdita, H. Shulman, and M. Waidner, "Behind the scenes of rpki," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1413–1426.

[25] "Hotstuff," 2022, https://github.com/relab/hotstuff.

[26] G. Huston, R. Loomans, and G. Michaelson, "Rfc 6481: A profile for resource certificate repository structure. internet engineering task force (ietf), 2012," 2012.

[27] G. Huston, G. Michaelson, and S. Kent, "Certification authority (ca) key rollover in the resource public key infrastructure (rpki)," Tech. Rep., 2012.

[28] G. Huston, G. Michaelson, C. Martinez, T. Bruijnzeels, A. Newton, and D. Shaw, "Resource public key infrastructure (rpki) validation reconsidered," Tech. Rep., 2018.

[29] IETF, "The internet engineering task force," 2022, https://www.ietf.org/.

[30] JAPNIC, "Service outage: Roaweb and rpki repository (resolved)," 2020, https://www.nic.ad.jp/en/topics/2020/20200521-01.html.

[31] "Service outage: Disk full caused lost roa validity," 2022, https://www.nic.ad.jp/en/topics/2022/20220202-01.html.

[32] S. Kent and D. Ma, "Adverse actions by a certification authority (ca) or repository manager in the resource public key infrastructure (rpki)," *IETF, Fremont, CA, USA, RFC*, vol. 8211, 2017.

[33] J. Kristoff, R. Bush, C. Kanich, G. Michaelson, A. Phokeer, T. C. Schmidt, and M. Wählisch, "On measuring rpki relying parties," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 484–491.

[34] D. R. Labs, "rsync," 2023, https://github.com/dragonresearch/rpki.net.

[35] N. Labs, "Routinator," 2021, https://github.com/NLnetLabs/routinator.

[36] LACNIC, "African network information centre," https://lacnic.net/.

[37] "Lacnic membership," 2023, https://www.lacnic.net/1008/2/lacnic/how-to-become-a-member.

[38] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," RFC 6480, Feb. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6480.txt

[39] D. Ma, D. Mandelberg, and T. Bruijnzeels, "Simplified local internet number resource management with the rpki (slurm)," Tech. Rep., 2018.

[40] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 31–42.

[41] México, "Fortrpki," 2023, https://github.com/NICMx/FORT-validator.

[42] R. NCC, "Rsync rpki repository downtime," 2020, https://www.ripe.net/support/service-announcements/rsync-rpki-repository-downtime.

[43] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 35–48.

[44] L. Qin, L. Chen, D. Li, H. Ye, and Y. Wang, "Understanding rov deployment in the real world and why manrs action 1 is not followed," in *Network and Distributed System Security Symposium (NDSS)*, 2024.

[45] L. Qin, D. Li, R. Li, and K. Wang, "Themis: Accelerating the detection of route origin hijacking by distinguishing legitimate and illegitimate {MOAS}," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 4509–4524.

[46] "Ripe ncc membership," 2023, https://www.ripe.net/participate/member-support.

[47] "Ripe rpki dataset," 2021, https://ftp.ripe.net/rpki/.

[48] "Routing history," 2023, https://stat.ripe.net/docs/02.data-api/routing-history.html.

[49] "Ripe network coordination centre," 2023, https://www.ripe.net/.

[50] "Rpki-prover," 2023, https://github.com/lolepezy/rpki-prover.

[51] V. Shoup, "Practical threshold signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 207–220.

[52] K. Shrishak and H. Shulman, "Privacy preserving and resilient rpki," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[53] S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson *et al.*, "Internet x. 509 public key infrastructure (pki) proxy certificate profile," RFC 3820 (Proposed Standard), Tech. Rep., 2004.

[54] "Validator3," 2023, https://github.com/RIPE-NCC/rpki-validator-3.

[55] K. van Hove, J. van der Ham, and R. van Rijswijk-Deij, "Rpkiller: Threat analysis from an rpki relying party perspective," *arXiv preprint arXiv:2203.00993*, 2022.

