

Mind the Gap: Proving and Improving RPKI

Shay Cohen
Hebrew University of Jerusalem
Jerusalem, Israel
shay.cohen6@mail.huji.ac.il

Nicholas Scaglione
University of Connecticut
Storrs, CT
nicholas.scaglione@uconn.edu

Yossi Gilad
Hebrew University of Jerusalem
Jerusalem, Israel
yossigi@cs.huji.ac.il

Hemi Leibowitz
The College of Management
Academic Studies
Rishon LeZion, Israel
menahemle@colman.ac.il

Bing Wang
University of Connecticut
Storrs, CT
bing@uconn.edu

Amir Herzberg
University of Connecticut
Storrs, CT
amir.herzberg@uconn.edu

Abstract

We present the first rigorous security analysis of the Resource Public Key Infrastructure (RPKI), an IETF standard for protecting inter-domain routing from basic yet effective attacks: prefix and subprefix hijacks. Existing evaluations of RPKI’s security focus on empirical studies: adoption measurements, simulations evaluating the security provided by different adoption levels, and identification of implementation and specification flaws. In contrast, we focus on *rigorous, modular security specifications and analysis* of RPKI, including its interactions with the Border Gateway Protocol (BGP) and the Internet Protocol (IP). We identify sufficient conditions for RPKI to provably achieve its goals (requirements), under explicit, well-defined assumptions (models). We show that existing RPKI deployments do not always meet these conditions, e.g., because of circular dependencies, and propose standard-compliant improvements that restore them.

1 Introduction

For more than 30 years, the Border Gateway Protocol (BGP) has been one of the fundamental protocols that make the Internet work, serving as the de facto protocol for routing information across Autonomous Systems (ASes). Unfortunately, like other protocols designed decades ago, BGP was not designed with security mechanisms, leaving it vulnerable to various attack vectors with significant consequences. For example, a rogue AS can announce an entire IP prefix assigned to another AS, an attack called a *prefix hijack*, letting it intercept traffic sent to that prefix.

To mitigate such attacks, BGP needs a Public-Key Infrastructure (PKI) to authenticate the origin of BGP announcements. The most prominent PKI solution is the *Resource Public Key Infrastructure (RPKI)* [36]. In RPKI, ASes are issued certificates that map their IP prefixes to their public key. A certificate lets an IP prefix owner sign a Route Origin Authorization (ROA), which links IP prefixes to an AS allowed to originate them, and publish the ROA (along with the

certificate) at RPKI Publication Points (PPs). ASes can then fetch these certificates and ROAs, and perform Route Origin Validation (ROV) on the received BGP advertisements. More advanced inter-domain routing security mechanisms that extend authentication beyond the origin, such as BGPsec [48] and other proposals [14, 57], all build on RPKI’s certificates, ROAs, and the ROV mechanism. Encouragingly, ROA coverage for IP prefixes has surged in recent years [11, 59], with more than 65% adoption as of 2026 [13].

Existing evaluations of RPKI’s security are empirical: they measure adoption, audit implementations, and demonstrate specific attacks (§2). However, despite RPKI’s importance and growing adoption, no rigorous analysis of its security guarantees exists.

It is natural to ask: *why was such rigorous analysis not done before?* The lack of a rigorous analysis of RPKI is not an anomaly. Internet protocols that run distributed across many parties are too complex for the standard security-analysis toolkit, which binds every assumption to every goal in a single monolithic specification [27, 83, 84]. Protocols of this kind tend to receive informal security arguments, or formal analyses that abstract away mechanisms which in practice decide whether the protocol is secure. In many cases, these analyses miss weaknesses that are detected only later, after an attack ([49, 81] are two well-known examples).

In particular, an often-overlooked key question is: how do ASes that adopt RPKI, e.g., to deploy ROV, obtain the relevant RPKI records (for ROV: ROAs and certificates)? We cannot realistically assume that ASes performing ROV have already obtained all existing certificates and ROAs. Analysis of RPKI, and of the security mechanisms that build on it, should therefore be precise and account for subtle aspects with significant security implications.

That said, this ‘norm’ is understandable; analyzing complex protocols with more realistic models has proven challenging. To illustrate why, consider RPKI’s object distribution mechanism, which requires connectivity to RPKI PPs. This creates a potential dependency cycle: relying parties need to retrieve a ROA covering the PP’s IP address to reach it securely, but

RPKI neither guarantees “cycle-free” ROA resolution nor attempts to achieve it [16]. As ROV can reject bogus routes only when it knows a ROA for that prefix, failing to reach a PP also creates a security risk.

This work addresses this security gap by providing the *first rigorous specifications and (modular) analysis of RPKI*. This analysis allowed us to identify the assumptions necessary for security, as well as additional concerns with current RPKI deployments. Using these insights, we provide protocol-compatible recommendations to improve the security of RPKI deployments, and present a prototype of a robust relying party that implements the client-side recommendation, which any AS can deploy today. We use the Composable Modular Security Specifications (CMoSS) framework [83] to define the requirements that interdomain routing should provide, and to analyze whether BGP, with and without RPKI, achieves these requirements under different models (assumptions). CMoSS is designed for *modularity*, making it well suited to *realistic yet rigorous* analysis of complex protocols such as RPKI deployed with BGP.

Our analysis proceeds in incremental steps. We begin by defining the connectivity requirements (§5), a non-trivial task: against an adversary that can originate announcements, the natural requirement is unattainable, so we also define a *stable routing* variant of the connectivity requirement. We then analyze BGP under a scheduler adversary that only determines when messages are delivered, and deduce the assumptions BGP needs to provide connectivity against it. Crucially, those assumptions are not sufficient when BGP is composed with RPKI, because a ROA can make an otherwise valid route invalid. In fact, this impact can be indirect: even if the ROAs in question do not concern either communication endpoint, preventing a relying party from accessing a PP and retrieving RPKI objects may cause that party to classify legitimate routes as invalid. This finding motivates us to analyze RPKI under its three deployment modes, which differ in who runs the certificate authority (CA) and who operates the PP (§3.2). In *hosted* mode the Regional Internet Registry (RIR) does both, in *delegated* mode the resource owner does both, and in *hybrid* mode the owner is its own CA but publishes at another entity’s PP. We prove our positive result for hybrid mode, which also covers hosted mode, and discuss the challenges of delegated mode (§7).

Next, we consider a stronger adversary that can also originate prefix hijacks. We show that even if the source, the destination, and every AS on the route between them enforce ROV and hold the destination’s ROA, the adversary can *misroute* the traffic onto a less-preferred path that traverses an AS it controls, which then discards it (§8). We give sufficient conditions on ROV deployment that prevent them.

To ensure that ASes can obtain RPKI objects and perform ROV, we propose standard-compliant extensions to RPKI relying parties and recommendations for PP operators to improve security. We provide RobustRP, an open-source relying party

that implements the client-side extensions, and show that it can fetch 40.3% of RPKI’s route authorizations from more servers than today’s software.

In summary, the contributions of this paper are as follows.

- We rigorously define *IP connectivity requirements* in two forms, with and without a stable routing precondition (§5).
- We formalize BGP as a CMoSS scheme and prove that it satisfies the IP connectivity requirement against a scheduler adversary, establishing the baseline against which we measure RPKI (§6).
- We prove that the RPKI specifications [33, 42] satisfy the requirements under well-defined models (assumptions), including models of the adversary’s capabilities, of communication and synchronization, and of topology and policy (e.g., valley-free routing) (§7). Against an adversary that also originates prefix hijacks, these models no longer suffice: it can misroute traffic even under ubiquitous ROV adoption. We identify the assumptions that restore stable routing IP connectivity (§8).
- Our analysis reveals potential attacks on the real-world RPKI deployment (where some of the models do not hold). We present extensions to RPKI relying parties and operational recommendations to publication-point operators that mitigate these attacks and secure RPKI deployment. Our proposals are compatible with the RPKI standard and today’s deployment. We implement an open-source RPKI relying party that integrates the extensions we propose and evaluate its benefits (§9).
- Our work is a step toward provable security for routing protocols, which typically assume a secure underlying RPKI. Because our analysis supports secure composition, such protocols can be analyzed on top of our results rather than on top of an assumption.

2 Related Work

Provable Security for Inter-Domain Routing. Rigorous security analysis in inter-domain routing has targeted routing protocols rather than the certification infrastructure they rely on. Boldyreva and Lychev [5] give a provable-security treatment of S-BGP [37] and other path-vector protocols. Clean-slate architectures such as SCION [4] carry a trust infrastructure of their own. BGP’s routing dynamics have likewise been formalized [22, 26]. Analyses of secure routing protocols that build on RPKI take the certification infrastructure as given: work on BGPsec [48, 77] and on BGP security under partial deployment [46] assumes that certificates and ROAs are available to the ASes modeled as adopters. To our knowledge, no prior work rigorously establishes RPKI’s own guarantees, or derives the conditions under which the infrastructure it depends on preserves them. We close this gap. Because our

analysis supports secure composition, mechanisms that build on RPKI, e.g., BGPsec [48], path-end validation [14], and BGP-iSec [57], can be analyzed on top of our results rather than on top of an assumption.

Empirical Evaluation of RPKI. Research on RPKI security is predominantly empirical, and establishes what today’s deployment looks like rather than what it guarantees. Longitudinal studies measure ROA coverage and ROV adoption [11, 31, 44, 59], and recent surveys assess the maturity of the deployment as a whole [51, 72]. A second line audits implementations: van Hove et al. [80] test known relying-party implementations against resource-exhaustion and traversal attacks, Mirdita et al. [53] report two zero-day defects that crash relying-party software, potentially affecting 84.9% of deployments, and CURE [52] finds 18 further vulnerabilities by fuzzing. A third demonstrates attacks that keep a relying party from retrieving objects, by abusing its rate limits or by inflating a delegation path [28, 29]. Cooper et al. [16] show that a misbehaving certificate authority can subvert the resources of its subordinates, and observe that RPKI’s reliance on the routing system it protects creates a circular dependency. Liu et al. [45] survey infrastructure threats to publication point reachability and report PPs whose addresses lack ROA coverage. We instead derive the assumptions under which connectivity is guaranteed, and show where today’s deployment does not meet them.

Limitations of Traditional Frameworks. Traditional frameworks for provable security, whether game-based [25] or simulation-based such as Universal Composability (UC) [10], formalize a specification as a single monolithic object that binds every assumption to every goal. RPKI is not a single object. It is a distributed system of RIRs, CAs, PPs, relying parties, and routers, operating under assumptions about topology, routing policy, and synchronization, and resting on two further infrastructures. Thus, we adopt CMoSS (§3.3), which specifies each assumption and each goal separately and composes the resulting proofs, rather than as a single indistinguishability game or ideal functionality.

Improving RPKI. A separate line of work improves RPKI in directions orthogonal to ours: improving repository privacy and resilience [74], resisting censorship through a decentralized repository architecture [78], simplifying the certification hierarchy [71], and strengthening ROV under partial adoption [43, 55].

3 Background

We next give the background on BGP (§3.1), RPKI (§3.2), and the CMoSS framework we use in our analysis (§3.3).

3.1 The Border Gateway Protocol

To route IP packets from one AS to another, i.e., to perform *inter-domain* routing, ASes first need to establish connections

between dedicated routers called *border-gateway* routers, which run BGP. In BGP, ASes establish connections based on business agreements, which dictate the routing policy. Specifically, there are two main types of business relationships: (1) customer-provider, where a customer AS pays a provider AS for routing traffic to and from the customer, and (2) peer-to-peer, where ASes, typically of similar size, agree to exchange traffic free of charge.

When an AS x announces to a neighboring AS a route to an IP prefix p assigned to AS x , we say that AS x is the *origin* of prefix p . Upon receiving a route announcement, an AS decides whether that route is its *best* route to the prefix and, if so, whether to export it further. The decision follows financial incentives, commonly modeled by the *Gao-Rexford valley-free* model [22]. The model consists of two policy conditions and a topology condition:

1. Customer-learned routes are preferred over peer-learned routes, which in turn are preferred over provider-learned routes.
2. Routes learned from providers or peers are exported only to customers, whereas routes learned from customers are exported to all.
3. The customer-provider graph is acyclic, i.e., no AS is a customer of itself, directly or transitively.

BGP does not authenticate route announcements. An AS can announce a route to any prefix, and its neighbors accept and propagate that route according to the conditions above. An announcement for a prefix not assigned to the announcing AS is a *prefix hijack*; it diverts traffic to that AS from every AS that selects the hijacked route as its best route. RPKI (§3.2) mitigates prefix hijacks by binding each prefix to the ASes authorized to originate it.

3.2 The Resource Public Key Infrastructure

RPKI cryptographically binds Internet number resources, i.e., IP prefixes and AS numbers (ASNs), to their resource owners. Such bindings take the form of *Resource Certificates* (RCs), which are extended PKIX certificates [33, 47]. The five Regional Internet Registries (RIRs) anchor the trust and issue RCs to resource owners, which use them to sign *Route Origin Authorizations* (ROAs) [76]; each ROA states which AS is authorized to originate a particular IP prefix. Each ROA also carries a *maxLength* field, bounding how specific an announced subprefix may be while remaining authorized [23].

RPKI objects are published at *Publication Points* (PPs), also called repositories. ASes that wish to use RPKI to validate BGP advertisements run *Relying Party* (RP) software, also called a *validator*, to retrieve RPKI objects from different PPs. The RP is bootstrapped with Trust Anchor Locator (TAL) files, where each TAL is a small file containing a RIR’s public key and the URI of its root certificate. Each certificate includes a Subject Information Access (SIA) extension that points to the PP that holds the objects it certifies.

Starting from each TAL, the RP walks the SIA chain, fetches each PP over rsync or the RPKI Repository Delta Protocol (RRDP) [6], and verifies every object it encounters. At the end of this process, the RP compiles a set of *Validated ROA Payloads* (VRPs), each being a (prefix, maxLength, origin AS) triple [54].

BGP routers receive the VRPs and use them to perform Route Origin Validation (ROV), which classifies each received route as *Valid*, *Invalid*, or *NotFound*. A route is *Valid* if some VRP covers the announced prefix, authorizes its origin AS, and permits its length, i.e., the announced prefix is no longer than the maxLength of that VRP; *Invalid* if some VRP covers the announced prefix but no VRP satisfies all three conditions; and *NotFound* if no VRP covers the announced prefix. Routers then drop routes classified as *Invalid*¹.

Each resource owner can operate under one of three deployment modes: *hosted*, *delegated*, or *hybrid*. These modes differ in two critical aspects of RPKI: object certification, performed by a *Certificate Authority* (CA), and object publication, performed at a PP. In hosted mode, the RIR or managing registry provides both services: it manages the cryptographic keys for issued certificates, uses these keys to sign RPKI objects on behalf of its customers, and publishes the resulting objects at its own PP. In delegated mode, the resource owner runs both services: it is its own CA, manages its private keys, and operates its own PP, making it responsible for the PP’s availability. In hybrid mode, the resource owner is its own CA, retaining its keys and issuing its own RPKI objects, but publishes them at a PP operated by another entity, e.g., a RIR. §A illustrates the three deployment modes.

3.3 Rigorous Analysis using CMoSS

MoSS [27] and CMoSS [83] allow *modular specifications*: a set of requirements (goals) and a set of models (assumptions), each defined rigorously and independently of the others. This separation matters for complex real-world protocols such as RPKI, which operate under multiple assumptions and must ensure multiple goals. It also lets different protocols be analyzed under the same reusable models.

In this work, we use *CMoSS* (*Composable MoSS*) [83], an extension of MoSS that supports provably secure composition of (provably secure) protocols. CMoSS improves on MoSS by allowing not only modular specifications but also *modular design and analysis*; i.e., protocols may use one or multiple *subprotocols*, with composition of the security analysis.

CMoSS defines a single adversary-driven execution process (‘game’) denoted $\text{Exec}_{\mathcal{A}, \mathcal{P}}$, where a protocol $\mathcal{P}$ is executed against an adversary $\mathcal{A}$, which controls most of the execution; both $\mathcal{P}$ and $\mathcal{A}$ are randomized algorithms. For example, $\mathcal{A}$ dictates which operations to invoke, at what time, on which entities, with what inputs. The execution process

fulfills the adversary’s instructions in an *event loop*, where each adversarial instruction is translated into a single event. At the end of execution, the process outputs a *transcript* T that details all executed events.

The execution process is oblivious to the protocol, models, and requirements. We define models and requirements using predicates, described informally here; see the rigorous definitions in [83].

Let T be the transcript (output) of a particular execution of $\text{Exec}_{\mathcal{A}, \mathcal{P}}$, and let π be a predicate over such transcripts. We say that execution T *satisfies* predicate π if $\pi(T) = \top$. Given a protocol $\mathcal{P}$ and an adversary $\mathcal{A}$, we use $\epsilon_{\mathcal{A}, \mathcal{P}}^\pi$ to denote their *failure probability* for predicate π , i.e., the probability that $\pi(T) = \perp$ where T is a random execution of $\mathcal{P}$ with $\mathcal{A}$. Models and requirements are defined as *specifications*, where each specification is a pair (π, β) such that π is a predicate and β is the base failure probability². A specification σ holds *unconditionally* for an adversary $\mathcal{A}$ and protocol $\mathcal{P}$, denoted $(\mathcal{A}, \mathcal{P}) \models \text{Uncond}(\sigma)$, if the failure probability $\epsilon_{\mathcal{A}, \mathcal{P}}^\pi$ never exceeds β ; if it exceeds β by at most a negligible amount, the specification holds *asymptotically*, denoted $(\mathcal{A}, \mathcal{P}) \models \text{Asymp}(\sigma)$.

A model is a set of specifications $\mathcal{M} = \{(\mu_i, \beta_i)\}$ that restricts the adversary’s capabilities. An adversary $\mathcal{A}$ satisfies a model $\mathcal{M}$ if for every PPT protocol $\mathcal{P}$ and every specification $\sigma \in \mathcal{M}$, σ holds for $\mathcal{A}$ and $\mathcal{P}$. Security requirements are also sets of specifications, $\mathcal{R} = \{(\rho_i, \beta_i)\}$. A protocol $\mathcal{P}$ satisfies requirement $\mathcal{R}$ assuming model $\mathcal{M}$ if for every PPT adversary $\mathcal{A}$ that satisfies $\mathcal{M}$ and every specification $\sigma \in \mathcal{R}$, σ holds for $\mathcal{A}$ and $\mathcal{P}$.

We recall the following observation from [83]: a specification satisfied unconditionally is also satisfied asymptotically.

$$(\mathcal{A}, \mathcal{P}) \models \text{Uncond}(\sigma) \Rightarrow (\mathcal{A}, \mathcal{P}) \models \text{Asymp}(\sigma) \quad (1)$$

Unconditional satisfaction is easier to use in proofs: it requires showing only that the adversary cannot win beyond β , whereas asymptotic satisfaction also requires accounting for the negligible number of executions beyond β . We use this observation in our analysis: we first show that BGP alone satisfies the connectivity requirement unconditionally when the model assumptions hold unconditionally, so Equation (1) yields asymptotic satisfaction as well. We then compose BGP with RPKI, whose blackbox PKI and DNS subprotocols satisfy their own requirements only asymptotically, so the guarantees we prove for BGP with RPKI are asymptotic.

CMoSS also supports *secure composition* of protocols. We leverage this by building on prior work on formalizing PKI security [84], which lets us formally address the underlying PKI rather than abstract it away. We analyze RPKI and prove it secure when composed with *any* PKI-secure scheme. Composing over a PKI may appear routine, since security analyses

¹To allow incremental deployment, RPKI is “fail-open”: announcements with no covering ROA are treated as *NotFound* and accepted.

²Often $\beta = 0$, but for example, $\beta = 0.5$ is used for indistinguishability.

commonly assume secure underlying schemes, but it is not: a PKI is not a single scheme like a signature scheme, but a distributed system with many entities and elements that affect its security. Using secure composition to design RPKI on top of a PKI scheme makes RPKI analysis feasible. Future work could likewise compose protocols on top of our RPKI analysis, which matters because many routing security proposals, e.g., BGPsec [48], rely on a deployed RPKI.

Some models and requirements are defined with respect to specific bounds. We use Δ to denote a function that maps strings to numbers; for example, $\Delta[\text{'clk'}]$ is the bound on clock drift in the $\mu_{\Delta[\text{'clk'}]}^{\text{Drift}}$ model predicate. Another example is the $(\delta_{e2e}, \delta_{boot})$ -IP connectivity requirement (Definition 3), where we require that a packet arrives at its destination no later than δ_{e2e} after it was sent. For a list of all the bounds used in the paper, see §B.