[56] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[57] Y. Zhang, S. Setty, Q. Chen, L. Zhou, and L. Alvisi, "Byzantine ordered consensus without byzantine oligarchy," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 633–649.

[58] H. Zou, D. Ma, Q. Shao, and W. Mao, "The horizontal inr conflict-detection algorithm: Revealing inr reallocation and reauthorization in rpki," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 459–465.

APPENDIX A
CERTIFICATE POLICY VERIFICATION ALGORITHM

In the following, we present pseudocode for the validity verification $CheckValidity()$, the signature verification $CheckSignature()$, and the CRP verification algorithms.

Algorithm 2: Check `VALIDITY` field in CIP

```
1  CIP_rc: dict{key = field, value = values} // the CIP of the
   ISSUER_RC
2  CIP: dict{key=field, value=values}
3  Function CheckValidity(CIP_rc, CIP):
4      V_rc = CIP_rc[VALIDITY]
5      V = CIP[VALIDITY]
6      if V.notBefore ≥ V.notAfter then
7          return false
8      if V.notAfter ≥ Time.now then
9          return false
10     if V.notBefore < V_rc.notBefore or V.notAfter >
         V_rc.notAfter then
11         return false
12     return true;
```

Algorithm 2 shows the verification process of the `VALIDITY` field in CIP. If the validity period of the protected certificate is earlier than the current time or outside the validity period of the `ISSUER_RC`, the algorithm will return an error. Algorithm 4 verifies the signatures of the CIP and CRP. The algorithm first obtains the hash and signature of the CP, then calculates whether the hash is correct, and finally decodes the signature.

Algorithm 3 shows the CRP validation process in pseudocode. *dRR _Tree* maintains all certificates currently protected by dRR, *i.e.*, certificates that have CIPs and have not been revoked. The algorithm first verifies whether the certificate to be revoked in the `CERT_SET` field is the currently protected certificate. Then, the algorithm gets the certificate revocation method. If the `R_M` field is *rir*, the algorithm

will further verify the identity of the CRP_ISSUER field, and obtain the P_K of *All-RIRs* to verify the signature (line 12˜17). After all the verification is done, the certificate in the CERT_SET field and their descendant certificates will be marked as revoked (line 18˜22). If the certificates are revoked by the INR holder, the algorithm will first verify the signature of the INR holder (line 25˜28). Then, the algorithm verifies whether the real owner of each revoked certificate is the CRP_ISSUER and whether it has descendant certificates. If the revoked certificate or its descendant certificate does not belong to the CRP_ISSUER, the algorithm will return an error. Otherwise, the certificate and its descendant certificates will be revoked (line 29˜38).

Algorithm 3: CRP Validation

```
1  CRP: dict{key=field, value=values}
2  INRh_reg: dict{key=INR holder ID, value=registration msg}, all INR
   holders' registration messages.
3  CS_reg: dict{key=CS ID, value=registration msg}, all CSs' registration
   messages.
4  CIPs: dict{key=cert_hash, value=CIP info}, all valid CIPs.
5  dRR _Tree: dict{key=cert_hash, value=descendant certificate list}, the
   current dRR-protected certificates tree.
6  Function VALIDATION(CRP, INRh_reg, CS_reg , CIPs, dRR _Tree):
7      if CRP[VERSION] ≠ 1 then
8          return false
9      if CRP[CERT_SET] ⊄ dRR_Tree.keys then
10         return false
11     if CRP[R_M] == rir then
12         if CRP[CRP_ISSUER] ≠ All−RIRs then
13             return false
14         else
15             KEY = (CS_reg[All−RIRs][P_K])
16         if !(CheckSignature(CRP,KEY)) then
17             return false
18         else
19             for cert in CERT_SET do
20                 for subcert in dRR_Tree[cert] do
21                     Revoke(subcert)
22                 Revoke(cert)
23             return [B_NUM,CRP_HASH]
24     else
25         subject = CRP[CRP_ISSUER]
26         KEY = (INRh_reg[subject][P_K])
27         if !(CheckSignature(CRP, KEY)) then
28             return false
29         for cert in CRP[CERT_SET] do
30             if CIP[cert][SUBJECT] ≠ subject then
31                 return false
32             else
33                 Revoke(cert)
34             for subcert in dRR_Tree[cert] do
35                 if subcert[SUBJECT] ≠ subject then
36                     return false
37                 else
38                     Revoke(subcert)
39         return [B_NUM,CRP_HASH]
```