4 Inter-Domain Routing Scheme

We now define an Inter-Domain Routing (IDR) scheme that manages routing decisions, enforces policies, and maintains connectivity between ASes, while handling packet transmission across AS boundaries based on the control plane's state. Using the IDR scheme, we can specify security requirements for protocols that implement it, as we do in §5.

We define an inter-domain routing scheme as the set $\text{IDR} = \{\text{INIT}, \text{ORIGINATE}, \text{WITHDRAW}, \text{SETPOLICY}, \text{SENDPACKET}, \text{RECEIVEPACKET}\}$. First, we explain the rationale for this specific set of operations in §4.1 and then present the formal definition of the IDR scheme in §4.2.

4.1 Functionality and Interfaces

A set $\mathcal{N}$ of stateful entities runs an inter-domain routing protocol. The local state of each of these entities is initialized using the IDR.INIT operation. Initialization ensures each entity has everything it needs to execute the protocol correctly, e.g., cryptographic keys, RIRs' root certificates, etc.

Once an entity is initialized, it can originate routes to prefixes it owns using IDR.ORIGINATE and withdraw these announcements using IDR.WITHDRAW . These announcements are propagated across the network as data packets using the IDR.SENDPACKET operation, which uses its internal state to decide to which neighboring AS to send the packet. Conversely, when a packet arrives from a neighboring AS, the entity processes it using the IDR.RECEIVEPACKET operation. This operation evaluates the incoming packet's destination against the active forwarding state to determine the appropriate action: forward the packet to the next hop, deliver it locally, or drop it (e.g., if no valid route exists).

The routing decisions that entities make are guided by *policies*, which are based on business relationships, economic constraints, or security preferences. Specifically, import policies dictate filtering rules that determine which incoming updates

are ignored, and export policies dictate ranking and propagation mechanisms upon which selected routes are ultimately advertised to neighbors. An entity's policy is configured using the IDR.SETPOLICY operation.

We define a policy *policy* as an ordered sequence of conditional rules, a structure that enables granular control over route attributes and propagation behavior.

DEFINITION 1 (Policy). *policy* = $(r_1, r_2, \dots, r_n)$ is a finite ordered list of rules, where each rule $r_i = (C_i, A_i)$ consists of:

- C_i : a finite ordered list of conditions $(\text{cond}_1, \text{cond}_2, \dots)$, where each $\text{cond}_j = (\text{ConditionName}, \text{params})$ includes a symbolic name (e.g., "prefixIn", "asPathContains") together with a tuple of fixed parameters.
- A_i : a finite ordered list of actions $(\text{act}_1, \text{act}_2, \dots)$, where each $\text{act}_k = (\text{ActionName}, \text{params})$ includes a symbolic name (e.g., "setLocalPref", "prepend", "accept", "reject", "continue") together with parameters. By convention, the final action must be accept, reject, or continue.

The rules of a policy *policy* are evaluated in order. The first rule whose conditions are all met is applied, i.e., the rule's actions are executed sequentially. The protocol's implementation interprets each action or condition according to its symbolic name and parameters. If the final action is accept, the route is accepted; if it is reject, the route is rejected; if it is continue, evaluation proceeds with the next rule. If no rule matches, the route is rejected by default.

4.2 Scheme Definition

We now formally define the IDR scheme.

DEFINITION 2 (Inter-Domain Routing Scheme). An inter-domain routing scheme *IDR* is a set containing (at least) the following PPT algorithms, where each algorithm receives a state *st*, and outputs an updated state *st'*.

- $\text{INIT}(\text{params})$: Initializes the state according to the input parameters *params*.
- $\text{ORIGINATE}(p) \rightarrow (\text{out})$: Processes the announcement of prefix *p* and outputs a set of name-value pairs *out* that include packet-type entries for neighboring ASes.
- $\text{WITHDRAW}(p) \rightarrow (\text{out})$: Represents the local withdrawal of a previously originated prefix *p*. This is the logical negation of $\text{ORIGINATE}(p)$. It returns a set of name-value pairs *out* that include packet-type entries (e.g., withdraw messages) for neighboring ASes.
- $\text{SENDPACKET}(pkt) \rightarrow (\text{out})$: Takes a packet *pkt* as input and returns a set of packet send requests *out*.
- $\text{RECEIVEPACKET}(pkt, \text{sender}) \rightarrow (\text{out})$: Takes a packet *pkt* as input and returns a set of processing or forwarding actions for the packet.

- **SETPOLICY(policy)**: Takes as input a policy object policy (a structured set of (name, value) rules) and returns $\top$ or $\perp$.

In case any of these operations fails, the output is $\perp$.

5 Connectivity Requirements

In this section, we define the fundamental goal of *IP connectivity*, i.e., the requirement that data-plane traffic is delivered to its desired destination.

While connectivity is often viewed as a *reliability* issue, i.e., ensuring reachability in the face of random failures, we regard it as a *security* requirement. This is because our goal is to ensure connectivity not just against benign failures, but in the presence of an adversary attempting to disrupt the network. A fully secure routing system may need further guarantees, such as preventing an adversary from observing traffic between honest parties. This work focuses on ensuring that an adversary cannot prevent communication between honest endpoints, without making guarantees regarding traffic confidentiality or privacy.

Intuitively, secure connectivity means that when an honest source AS sends a packet to an IP address legitimately announced by one or more destination ASes, the packet will successfully reach at least one of those destinations within a bounded delay δ_{e2e} , regardless of adversarial interference. We further allow delay, denoted δ_{boot} , for bootstrapping the routing infrastructure.

Note that unlike *control-plane reachability*, i.e., the requirement that a path exists in the routing tables, the IP connectivity requirement ensures that packets are successfully delivered in the *data plane*.

DEFINITION 3 (($\delta_{e2e}, \delta_{boot}$)-IP Connectivity Requirement). Let AS_{src} be an AS, AS_{Dsts} be a set of ASes, ip be an IP address, p be the most specific prefix containing ip , and pkt_id identify a packet sent at time t .

The ($\delta_{e2e}, \delta_{boot}$)-IP connectivity requirement, defined as $\mathcal{R}_{\delta_{e2e}, \delta_{boot}}^{IP-CON} \equiv \{(\rho_{\delta_{e2e}, \delta_{boot}}^{IP-CON}, 0)\}$ and formally captured in *Algorithm 1* (see §C), holds for a transcript T , if given that: (1) All ASes in AS_{Dsts} are benign, and (2) All ASes in AS_{Dsts} announced prefix p prior to time $t - \delta_{e2e}$, and at least δ_{boot} after every benign entity was initialized, and (3) None of them withdrew prefix p before time $t + \delta_{e2e}$. Then, if AS_{src} sends a packet pkt_id towards ip at time t , it is received by an AS in the set AS_{Dsts} by time $t + \delta_{e2e}$. $\square$

Somewhat surprisingly, although one might naturally expect BGP with RPKI to satisfy the ($\delta_{e2e}, \delta_{boot}$)-IP connectivity requirement described above, our analysis found that it is not true even under reasonable assumptions (see §7), because an adversary could

execute a precisely³ timed data-plane hijack that violates the ($\delta_{e2e}, \delta_{boot}$)-IP connectivity requirement (see an example attack in §D). Instead, we found that only a weaker requirement is satisfied, which we term *Stable Routing (SR)* ($\delta_{e2e}, \delta_{boot}$)-IP connectivity (Definition 4). This weaker requirement mandates that for a duration of δ_{e2e} before and after the packet sending event at time t , the adversary is prohibited from performing any BGP announcements or withdrawals affecting the target IP address. In other words, the requirement enforces a stable $2\delta_{e2e}$ routing window around t , ensuring that the BGP state at the AS that sends the packet fully converges strictly prior to the send event.

DEFINITION 4 (Stable Routing ($\delta_{e2e}, \delta_{boot}$)-IP Connectivity). Let AS_{src} be an AS, AS_{Dsts} be a set of ASes, ip be an IP address, p be the most specific prefix containing ip , and pkt_id identify a packet sent at time t . The stable routing ($\delta_{e2e}, \delta_{boot}$)-IP connectivity requirement, defined as $\mathcal{R}_{\delta_{e2e}, \delta_{boot}}^{SR-IP-CON} \equiv \{(\rho_{\delta_{e2e}, \delta_{boot}}^{SR-IP-CON}, 0)\}$ and formally captured in *Algorithm 2* (see §C), holds for a transcript T , if given that: (1) All ASes in AS_{Dsts} are benign, and (2) All ASes in AS_{Dsts} announced p prior to $t - \delta_{e2e}$, and at least δ_{boot} after every benign entity was initialized, and (3) None of them withdrew p before $t + \delta_{e2e}$, and (4) No BGP announcements or withdrawals of any prefix containing ip were sent during $[t - \delta_{e2e}, t + \delta_{e2e}]$, then, if AS_{src} sends a packet pkt_id towards ip at t , it is received by an AS in AS_{Dsts} by $t + \delta_{e2e}$. $\square$

6 Analyzing BGP

We now analyze BGP based on RFC 4271 [65]. In §E, we detail our formalization of RFC 4271 into CMOSS predicates, and denote it as BGP. We define model assumptions for BGP in §6.1 and analyze BGP under these assumptions in §6.2, demonstrating that BGP guarantees IP connectivity against an adversary that controls message scheduling but no AS. This analysis is the precursor to our introduction of RPKI in §7.

6.1 BGP Model Assumptions

We now define model assumptions for BGP using the $\mu_{f,\Delta}^{BGP}$ predicate, which is a conjunction of the following:

$$\mu_{f,\Delta}^{BGP} \equiv \left\{ \begin{array}{l} \mu_{\Delta}^{Drift} \wedge \mu_{\Delta}^{Com} \wedge \mu^{Init} \wedge \mu^{BgpInit} \wedge \\ \mu^{WellFormedGraph} \wedge \mu^{VF} \wedge \mu^{PreferSelf} \wedge \\ \mu^{MaxPrefixLength} \wedge \mu^{F \leq f} \wedge \mu^{SHO} \end{array} \right\} \quad (2)$$

The rationale is twofold: first, capturing assumptions as small predicates improves readability; second, it promotes modularity. Specifically, in (2), we re-use model predicates that were defined in previous works, and allow future works

³Performing such a perfectly timed attack is extremely difficult in practice. Moreover, if performed repeatedly, it would generate a high volume of traffic and immediately alert network administrators.

to re-use model predicates which we define in this work. We now explain each model assumption and its rationale.

Clock Synchronization. We assume a partially synchronous system in which entities have access to local clocks that proceed at a rate comparable to real time. While we do not require perfectly synchronized clocks, we assume that the drift of any honest entity's local clock from the actual time is bounded. This is captured by the $\mu_{\Delta[\text{'clk'}]}^{\text{Drift}}$ predicate⁴ from [27].

Network Delay. We also assume that the underlying link layer provides reliable message delivery between connected, non-faulty ASes. Specifically, if an honest entity i sends a message m to a connected honest neighbor j at time t , then j will process m within a bounded delay $\Delta[\text{'com'}]$, justified by BGP's reliance on TCP. This assumption is captured by the $\mu_{\Delta[\text{'com'}]}^{\text{Com}}$ predicate from [84]. Formally, if an operation outputs a triplet ('send', pkt , j), then j receives a call to RECEIVEPACKET with parameters (pkt , i).

Initialization. We assume that all entities are initialized at the start of the execution via the μ^{Init} predicate from [84]. In addition, we assume that all physical peering links are initialized as secure /31 point-to-point subnets and do not overlap with the AS's assigned IP blocks. This assumption is captured by the μ^{BgpInit} predicate (Algorithm 61). It prevents trivial data-plane hijacking by ensuring infrastructure addresses remain distinct from the routable prefix space. We further assume that each AS prioritizes its own originated routes above all others. This principle is captured by the $\mu^{\text{PreferSelf}}$ (Algorithm 59) predicate.

Topology. We model the Internet as an undirected graph of ASes that form relations based on customer-provider and peering business relationships (Definition 5). We do not consider other relationships, such as sibling-to-sibling, consistent with prior work (e.g., [55], [57]). We assume there exists a *Tier-1 clique* of top-tier ASes that have no providers, maintain full-mesh peering relationships, and exhibit *global reachability*, i.e., they are directly reachable from any AS. This results in a topology that contains *no islands*. These assumptions are captured in the $\mu^{\text{WellFormedGraph}}$ predicate (Algorithm 42).

DEFINITION 5 (AS Topology Graph). *An AS topology graph G is an undirected graph $G = (V, E, R)$, where V is the set of ASes, E is the set of edges representing direct BGP links between neighboring ASes and each edge $e \in E$ is annotated with a business relationship by a function $R : E \rightarrow \{p2c, c2p, p2p\}$, where the label indicates the economic relationship between the two ASes connected by e : customer-to-provider (c2p), where the customer pays the provider for transit; provider-to-customer (p2c), the reverse; and peer-to-peer (p2p), where traffic is exchanged without payment.*

⁴MoSS uses the π notation for predicates, regardless of them being model predicates or requirement predicates. However, CMOSS uses μ for model predicates and ρ for requirement predicates. Since we follow the CMOSS notation, we use μ for consistency, even though the cited paper used π . The same applies to other predicates we re-use from previous works.

Routing Policy. We assume the *valley-free (VF)* routing policy (see §3.1) and formalize it as the μ^{VF} predicate:

$$\mu^{\text{VF}} \equiv \left\{ \mu^{\text{VFexport}} \wedge \mu^{\text{VFpref}} \wedge \mu^{\text{NoProviderCustomerCycle}} \right\}$$

The first two predicates specify the export policy and route preference, respectively, and the third predicate ensures no cycles of customer-provider relationships in the AS topology (Algorithms 56 to 58). Collectively, these policies prevent the formation of “valleys” in AS paths. While only mostly followed in practice [2], they remain the standard for theoretical BGP modeling (e.g., [57], [21]).

Routes propagate only along valley-free paths, so the propagation time is governed by the length of these paths rather than by the graph distance. We therefore measure the extent of G by its valley-free diameter.

DEFINITION 6 (Valley-free diameter). *Let $G = (V, E, R)$ be a well-formed AS topology graph (Algorithm 42). For $u, v \in V$, let $d_{\text{VF}}(u, v)$ be the length of a shortest valley-free path from u to v . The valley-free diameter is $\text{diam}_{\text{VF}}(G) = \max_{u, v \in V} d_{\text{VF}}(u, v)$.*

Following RFC 7454 [17], we assume that special-purpose prefixes such as 0.0.0.0/0 are filtered, along with prefixes which are too specific, i.e., prefixes beyond /24. This is captured by the $\mu^{\text{MaxPrefixLength}}$ predicate (Alg. 43).

Adversarial Model. We assume that the set of corrupted entities F contains at most f entities, where $\mu^{|F| \leq f}$ is defined in [27]. We assume a single benign origin, specified by the predicate μ^{SHO} (Algorithm 62); that said, the analysis can be easily extended to multiple benign origins.

6.2 Connectivity Under a Scheduler Adversary

We now show that BGP ensures connectivity in the absence of corrupted ASes. That is, we assume an adversary which controls the network scheduling, i.e., the delivery order and the delay of messages up to the bound $\Delta[\text{'com'}]$, but does not control any AS and therefore cannot inject, modify or drop valid messages; we refer to such adversary as a *scheduler* adversary. This adversary is captured by setting $f = 0$ in $\mu_{f=0, \Delta}^{\text{BGP}}$, and we define the $\mathcal{M}_{\Delta}^{\text{sch}}$ model as $\mathcal{M}_{\Delta}^{\text{sch}} \equiv \{(\mu_{f=0, \Delta}^{\text{BGP}}, 0)\}$.

We first show in Lemma 1 the time it takes for a BGP advertisement to propagate under the assumptions in $\mathcal{M}_{\Delta}^{\text{sch}}$.

LEMMA 1. *Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Under model $\mathcal{M}_{\Delta}^{\text{sch}}$, if $\text{AS}_{\text{Dst}} \in V$ originates a prefix p at time t , and AS_{Dst} does not withdraw it before $t + c_{\text{prop}}$, then every benign $v \in V$ has installed a route to p with origin AS_{Dst} by time $t + c_{\text{prop}}$.*

Proof. See Appendix F.1. □

In Lemma 2 we show that an IP packet arrives at its destination successfully under the assumptions in $\mathcal{M}_{\Delta}^{\text{sch}}$.

LEMMA 2. Let $G = (V, E, R)$ be an AS topology graph, ip be an IP address and p be an IP prefix and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Under model $\mathcal{M}_{\Delta}^{\text{sch}}$, if:

- p is the most specific prefix covering ip , and
- Benign $\text{AS}_{\text{Dst}} \in V$ originates p by time $t - c_{\text{prop}}$, and
- Does not withdraw it before $t + c_{\text{prop}}$, and
- Benign $\text{AS}_{\text{Src}} \in V$ sends an IP packet identified by pkt_id towards ip at time t

Then AS_{Src} 's packet pkt_id is received by AS_{Dst} by $t + c_{\text{prop}}$.

Proof. See Appendix F.2. $\square$

Theorem 1, below, uses Lemmas 1 and 2 to show that BGP unconditionally satisfies the $(\delta_{e2e}, \delta_{\text{boot}})$ -IP connectivity requirement, $\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}}$, assuming the $\mathcal{M}_{\Delta}^{\text{sch}}$ model and a sufficiently large δ_{e2e} .

THEOREM 1 (BGP satisfies $(\delta_{e2e}, \delta_{\text{boot}})$ -IP connectivity under a scheduler adversary). Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. BGP unconditionally satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}}$ requirement assuming the $\mathcal{M}_{\Delta}^{\text{sch}}$ model holds unconditionally, provided that $\delta_{e2e} \geq c_{\text{prop}}$ and $\delta_{\text{boot}} \geq 0$.

Proof. Assume to the contrary that the theorem does not hold. This means, that following Definition 3, there must exist a transcript T in which, even though a benign $\text{AS}_{\text{Src}} \in V$ sends an IP packet destined to an IP address ip at time t , and a benign $\text{AS}_{\text{Dst}} \in V$ originated prefix p that covers ip at time $t_0 = t - \delta_{e2e}$, and did not withdraw it before $t + \delta_{e2e}$, yet AS_{Dst} did not receive the packet by $t + \delta_{e2e}$. Since $\delta_{e2e} \geq c_{\text{prop}}$, the origination at t_0 is prior or equal to $t - c_{\text{prop}}$, and p is not withdrawn before $t + c_{\text{prop}}$, so the conditions of both lemmas are met. From Lemma 1, we know that AS_{Src} receives a route r to prefix p with AS_{Dst} as the origin within $c_{\text{prop}} \leq \delta_{e2e}$. Since AS_{Dst} is the only originator of p (guaranteed by μ^{SHO}) and that all nodes are benign (guaranteed by $\mu^{|\text{F}| \leq 0}$), AS_{Src} will not receive any other routes with different origins for the same prefix. Therefore, AS_{Src} will install route r to prefix p with AS_{Dst} as the origin, by time t . From Lemma 2, an IP packet sent at time t reaches AS_{Dst} by time $t + c_{\text{prop}} \leq t + \delta_{e2e}$, contradicting the assumption that such a transcript T exists. $\square$

7 BGP with RPKI, and its Security Against a Scheduler Adversary

In this section, we extend the BGP implementation by adding support for RPKI. We then show a somewhat surprising result: BGP with RPKI may fail to ensure connectivity against the scheduler adversary, although ‘plain’ BGP does ensure connectivity against that same adversary. As we show, when certain ROAs are not received by an RPKI validator, this

can cause the validator to incorrectly consider some legitimate routes as invalid (and drop the corresponding announcements).

We begin, in §7.1, by presenting the implementation of BGP_{RPKI} . In §7.2 we discuss how RPKI uses and depends on DNS and PKIX. We examine the challenges in RPKI deployment (§7.3), the effect of PPs in ROA retrieval, and the circular dependency between RPKI and BGP; we show how this may cause loss of connectivity for BGP with RPKI (due to dropping of legitimate announcements). We then (§7.4 and 7.5) show a positive result, namely, we prove that *connectivity is guaranteed* for RPKI’s hybrid/hosted deployments.

7.1 RPKI Implementation

We define BGP_{RPKI} , which extends the BGP implementation from §6 with two mechanisms: ROV, following RFC 6811 [54], and the retrieval and validation of RPKI objects, following RFC 6480 [41]. Our formalization follows the current object profiles: resource certificates use the PKIX profile of RFC 6487 [33], ROAs follow RFC 9582 [76], and manifests follow RFC 9286 [3]. We model the certification layer using the PKI scheme of [84]. The full construction, mapping these RFCs to CMOSS predicates, is detailed in §G.

7.2 Blackbox Protocols Assumed by BGP_{RPKI}

RPKI relies on two underlying mechanisms: (1) a PKI scheme that facilitates the issuance and validation of RCs and ROAs, and (2) the Domain Name System (DNS). DNS is used by RPKI to resolve PP locations (using the SIA field in certificates and the TAL URIs). Importantly, failures of the PKI and/or DNS mechanisms may cause failures of RPKI. We demonstrate this in Appendix H, which shows how a single wrong DNS answer can cause a benign relying party (router enforcing ROV) to drop a legitimate BGP announcement.

To address this dependency of RPKI on PKI and DNS, we present and analyze BGP_{RPKI} , the RPKI protocol, as using two corresponding *blackbox subprotocols*: one for DNS and one for PKI. The DNS subprotocol has to satisfy the DNS correctness requirement $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, defined in §I, i.e., correct resolution of a domain name in a bounded time $\Delta[\text{'DNS'}]$; and the PKI subprotocol has to satisfy the $\mathcal{R}^{\text{PKI}}$ requirement, which is the conjunction of the PKI requirements defined in [84]: *existential-unforgeability, accountability, revocation and transparency*. See Figure 1.

Using protocol composition, we can simplify the analysis of BGP_{RPKI} without ignoring key security aspects of the underlying infrastructure. Hence, we can analyze whether a PKI/DNS implementation satisfies the blackbox requirements (and thus benefits from modularity and simplicity), and separately whether RPKI satisfies the connectivity requirement assuming such blackbox protocols. Due to the secure composability properties proved in [83], composing *any* PKI/DNS

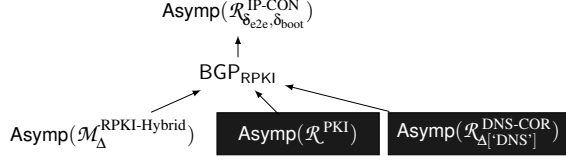


Figure 1: The BGP_{RPKI} protocol asymptotically-satisfies the IP connectivity requirement, $R_{e2c, \delta_{boot}}^{IP-CON}$, assuming the asymptotic $M_{\Delta}^{RPKI-Hybrid}$ model, and subprotocols that asymptotically satisfy the R^{PKI} and the $R_{\Delta}^{DNS-COR}$ blackbox requirements.

protocols that satisfy the blackbox requirements with RPKI yields a protocol that ensures connectivity.

7.3 Failures Against Scheduling-Only Adversary

Theorem 1 shows that BGP ensures IP connectivity against a scheduling-only adversary. We now show the somewhat surprising result that in certain scenarios, the BGP_{RPKI} protocol *fails to ensure IP connectivity* against this scheduling-only adversary, i.e., an adversary that cannot control (corrupt) any AS, and can only control scheduling and delays. Later in this section, we identify assumptions (models) that suffice to avoid this ‘anomaly’, i.e., under which BGP_{RPKI} does ensure IP connectivity.

We first recall how RPKI objects are fetched, for the different RPKI hosting models. Fetching is crucial; if an RPKI RP is unable to obtain some RPKI objects, then the RP may produce an incomplete set of VRPs, which could cause BGP routers to make wrong decisions in the ROV process.

Under both the hosted and hybrid models, RPKI objects are published in the RIRs’ repositories, so RPs can compile a complete set of VRPs by accessing these repositories. In contrast, under the delegated model, fetching is more complicated: RPKI objects are distributed among multiple PPs hierarchically. Fetching RPKI objects in the delegated model depends on routing - the very infrastructure that RPKI protects. This creates a *circular dependency* – and an additional attack surface.

Let us see how RPKI may break connectivity - even for a scheduling-only adversary. Consider an announcement α for a prefix p , which is covered by two valid ROAs, R_1 and R_2 , where α is valid considering both but invalid considering only R_1 (different origin or too-specific). An ROV router aware only of R_1 may ignore α since it would be invalid. This can occur under the delegated model, where the parent contains only the child’s resource certificate. Retrieving the child’s ROAs may then itself rely on insecure routing paths, or, worse, on announcements that the RP already classified as Invalid and dropped. Figure 2 illustrates this concern with an “ROA deadlock”: a valid super-prefix ROA inadvertently

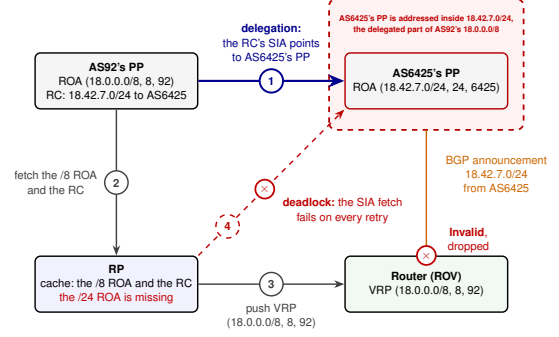


Figure 2: The ROA deadlock in RPKI’s delegated model. The addresses and AS numbers are illustrative, not measured. AS92 delegates 18.42.7.0/24 to AS6425, issuing a certificate whose SIA points to AS6425’s PP (1), which sits inside this delegated /24. The RP fetches AS92’s PP (2) and pushes the resulting VRP to its router (3). From that moment the router classifies AS6425’s announcement of 18.42.7.0/24 as Invalid. AS92 does not announce the /8 itself, using the ROA only to prevent hijacks, so that announcement is the only route into the /24. The RP therefore cannot follow the SIA (4), and every retry fails, so it never learns the ROA that would make the announcement Valid.

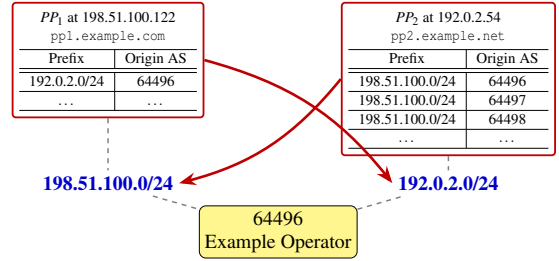


Figure 3: A circular dependency between two PPs (measured August 2026). Addresses and ASNs are anonymized. PP_1 publishes the ROA for PP_2 ’s prefix 192.0.2.0/24, and PP_2 publishes the ROAs for PP_1 ’s prefix 198.51.100.0/24. Thus, on an RP’s initial fetch, neither PP’s protecting ROA is known, leaving both vulnerable to hijacking. Dashed lines link each PP to its covering prefix and both prefixes to their originating AS; red arrows map PPs to the prefixes their ROAs cover.

breaks connectivity to a sub-prefix, which prevents fetching the sub-prefix’s ROA, because that ROA is hosted at a PP within the invalidated sub-prefix. Such scenarios arise from the deployment itself rather than from an engineered attack (§2).

Further, we found instances of PPs with circular dependencies. That is, PP_1 holds the ROAs that cover the IP prefixes of PP_2 , and PP_2 also holds the ROAs for PP_1 (see Figure 3). This creates an absurd scenario in which an RP needs to contact PP_1 to retrieve the ROA for PP_2 , which can only happen after the RP has contacted PP_2 to retrieve the ROA for PP_1 .

In summary, obtaining RPKI objects today involves satisfying a complex dependency chain between objects and repository location. Hence, we start our analysis of RPKI in §7 under the hybrid model (which also covers the hosted model). We begin with a *weak scheduler* adversary that controls only message scheduling and delivery and can thus control which ROAs each RP holds when an announcement arrives, turning such conflicts into deliberate disconnections. We then progress to adversaries that also corrupt ASes and originate BGP announcements. In §9, we further discuss the delegated deployment model and suggest RPKI improvements based on our analysis. Formal analysis of RPKI under the delegated model is left for future work.

7.4 Hybrid Model Assumptions

We now define model assumptions for RPKI in the hybrid model as a conjunction of the following sub-predicates:

$$\mu_{f,\Delta}^{\text{RPKI-Hybrid}} \equiv \left\{ \begin{array}{l} \mu_{f,\Delta}^{\text{BGP}} \wedge \mu^{\text{RPKIHybrid}} \wedge \mu^{\text{RIRDeploy}} \wedge \\ \mu_{\Delta[\text{'orig'}]}^{\text{RPKIInit}} \wedge \mu_{\Delta[\text{'conf'}]}^{\text{NoROACConflict}} \wedge \\ \mu_{\Delta[\text{'ask'}]}^{\text{ASK}} \wedge \pi_{\Delta[\text{'clk'}],\Delta[\text{'wake'}]}^{\text{WakeAt}} \end{array} \right\} \quad (3)$$

As expressed in Equation (3), the RPKI hybrid model adds additional assumptions to those captured by $\mu_{f,\Delta}^{\text{BGP}}$. We now explain these assumptions and their rationale.

Key Management and Hosting. We assume that all CAs manage their own RCs and secret keys, while RIRs host certificates via their PPs, formalized in the $\mu^{\text{RPKIHybrid}}$ model predicate (Algorithm 67).

RIRs Deployment. Based on an empirical analysis we conducted on how the RIR-level PPs are deployed today (see Appendix J for the details), we make the following assumptions and capture them in the $\mu^{\text{RIRDeploy}}$ model predicate (Algorithm 69): (1) Each RIR hosts its TA root certificate, PP, and RP on the same AS, (2) All of these critical locations are covered by valid, exact-match ROAs, (3) No conflicting ROA exists for those critical IPs.

Infrastructure Reachability. We assume that every benign entity hosting a TAL, a PP, or an RP originates a prefix covering that address within $\Delta[\text{'orig'}]$ of its initialization and never withdraws it, captured by $\mu_{\Delta[\text{'orig'}]}^{\text{RPKIInit}}$ (Algorithm 68). Otherwise, the adversary could cause disconnection by exploiting the unreachability of this infrastructure.

ROA Conflicts. For $(p, \text{AS}_{\text{Dst}})$ and every benign RP's self-originated prefix covering its RP address, we do not forbid conflicting ROAs. Instead, we require a validating ROA to be published prior to any potentially conflicting ROA. Otherwise, the disconnection is obtained by a conflicting ROA, that is by CAs misbehaving, which lies outside the scheduler-adversary model. This model is formalized in $\mu_{\Delta[\text{'conf'}]}^{\text{NoROACConflict}}$ (Algorithm 65). The $\Delta[\text{'conf'}]$ sets the required lead time between the publication of the validating ROA and that of any conflicting one.

Bounded Blackbox Interaction. We assume the $\mu_{\Delta[\text{'ask'}]}^{\text{ASK}}$ model from [83], which ensures that the interaction between BGP_{RPKI} and the blackbox protocols is bounded by $\Delta[\text{'ask'}]$, the delay from a protocol's *ask* until the adversary invokes the operation, bounding a single hand-off.

Entities' Corruption. Many aspects of RPKI can prevent RPs from retrieving ROAs from PPs, and hence cause security failures (§2). In this work, we do not consider RP and PP failures and assume benign, bug-free RP and PP implementations. We consider different adversarial models, gradually moving from a scheduler adversary to one capable of launching prefix hijacks.

Scheduled operations. We reuse $\pi_{\Delta[\text{'clk'}],\Delta[\text{'wake'}]}^{\text{WakeAt}}$ from [30]. This predicate ensures timely scheduled operations. If an operation's output includes $(\text{'WakeAt'}, t)$, a WAKEUP operation is invoked at the requesting entity within $\Delta[\text{'wake'}]$ time from the output time.

7.5 Connectivity under the Hybrid Model

We show that RPKI in hybrid mode satisfies $\mathcal{R}_{\delta_{e2e}, \delta_{boot}}^{\text{IP-CON}}$ against a scheduler adversary under model $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}} \equiv \{(\mu_{0,\Delta}^{\text{RPKI-Hybrid}}, 0)\}$, matching the baseline of BGP.

THEOREM 2 (BGP_{RPKI} satisfies $(\delta_{e2e}, \delta_{boot})$ -IP connectivity under a scheduler adversary). *Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{boot}}^{\text{IP-CON}}$ requirement assuming the $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}}$ model holds asymptotically and blackbox protocols that asymptotically satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, provided that:*

$$\begin{aligned} \delta_{e2e} &\geq c_{\text{prop}} \\ \Delta[\text{'conf'}] &\geq \Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2 \cdot c_{\text{prop}} \\ \delta_{boot} &\geq \Delta[\text{'orig'}] + c_{\text{prop}} + 2\Delta[\text{'conf'}] \end{aligned}$$

Proof. See Appendix F.3. $\square$

8 BGP with RPKI Against a Prefix Adversary

In §7 we showed that RPKI can guarantee IP connectivity against a scheduler adversary if additional RPKI assumptions are made. We now analyze BGP_{RPKI} under an adversary that can also originate prefix hijacks. In §L, we show that RPKI does not guarantee IP connectivity against such adversary, because the adversary can launch *misrouting attacks*, i.e., force the active route onto a plausible path that might not be fully ROV-adopting. Interestingly, we found that RPKI is susceptible to such attacks regardless of the deployment model and distribution of ROAs. That is, even when we assume that every ROV-adopting AS already holds a valid ROA for the originated prefix, the adversary could still successfully launch these misrouting attacks. Surprisingly, these attacks remain

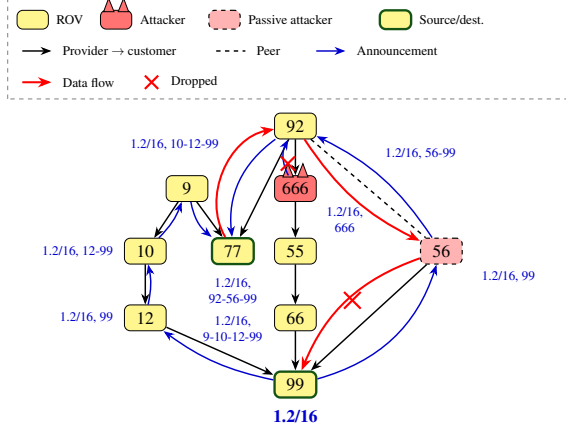


Figure 4: Prefix hijack under full ROV adoption. The ROV adopter AS92 rejects AS666’s hijack of AS99’s prefix 1.2/16. This causes AS92 to fall back to a valid but less-preferred path via its peer AS56, announcing this new route to AS77. AS77 prefers this new, shorter route and steer its traffic away from the honest best path. Connectivity fails because AS56 acts as a passive malicious AS and drops the rerouted traffic.

effective even under ubiquitous ROV adoption. Figure 4 gives an example: AS77 fails to reach AS99 even though every AS adopts ROV and holds AS99’s ROA. The adopter AS92 correctly rejects AS666’s hijack, but falls back to a less-preferred path through AS56, a corrupted AS that discards the rerouted traffic. This is within the $\mu_{\text{Prefix}}^{\text{HijackAdv}}$ model that constrains only the adversary’s BGP announcements (Algorithm 45), so a corrupted AS on the forwarding path may still drop packets.

Further attacks and analysis described in §L allowed us to identify the additional assumptions that need to be made for RPKI to guarantee the IP connectivity requirement. We now define and motivate these additional assumptions using the following model predicate:

$$\mu_{f,\Delta}^{\text{HonestROVPref}} \equiv \left\{ \begin{array}{l} \mu_{f,\Delta}^{\text{BGP}} \wedge \mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{HonestPaths}} \wedge \mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{AdoptedBP}} \wedge \\ \mu_{\text{Prefix}}^{\text{HijackAdv}} \wedge \mu_{\delta_{2e}}^{\text{ROA-InCache}} \end{array} \right\} \quad (4)$$

The $\mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$ (Algorithm 47) predicate ensures all plausible paths are benign, so the attacker cannot simply sit on the path and drop traffic. The $\mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$ (Algorithm 48) ensures that all ASes on the best paths are ROV-adopting. The $\mu_{\text{Prefix}}^{\text{HijackAdv}}$ predicate (Algorithm 45) restricts the adversary’s announcements to prefix hijacks and rules out subprefix hijacks and corrupted CAs. The $\mu_{\delta_{2e}}^{\text{ROA-InCache}}$ predicate (Algorithm 63) ensures that the adopters actually obtain the ROAs they need in time (prior to $t - \delta_{2e}$), allowing to focus on prefix attacks that do not target the RPKI infrastructure. In Section 9, we further discuss why obtaining these ROAs in time is challenging in today’s deployment and propose solutions.

However, an adversary can still hijack a packet in flight by timing the malicious announcement to arrive just after

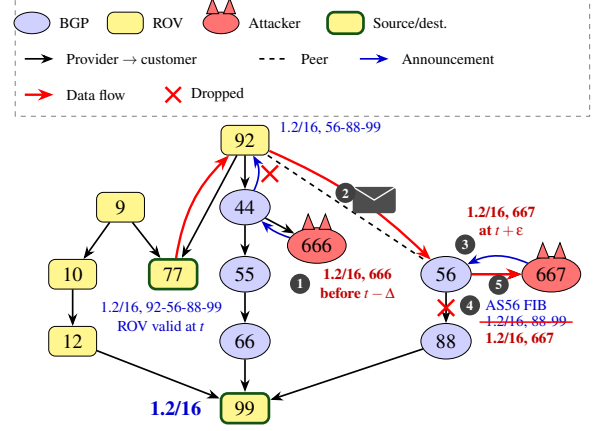


Figure 5: Why Theorem 3 assumes stable routing. $\mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$ holds since attacker ASes are not on any valley-free path from AS77 to AS99. (1) Before the stable routing window, AS666’s hijack forces AS77 onto honest path 77-92-56-88-99. (2) AS77 sends packet at t . (3) At $t + \epsilon$, AS667 announces 1.2/16 to AS56. (4) Lacking ROV, AS56 prefers this new customer route. (5) AS56 diverts the arriving packet to AS667. Had AS667 announced earlier, AS92 would discard it as invalid, routing AS77 via 9-10-12-99.

the packet is sent, as illustrated in Figure 5. Therefore, we prove that BGP_{RPKI} satisfies the stable routing connectivity requirement $\mathcal{R}_{\delta_{2e},\delta_{\text{boot}}}^{\text{SR-IP-CON}}$ against a prefix adversary under model $\mathcal{M}_{f,\Delta}^{\text{HonestROVPref}} \equiv \{(\mu_{f,\Delta}^{\text{HonestROVPref}}, 0)\}$.

THEOREM 3 (BGP_{RPKI} satisfies $(\delta_{2e}, \delta_{\text{boot}})$ -SR-IP Connectivity under a prefix adversary assuming available ROA). *Let G be an AS topology graph. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{2e},\delta_{\text{boot}}}^{\text{SR-IP-CON}}$ requirement assuming the $\mathcal{M}_{f,\Delta}^{\text{HonestROVPref}}$ model holds asymptotically and black-box protocols that asymptotically satisfy requirements $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta}^{\text{DNS-COR}}$, provided that $\delta_{2e} \geq \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{‘com’}] + \Delta[\text{‘clk’}])$ and $\delta_{\text{boot}} \geq 0$.*

Proof. See Appendix F.4. $\square$

However, even under these restrictions, an adversary can still hijack or intercept communication via *subprefix* attacks. The reason is that under partial deployment, even under $\mu_{\text{AS}_{\text{Src}},\text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$, an ROV adopter’s traffic may still traverse non-adopting intermediaries where an adversary can hijack it, although the adopter itself never selects an invalid announcement as a route; see §L.2.

9 Ensuring RPKI’s Repository Reachability

Our analysis shows the importance of successful interactions with RPKI repositories. Thus, we examine the current RPKI deployment, identifying reachability challenges (§9.1), and

propose mitigation compatible with the current RPKI (§9.2). We then prove that, under hybrid deployment, our proposal guarantees stable routing IP connectivity, and we evaluate it empirically (§9.3) using a prototype called *RobustRP*.

9.1 Challenges

Even under full deployment, rogue ASes can disrupt connectivity (§8), so topology and adoption highly affect what RPKI can guarantee. We now highlight two issues with RPKI PPs.

9.1.1 ROA and ROV Coverage of RPKI PPs

We crawled the global RPKI repository tree using Routinator [61], starting from the five RIRs, resolving 80 of the 83 PP hostnames we identified to 110 IPv4 addresses covered by 69 ROAs. About half of the infrastructure RPKI depends on is not protected by the mechanisms RPKI itself provides: only 42 of these 80 PPs have all their addresses correctly covered by a ROA, and only 43 sit in an AS that at least one dataset in *rov_collector* [19] reports as ROV-adopting, consistent with a recent study [45]. A PP without a correctly configured ROA is exposed to the same subprefix hijacks RPKI exists to prevent. See §K for methodology and further results.