APPENDIX B
THE ABBREVIATIONS USED IN THE PAPER

In the following, we present the abbreviations used in the paper, including the standard RPKI-related abbreviations in Table VII and the abbreviations specifically defined in dRR in Table VI.

Algorithm 4: Check the signature on the CP

```
1  CP: dict{key=field, value=values} // the certificate policy: a
   CIP or a CRP
2  KEY: the key used to sign the certificate policy
3  Function CheckValidity(CP, KEY):
4      CP_hash = NULL
5      CP_sig = NULL
6      if CP.type == CIP then
7          CP_hash = CP[CIP_HASH]
8          CP_sig = CP[CIP_SIG]
9      else
10         CP_hash = CP[CRP_HASH]
11         CP_sig = CP[CRP_SIG]
12     if HASH(CP) ≠ CP_hash then
13         return false
14     if Decode(CP_sig,KEY) ≠ CP_hash then
15         return false
16     return true;
```

TABLE VI: The list of abbreviations defined in dRR

The abbreviation	The full form
dRR	decentralized RPKI Repository
CS	Certificate Server
CP	Certificate Policy
CUL	Certificate Update List
CIP	Certificate Issuance Policy
CRP	Certificate Revocation Policy

TABLE VII: The list of standard abbreviations

The abbreviation	The full form
INR holder	Internet Number Resource holder
RP	Relying Party
pp	Publication Point
ROA	Route Origin Authorization
CA	Certification Authority
RC	Resource Certificate
RIR	Regional Internet Registry
NIR	National Internet Registry
ISP	Internet Service Provider
ROV	Route Origin Validation
SIA	Subject Information Access
CRL	Certificate Revocation List
RRDP	RPKI Repository Delta Protocol

APPENDIX C
THE MEASUREMENT OF RPKI REPOSITORY

We utilize 20,000 globally distributed DNS resolvers to resolve the domain name of 61 RRDP files, with the aim of finding the CNAME and all IP address records for each PP. Then, we use the IP-to-AS mapping information maintained by RIPE NCC [48] to map the IP addresses to ASNs. We try to determine whether the services are hosted on CDNs from the following aspects: (1) Whether the domain name and CNAME record contain information about CDN service providers; (2) The number of IP addresses returned by DNS resolvers and the geographical distribution of these IP addresses; (3) Whether the HTTPS request headers contain information about mainstream CDN providers; (4) The latency of accessing RRDP files from different geographic locations around the world. Typically, CDN service providers will display their information in the domain name and CNAME record, such as including "cdn.cloudflare.net," or specify the CDN in the X-Via-CDN field, cache status field, or Server field in the HTTPS headers. In addition, if CDN acceleration is used, the latency of requesting HTTPS services from different geographic locations around the world will be relatively low.

TABLE VIII: The measurement results of 61 repository publication points.

RRDP domain name (CNAME)	IPv4	IPv6	ASN	Silicon Valley (ms)	Frankfurt (ms)	Sao Paulo (ms)	Bombay (ms)	HTTPs header	CDN	Affiliation
sakuya.nat.moe	141.193.21.22	2602:feda:4:dead:216:3eff:fec7:efab	AS46997	189	222	186	239	nginx	-	NATO lab (organization)
rrdp-rps.arin.net	199.5.26.148 199.71.0.148 199.212.0.148	2001:500:13::148 2001:500:31::148 2001:500:a9::148	AS394018 AS393220 AS393225	-	-	-	-	nginx	-	ARIN (RIR)
rrdp.twnic.tw	103.235.88.190	2406:da14:7c1:9201:34e6:8d27:69af:4b87	AS16509	103	250	377	139	nginx	-	TWNIC (Region)
rrdp.taaa.eu	62.109.150.6	2001:1ab0:7e1e:150::6	AS29134	159	8.1	221	131	nginx	-	-
rrdp.sub.apnic.net	203.119.101.91	2001:dd8:9:2::101:91	AS4608	164	297	332	154	Apache	-	APNIC (RIR)
rrdp.rp.ki (r.rp.ki)	5.161.43.0	2a01:4ff:f0:9fe::1	AS213230	68	96	120	289	nginx	-	-
rrdp.roa.tohunet.com	104.21.68.185 172.67.197.210	2606:4700:3030::6815:44b9 2606:4700:3031::ac43:c5d2	AS13335	3.3	1.0	113	71	cloudflare	cloudflare	-
rrdp.ripe.net (rrdp.ripe.net.akamaized.net)	104.18.8.176 104.18.9.176	2606:4700::6812:8b0 2606:4700::6812:9b0	AS13335	1.8	1.0	2.1	1.3	cloudflare	cloudflare	RIPE NCC (RIR)
rrdp.paas.rpki.ripe.net (rrdp.paas.rpki.ripe.net.cdn.cloudflare.ne)	104.18.12.188 104.18.13.188	2606:4700::6812:cbc 2606:4700::6812:dbc	AS13335	2.2	1.0	2.1	1.3	cloudflare	cloudflare	-
rrdp.lacnic.net	200.3.14.186	2001:13c7:7002:4128::186	AS28001	60.0	193	134	324	nginx	-	LACNIC (RIR)
rrdp.krill.cloud (prod-ps.krill.cloud)	64.227.73.107	2a03:b0c0:2:d0::126b:3001	AS14061	138.0	13	206	132	nginx	-	-
rrdp.arin.net	199.71.0.149 199.5.26.149 199.212.0.149	2001:500:13::149 2001:500:31::149 2001:500:a9::149	AS393220 AS394018 AS393225	-	-	-	-	nginx	-	ARIN (RIR)
rrdp.apnic.net (rrdp.apnic.net.cdn.cloudflare.net)	104.18.235.68 104.18.236.68	2606:4700::6812:eb44 2606:4700::6812:ec44	AS13335	2.6	0.9	2.3	1.2	cloudflare	cloudflare	APNIC (RIR)
rrdp.afrinic.net	196.216.2.29	2001:42d0:0:201::29	AS33764	337	218	361	410	nginx	-	AFRINIC (RIR)
rpki-test-repo.net.kagl.me	104.37.40.238	-	AS970	24	145	190	246	nginx	-	Keaton Alexander Guger (individual)
rpki-rrdp.us-east-2.amazonaws.com (d27hng7mrok6ld.cloudfront.net)	108.138.246.4 108.138.246.98 108.138.246.62 108.138.246.15	-	AS16509	1.3	1.5	8.2	1.4	AmazonS3	Amazon	-
rpki-rrdp.mnihyc.com (rpki-rrdp.mnihyc.com.cdn.cloudflare.net)	104.21.23.24 172.67.208.113	2606:4700:3035::6815:1718 2606:4700:3037::ac43:d071	AS13335	1.8	0.9	113	57	cloudflare	cloudflare	-
rpki-repository.nic.ad.jp	192.41.192.213	2001:dc2:1000:8000::2	AS2515	106	268	289	140	Apache	-	JPNIC (Country)
rpki-repo.registro.br	200.160.2.50	2001:12ff:0:2::50	AS22548	179	187	1.8	321	nginx	-	Brasil (Country)
rpki-publication.haruue.net	172.67.146.105 104.21.33.146	2606:4700:3037::6815:2192 2606:4700:3032::ac43:9269	AS13335	3.2	1.0	113	69	cloudflare	cloudflare	-
rpkica.mckay.com	51.75.161.87	-	AS16276	149	13.5	198	123	Apache	-	-
rpki1.terratransit.de	77.237.224.23	2a01:6f0:101:17::1	AS42366	155	6.7	216	115	nginx	-	TerraTransit AG (ISP)
rpki1.rpki-test.sit.fraunhofer.de	141.12.174.14	-	AS28714	152	1.8	210	115	Apache	-	Fraunhofer (ISP)
rpki.zappiehost.com	23.160.160.185	2602:fd92:a85::	AS39618	45	121	136	245	nginx	-	-
rpki.xindi.eu	185.173.16.136	2a0b:2f00:0:234::136	AS48112	178	27	234	143	Apache	-	XINDI Networks (ISP)