9.1.2 Single Points of Failure in RPKI Data

Although the RPKI framework is distributed worldwide, each CA publishes its RPKI data under a single publication point name [78]. Although RFCs allow redundancy via additional PPs specified in the SIA field, as of June 2026, our measurements show that none of the publicly available RCs use this as an alternative. Some efforts to increase data availability include supporting a legacy rsync backup or using CDNs. Such attempts still might fail. If an RP cannot reach the PP, it has no alternative hostname to try. It attempts to connect to every IP address returned by DNS resolution, but a resolution failure, DNS poisoning, or a prefix hijack can still prevent access. Furthermore, relying on CDNs can complicate ensuring ROA coverage at each endpoint [45]. When retrieval fails, RPs typically fall back to the legacy rsync service, which is often hosted on the same server as the primary RRDP service [58], creating a single point of failure.

9.2 Improving Repository Reachability

We next propose two extensions to RPs and two operational recommendations to enhance RPKI and ensure connectivity. Our techniques are compatible with the RPKI specification and can be deployed incrementally today.

9.2.1 Initial RP Cache

Shipping the RP software with only the TAL files results in a bootstrap procedure where the RP, while fetching the RIR

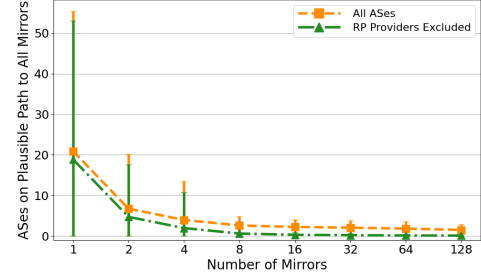


Figure 6: The number of ASes on a plausible path to all repository mirrors. The count is plotted against the number of mirrors, giving the average and 95% confidence intervals over 1,000 simulation runs.

certificates and later accessing their PPs, is exposed to prefix hijacks that could disable ROV entirely. Thus, our first proposal is to modify the initial RP cache to include the root certificates and the ROAs that cover the RIR-level PPs (along with TAL files). This would allow to secure the connection to RIRs’ PPs using ROV. Following the initial bootstrap, updated certificates can be fetched through the existing RPKI. This requires minor changes to RP software and introduces a small overhead for RIRs. Since the cached content remains relatively static, the overhead for RPs is fairly reasonable. Specifically, four RIRs have certificate validity of two or ten years (only RIPE uses a shorter validity period of 3 months), while the relevant ROAs are typically valid for about 90 days.

9.2.2 Redundancy Through Shared Mirroring

Our second proposal is to use shared mirroring to facilitate repository redundancy. Since RPKI objects are signed, a mirror need not be trusted to serve them, so any reachable mirror suffices, and an adversary must disrupt the paths to all mirrors at once to withhold a PP’s data. Sharing the same mirrors across PPs increases the redundancy of every participating PP, so the deployment cost is split among PPs.

With a sufficient number of mirrors, we expect that an attacker’s options are severely limited. To verify this conjecture, we simulated (see §M.1) an RP and a set of mirrors on an Internet-scale topology [9] using the BGP simulator [20], and measured how many ASes are on a plausible path between the RP and *all* mirrors, and can therefore cut it off from all of them at once.

The simulation results are in Figure 6. The “All ASes” line includes all ASes on the path; the “RP Providers Excluded” line omits the RP’s direct providers. With only one mirror/repository, on average, 20 ASes lie on a plausible path between the RP and that mirror, though with large variance depending on RP and mirror selection. As the number of mirrors increases, the number of ASes on a plausible path from the RP to all mirrors decreases. If direct providers are excluded, it approaches zero, indicating that the only ASes

remaining on a plausible path to all mirrors are the RP’s direct providers. This is expected, as a stub or multi-homed AS will always route through its direct providers regardless of mirror placement. Beyond 8 mirrors shows negligible difference.

Multiple methods can support mirroring in the current RPKI. The first is to utilize the SIA field, which allows, per RFC 6487 [33], a list of redundant mirrors for the same PP. This method names each mirror in the certificate; we acknowledge that it increases the overhead associated with introducing new PPs and certifying new RCs with updated SIA fields, as well as higher maintenance overhead for the PPs. The second option is to leverage the CDNs already deployed: prior work reports that 46% of PPs use one [58], and we identify 33 of the 83 PPs we surveyed as CDN-fronted. This does not conflict with §9.1: a CDN supplies the address diversity that mirroring needs, but it does not by itself supply ROA coverage for every edge address, so a CDN-fronted PP is a usable mirror set only once each edge prefix carries a valid ROA.

9.2.3 Operational Recommendations

Our first recommendation is to secure PP delegation. We propose that a parent PP host both the child PP’s CA certificate and the ROAs for the child PPs (analogous to distributing public keys in DNSSEC). Thus, any retrieval from a child PP benefits from ROV protection. While this will impose additional overhead, mechanisms such as the parent CA generating the necessary ROAs to cover the child PP make it manageable. The above mechanisms, together with extending the initial RP cache (§9.2.1), aim to ensure that at every step of the traversal down the RPKI hierarchy, the RP already holds the relevant ROA before it connects. Thus, they mitigate the limitations of Inter-PP dependencies.

Our second recommendation is to improve PP and RP security through several practices. While previous works have largely focused on protecting DNS as a critical component [1, 30], both PPs and RPs are equally critical infrastructure, and their operators should treat them as such in security design and maintenance. First, a simple and effective defense is to use the longest-allowed IP prefixes (/24 in IPv4) when possible, as this foils subprefix hijacks. Second, operators should cover both PPs and RPs with valid ROAs. Third, they should host both in ROV-adopting ASes, protecting communication in both directions.

9.3 Analysis, Prototype and Evaluation

We formally analyze our proposals by adding two models. The first ensures safe cache initialization at the RP, including the required ROA to safely bootstrap the retrieval process. The second is shared mirroring, which adds redundancy under the assumption that at least one mirror is well-connected. Under both assumptions and the $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}}$ model, we prove that the RPKI retrieval process is guaranteed (see Lemma 5 in

Publication point	Baseline	RobustRP	Gain	VRP share
rpki-rrdp.us-east-2.amazonaws.com	12	73	+61	2.18%
rrdp.paas.rpki.ripe.net	4	29	+25	} 35.67%
rrdp.ripe.net	4	29	+25	
rrdp.lacnic.net	2	4	+2	2.42%
Total	22	135	+113	40.3%

Table 1: IP pool expansion per publication point and corresponding VRP share, enhancing the resilience of 40.3% of the global VRP set and discovering 113 more candidate IPs than the baseline Routinator.

§N), providing the same assumptions used in Theorem 3. This results in Theorem 4, showing that our proposal allows RPKI to guarantee stable routing IP connectivity under $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Robust}} \equiv \{(\mu_{f,\Delta}^{\text{Hybrid-Robust}}, 0)\}$, while assuming an adversary model capable of launching prefix hijacks. See §N.

THEOREM 4 (BGP_{RPKI} satisfies $(\delta_{e2e}, \delta_{boot})$ -SR-IP connectivity under the Robust model). *Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{‘com’}] + \Delta[\text{‘clk’}]), c_{\text{fetch}} \equiv \Delta[\text{‘wake’}] + \Delta[\text{‘DNS’}] + 2\Delta[\text{‘ask’}] + 2 \cdot c_{\text{prop}}$. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{boot}}^{\text{SR-IP-COR}}$ requirement assuming the $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Robust}}$ model holds asymptotically and blackbox protocols that asymptotically satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{‘DNS’}]}^{\text{DNS-COR}}$, provided that:*

$$\begin{aligned}
\delta_{e2e} &\geq c_{\text{prop}} \\
\Delta[\text{‘conf’}] &\geq \Delta[\text{‘mirror’}] \\
\Delta[\text{‘dest’}] &\geq \delta_{e2e} + \Delta[\text{‘mirror’}] + c_{\text{fetch}} \\
\delta_{boot} &\geq \Delta[\text{‘orig’}] + \Delta[\text{‘mirror’}] \\
&\quad + \Delta[\text{‘dest’}] + (\text{diam}_{\text{VF}}(G) + 1) \cdot c_{\text{fetch}}
\end{aligned}$$

Evaluation. We evaluate *RobustRP*, our enhanced RP prototype following the guidelines of §9.2. Across the PPs listed in Table 1, RobustRP discovers 135 candidate addresses compared to the baseline RP’s 22, enhancing the resilience of 40.3% of the global verified ROA payload set. The larger pool enables a fallback mechanism: when a connection to a PP fails, it tries the next address until it succeeds or exhausts the pool, extending redundancy through existing infrastructure as advised in §9.2.2 (Appendix O).

10 Conclusion

We present the first rigorous security analysis of RPKI. Through this analysis, we find key challenges to obtaining security benefits from RPKI in practice, including the circular dependency on IP connectivity and the need for ROV along all plausible paths between source and destination. We propose protocol-compatible mechanisms and evaluate RobustRP, a prototype implementing the client-side ones.

Acknowledgments

This work was supported by grant no. 2022701 from the United States-Israel Binational Science Foundation (BSF), by the Research Authority Fund of the College of Management Academic Studies, Rishon LeZion, Israel, by grants 2247810, 2410268, and 2149765 of the National Science Foundation, and by Dr. Herzberg's endowment from Comcast. The opinions expressed in the paper are those of the researchers and not of their institutions or funding sources.

Ethical Considerations

Our measurements in §9.1 and §J.3 revealed vulnerabilities regarding PP prefixes (i.e., either no ROA or an ROA with a misconfigured maxLength). Before submitting this work, we notified the operators of the affected PPs by email using contact information published in RDAP and WHOIS.

Open Science

We provide *RobustRP*, our fork of Routinator [61] described in Appendix O. We make its full source code publicly available at <https://github.com/shaycohen988/routinator-multi-resolver> [15].

References

- [1] American Registry for Internet Numbers (ARIN). Number resource policy manual (NRPM). <https://www.arin.net/participate/policy/nrpm/>, 2026. Version 2026.1. Accessed: June 10, 2026.
- [2] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Basnett. Investigating interdomain routing policies in the wild. In *Proceedings of the 2015 Internet Measurement Conference*, pages 71–77, 2015.
- [3] R. Austein, G. Huston, S. Kent, and M. Lepinski. Manifests for the Resource Public Key Infrastructure (RPKI). RFC 9286 (Proposed Standard), June 2022. URL: <https://www.rfc-editor.org/rfc/rfc9286.txt>, doi:10.17487/RFC9286.
- [4] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M Reischuk, and Pawel Szalachowski. The SCION internet architecture. *Communications of the ACM*, 60(6):56–65, 2017.
- [5] Alexandra Boldyreva and Robert Lychev. Provable security of S-BGP and other path vector protocols: model, analysis and extensions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 541–552, 2012.
- [6] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein. The RPKI Repository Delta Protocol (RRDP). RFC 8182 (Proposed Standard), July 2017. URL: <https://www.rfc-editor.org/rfc/rfc8182.txt>, doi:10.17487/RFC8182.
- [7] R. Bush. Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI). RFC 7115 (Best Current Practice), January 2014. URL: <https://www.rfc-editor.org/rfc/rfc7115.txt>, doi:10.17487/RFC7115.
- [8] R. Bush and R. Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. RFC 8210 (Proposed Standard), September 2017. URL: <https://www.rfc-editor.org/rfc/rfc8210.txt>, doi:10.17487/RFC8210.
- [9] CAIDA. CAIDA Serial 2 Data Set, April 2022. URL: <https://publicdata.caida.org/datasets/as-relationships/serial-2/>.
- [10] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001. Online at <https://eprint.iacr.org/2000/067.pdf>, last updated Dec. 2018.
- [11] Taejoong Chung, Emile Aben, Tim Bruijnzeels, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Roland van Rijswijk-Deij, John Rula, and Nick Sullivan. RPKI is Coming of Age: A Longitudinal Study of RPKI Deployment and Invalid Route Origins. In *Proc. of IMC*. ACM, 2019.
- [12] Cloudflare. Is BGP safe yet?, 2023. URL: <https://isbgpsafeyet.com/>.
- [13] Inc. Cloudflare. Routing statistics worldwide, 2026. Cloudflare Radar. URL: <https://radar.cloudflare.com/routing/rpki#routing-statistics>.
- [14] Avichai Cohen, Yossi Gilad, Amir Herzberg, and Michael Schapira. Jumpstarting BGP Security with Path-End Validation. In *Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16*, page 342–355, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2934872.2934883.
- [15] Shay Cohen. RobustRP: A multi-resolver fork of Routinator, 2026. URL: <https://github.com/shaycohen988/routinator-multi-resolver>.
- [16] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. On the Risk of Misbehaving RPKI Authorities. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII*, pages 16:1–16:7, New York, NY, USA, 2013. ACM. URL: <http://doi.acm.org/10.1145/2535771.2535787>, doi:10.1145/2535771.2535787.
- [17] J. Durand, I. Pepelnjak, and G. Doering. BGP Operations and Security. RFC 7454 (Best Current Practice), February 2015. URL: <https://www.rfc-editor.org/rfc/rfc7454.txt>, doi:10.17487/RFC7454.

- [18] Yosef Edery, Justin Furuness, Jie Kong, Nicholas Scaglione, Hemi Leibowitz, Amir Herzberg, Bing Wang, and Yossi Gilad. Suppressing BGP zombies with route status transparency, 2025.
- [19] Justin Furuness. `rov_collector`: Collects ROV data for ASes. https://github.com/jfuruness/rov_collector, 2023. GitHub repository.
- [20] Justin Furuness, Cameron Morris, Reynaldo Morillo, Amir Herzberg, and Bing Wang. BGPpy: The BGP Python Security Simulator. In *Proc. of Cyber Security Experimentation and Test (CSET)*, 2023.
- [21] Justin Furuness, Cameron Morris, Reynaldo Morillo, Arvind Kasiliya, Bing Wang, and Amir Herzberg. Securing BGP ASAP: ASPA and other post ROV defenses. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, San Diego, CA, USA, February 2025. doi:10.14722/ndss.2025.240675.
- [22] Lixin Gao and Jennifer Rexford. Stable Internet Routing without Global Coordination. *IEEE/ACM Trans. Netw.*, 9(6):681–692, 2001.
- [23] Y. Gilad, S. Goldberg, K. Sriram, J. Snijders, and B. Maddison. The Use of maxLength in the Resource Public Key Infrastructure (RPKI). RFC 9319 (Best Current Practice), October 2022. URL: <https://www.rfc-editor.org/rfc/rfc9319.txt>, doi:10.17487/RFC9319.
- [24] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. Are we there yet? On RPKI’s deployment and security. In *NDSS. The Internet Society*, 2017. URL: https://www.researchgate.net/publication/309379229_Are_We_There_Yet_On_RPKI's_Deployment_and_Security.
- [25] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, apr 1984.
- [26] Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.
- [27] Amir Herzberg, Hemi Leibowitz, Ewa Syta, and Sara Wrótniak. MoSS: Modular Security Specifications framework. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 33–63, Cham, 2021. Springer International Publishing. Full version at: <https://eprint.iacr.org/2020/1040>.
- [28] Tom Hlavacek, Philipp Jeitner, Donika Mirdita, Haya Shulmann, and Michael Waidner. Staloris: RPKI downgrade attack. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association. URL: <https://www.usenix.org/conference/usenixsecurity22/presentation/hlavacek>.
- [29] Tomas Hlavacek, Philipp Jeitner, Donika Mirdita, Haya Schulmann, and Michael Waidner. Beyond limits: How to disable validators in secure networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*. ACM, 2023. doi:10.1145/3603269.3604861.
- [30] Tomas Hlavacek, Philipp Jeitner, Donika Mirdita, Haya Shulman, and Michael Waidner. Behind the scenes of RPKI. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS ’22*, page 1413–1426, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3560645.
- [31] Tomas Hlavacek, Haya Schulmann, Niklas Vogel, and Michael Waidner. Keep your friends close, but your routeservers closer: Insights into RPKI validation in the internet. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4841–4858. USENIX Association, 2023.
- [32] R. Housley. Cryptographic Message Syntax (CMS). RFC 5652 (Internet Standard), September 2009. Updated by RFC 8933. URL: <https://www.rfc-editor.org/rfc/rfc5652.txt>, doi:10.17487/RFC5652.
- [33] G. Huston, G. Michaelson, and R. Loomans. A Profile for X.509 PKIX Resource Certificates. RFC 6487 (Proposed Standard), February 2012. Updated by RFCs 7318, 8209. URL: <https://www.rfc-editor.org/rfc/rfc6487.txt>, doi:10.17487/RFC6487.
- [34] IPinfo. CDN ASN tag, 2025. URL: <https://ipinfo.io/tags/cdn>.
- [35] IPinfo. Cloud ASN tag, 2025. URL: <https://ipinfo.io/tags/cloud>.
- [36] S. Kent and K.seo. An Infrastructure to Support Secure Internet Routing. RFC 6480, The Internet society, February 2012. URL: <http://tools.ietf.org/html/rfc6480>.
- [37] Stephen T. Kent, Charles Lynn, and Karen Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.
- [38] APNIC Labs. Rpki, 2023. URL: <https://stats.labs.apnic.net/rpki>.
- [39] P. Lapukhov, A. Premji, and J. Mitchell (Ed.). Use of BGP for Routing in Large-Scale Data Centers. RFC 7938 (Informational), August 2016. URL:

- <https://www.rfc-editor.org/rfc/rfc7938.txt>, doi:10.17487/RFC7938.
- [40] M. Lepinski, A. Chi, and S. Kent. Signed Object Template for the Resource Public Key Infrastructure (RPKI). RFC 6488 (Proposed Standard), February 2012. URL: <https://www.rfc-editor.org/rfc/rfc6488.txt>, doi:10.17487/RFC6488.
 - [41] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational), February 2012. URL: <https://www.rfc-editor.org/rfc/rfc6480.txt>, doi:10.17487/RFC6480.
 - [42] M. Lepinski, S. Kent, and D. Kong. A Profile for Route Origin Authorizations (ROAs). RFC 6482 (Proposed Standard), February 2012. URL: <https://www.rfc-editor.org/rfc/rfc6482.txt>, doi:10.17487/RFC6482.
 - [43] Weitong Li, Yuze Li, and Taejoong Chung. ImpROV: Measurement and practical mitigation of collateral damage in RPKI route origin validation. In *Proc. of USENIX Security*, August 2025.
 - [44] Weitong Li, Zhexiao Lin, Md. Ishtiaq Ashiq, Emile Aben, Romain Fontugne, Amreesh Phokeer, and Taejoong Chung. RoVista: measuring and analyzing the route origin validation (ROV) in RPKI. In *ACM IMC*, 2023.
 - [45] Yunhao Liu, Jessie Hui Wang, Yuedong Xu, Zongpeng Li, Yangyang Wang, and Jilong Wang. Crack in the armor: Underlying infrastructure threats to RPKI publication point reachability. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2026.
 - [46] Robert Lychev, Sharon Goldberg, and Michael Schapira. BGP security in partial deployment: Is the juice worth the squeeze?, July 10 2013. URL: <http://arxiv.org/abs/1307.2690>.
 - [47] C. Lynn, S. Kent, and K. Seo. X.509 Extensions for IP Addresses and AS Identifiers. RFC 3779 (Proposed Standard), June 2004. URL: <https://www.rfc-editor.org/rfc/rfc3779.txt>, doi:10.17487/RFC3779.
 - [48] M. Lepinski (Ed.) and K. Sriram (Ed.). BGPsec Protocol Specification. RFC 8205, sep 2017. URL: <https://tools.ietf.org/html/rfc8205>.
 - [49] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. A cross-protocol attack on the TLS protocol. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 62–72. ACM, 2012. URL: <http://dl.acm.org/citation.cfm?id=2382196>.
 - [50] Donika Mirdita, Haya Schulmann, and Michael Waidner. Poster: Exploring the landscape of RPKI relying parties. pages 4773–4775, 11 2025. doi:10.1145/3719027.3760721.
 - [51] Donika Mirdita, Haya Schulmann, and Michael Waidner. SoK: An introspective analysis of RPKI security. In *USENIX Security Symposium (USENIX Security)*, pages 3649–3665. USENIX Association, August 2025.
 - [52] Donika Mirdita, Haya Shulman, Niklas Vogel, and Michael Waidner. The CURE to vulnerabilities in RPKI validation. In *Proceedings of the 2024 Network and Distributed System Security (NDSS) Symposium*, 2024.
 - [53] Donika Mirdita, Haya Shulman, and Michael Waidner. Poster: RPKI kill switch. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22*, page 3423–3425, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3563536.
 - [54] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. BGP Prefix Origin Validation. RFC 6811 (Proposed Standard), January 2013. Updated by RFCs 8481, 8893. URL: <https://www.rfc-editor.org/rfc/rfc6811.txt>, doi:10.17487/RFC6811.
 - [55] Reynaldo Morillo, Justin Furuness, Cameron Morris, James Breslin, Amir Herzberg, and Bing Wang. ROV++: Improved deployable defense against BGP hijacking. In *USENIX Network and Distributed System Security (NDSS) Symposium*, 2021.
 - [56] Cameron Morris, Amir Herzberg, Bing Wang, and Samuel Secondo. BGP-iSec: improved security against Post-ROV routing attacks (full version). https://www.researchgate.net/publication/375553362_BGP-iSec_Improved_Security_of_Internet_Routing_Against_Post-ROV_Attacks, 2023.
 - [57] Cameron Morris, Amir Herzberg, Bing Wang, and Samuel Secondo. BGP-iSec: Improved security of internet routing against post-ROV attacks. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*. The Internet Society, 2024. URL: <https://www.ndss-symposium.org/wp-content/uploads/2024-1035-paper.pdf>.
 - [58] Moritz Müller-Brus, Lisa Bruder, Caspar Schutijser, and Ralph Koning. A first look at common RPKI publication practices, 12 2025. doi:10.48550/arXiv.2512.16369.
 - [59] National Institute of Standards and Technology (NIST). NIST RPKI Monitor, version 2.0. <https://>