rpki.tools.westconnect.ca	23.129.32.54	2602:fd60:11::54	AS53356	2.2	143	173	256	Apache	-	-
rpki.telecentras.lt	86.38.8.173	-	AS15419	-	-	-	-	nginx	-	-
rpki.sailx.co	208.82.103.214	-	AS396097	2.5	140	177	261	nginx	-	sail.X (Company)
rpki.roa.net	62.133.35.21	2a09:0:8::21	AS3214	147	4.3	194	129	nginx	-	xTom GmbH (ISP)
rpki.rand.apnic.net	203.133.248.19	2401:2000:6660::19	AS4608	167	292	328	160	nginx	-	APNIC (RIR)
rpki.qs.nu (rpki-repo.qs.nu)	95.216.140.6	2a01:4f9:c010:cbd::1	AS24940	168	28	221	166	nginx	-	-
rpki.pedjoeang.group	104.167.215.189	-	AS60841	40	121	148	268	Caddy	-	-
rpki.owl.net	5.161.48.81	2a01:4ff:f0:d40::1	AS213230	68	92.2	121	306	nginx	-	-
rpki.multacom.com	198.211.9.2	-	AS35916	9	144	178	243	Apache	-	MULTACOM (ISP)
rpki.luys.cloud (ias.agent.idns.cloud)	108.166.211.224 45.145.75.91 185.249.221.37 185.249.221.38 185.249.221.39	-	AS35916 AS201106 AS40065	160	159	206	107	openresty	-	-
rpki.kitten.network	104.167.215.30	-	AS60841	46	120	151	263	KittenSystems	-	-
rpki.folf.systems	45.79.192.20	2600:3c02::f03c:92ff:fe46:72f0	AS63949	60	103	122	235	nginx	-	-
rpki.ezdomain.ru	-	-	-	-	-	-	-	-	-	-
rpki.cnnic.cn	218.241.105.61 42.83.145.17	-	AS24151	145	215	286	186	Apache	-	CNNIC (country)
rpki.caramelfox.net	149.56.154.148	2001:678:f68:1099:5054:ff:fec3:d6c3	AS16276	-	-	-	-	nginx	-	CaramelFox Networks (ISP)
rpki.berrybyte.network	104.167.214.10	-	AS60841	159	18	220	313	Caddy	-	BerryByte Limited (ISP)
rpki.august.tw	165.140.142.125	2602:fc23:18::3	AS50058	48	133	149	282	nginx	-	August Internet Limited (ISP)
rpki.apernet.io	141.164.40.93	2401:c080:1c01:6da:5400:2ff:fedc:c925	AS20473	136	280	332	353	nginx	-	-
rpki.akrn.net	43.129.24.158	-	AS132203	147	193	371	106	nginx	-	-
rpki.admin.freerangecloud.com	172.98.192.101	-	AS31863	68	117	130	308	Apache	-	-
rpki.0i1.eu	-	2a03:4000:20:b7::106	AS197540	-	-	-	-	Apache	-	-
rov-measurements.nlnetlabs.net	64.225.72.59	2a03:b0c0:2:d0::87b:e001	AS14061	139	18.3	207	133	nginx	-	NLnet lab (organization)
repo-rpki.idnic.net	116.193.189.25	-	AS63515	186	164	394	66	nginx	-	IDNIC (country)
parent.rov.koenvanhove.nl	203.119.23.1	2a04:b905:8000::1	AS211321	-	-	-	-	-	-	Koen van Hove (individual)
magellan.ipxo.com	104.22.46.221 172.67.20.88 104.22.47.221	2606:4700:10::6816:2edd 2606:4700:10::ac43:1458 2606:4700:10::6816:2fdd	AS13335	1.9	0.96	2.5	1.3	cloudflare	cloudflare	-
krill.rayhaan.net (pr01.rue.rayhaan.net)	185.95.219.37	2a05:fc87:1:12::b	AS39540	150	26	204	130	Apache	-	-
cloudie.rpki.app	192.189.65.154	2607:2c40:beef:1b::44	AS12186	49	123	146	249	nginx	-	-
chloe.sobornost.net	46.23.94.25	2a03:6000:6f66:615::25	AS60131	148	8	207	124	OpenBSD httpd	-	-
ca.rg.net	198.180.152.6	2001:418:3807::6	AS4128	47	117	137	272	Apache	-	-
ca.nat.moe (kanako.nat.moe)	208.167.242.134	2001:19f0:5:3b99:5400:2ff:fead:cf71	AS20473	70	85	118	193	nginx	-	-
0.sb	185.255.55.88	2a0c:59c0::88	AS3214	145	8.1	191	125	nginx	-	-
repo.kagl.me	104.37.40.237	-	AS970	23	151	192	248	nginx	-	-
rpki.co	69.197.133.102	2606:b0c0:1001::65	AS50058	46	116	149	275	openresty	-	-
rpki.cc	104.167.215.30	-	AS60841	43	120	146	260	KittenSystems	-	-
krill.accuristechnologies.ca	96.126.104.114	2600:3c03::f03c:93ff:fe17:c5b3	AS63949	65	88	129	196	nginx	-	-
rpki-01.pdxnet.uk	212.111.43.214	2a01:7e00::f03c:93ff:fe17:38fe	AS63949	148	13	181	136	Caddy	-	-