- rpki-monitor.antd.nist.gov/. Accessed November 2023.
- [60] National Institute of Standards and Technology (NIST). NIST RPKI Monitor. <https://rpki-monitor.antd.nist.gov/>, 2026. Accessed: 2026-04-06.
 - [61] NLnet Labs. Routinator: An RPKI validator and RTR server, 2017. URL: <https://github.com/NLnetLabs/routinator>.
 - [62] Chris Parker. Your 'multihop' BGP session probably isn't multihop. APNIC Blog, January 2023. URL: <https://blog.apnic.net/2023/01/20/your-multihop-bgp-session-probably-isnt-multihop>.
 - [63] PeeringDB. PeeringDB: The interconnection database, 2025. URL: <https://www.peeringdb.com>.
 - [64] Aaron Peters and Sajal Kayan. CDN planet: CDN provider index, 2025. URL: <https://www.cdnplanet.com/cdns-by-country/>.
 - [65] Y. Rekhter (Ed.), T. Li (Ed.), and S. Hares (Ed.). A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705, 8212, 8654, 9072. URL: <https://www.rfc-editor.org/rfc/rfc4271.txt>, doi:10.17487/RFC4271.
 - [66] A. Retana, R. White, V. Fuller, and D. McPherson. Using 31-Bit Prefixes on IPv4 Point-to-Point Links. RFC 3021 (Proposed Standard), December 2000. URL: <https://www.rfc-editor.org/rfc/rfc3021.txt>, doi:10.17487/RFC3021.
 - [67] Andreas Reuter, Randy Bush, Ítalo Cunha, Ethan Katz-Bassett, Thomas C Schmidt, and Matthias W"ahlich. Towards a rigorous methodology for measuring adoption of RPKI route validation and filtering. *ACM SIGCOMM Computer Communication Review*, 48(1):19–27, 2018.
 - [68] RIPE Network Coordination Centre. RIPE NCC Statistics. <https://www.ripe.net/analyse/statistics>, 2016.
 - [69] Nils Rodday, Ítalo S. Cunha, Randy Bush, Ethan Katz-Bassett, Gabi Dreo Rodosek, Thomas C. Schmidt, and Matthias W"ahlich. Revisiting RPKI route origin validation on the data plane. In Vaibhav Bajpai, Hamed Haddadi, and Oliver Hohlfeld, editors, *5th Network Traffic Measurement and Analysis Conference, TMA 2021, Virtual Event, September 14-15, 2021*, pages 1–14, Virtual Event, 2021. IFIP. URL: <http://dl.ifip.org/db/conf/tma/tma2021/tma2021-paper11.pdf>.
 - [70] rpki-client Project. rpki-client console: VRP and RPKI object browser, 2026. 2026-07-12. URL: <https://console.rpki-client.org>.
 - [71] Haya Schulmann and Niklas Vogel. Pruning the tree: Rethinking RPKI architecture from the ground up. In *Proc. of NDSS*, February 2026.
 - [72] Haya Schulmann, Niklas Vogel, and Michael Waidner. Are we actually there? assessing RPKI maturity. *Communications of the ACM*, 69(5):78–87, 2026.
 - [73] secynic. ipwhois, 2024. URL: <https://github.com/secynic/ipwhois>.
 - [74] Kris Shrishak and Haya Shulman. Privacy preserving and resilient RPKI. In *IEEE INFOCOM*. IEEE, 2021.
 - [75] Haya Shulman, Niklas Vogel, and Michael Waidner. Poster: Insights into global deployment of RPKI validation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22*, page 3467–3469, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3563523.
 - [76] Job Snijders, Ben Maddison, Matt Lepinski, Derrick Kong, and Stephen Kent. A Profile for Route Origin Authorizations (ROAs). RFC 9582, May 2024. URL: <https://www.rfc-editor.org/info/rfc9582>, doi:10.17487/RFC9582.
 - [77] Vinay K Sriram and Doug Montgomery. Design and analysis of optimization algorithms to minimize cryptographic processing in BGP security protocols. *Computer communications*, 106:75–85, 2017.
 - [78] Yingying Su, Dan Li, Li Chen, Qi Li, and Sitong Ling. dRR: A decentralized, scalable, and auditable architecture for RPKI repository. In *Proc. of NDSS*, February 2024.
 - [79] U.S. Department of the Interior. Cloud service providers, 2025. URL: <https://www.doi.gov/cloud/portfolio/CSPs>.
 - [80] Koen van Hove, Jeroen van der Ham-de Vos, and Roland Rijswijk-Deij. rpkiller: Threat analysis of the BGP resource public key infrastructure. *Digital Threats: Research and Practice*, 4, 08 2023. doi:10.1145/3617182.
 - [81] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA*, pages 1313–1328. ACM, 2017. URL: <http://doi.acm.org/10.1145/3133956>.

- [82] S. Weiler, A. Sonalker, and R. Austein. A Publication Protocol for the Resource Public Key Infrastructure (RPKI). RFC 8181 (Proposed Standard), July 2017. URL: <https://www.rfc-editor.org/rfc/rfc8181.txt>, doi:10.17487/RFC8181.
- [83] Sara Wrótniak, Hemi Leibowitz, Ewa Syta, and Amir Herzberg. CMoSS: Composible Modular Security Specifications. <https://eprint.iacr.org/2026/1190>, 2026.
- [84] Sara Wrótniak, Hemi Leibowitz, Ewa Syta, and Amir Herzberg. Provable security for PKI schemes. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS, CCS '24*, pages 1552–1566. Association for Computing Machinery, 2024. Full version at <https://eprint.iacr.org/2019/807>. doi:10.1145/3658644.3670374.

A Deployment Models

Figure 7 illustrates RPKI’s deployment models.

B List of Bounds

Table 2 lists all the bounds that are used in the paper.

Notation	Purpose
$\Delta[\text{'com'}]$	Bound on message delay.
$\Delta[\text{'clk'}]$	Bound on clock drift.
$\Delta[\text{'wake'}]$	Bound on the interval between consecutive wake-up operations.
$\Delta[\text{'ask'}]$	Bound on the time it takes for composed protocols to interact.
$\Delta[\text{'orig'}]$	Bound on the time it takes to originate a covering BGP prefix for the RPKI infrastructure.
$\Delta[\text{'DNS'}]$	Bound on the delay from a DNS query to its resolution response.
$\Delta[\text{'mirror'}]$	Bound on the replication time between mirrors.

Table 2: List of bounds used in the paper.

C Δ -IP Connectivity Requirement Predicates

The $\rho_{\delta_{2e}, \delta_{boot}}^{\text{IP-CON}}$ predicate is in Algorithm 1 and $\rho_{\delta_{2e}, \delta_{boot}}^{\text{SR-IP-CON}}$ predicate is in Algorithm 2.

D Mid-Flight Attack

We now detail an attack that arises from packet propagation delays, which we call a *mid-flight prefix hijack*. It is the precisely timed data-plane hijack referenced in Section 5, and it motivates the Stable Routing condition of our connectivity requirement.

This attack defeats the ROV enforcement of the sending AS by exploiting the interval during which a packet traverses a single link. Figure 8 illustrates the scenario: AS99 legitimately originates 1.2/16, the adversary AS666 hijacks that

prefix, and AS77, an ROV adopter, sends a packet to an IP address in 1.2/16 along the route 77-56-88-99. Because AS77 enforces ROV, the route it selects at the time of sending has an RPKI-valid origin. Once the packet leaves AS77, the sender no longer controls the forwarding decisions that follow: each AS on the path matches the packet against its own Forwarding Information Base (FIB) at the time the packet arrives (Algorithm 14). AS666 therefore times its hijack announcement to reach AS56, the next hop of AS77, while the packet is still in flight on the AS77-AS56 link. AS56 does not adopt ROV, so it accepts the announcement and installs the hijacked route in its FIB before the packet arrives. AS56 then forwards the packet to AS666, even though AS77 validated the route it selected.

The attack is difficult to perform in practice: the adversary must place its announcement at AS56 within the propagation delay of a single link, and repeating the attack produces multiple announcements that alert network administrators.

The attack is visible only in an analysis that models the delay between the sending of a packet and its arrival at the next hop. Prior analyses of prefix hijacks evaluate a fixed snapshot of the routing graph and of the relationships among ASes [24, 55], so a hijack announcement and a packet in flight never coexist in their model. In CMoSS [83], the adversary decides when each operation runs and when each message is delivered, so it can schedule the hijack announcement to arrive at AS56 in the interval between the send event at AS77 and the receive event at AS56. Our connectivity requirement therefore carries the Stable Routing condition, which denies the adversary that scheduling freedom. We found this attack while attempting to prove the IP connectivity requirement, not by inspecting the topology.

E BGP Implementation

E.1 Entities and State

The protocol is run by BGP routers, where each router represents a single AS. The set of all entities (BGP routers) is denoted by $\mathcal{N}$ and each entity maintains the following local state st :

- $st.i$: A unique identifier of the entity.
- $st.asn$: The AS number.
- $st.peers$: Maps the unique identifier j of each neighboring AS to the tuple $(rel, asn_j, ip_{src}, ip_{dst})$. Fields are read by name, for example $st.peers[j].rel$. Here $rel \in \{\text{Customer}, \text{Peer}, \text{Provider}\}$ denotes the AS relationship, asn_j is the neighbor’s ASN, ip_{src} is the IP address of the local router’s interface, and ip_{dst} is the IP address of the neighboring router on the shared subnet.
- $st.ipBlocks$: The set of IP prefixes assigned to this AS.

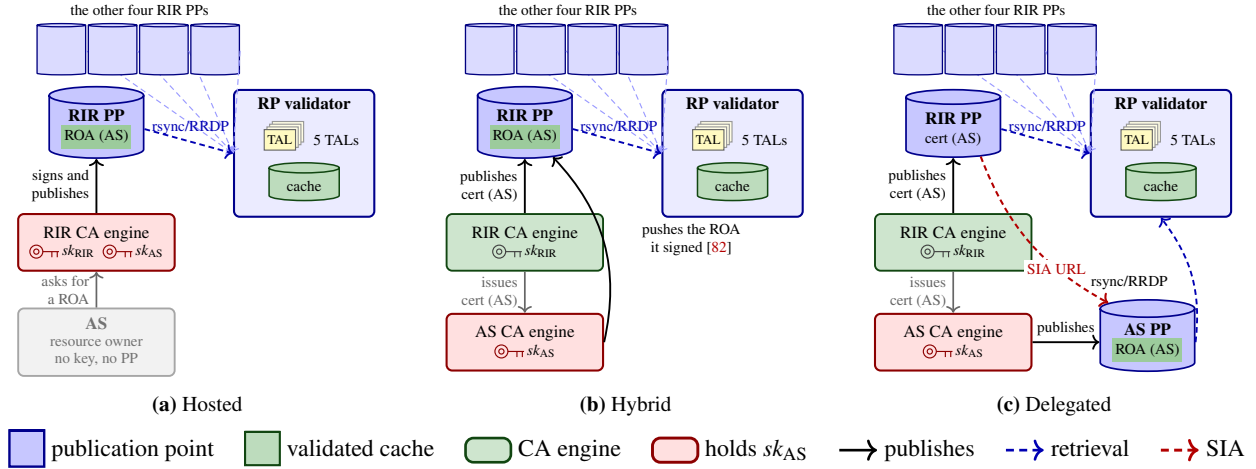


Figure 7: The three RPKI deployment modes. Every relying party starts from its five TALs, retrieves the five RIR publication points (upper left) over rsync or RRDp and builds its validated cache. (a) Hosted: the RIR holds both keys, so the AS operates nothing and one retrieval returns everything. (b) Hybrid: the AS holds sk_{AS} but pushes the signed ROA into the RIR’s PP, so the key moves and the retrieval does not. (c) Delegated: the AS also operates its own PP, named only by the SIA URL in cert (AS), the only link to it; when that retrieval fails, the validator caches no ROA for the AS.

Algorithm 1 $\rho_{\delta_{e2e}, \delta_{boot}}^{\text{IP-CON}}$: $(\delta_{e2e}, \delta_{boot})$ -IP Connectivity Requirement

Input: Transcript T .

Output: $\perp$ if the adversary demonstrates that AS_{src} sent a specific packet (identified by pkt_id) at time t to an IP address legitimately announced by the set of honest destinations AS_{dsts} , which was announced at least δ_{boot} after the initialization of every benign entity, but no AS in AS_{dsts} received that packet by time $t + \delta_{e2e}$; otherwise, $\top$

```

1: requirement  $\rho_{\delta_{e2e}, \delta_{boot}}^{\text{IP-CON}}(T)$ 
2:  $(AS_{src}, AS_{dsts}, ip, p, pkt\_id, t, label, aux) \leftarrow T.out_A$ 
    $\left[ \begin{array}{l} AS_{src} \in T.N - T.F \wedge \\ \emptyset \neq AS_{dsts} \subseteq T.N - T.F \wedge \\ ip \subseteq p \wedge \\ \text{BOOTSTRAP ELAPSED}(T, AS_{dsts}, p, \delta_{boot}) \wedge \\ \text{ORIGINATE MOST SPECIFIC PREFIX}(T, t - \delta_{e2e}, AS_{dsts}, ip, p) \wedge \\ \text{PREFIX NOT WITHDRAWN}(T, t + \delta_{e2e}, AS_{dsts}, p) \wedge \\ \text{PACKET SENT}(T, t, AS_{src}, ip, pkt\_id) \wedge \\ \text{PACKET NOT RECEIVED}(T, t + \delta_{e2e}, AS_{dsts}, ip, pkt\_id) \end{array} \right]$ 
3: if
4:   return  $\perp$ 
5: end if
6: return  $\top$ 
7: end requirement

```

$\triangleright$ Extract adversary's output
 $\triangleright$ Sender is benign
 $\triangleright$ A destination exists and benign
 $\triangleright$ Bootstrap is done before announcing (Alg. 3)
 $\triangleright$ Dest establishes valid route (Alg. 4)
 $\triangleright$ Route remains available (Alg. 5)
 $\triangleright$ Source attempts delivery of packet (Alg. 6)
 $\triangleright$ But exact packet never arrives (Alg. 7)
 $\triangleright$ Adversary wins (connectivity broken)
 $\triangleright$ Adversary fails (connectivity maintained)

- $st.RIB_{in}$: Maintains the raw routing information received from peers, mapping each peer j and destination prefix p to its latest set of path attributes α (see Appendix E.2). Routes that are self-originated are kept in the same table, under the entity’s own identifier $st.i$.
- $st.locRIB$: Stores the best route for each prefix p , mapping it to the attributes α selected after processing the RIB_{in} through import policies and the decision algorithm.

- $st.RIB_{out}$: Maintains the outgoing routing information for each peer, mapping each peer j and destination prefix p to the path attributes α of the most recently advertised route after applying the export policy.
- $st.policy_{in}, st.policy_{out}$: The routing policies applied to incoming and outgoing routes, respectively, controlling how routes are imported and exported.
- $st.fib$: Maps each IP prefix to the next hop used by the data plane for matching packets. A next hop is either

Algorithm 2 $\rho_{\delta_{e2e}, \delta_{boot}}^{\text{SR-IP-CON}}$: Stable Routing ($\delta_{e2e}, \delta_{boot}$)-IP Connectivity Requirement

Input: Transcript T .

Output: $\perp$ if the adversary demonstrates that AS_{Src} sent a specific packet (identified by pkt_id) at time t to an IP address legitimately announced by the set of honest destinations AS_{Dsts} , which was announced at least δ_{boot} after the initialization of every benign entity, but no AS in AS_{Dsts} received that packet by time $t + \delta_{e2e}$, provided that routing is stable; otherwise, $\top$

```

1: requirement  $\rho_{\delta_{e2e}, \delta_{boot}}^{\text{SR-IP-CON}}(T)$ 
2:    $(\text{AS}_{\text{Src}}, \text{AS}_{\text{Dsts}}, \text{ip}, p, \text{pkt\_id}, t, \text{label}, \text{aux}) \leftarrow T.out_{\mathcal{A}}$  ▷ Extract adversary's output
3:   if  $\neg \rho_{\delta_{e2e}, \delta_{boot}}^{\text{IP-CON}}(T)$  and  $\text{STABLEROUTING}(T, t, \text{ip}, \delta_{e2e})$  then ▷ Connectivity broken (Alg. 1) despite stable routing (Alg. 8)
4:     return  $\perp$  ▷ Adversary wins (connectivity broken)
5:   end if
6:   return  $\top$  ▷ Adversary fails (connectivity maintained or routing not stable)

7: end requirement

```

Algorithm 3 Ensures AS_{Dsts} announced p at least δ_{boot} after every benign entity was initialized.

```

1: procedure BOOTSTRAPELAPSED( $T, \text{AS}_{\text{Dsts}}, p, \delta_{boot}$ )
2:   return  $\forall \hat{e}_i, \hat{e}_o$  s.t.
      $T.opr[\hat{e}_i] = \text{'Init'} \wedge T.ent[\hat{e}_i] \in T.N - T.F \wedge$ 
      $T.opr[\hat{e}_o] = \text{'ORIGINATE'} \wedge T.ent[\hat{e}_o] \in \text{AS}_{\text{Dsts}} \wedge$ 
      $T.inp[\hat{e}_o].p = p : \quad \triangleright \text{bootstrap finished before the announcement}$ 
      $T.\tau[\hat{e}_o] \geq T.\tau[\hat{e}_i] + \delta_{boot}$ 
3: end procedure

```

Algorithm 4 Ensures AS_{Dsts} announced the most specific prefix p for ip by time t .

```

1: procedure ORIGINATEMOSTSPECIFICPREFIX( $T, t, \text{AS}_{\text{Dsts}}, \text{ip}, p$ )
2:   return  $(\forall \text{AS} \in \text{AS}_{\text{Dsts}}, \exists \hat{e}$  s.t. ▷ All origins in the set announced  $p$ 
3:      $T.ent[\hat{e}] = \text{AS} \wedge$ 
4:      $T.opr[\hat{e}] = \text{'ORIGINATE'} \wedge$ 
5:      $T.inp[\hat{e}].p = p \wedge$ 
6:      $T.\tau[\hat{e}] \leq t) \wedge$ 
7:      $(\nexists \hat{e}'$  s.t. ▷ No honest AS announced a strictly more specific prefix for  $\text{ip}$ 
8:        $T.ent[\hat{e}'] \in T.N - T.F \wedge$ 
9:        $T.opr[\hat{e}'] = \text{'ORIGINATE'} \wedge$ 
10:       $T.\tau[\hat{e}'] \leq t \wedge$ 
11:       $\text{ip} \subseteq T.inp[\hat{e}'].p \subset p)$ 
12: end procedure

```

Algorithm 5 Ensures no origin in the destination set withdrew the prefix p by time t .

```

1: procedure PREFIXNOTWITHDRAWN( $T, t, \text{AS}_{\text{Dsts}}, p$ )
2:   return  $\nexists \hat{e}$  s.t.
3:      $T.ent[\hat{e}] \in \text{AS}_{\text{Dsts}} \wedge$  ▷ Event issued by any AS in  $\text{AS}_{\text{Dsts}}$ 
4:      $T.opr[\hat{e}] = \text{'Withdraw'} \wedge$  ▷ Operation is Withdraw
5:      $T.inp[\hat{e}].p = p \wedge$  ▷ Withdrawn prefix is exactly  $p$ 
6:      $T.\tau[\hat{e}] \leq t$  ▷ Withdrawal happened by  $t$ 
7: end procedure

```

the IP address of a neighboring router, or one of the values *Self* and *LocalNetwork*, which mark a destination inside this AS that the router delivers locally instead of forwarding.