Table VIII presents our measurement results. Among them, 8 RRDP files are explicitly hosted on CDN, with significantly lower global access latency compared to others, and the HTTPS headers also contain information about the CDN service provider. Among the remaining RRDP files, 50 of them resolved to a single IPv4 address, while 3 of them resolved to multiple IPv4 addresses. For these 50 single-address RRDP files, their global access latency is an order of magnitude higher than that of services hosted in CDNs, and there is no obvious CDN provider information in the domain name and HTTPS header. Therefore, it is highly likely that they are not hosted on a CDN. Among the 3 multi-address RRDP files, "rpki.luys.cloud" has three different geographical locations for the IP addresses. We speculate that it might be utilizing multiple servers to host RPKI objects. Furthermore, based on the domain information or corresponding webpage (if they have one), we also inferred the affiliation of the PPs. The results are presented in the last column, which shows that there are two PPs operated by individuals. Additionally, we found that the RRDP files for some PPs were in an inaccessible state, which showed that the service was unstable.

APPENDIX D
THE REAL WORLD SURVEY

We obtained a list of ASes that have deployed ROA from ROA data and conducted a survey with the administrators of 2,500 randomly selected ASes. We distributed our questionnaires via email. We also randomly sent questionnaires to the administrators of 3,000 ASes that have not deployed ROA.

The main issues we focus on include (1) Future consideration of adopting delegated RPKI; (2) Deployment status of ROV; (3) Reasons for not deploying ROV; (4) Considerations regarding malicious behavior from RPKI authorities. We have received responses from administrators of 68 ASes that have deployed ROA and 35 ASes that have not deployed ROA. Figure 17 to Figure 21 show the results from ROA deployers, while Figure 22 to Figure 25 show the results from non-ROA deployers. We believe our result can somewhat represent the industry's concerns about RPKI.

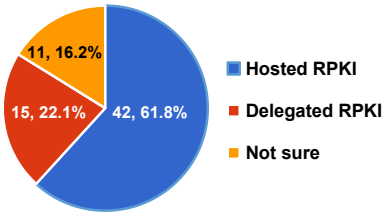


Fig. 17: Which RPKI ROA deployment mode does your organization use? (w/ROA).