- *st.LocalMap*: Maps the next-hop IP addresses of adjacent peers to their corresponding neighbor identifiers j ,

Algorithm 6 Ensures the source sent successfully

```

1: procedure PACKETSENT( $T, t, \text{AS}_{\text{Src}}, \text{ip}, \text{pkt\_id}$ )
2:   return  $\exists \hat{e}$  s.t.
3:      $T.ent[\hat{e}] = \text{AS}_{\text{Src}} \wedge$  ▷ Event issued by  $\text{AS}_{\text{Src}}$ 
4:      $T.opr[\hat{e}] = \text{'SendPacket'} \wedge$  ▷ Operation is SendPacket
5:      $T.inp[\hat{e}].pkt.dstip = \text{ip} \wedge$  ▷ Destination IP matches  $\text{ip}$ 
6:      $T.inp[\hat{e}].pkt.pkt\_id = \text{pkt\_id} \wedge$  ▷ Specific packet ID matches
7:      $T.\tau[\hat{e}] = t$  ▷ Send occurred exactly at  $t$ 
8:      $T.out[\hat{e}] \neq \perp$  ▷ Send was successful
9: end procedure

```

Algorithm 7 Ensures no origin in the destination set received the packet by time t .

```

1: procedure PACKETNOTRECEIVED( $T, t, \text{AS}_{\text{Dsts}}, \text{ip}, \text{pkt\_id}$ )
2:   return  $\nexists \hat{e}$  s.t.
3:      $T.ent[\hat{e}] \in \text{AS}_{\text{Dsts}} \wedge$  ▷ Event issued by any AS in  $\text{AS}_{\text{Dsts}}$ 
4:      $T.opr[\hat{e}] = \text{'ReceivePacket'} \wedge$  ▷ Operation is ReceivePacket
5:      $T.inp[\hat{e}].pkt.dstip = \text{ip} \wedge$  ▷ Destination IP matches  $\text{ip}$ 
6:      $T.inp[\hat{e}].pkt.pkt\_id = \text{pkt\_id} \wedge$  ▷ Specific packet ID matches
7:      $T.\tau[\hat{e}] \leq t$  ▷ Received by deadline  $t$ 
8: end procedure

```

Algorithm 8 Ensures no announcement or withdrawal of a prefix covering ip occurs within δ_{e2e} window of sending event at time t .

```

1: procedure STABLEROUTING( $T, t, \text{ip}, \delta_{e2e}$ )
2:   return  $\nexists \hat{e} \in \{1, \dots, T.e\}$  s.t.
3:      $T.opr[\hat{e}] \in \{\text{'ORIGINATE'}, \text{'Withdraw'}\} \wedge$  ▷ Scan for routing updates
4:      $T.\tau[\hat{e}] \in [t - \delta_{e2e}, t + \delta_{e2e}] \wedge$  ▷ within the connectivity window
5:      $\text{ip} \subseteq T.inp[\hat{e}].p$  ▷ targeting the relevant IP
6: end procedure

```

representing the underlying physical topology of point-to-point links.

Both *st.RIB_{in}* and *st.RIB_{out}* store only the most recent route, since RFC 4271 [65] treats previous updates from the same neighbor for the same prefix as implicitly withdrawn.

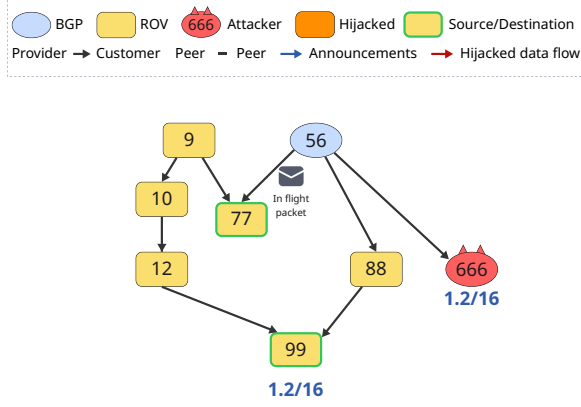


Figure 8: A mid-flight prefix hijack. A hijack announcement from AS666 reaches AS56 while the packet is in flight on the AS77-AS56 link. The resulting FIB update at AS56 redirects the packet to AS666, although AS77 enforces ROV.

E.2 Simplifications

Single Node AS We treat each AS as a single node, ignoring internal routing details. This standard abstraction is found in foundational works like [22] and [26], and is widely adopted by BGP simulators (e.g., [20]).

Initialization, Direct Peering Assumption, and Processing Time BGP operates over TCP/IP, using the NEXT_HOP attribute to identify the next router. Typically, different ASes (eBGP peers) share a direct physical link or a common subnet [65]. Although not explicitly mandated by the RFC, vendors enforce this proximity with a default TTL of 1 to mitigate spoofing attacks [62].⁵ Furthermore, establishing a peering requires a TCP handshake and stateful session maintenance, which rely on a manual bootstrapping phase where operators configure local interfaces, allocate IP subnets, and explicitly define peers for physical links. Because our security analysis focuses on control-plane route propagation and data-plane packet delivery in an adversary-driven environment where the adversary dictates the initial state, we abstract away the session establishment phase and strictly constrain the adversary’s configuration choices. Specifically, we mandate that physical peering subnets adhere to secure point-to-point topologies using /31 prefixes, as recommended by RFC 3021 [66], and communication between adjacent peers is secured by the $\mu_{\Delta[\text{com}]}^{\text{Com}}$ model. We further assume that BGP routers propagate received announcements instantly, without any scheduled events. This means that upon a change to the local routing table, the router propagates the change to its neighbors without any caching delay or scheduled propagation.

⁵Multihop eBGP is possible via manual configuration and is often used for load balancing [39].

Routes, Decision Process, and Policies A BGP router advertises a *route* to a neighboring peer to share network reachability information. Each route comprises an IP prefix (or a set of prefixes) together with path attributes that provide detailed information about the path to those destinations. During an internal decision process, the router evaluates these attributes according to local policy, ranks route preferences, and ultimately selects the best route for a given prefix.

While the BGP decision process is formally defined in RFC 4271 [65], it is mostly influenced by the *localPref* attribute which is determined by local routing policies. Because our implementation grants the adversary control over these policies under some basic assumptions, the adversary already possesses a significant degree of freedom. Therefore, to avoid needlessly complicating the formal security analysis, we abstract away several BGP attributes. Other than some optional attributes, we also omit the mandatory Origin attribute and the decision process’s reliance on external intra-domain state (such as OSPF metrics), since we do not consider intra-AS routing.

We use a simplified route structure where each route contains a single prefix p and its associated path attributes α , represented as a tuple (p, α) , where the route attributes α are defined as:

- *nextHop*: the IP address of the adjacent router used for packet forwarding.
- *asPath*: ordered sequence of ASNs traversed to reach p .
- *localPref*: local preference value used to rank multiple paths to p . Note that localPref is for local use only and is not propagated across ASes.

While this simplified structure captures the necessary mechanics for our proofs, our implementation can be easily extended to support further attributes if required.

E.3 Scheme Implementation

We now present the formal implementation of BGP, which implements the IDR scheme defined in Definition 2.

BGP.INIT Alg. 9 This operation initializes the local state of an AS. It first checks whether st has already been initialized and aborts if so (Line 2). Otherwise, it saves the identity of the entity (Line 3) and sets the local AS number (Line 4). The function then sets the FIB and LocalMap to empty (Lines 5 and 6).

It then populates the peer relationships by iterating over the network links provided in $params.\mathcal{N}$ (Lines 7 and 8). During this iteration, it maps the physical subnet directly to the neighbor’s IP (Line 10), and updates the LocalMap to link neighbor IPs to their unique identifiers (Line 11).

Following this, the function sets the import and export policies (Line 13) and the set of IP blocks owned by this AS

(Line 14). The function proceeds to reset all routing information bases to empty (Lines 15–17). Finally, it returns the fully initialized st (Line 18).

BGP.ORIGINATE Alg. 10 This operation originates a prefix p that lies within the address space that the AS owns. It first checks that the AS owns the requested prefix; that is, that there exists $b \in st.ipBlocks$ with $p \subseteq b$, and returns without action if no such block exists (Line 2). If allowed, it constructs a new route with the AS path set to $[st.asn]$ and the next hop set to $LocalNetwork$ (Line 3)⁶. It then stores this self originated route under $st.RIB_{in}[st.i][p]$ (Line 4) and calls the decision process described below (Line 5). The procedure finally returns the updated state and any resulting outbound messages (Line 6).

BGP.WITHDRAW Alg. 11 The BGP.WITHDRAW operation withdraws a previously originated prefix. It checks whether that prefix was previously originated (i.e., if p is present in $st.RIB_{in}$ under its own entity identifier $st.i$). If it is not, the function returns without effect (Line 2). If the prefix is present, the entry is cleared (Line 3), and the decision process is invoked to update the RIBs and propagate the withdrawal (Line 4). The procedure then returns the updated local state and any resulting messages (Line 5).

BGP.SETPOLICY Alg. 12 The BGP.SETPOLICY operation updates the local routing policies of an AS. It receives a policy object specifying inbound and outbound rules, and replaces the existing policies in st with the new ones (Line 2). It then returns the updated local state (Line 3).

BGP.SENDPACKET Alg. 13 The BGP.SENDPACKET operation forwards a packet to the relevant adjacent AS. It looks up the packet’s destination IP in the FIB, using a Longest Prefix Match (LPM). If the next hop is *Self* or *LocalNetwork*, the operation is rejected and returns $\perp$ (Line 4). Otherwise, it repeats the lookup (Line 7) until it holds an address that the LocalMap maps to a neighbor. It drops the packet if a lookup ends unsuccessfully, or if the result is *Self* or *LocalNetwork* (Line 9). It then finds the relevant neighbor from the LocalMap (Line 13) and outputs an instruction to send the packet (Line 14).

BGP.RECEIVEPACKET Alg. 14 This operation processes incoming packets, determining whether to deliver them locally or to forward them. It first performs an LPM lookup on the packet’s destination IP (Line 3). If the resulting next hop is *Self* or *LocalNetwork*, the destination lies within this AS,

and the function hands the packet to PACKETROUTER (Algorithm 17) for local delivery (Line 4). Otherwise it treats the packet as transit traffic and calls BGP.SENDPACKET (Line 8).

In addition to the above BGP operations, several internal logic mechanisms are critical to the operation of the BGP system. We provide an overview of these core processes:

Update Alg. 15 The Update function processes incoming BGP updates. It first unpacks the prefix, route, and action from the message, and rejects the update when the sender is not a known neighbor (Line 2). For an announcement, it validates that the next hop matches the sender’s IP and that the path is loop free, rejecting the update otherwise (Line 4), and stores the accepted route in $st.RIB_{in}$ (Line 7). For a withdrawal, it sets the corresponding entry to $\perp$ (Line 9). It rejects any other action (Line 11). The function then calls the decision process for the affected prefix to reconsider the best route (Line 13), and returns the updated local state and any resulting messages (Line 14).

DECISION Alg. 16 The DECISION function selects the best route for a given prefix from all available candidates (Line 2). If no candidate remains and a route was previously installed, the function clears the entry in $st.locRIB$ (Line 5) and the corresponding forwarding entry in $st.fib$ (Line 6), then propagates the withdrawal to its neighbors and returns the updated state and messages. If candidates exist, the function computes the best route (Line 12) and returns without further action when that route matches the one already installed. Otherwise it installs the new route in $st.locRIB$, writes its next hop to $st.fib$ (Line 17), and calls the propagation process to inform its neighbors (Line 18). The function returns the updated local state and any resulting messages (Line 19).

Policy Utilities Alg. 18 This procedure evaluates routing policies. ApplyPolicy iterates through each policy rule, checking conditions (Lines 16, 19, 21 and 23) and applying actions (Lines 29–37). Rules may immediately accept or reject the route, or modify it and continue. If no rule matches, the route is rejected by default (Lines 23 and 42). EvalCond supports checking receiver or sender relationships (Lines 16 and 19), unconditional match (Line 21), or unknown conditions (Line 23). ExecActs executes actions including reject, accept, continue, set local preference, and prepend ASN (Lines 29–37); unknown actions trigger immediate rejection (Line 39), and if no explicit decision occurs, the modified route is returned (Line 42).

F Formal Proofs

In this appendix, we provide the full formal proofs that were omitted or sketched in the main body of the paper.

⁶This represents a “local” next hop, telling the router that the packet has reached its destination and should be delivered internally instead of being sent to another AS.

Algorithm 9 BGP.INIT: Initialization

```

1: operation BGP.INIT( $st, params$ )
2:   if  $st \neq \perp$  then return  $\perp$  ▷ Skip if already initialized
3:    $st.\mathbf{1} \leftarrow params.\mathbf{1}$  ▷ Save identity of the AS entity
4:    $st.asn \leftarrow params.asn$  ▷ Set AS number
5:    $st.fib \leftarrow \emptyset$ 
6:    $st.LocalMap \leftarrow \emptyset$ 
7:   for all  $(j, rel, asn_j, ip_{src}, ip_{dst}, Plink) \in params.\mathcal{A}$  do
8:      $st.peers[j] \leftarrow (rel, asn_j, ip_{src}, ip_{dst})$ 
9:      $st.fib[ip_{src}/32] \leftarrow Self$  ▷ Control Plane (/32)
10:     $st.fib[Plink] \leftarrow ip_{dst}$  ▷ Map physical /31 subnet directly to neighbor IP
11:     $st.LocalMap[ip_{dst}] \leftarrow j$  ▷ Entity map for direct routing
12:   end for
13:    $(st.policy_{in}, st.policy_{out}) \leftarrow params.policy$  ▷ Set inbound and outbound policies
14:    $st.ipBlocks \leftarrow params.ipBlocks$  ▷ Assign IP prefixes owned by this AS
15:    $st.RIB_{in} \leftarrow \perp$  ▷ Raw routes per neighbor and prefix
16:    $st.locRIB \leftarrow \perp$  ▷ Best routes per prefix
17:    $st.RIB_{out} \leftarrow \perp$  ▷ Routes advertised
18:   return  $st$  ▷ Return initialized state
19: end operation

```

Algorithm 10 BGP.ORIGINATE: announce an owned prefix

```

1: operation BGP.ORIGINATE( $st, \perp$ )
2:   if  $\nexists b \in st.ipBlocks : p \subseteq b$  then return  $(st, \perp)$  ▷ Announce only prefixes legitimately owned
3:    $r \leftarrow \{asPath : [st.asn], nextHop: LocalNetwork\}$  ▷ Create route
4:    $st.RIB_{in}[st.\mathbf{1}][p] \leftarrow r$  ▷ Install self-originated route in  $RIB_{in}$ 
5:    $(st, out) \leftarrow DECISION(st, p)$  ▷ Re-evaluate best route via Algorithm 16
6:   return  $(st, out)$  ▷ Return updated state and outgoing messages
7: end operation

```

F.1 Proof of Lemma 1

LEMMA 1. Let $G = (V, E, R)$ be an AS topology graph and denote $c_{prop} \equiv \text{diam}_{VF}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Under model $\mathcal{M}_{\Delta}^{sch}$, if $AS_{Dst} \in V$ originates a prefix p at time t , and AS_{Dst} does not withdraw it before $t + c_{prop}$, then every benign $v \in V$ has installed a route to p with origin AS_{Dst} by time $t + c_{prop}$.

Proof. Denote $c_{prop} = \text{diam}_{VF}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Assume to the contrary that the lemma does not hold. This means that there exists a transcript T where the adversary managed to prevent an AS $v_{\mathcal{A}} \in V$ from installing a route to p with origin AS_{Dst} by time $t + c_{prop}$.

From $\mu^{\text{WellFormedGraph}}$ (Algorithm 42), we know that there exists at least one valley-free path from AS_{Dst} to any $v \in V$. Let $P = (v_0, \dots, v_k)$ be the shortest valley-free path such that $v_0 = AS_{Dst}$ and $v_k = v$. Following Definition 6, $k = d_{VF}(AS_{Dst}, v) \leq \text{diam}_{VF}(G)$. From $\mu^{|F| \leq 0}$, we know that all ASes are benign, and benign ASes only announce a prefix if they own a block covering it (Algorithm 10, Line 2). That, combined with μ^{SHO} , makes AS_{Dst} the only AS that originated p . Since a benign AS extends an AS-Path only by prepending the local ASN (Algorithm 21, line 14), every route to p stored by any AS has origin AS_{Dst} .

Let $t_i = t + i \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. We prove by induction on i that by time t_i , node $v_i \in P$ has installed a route to p in its $locRIB$, and, for $i < k$, its export policy permits sending that route to v_{i+1} .

Algorithm 11 BGP.WITHDRAW: revoke an announced prefix

```

1: operation BGP.WITHDRAW( $st, p$ )
2:   if  $st.RIB_{in}[st.\mathbf{1}][p] = \perp$  then return  $(st, \perp)$  ▷ Nothing to withdraw
3:    $st.RIB_{in}[st.\mathbf{1}][p] \leftarrow \perp$  ▷ Remove self-originated route in  $RIB_{in}$ 
4:    $(st, out) \leftarrow DECISION(st, p)$  ▷ Re-evaluate best route via Algorithm 16
5:   return  $(st, out)$  ▷ Return updated state and withdrawals
6: end operation

```

Algorithm 12 BGP.SETPOLICY: Set local policy

```

1: operation BGP.SETPOLICY( $st, policy$ )
2:    $(st.policy_{in}, st.policy_{out}) \leftarrow policy$  ▷ Update inbound and outbound policies from input
3:   return  $(st, \perp)$  ▷ Return the updated local state
4: end operation

```

Base Case ($i = 0$): At time t , the node $v_0 = AS_{Dst}$ originates p by executing BGP.ORIGINATE (Algorithm 10), which stores the self-originated route in $st.RIB_{in}$ (Line 4) and immediately calls the decision process DECISION (Algorithm 16). GATHERCANDIDATES (Line 2) gathers the RIB-in entries of the neighbors together with that of v_0 itself, so the self-originated route is a candidate; since $\mu^{\text{PreferSelf}}$ gives it the maximal local preference, it is selected by **FINDBESTROUTE** (Line 12) and installed into v_0 's $locRIB$ (Line 17). From μ^{VFexport} (Algorithm 56), a self-originated route is exported to every neighbor, in particular to v_1 when $k \geq 1$, and the time of origination is exactly $t = t_0$.

Inductive Step: Assume the claim holds for v_{i-1} , and let r be the route it has installed by time t_{i-1} ; hence, v_{i-1} exports r to v_i . Because P is valley-free, the relationship between v_{i-1} and v_i adheres to the export conditions of μ^{VFexport} , so r is not filtered by v_{i-1} 's export policy. PROPAGATE was called with r at the step that installed it (Algorithm 16, Lines 17 and 18), so the export policy check (Algorithm 21, Line 11) evaluates to true for v_i , leading to the generation of an update message (Line 18). A routing packet is then invoked (Line 28) and passed to the data plane (Algorithm 13). Because the μ^{Bgplnit} model predicate ensures all local subnets and local topological mappings are properly established prior to execution, the data plane safely performs a recursive LPM lookup (Line 8) to resolve the next-hop IP into a physical *target_entity* from the *LocalMap* (Line 13), eventually sending the packet (Line 14) to v_i .