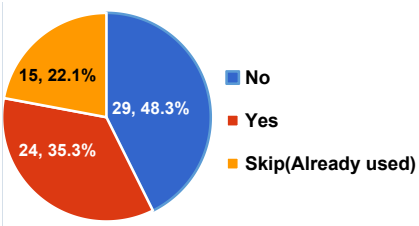


Fig. 18: For P3. Will you consider using delegated RPKI and running your own PP in the future? (w/ROA).

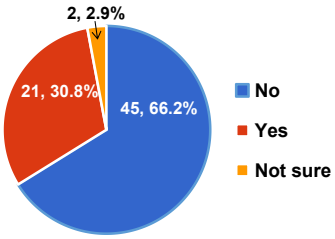


Fig. 19: Has your AS deployed ROV (Use RPKI data to verify the BGP updates that you receive)? (w/ROA).

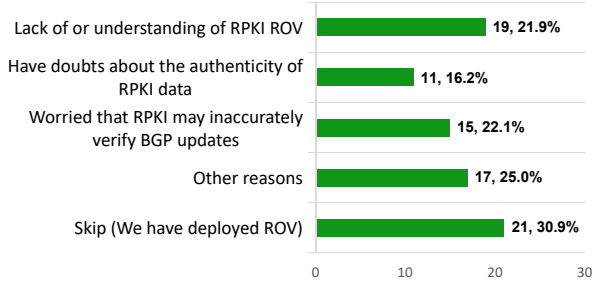


Fig. 20: What are the reasons for not deploying ROV? (multi-selection) (w/ROA).

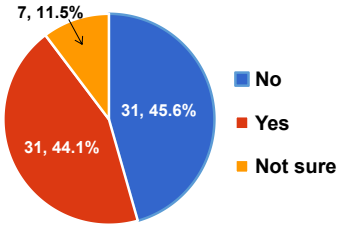


Fig. 21: For P1. Are you worried that RPKI authorities maliciously compromise your certificates, which could affect the legitimacy of your BGP updates? (w/ROA).

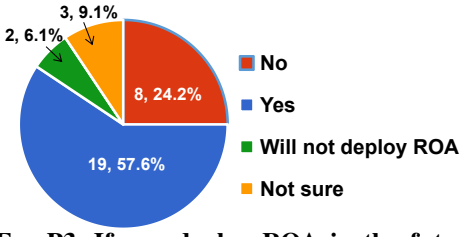


Fig. 22: For P3. If you deploy ROA in the future, would you consider adopting delegated RPKI and running your own PP? (wo/ROA).

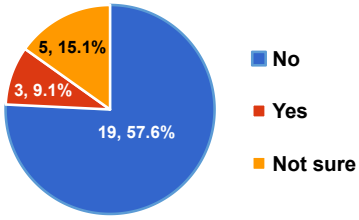


Fig. 23: Has your AS deployed ROV (Use RPKI data to verify all BGP advertisements that you receive)? (wo/ROA).

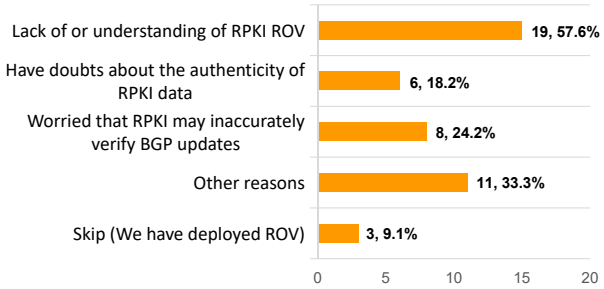


Fig. 24: What are the reasons for not deploying ROV? (multi-selection) (wo/ROA).

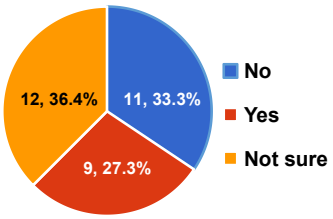


Fig. 25: For P1. Are you worried that RPKI authorities maliciously compromise your certificates, which could affect the legitimacy of your BGP updates? (wo/ROA).