Thus, v_i processes the announcement (Algorithm 15) within an additional time of $(\Delta[\text{'com'}] + \Delta[\text{'clk'}])$, since $\mu^{\text{Drift}}_{\Delta[\text{'clk'}]}$ ensures up to $\Delta[\text{'clk'}]$ deviation between the clocks and $\mu^{\text{Com}}_{\Delta[\text{'com'}]}$ ensures neighbor communication within $\Delta[\text{'com'}]$, that is, by time $t_{i-1} + (\Delta[\text{'com'}] + \Delta[\text{'clk'}]) = t_i$. If v_i does not appear on the AS-Path of r , the loop guard (Line 4) does not prevent the route from being stored in v_i 's RIB_{in} (Line 7) and DECISION is invoked (Line 13) with a non-empty candidate set, so v_i installs a route to p at that step, unless its $locRIB$ already holds the selected route, in which case it installed one earlier. If instead v_i does appear on that AS path, then v_i exported an

Algorithm 13 BGP.SENDPACKET

```

1: operation BGP.SENDPACKET( $st, pkt$ )
2:    $current\_ip \leftarrow pkt.dst\_ip$ 
3:    $next\_hop \leftarrow LPM(st.fib, current\_ip)$ 
4:   if  $next\_hop \in \{Self, LocalNetwork\}$  then
5:     return ( $st, \perp$ )  $\triangleright$  Outgoing packet cannot target internal state
6:   end if
7:   while  $st.LocalMap(current\_ip) = \perp$  do
8:      $current\_ip \leftarrow LPM(st.fib, current\_ip)$ 
9:     if  $current\_ip = \perp \vee current\_ip \in \{Self, LocalNetwork\}$  then
10:      return ( $st, \perp$ )  $\triangleright$  Network unreachable
11:    end if
12:  end while
13:   $neighbor \leftarrow st.LocalMap(current\_ip)$ 
14:   $out \leftarrow \{('Send', pkt, neighbor)\}$ 
15:  return ( $st, out$ )
16: end operation

```

Algorithm 14 BGP.RECEIVEPACKET

```

1: operation BGP.RECEIVEPACKET( $st, pkt, sender$ )
2:    $current\_ip \leftarrow pkt.dst\_ip$ 
3:    $next\_hop \leftarrow LPM(st.fib, current\_ip)$ 
4:   if  $next\_hop \in \{Self, LocalNetwork\}$  then
5:     ( $st, out$ )  $\leftarrow$  PACKETROUTER( $st, pkt, sender$ )  $\triangleright$  See Algorithm 17
6:     return ( $st, out$ )
7:   else
8:     ( $st, out$ )  $\leftarrow$  BGP.SENDPACKET( $st, pkt$ )  $\triangleright$  See Algorithm 13
9:     return ( $st, out$ )
10:  end if
11: end operation

```

ancestor of r at an earlier step, and had therefore installed a route to p before t_{i-1} . In both cases v_i has installed a route to p , which we denote r' , by time t_i .

For $i < k$, v_i may in turn export r' to v_{i+1} : the three ASes v_{i-1} , v_i and v_{i+1} are consecutive on the valley-free path P , so the relationship between v_i and v_{i+1} adheres to the export conditions of $\mu^{V_{\text{export}}}$ (Algorithm 56), and r' is not filtered by v_i 's export policy. This completes the inductive step.

For $i = k$, node $v_k = v$ has installed a route to p by time $t_k = t + k \cdot (\Delta['com'] + \Delta['clk'])$, and its origin is AS_{Dst} . Since $k \leq \text{diam}_{VF}(G)$, this is no later than $t + \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk']) = t + c_{\text{prop}}$, and since AS_{Dst} does not withdraw p within the window, the route is not retracted at the source before $t + c_{\text{prop}}$.

The above holds for any $v \in V$, and in particular for $v_{\mathcal{A}}$. Therefore, $v_{\mathcal{A}}$ must have installed a route to p with origin AS_{Dst} by time $t + c_{\text{prop}}$, which contradicts the assumption. $\square$

F.2 Proof of Lemma 2

LEMMA 2. *Let $G = (V, E, R)$ be an AS topology graph, ip be an IP address and p be an IP prefix and denote $c_{\text{prop}} \equiv \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$. Under model $\mathcal{M}_{\Delta}^{sch}$, if:*

- p is the most specific prefix covering ip , and
- Benign $AS_{Dst} \in V$ originates p by time $t - c_{\text{prop}}$, and
- Does not withdraw it before $t + c_{\text{prop}}$, and

Algorithm 15 Update: process a received BGP update

```

procedure Update( $st, update, Sender$ )
1: ( $p, r, action$ )  $\leftarrow$  update
2: if  $Sender \notin \text{keys}(st.peers)$  then return ( $st, \perp$ )
3: if  $action = \text{Announce}$  then
4:   if  $r.nextHop \neq st.peers[Sender].ip_{dst} \vee st.asn \in r.asPath$  then
5:     return ( $st, \perp$ )  $\triangleright$  Invalid nextHop or loop detected
6:   end if
7:    $st.RIB_{in}[Sender][p] \leftarrow r$   $\triangleright$  Store attributes for  $p$  from Sender
8: else if  $action = \text{Withdraw}$  then
9:    $st.RIB_{in}[Sender][p] \leftarrow \perp$   $\triangleright$  Mark route as withdrawn in Adj-RIB-In
10: else
11:   return ( $st, \perp$ )
12: end if
13: ( $st, out$ )  $\leftarrow$  DECISION( $st, p$ )  $\triangleright$  Reconsider best route via Algorithm 16
14: return ( $st, out$ )  $\triangleright$  Return updated state and decision result
end procedure

```

Algorithm 16 DECISION: select the best route for p

```

procedure DECISION( $st, p$ )
1:  $out \leftarrow \emptyset$ 
2:  $Candidates \leftarrow \text{GATHERCANDIDATES}(st, p)$   $\triangleright$  see Algorithm 19
3: if  $Candidates = \emptyset$  then
4:   if  $st.locRIB[p] \neq \perp$  then
5:      $st.locRIB[p] \leftarrow \perp$   $\triangleright$  Remove route from local tables
6:      $st.fib[p] \leftarrow \perp$   $\triangleright$  Remove route from fib
7:     ( $st, out$ )  $\leftarrow$  PROPAGATE( $st, p, \perp$ )  $\triangleright$  Tell peers we lost the route, see Algorithm 21
8:   return ( $st, out$ )
9: end if
10: return ( $st, \perp$ )
11: end if
12:  $BestRoute \leftarrow \text{FINDBESTROUTE}(p, Candidates)$   $\triangleright$  See Algorithm 20
13: if  $BestRoute = st.locRIB[p]$  then
14:   return ( $st, \perp$ )  $\triangleright$  No change
15: end if
16:  $st.locRIB[p] \leftarrow BestRoute$ 
17:  $st.fib[p] \leftarrow BestRoute.nextHop$   $\triangleright$  Install new best route
18: ( $st, out$ )  $\leftarrow$  PROPAGATE( $st, p, BestRoute$ )  $\triangleright$  Announce or withdraw to neighbors, see Algorithm 21
19: return ( $st, out$ )  $\triangleright$  Return BestRoute and messages
end procedure

```

- Benign $AS_{Src} \in V$ sends an IP packet identified by pkt_id towards ip at time t

Then AS_{Src} 's packet pkt_id is received by AS_{Dst} by $t + c_{\text{prop}}$.

Proof. Denote $c_{\text{prop}} = \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$. Assume to the contrary that the lemma does not hold. This means that there exists a transcript T where the adversary managed to prevent an AS_{Dst} from receiving the packet identified by pkt_id by time $t + c_{\text{prop}}$.

Let $t_0 \leq t - c_{\text{prop}}$ be the time at which AS_{Dst} originates p . Since AS_{Dst} does not withdraw p before $t + c_{\text{prop}}$, it in particular does not withdraw it before $t_0 + c_{\text{prop}}$, so Lemma 1 applies at t_0 : every $v \in V$ has installed a route to p with origin AS_{Dst} by time $t_0 + c_{\text{prop}} \leq t$. Moreover, under $\mu^{|F| \leq 0}$ every AS is honest and, following μ^{SHO} , AS_{Dst} is the only AS that originates p , so no matter which route a node's decision process installs, its origin is AS_{Dst} .

Every AS therefore holds a forwarding entry for p from time t onward, and p is the entry that the data plane matches for ip : an AS holds a route only for a prefix originated in

Algorithm 17 BGP.PACKETROUTER: demultiplex packets by protocol

```

1: procedure PACKETROUTER( $st, pkt, sender$ )
2:    $out \leftarrow \perp$   $\triangleright$  Init output
3:   switch ( $pkt.protocol$ ) do
4:     case BGP:
5:        $(st, out) \leftarrow \text{Update}(st, pkt.payload, sender)$   $\triangleright$  Pass routing updates to BGP
6:     default:
7:        $out \leftarrow \perp$   $\triangleright$  Drop unsupported protocols
8:   return ( $st, out$ )
9: end procedure

```

T , and p is the most specific such prefix covering ip , so the longest prefix match for ip in any AS's FIB is p .

When AS_{src} sends the packet pkt_id at time t by executing BGP.SENDPACKET (Algorithm 13), the procedure performs a recursive Longest Prefix Match (LPM) lookup (line 8) to resolve the destination IP ip to the covering prefix p and determine the $next\_hop$. The lookup does not fail (line 9), since by the previous paragraph, AS_{src} holds an entry for p in its FIB. Since $\mu^{BgpInit}$ model predicate ensures all local subnets and local topological mappings are properly established prior to execution, the recursive resolution continues until the next hop maps to a physical *target_entity* in the node's *LocalMap* (line 13), at which point the packet is emitted (line 14).

As the packet traverses the network, each intermediate node v processes it via BGP.RECEIVEPACKET (Algorithm 14). For any node where the destination is not local (line 4), the forwarding logic invokes BGP.SENDPACKET to transition the packet to the next neighbor according to v 's local FIB, and the argument of the previous paragraph applies directly to v . At AS_{dst} the match is local, and the packet is handed to PACKETROUTER (Algorithm 17), which is what it means for AS_{dst} to receive pkt_id .

Since AS_{dst} does not withdraw p before $t + c_{prop}$, and since the FIB entry for p is rewritten only when the decision process installs a new best route (Algorithm 16, line 17), which by the first paragraph again has origin AS_{dst} , every node the packet visits forwards it toward AS_{dst} throughout the packet's journey. This from a forwarding path to AS_{dst} of at most $\text{diam}_{VF}(G)$ hops, and each data-plane hop introduces a delay of at most $\Delta['com'] + \Delta['clk']$, since $\mu_{\Delta['com']}^{Com}$ ensures neighbor communication within $\Delta['com']$ and $\mu_{\Delta['clk']}^{Drift}$ ensures up to $\Delta['clk']$ deviation between the two local clocks. The end-to-end data-plane delay is therefore bounded by:

$$\text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk']) = c_{prop}$$

Therefore, AS_{dst} receives pkt_id no later than time $t + c_{prop}$, which contradicts the assumption that a transcript T in which the adversary managed to prevent AS_{dst} from receiving pkt_id by $t + c_{prop}$ exists. $\square$

Algorithm 18 Policy utilities: evaluate conditions and actions

```

procedure ApplyPolicy( $st, policy, r, peer$ )
1: for  $i = 1 \rightarrow |policy|$  do  $\triangleright$  Evaluate rule  $r_i = (C_i, A_i)$  in order
2:   if  $\forall cond \in C_i : \text{EvalCond}(cond, st, r, peer)$  then
3:      $r', continue \leftarrow \text{ExecActs}(A_i, r)$ 
4:     if  $continue = \perp$  then
5:       return  $r'$   $\triangleright$  Accept/Reject decided in this rule
6:     else
7:        $r \leftarrow r'$   $\triangleright$  Preserve modifications for next rule
8:     end if
9:   end if
10: end for
11: return  $\perp$   $\triangleright$  Reject if no rule matches
end procedure

12: procedure EvalCond( $cond, st, r, peer$ )
13:   if  $cond = \text{selfOriginated}$  then
14:     return ( $r.nextHop = \text{LocalNetwork}$ )  $\triangleright$  Route originated locally
15:   else if  $cond = \text{receiverRelationships with params} = p$  then
16:     return ( $st.peers[peer].rel \in p$ )  $\triangleright$  Export receiver's relationship is  $p$ 
17:   else if  $cond = \text{senderRelationships with params} = p$  then
18:      $j \leftarrow st.LocalMap(r.nextHop)$   $\triangleright$  Neighbor that sent the route,  $\perp$  if self-originated
19:     return ( $j \neq \perp \wedge st.peers[j].rel \in p$ )  $\triangleright$  Route was learned from a neighbor of type  $p$ 
20:   else if  $cond = \text{true}$  then
21:     return  $\top$   $\triangleright$  Unconditional match (catch-all)
22:   else
23:     return  $\perp$ 
24:   end if
25: end procedure

26: procedure ExecActs( $A, r$ )
27:   for  $act \in A$  do
28:     if  $act = \text{reject}$  then
29:       return ( $\perp, \perp$ )  $\triangleright$  Reject route
30:     else if  $act = \text{accept}$  then
31:       return ( $r, \perp$ )  $\triangleright$  Accept route
32:     else if  $act = \text{continue}$  then
33:       return ( $r, \top$ )  $\triangleright$  Continue with next rule
34:     else if  $act = \text{setLocalPref with params} = x$  then
35:        $r.localPref \leftarrow x$   $\triangleright$  Set local preference to  $x$ 
36:     else if  $act = \text{prepend with params} = (asn, k)$  then
37:        $r.aspath \leftarrow [asn]^k \parallel r.aspath$   $\triangleright$  Prepend ASN  $asn$   $k$  times
38:     else
39:       return ( $\perp, \perp$ )
40:     end if
41:   end for
42:   return ( $r, \perp$ )
43: end procedure

```

Algorithm 19 GATHERCANDIDATES: collect candidate routes for p

```

1: procedure GATHERCANDIDATES( $st, p$ )
2:    $Candidates \leftarrow \emptyset$ 
3:   for all  $n \in \text{keys}(st.peers) \cup \{st.\perp\}$  do
4:     if  $st.RIB_n[n][p] \neq \perp$  then
5:        $r' \leftarrow \text{ApplyPolicy}(st, st.policy_{in}, st.RIB_n[n][p], n)$   $\triangleright$  Apply local import policy
6:       if  $r' = \perp$  then
7:         continue  $\triangleright$  Skip route rejected by policy
8:       end if
9:        $Candidates.Add(r')$ 
10:     end if
11:   end for
12:   return  $Candidates$   $\triangleright$  Return post-policy candidate set
13: end procedure

```

Algorithm 20 FINDBESTROUTE: rank candidates routes and pick the best

```

1: procedure FINDBESTROUTE( $p, C$ )
2:    $\text{maxPref} \leftarrow \max_{r \in C} r.\text{localPref}$ 
3:    $C \leftarrow \{r \in C \mid r.\text{localPref} = \text{maxPref}\}$   $\triangleright$  Step 1: Prefer highest LocalPref
4:    $\text{minLen} \leftarrow \min_{r \in C} |r.\text{asPath}|$ 
5:    $C \leftarrow \{r \in C \mid |r.\text{asPath}| = \text{minLen}\}$   $\triangleright$  Step 2: Prefer shortest AS-PATH
6:    $\text{minOrig} \leftarrow \min_{r \in C} r.\text{asPath}[0]$ 
7:    $C \leftarrow \{r \in C \mid r.\text{asPath}[0] = \text{minOrig}\}$   $\triangleright$  Step 3: Prefer lowest sender ASN, ties break
8:   return  $r \in C$ 
9: end procedure

```

Algorithm 21 PROPAGATE: send the best route to neighbors

```

1: procedure PROPAGATE( $st, p, r$ )
2:    $\text{out} \leftarrow \emptyset$ 
3:   for all  $j \in \text{keys}(st.\text{peers})$  do
4:      $\text{payload} \leftarrow \perp$ 
5:     if  $r = \perp$  then  $\triangleright$  withdrawal, no route to apply export policy to
6:       if  $st.\text{RIB}_{\text{out}}[j][p] \neq \perp$  then  $\triangleright$  If previously sent
7:          $st.\text{RIB}_{\text{out}}[j][p] \leftarrow \perp$   $\triangleright$  Clear RIB-out
8:          $\text{payload} \leftarrow \left\{ \begin{array}{l} ('Action', \text{Withdraw}), \\ ('Prefix', p) \end{array} \right\}$   $\triangleright$  Build withdrawal
9:       end if
10:    else
11:       $r' \leftarrow \text{ApplyPolicy}(st, st.\text{policy}_{\text{out}}, r, j)$   $\triangleright$  Apply export policy
12:      if  $r' \neq \perp$  then  $\triangleright$  Announcement case
13:        if  $\neg[|r'.\text{asPath}| = 1 \wedge r'.\text{asPath}[0] = st.\text{asn}]$  then
14:           $r'.\text{asPath} \leftarrow [st.\text{asn}] \parallel r'.\text{asPath}$   $\triangleright$  Prepend local ASN
15:        end if
16:         $r'.\text{nextHop} \leftarrow st.\text{peers}[j].ip_{\text{src}}$   $\triangleright$  Set next-hop IP
17:         $st.\text{RIB}_{\text{out}}[j][p] \leftarrow r'$   $\triangleright$  Update RIB-out
18:         $\text{payload} \leftarrow \left\{ \begin{array}{l} ('Action', \text{Announce}), \\ ('Prefix', p), \\ ('Route', r') \end{array} \right\}$   $\triangleright$  Build announce
19:      else  $\triangleright$  Withdrawal case
20:        if  $st.\text{RIB}_{\text{out}}[j][p] \neq \perp$  then  $\triangleright$  If previously sent
21:           $st.\text{RIB}_{\text{out}}[j][p] \leftarrow \perp$   $\triangleright$  Clear RIB-out
22:           $\text{payload} \leftarrow \left\{ \begin{array}{l} ('Action', \text{Withdraw}), \\ ('Prefix', p) \end{array} \right\}$   $\triangleright$  Build withdrawal
23:        end if
24:      end if
25:    end if
26:    if  $\text{payload} \neq \perp$  then  $\triangleright$  If update exists
27:       $\text{pkt} \leftarrow \left\{ \begin{array}{l} ('dst\_ip', st.\text{peers}[j].ip_{\text{dst}}), \\ ('protocol', \text{BGP}), \\ ('payload', \text{payload}), \\ ('id', \text{GetRandomPktId}()) \end{array} \right\}$   $\triangleright$  Create IP packet
28:       $(st, \text{out}_{\text{send}}) \leftarrow \text{BGP.SENDPACKET}(st, \text{pkt})$ 
29:       $\text{out} \leftarrow \text{out} \cup \text{out}_{\text{send}}$ 
30:    end if
31:  end for
32:  return  $(st, \text{out})$ 
33: end procedure

```

F.3 Proof of Theorem 2

THEOREM 2 (BGP_{RPKI} satisfies $(\delta_{e2e}, \delta_{\text{boot}})$ -IP connectivity under a scheduler adversary). Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta['com'] + \Delta['clk'])$. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}}$ requirement assuming the $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}}$ model holds asymptotically and blackbox protocols that asymptotically

satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta['DNS']}^{\text{DNS-COR}}$, provided that:

$$\begin{aligned}
\delta_{e2e} &\geq c_{\text{prop}} \\
\Delta['conf'] &\geq \Delta['wake'] + \Delta['DNS'] + 2\Delta['ask'] + 2 \cdot c_{\text{prop}} \\
\delta_{\text{boot}} &\geq \Delta['orig'] + c_{\text{prop}} + 2\Delta['conf']
\end{aligned}$$

Proof. We will prove that the requirement holds unconditionally assuming the model and blackbox protocols are also satisfied unconditionally. From Equation (1), this means that it also holds under asymptotic satisfaction, as the theorem states.

Assume to the contrary that BGP_{RPKI} does not satisfy $\text{Uncond}(\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}})$. This means that there must exist a PPT adversary $\mathcal{A}$ that unconditionally satisfy model $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}}$ and can break the requirement in a specific execution of BGP_{RPKI} . Let T be that execution.

Following the definition of $\rho_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}}$ (Algorithm 1), this means that the execution in T depicts a transcript where:

- A benign AS AS_{src} sent a packet with packet identifier pkt_id towards ip at time t , and
- The single benign originator AS_{dst} (ensured by μ^{SHO}) originated p by $t_0 = t - \delta_{e2e}$, at least δ_{boot} after every benign entity was initialized, and never withdrew it before $t + \delta_{e2e}$.

And yet, AS_{dst} did not receive pkt_id by $t + \delta_{e2e}$.

Writing t^{init} for the time by which every benign entity has initialized, the origination occurs at least δ_{boot} after t^{init} and no later than $t - \delta_{e2e}$, resulting in $t^{\text{init}} \leq t - \delta_{e2e} - \delta_{\text{boot}}$.

The BGP_{RPKI} decision process (Algorithm 28) differs from the BGP decision process only with respect to the ROV filter. Following Claim 1, which holds under the theorem conditions, the ROV filter never removes a route to p with origin AS_{dst} from any benign candidate set during $[t_0, t + \delta_{e2e}]$, and under $\mu^{\text{F}| \leq 0} \wedge \mu^{\text{SHO}}$ no route to p with any other origin ever exists. Hence on p every benign AS's BGP_{RPKI} decision coincides with its BGP decision throughout the window, and both Lemma 1 and Lemma 2 hold, since $\delta_{e2e} \geq \text{diam}_{\text{VF}}(G) \cdot (\Delta['com'] + \Delta['clk'])$, AS_{dst} originated p by $t_0 = t - \delta_{e2e}$, and it does not withdraw it before $t + \delta_{e2e}$. By Lemma 1, every benign AS, in particular AS_{src} , installs a route to p with origin AS_{dst} by time t , and by Lemma 2 the packet sent at t reaches AS_{dst} by $t + \delta_{e2e}$. This contradicts the assumed failure, so $\rho_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{IP-CON}}$ holds and delivery occurs by $t + \delta_{e2e}$, contradicting the assumption that such transcript T exists. $\square$

CLAIM 1. Let $G = (V, E, R)$ be an AS topology graph, let p and t be the destination prefix and the send time in the adversary output $T.\text{out}_{\mathcal{A}}$, and let AS_{dst} be the single benign originator of p , fixed by μ^{SHO} . Under model $\mathcal{M}_{\Delta}^{\text{RPKI-Hybrid}}$, and assuming blackbox protocols that satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta['DNS']}^{\text{DNS-COR}}$, if:

- Every benign entity was initialized by time $t - \delta_{e2e} - \delta_{boot}$, and
- $\delta_{e2e} \geq \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$, and
- $\Delta['conf'] \geq \Delta['wake'] + \Delta['DNS'] + 2\Delta['ask'] + 2\text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$, and
- $\delta_{boot} \geq \Delta['orig'] + \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk']) + 2\Delta['conf']$

Then no benign ROV adopter classifies a route to p with origin AS_{Dst} as Invalid at any time in $[t - \delta_{e2e}, t + \delta_{e2e}]$, except with negligible probability.

Proof. Assume to the contrary that there exists a transcript T in which a benign ROV adopter r classifies a route to p with origin AS_{Dst} as Invalid at time $t_{rov} \in [t - \delta_{e2e}, t + \delta_{e2e}]$.

According to CHECKROV (Algorithm 28), this means that r 's cache at time t_{rov} must lack any VRP that validates (p, AS_{Dst}) and also must contain a VRP that conflicts with (p, AS_{Dst}) . The conflicting VRP must have been created by the retrieval of a valid ROA ψ_{conf} during r 's last repository refresh at time t_{wake} . This means that ψ_{conf} was published in its RIR's PP by time $t_{pp} \leq t_{wake}$.

However, $\mu_{\Delta['conf']}^{\text{NoROAConflict}}$ ensures that a valid ROA ψ_v for (p, AS_{Dst}) must have been published by $t_{pp} - \Delta['conf']$ and must remain valid at t_{rov} . By Lemma 3, which hold the same conditions, by time $t - \delta_{e2e}$ every benign RIR's relying party holds the roots and every ROA published at an RIR PP by $t - \delta_{e2e} - \Delta['conf']$, in particular the ROA covering r 's RP prefix, so the RIR serves r 's fetch and each refresh completes. Under the $\mu^{\text{RPKIHybrid}}$ model assumption, r retrieves all the published objects from every RIR's PP during each refresh, and since $\Delta['conf'] \geq \Delta['wake']$, we get $t_{pp} - \Delta['conf'] \leq t_{wake} - \Delta['wake']$. Since the maximum interval between r 's consecutive wake-ups is $\Delta['wake']$, ψ_v must have been published prior to t_{wake} , which means that even if ψ_v and ψ_{conf} are published in different PPs and retrieved in different transactions, ψ_v must have been retrieved by r in the previous wake-up cycle. Therefore, a valid VRP for (p, AS_{Dst}) must have been installed at r and persisted by t_{rov} , contradicting the assumption that such a transcript T exists. $\square$

LEMMA 3. Let $G = (V, E, R)$ be an AS topology graph and t be a time. Under model $\mathcal{M}^{\text{RPKI-Hybrid}}$, and assuming blackbox protocols that satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta['DNS']}^{\text{DNS-COR}}$, if:

- Every benign entity was initialized by time $t - \delta_{e2e} - \delta_{boot}$, and
- $\delta_{e2e} \geq \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$, and
- $\Delta['conf'] \geq \Delta['wake'] + \Delta['DNS'] + 2\Delta['ask'] + 2\text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$, and
- $\delta_{boot} \geq \Delta['orig'] + \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk']) + 2\Delta['conf']$

Then, by time $t - \delta_{e2e}$, the relying party of every benign RIR holds, validated in its cache, every RIR root certificate and every ROA published at an RIR PP by time $t - \delta_{e2e} - \Delta['conf']$, except with negligible probability.

Proof. Let v be a benign RIR, and let t^{init} be the time by which every benign entity has initialized. By the first assumption, $t^{\text{init}} \leq t - \delta_{e2e} - \delta_{boot}$.

By $\mu^{\text{RIRDeploy}}$, no benign CA issues a ROA that conflicts on a prefix covering an RIR PP, root anchor location, or the RP address, and by $\mathcal{R}^{\text{PKI}}$, no forged certificate exists, except with negligible probability. Since a conflicting VRP could enter a benign cache only via a ROA that was either benignly issued or forged, except with negligible probability, no benign cache ever holds a conflicting VRP for one of these prefixes, at any time and regardless of sync state, so ROV never marks a route to a PP, a root anchor location, or an RP as Invalid.

Denote $c_{\text{prop}} = \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk'])$. We now show that both Lemma 1 and Lemma 2 are applicable to the RIRs. Let $\mathcal{R}$ denote the benign RIRs. By $\mu_{\Delta['orig']}^{\text{RPKIInit}}$, each $w \in \mathcal{R}$ originates prefixes covering each of the three addresses it hosts within $\Delta['orig']$ of its own init, those are never withdrawn; since every benign entity has initialized by t^{init} , every one of these originations occurs by $t^{\text{init}} + \Delta['orig']$. Since ROV never marks these routes Invalid, Lemma 1 applies to each prefix, and it is installed at any AS within c_{prop} of its origination by the time $t^{\text{reach}} = t^{\text{init}} + \Delta['orig'] + c_{\text{prop}}$. Therefore, by t^{reach} , every AS holds a route toward every RIR root anchor location, PP, and RP address, and since none of these prefixes is ever withdrawn, these routes persist for the rest of the execution. Consequently, from t^{reach} onward, each RIR is two-way reachable, and by Lemma 2 every message between an RIR RP and a root anchor location or PP is delivered within c_{prop} of being sent, since the request travels on the prefix covering the responder's address, and the reply on the prefix covering the requester's RP address.

We now prove that every RIR can sync. Let t_1 be the time of v 's first wake-up at or after t^{reach} . Since the RP wakes every $\Delta['wake']$, we have $t_1 \leq t^{\text{reach}} + \Delta['wake']$. At init, v receives the TAL files, each naming a root location and its public key. At t_1 all the routes above are installed, so v 's fetch of each root certificate can complete, since the root anchor location name resolves to the genuine address within $\Delta['DNS'] + 2\Delta['ask']$; the request is a one-way delivery on the prefix covering the root anchor location, arriving within c_{prop} ; and the reply is a one-way delivery on the prefix covering v 's RP address, arriving within another c_{prop} , where it is validated against the TAL key held since init. Hence v holds every RIR root by $t_1 + \Delta['DNS'] + 2\Delta['ask'] + 2c_{\text{prop}} \leq t^{\text{reach}} + \Delta['wake'] + \Delta['DNS'] + 2\Delta['ask'] + 2c_{\text{prop}}$.

To simplify the notation, let $\Delta_{\text{sync}} = \Delta['wake'] + \Delta['DNS'] + 2\Delta['ask'] + 2c_{\text{prop}}$. From $t^{\text{reach}} + \Delta_{\text{sync}}$ onward, v holds the roots, whose SIAs name the RIR PPs, so v knows every RIR PP. We claim that for any time $s \geq t^{\text{reach}} + \Delta_{\text{sync}}$,

the RP of v holds by time $s + \Delta_{\text{sync}}$, validated in its cache, every ROA published at an RIR PP prior to s .

v 's first wake-up at or after s occurs by $s + \Delta[\text{'wake'}]$; it then fetches from each PP, with DNS resolving within $\Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}]$ and the request and the reply taking $2c_{\text{prop}}$ in total, so the fetch completes by $s + \Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}} = s + \Delta_{\text{sync}}$. The reply leaves the PP at some time $t' \geq s$ and carries the PP's store as of t' , hence every ROA published by s . By the hybrid deployment mode $\mu^{\text{RPKIHybrid}}$, each PP serves every ROA together with its whole issuing chain, so each retrieved ROA is validated against the roots.

By the first assumption, $t^{\text{init}} \leq t - \delta_{e2e} - \delta_{\text{boot}}$, and since $\delta_{\text{boot}} \geq \Delta[\text{'orig'}] + c_{\text{prop}} + 2\Delta[\text{'conf'}]$ and $\Delta[\text{'conf'}] \geq \Delta_{\text{sync}}$, we have $t^{\text{reach}} + 2\Delta_{\text{sync}} = t^{\text{init}} + \Delta[\text{'orig'}] + c_{\text{prop}} + 2\Delta_{\text{sync}} \leq t - \delta_{e2e}$. Hence, by time $t - \delta_{e2e}$, v holds every RIR root certificate and every RPKI object published by $t - \delta_{e2e} - \Delta_{\text{sync}}$, in particular every one published by $t - \delta_{e2e} - \Delta[\text{'conf'}]$. Since v is an arbitrary benign RIR, the same holds for the RP of every benign RIR. $\square$

F.4 Proof of Theorem 3

THEOREM 3 (BGP_{RPKI} satisfies $(\delta_{e2e}, \delta_{\text{boot}})$ -SR-IP Connectivity under a prefix adversary assuming available ROA). *Let G be an AS topology graph. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{SR-IP-CON}}$ requirement assuming the $\mathcal{M}_{f, \Delta}^{\text{HonestROVPref}}$ model holds asymptotically and blackbox protocols that asymptotically satisfy requirements $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, provided that $\delta_{e2e} \geq \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$ and $\delta_{\text{boot}} \geq 0$.*

Proof of Theorem 3. We will prove that the requirement holds unconditionally assuming the model and blackbox protocols are also satisfied unconditionally. From Equation (1), this means that it also holds under asymptotic satisfaction, as the theorem states.

By Lemma 4, AS_{Src} must have learned of a route for prefix p by time t with AS_{Dst} as the origin. That lemma is proved by induction on a best path from AS_{Dst} to AS_{Src}: if AS_{Dst} originated the prefix at time t_0 , it must prefer its own route based on the $\mu^{\text{PreferSelf}}$ model, and since the path follows GR rules and exports to all, it must export to the next AS on the path. The path is made of ROV adopters, and the $\mu_{\delta_{e2e}}^{\text{ROA-InCache}}$ ensures that each benign adopter receives the full certificate chain validating a ROA (ensuring p is originated by AS_{Dst}). Therefore, if the i -th AS on the path selects a path to the prefix originated by AS_{Dst}, the $(i+1)$ -th AS receives it and will also prefer it.

If, for the sake of contradiction, the $(i+1)$ -th AS does not prefer it, this means there is a different valid ROA that defines AS_{bad} as a different origin for the same prefix. Since we assume a single originator, no other benign AS should announce

the prefix. Furthermore, under $\mu_{\text{Prefix}}^{\text{HijackAdv}}$, the adversary cannot control a CA to create a fraudulent ROA. Therefore, the only way for AS $i+1$ to prefer a different origin is if the adversary is able to create a fraudulent ROA, which breaks the existential unforgeability guaranteed by $\mathcal{R}^{\text{PKI}}$.

Thus, by induction, AS_{Src} learns of a valid route by the time $t_0 + \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$, which is at most t since $\delta_{e2e} \geq \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$ and $t_0 = t - \delta_{e2e}$. Regarding packet delivery, $\mathcal{P}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{SR-IP-CON}}$ ensures a stable routing window of $2\delta_{e2e}$ around t , meaning that by time t , all ASes have selected some path. If an AS along the path selected by AS_{Src} had changed its selection to a path leading to a different origin, this decision would have propagated to AS_{Src} prior to t (since this AS is benign and follows the protocol), which would have changed its own selected route. Because AS_{Dst} did not withdraw the route, AS_{Src} still has a path to AS_{Dst}. Finally, since $\mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$ ensures that AS_{Src} sends over an benign path, the packet is surely received by AS_{Dst}, thus satisfying the global connectivity requirement. $\square$

Supporting lemma

LEMMA 4. *Let $G = (V, E, R)$ be an AS topology graph, q be an IP prefix, ψ be a ROA, and AS_{Dst}, $u \in V$ be a benign ROV-adopting ASes.*

Under the $\mu_{f, \Delta}^{\text{BGP}} \wedge \mu_{\text{Prefix}}^{\text{HijackAdv}} \wedge \mu_{u, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}} \wedge \mu_{u, \text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$ model, and blackbox protocols that satisfy requirement $\mathcal{R}^{\text{PKI}}$, if:

- ψ validates q for AS_{Dst}, and
- AS_{Dst} originates q at time t and does not withdraw it before $t + \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$, and
- every AS on the best path from AS_{Dst} to u holds ψ , validated, in its cache throughout $[t, t + \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])]$

Then u has installed a route to q with AS_{Dst} as its origin by time $t + \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$, except with negligible probability.

Proof of Lemma 4. Let $P = (v_0, v_1, \dots, v_k)$ be a best path from AS_{Dst} to u , which consists of benign ROV adopters by $\mu_{u, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$ and $\mu_{u, \text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$, where $v_0 = \text{AS}_{\text{Dst}}$, $v_k = u$. Since P is a simple path in G , its length satisfies $k \leq \text{diam}_{\text{VF}}(G)$.

We claim that every node v_i along P will **select** a valley-free route to q originated by AS_{Dst} by time $t + i \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. We prove this by induction on i :

Base case ($i = 0$). At time t , the node $v_0 = \text{AS}_{\text{Dst}}$ originates q by executing BGP_{RPKI}.ORIGINATE (which wraps the underlying BGP functionality). This operation installs the self-originated route into $st.BGP.RIB_{in}$ and immediately invokes the decision process DECISION (Algorithm 28).

Within the decision function, the *GatherCandidates* sub-procedure (Line 2) retrieves all available candidate routes to

q , which necessarily includes the newly self-originated route. Because ROV is enabled for v_0 ($st.rov = \top$), this candidate route is evaluated by *CheckROV* (Line 23).

Since v_0 holds ψ which is valid throughout the window, UpdateVRPs (Algorithm 35, Lines 6–12) extracts it into $st.VRPs$, so the VRP set of v_0 contains a VRP that covers q and authorizes AS_{Dst} , and $AS_{Dst} = v_0$ is the announcing origin, so the VRP matches both the origin AS and the prefix length. Consequently *CheckROV* returns *Valid* (Line 28), allowing the self-originated route to bypass the *Invalid* filter (Line 4).

Finally, because the $\mu^{PreferSelf}$ model predicate ensures that locally originated routes are strictly preferred over all externally received routes, the self-originated route is definitively selected by *FindbestRoute* (Line 15). The node's state is subsequently updated by installing this route into v_0 's *locRIB* (Line 19) and programming the forwarding entry into the FIB (Line 20). Thus, at time t , v_0 selects a route to q with the correct origin.

Inductive step ($i \rightarrow i+1$). Assume that node v_i ($0 \leq i < k$) has already selected a valley-free path to q with AS_{Dst} as the origin through the execution of DECISION (Algorithm 28). Because the path $P = (v_0, v_1, \dots, v_k)$ is valley-free and consistent with the commercial structure encoded in $\mu_{f,\Delta}^{BGP}$, and by the export-to-all policy, v_i forwards its selected route to v_{i+1} .

Within the implementation, this is handled by the call to *Propagate* (Algorithm 28, Line 21). Inside this procedure, the export policy check (Algorithm 21, Line 11) evaluates to true for neighbor v_{i+1} . This leads to the generation of an announcement (Line 18) and the encapsulation of the routing packet (Line 28), which is then passed to the data-plane (Algorithm 13).

Because the $\mu^{BgpInit}$ model predicate ensures all local subnets and local topological mappings are properly established prior to execution, the data-plane safely performs a recursive LPM lookup (Algorithm 13, Line 8) to resolve the next-hop IP into a physical *target_entity* from the *LocalMap* (Line 13), eventually emitting the packet (Line 14) to v_{i+1} .

Upon arrival at v_{i+1} , the packet is processed by the *PacketReceive* procedure (Algorithm 14). Since the packet is addressed to the node's local interface (Line 4), it is handed to the *PacketRouter*, which in turn calls the Update (Algorithm 15) function to update the local *RIB_{in}*. Because $\mu_{\Delta['com']}^{Com}$ bounds neighbor communication by $\Delta['com']$ and $\mu_{\Delta['clk']}^{Drift}$ bounds clock deviation by $\Delta['clk']$, v_{i+1} completes this reception no later than:

$$t + (i+1) \cdot (\Delta['com'] + \Delta['clk']).$$

Once stored, v_{i+1} invokes DECISION, collecting the new route as a candidate. We now prove that v_{i+1} must select a route originated by AS_{Dst} . Suppose, for the sake of contradiction, that v_{i+1} selects a candidate route originated by an adversarial $AS_{bad} \neq AS_{Dst}$. Because v_{i+1} enforces ROV

($st.rov = \top$), any selected route must not evaluate to *Invalid*. Since v_{i+1} holds ψ validated throughout the window, UpdateVRPs places the VRP extracted from ψ in its VRP set, and that VRP covers q , so no announcement for q evaluates to *NotFound*: every candidate for q is classified either *Valid* or *Invalid*. That same VRP authorizes AS_{Dst} and not AS_{bad} , so the candidate announced by AS_{bad} is *Invalid* unless the cache of v_{i+1} holds a *further* validated ROA authorizing AS_{bad} for a prefix covering q .

Such a ROA validates against a benign RIR trust anchor and authorizes AS_{bad} to announce q . Under the $\mu_{Prefix}^{HijackAdv}$ model the adversary controls no CA, and no benign CA issues a ROA for AS_{bad} , so no genuine CERTIFY event produced it. By $\mathcal{R}^{PKI}$ no such certificate exists, except with negligible probability.

Thus, with overwhelming probability, v_{i+1} securely installs the ROV-valid, valley-free route to q originated by AS_{Dst} .

By induction, $v_k = u$ has installed a route to q with AS_{Dst} as origin by time

$$\begin{aligned} t + k \cdot (\Delta['com'] + \Delta['clk']) \\ \leq t + \text{diam}_{VF}(G) \cdot (\Delta['com'] + \Delta['clk']) \end{aligned}$$

concluding the proof. $\square$

G RPKI Implementation

G.1 Entities and State

In the BGP_{RPKI} , the set of stateful entities comprises two distinct types: Certificate Authorities (CAs) and BGP routers. CAs are responsible for issuing and revoking resource certificates. They are categorized as either *root CAs* (trust anchors), such as the five RIRs which relying parties trust directly via a preseeded TAL file, or *intermediate CAs*, which are trusted based on a certificate chain terminating at a root CA. BGP routers are able to announce and withdraw prefixes over the BGP network. Each entity $\mathfrak{t}$ maintains a local state st , the components of which are determined by the entity type ($st.roles$). All entities possess a unique identifier, denoted $st.i$. A CA's state st includes parameters that are needed to support the underlying PKI of RPKI, including:

- *st.certs*: Holds all valid certificates issued by the underlying PKI.
- *st.manifest*: Holds the most recent manifest.
- *st.manifestNum*: Indicates the current manifest number.
- *st.EEKeys*: Holds the key pairs of the one-time-use EE certificates.
- *st.IPs*, *st.ASNs*: The IP blocks and ASNs this CA holds, and may delegate.

PKI operations are supported by invoking an underlying PKI defined in [84].

BGP routers are either adopting ROV or not. Non-adopting routers have the exact state defined in Section 6. This state is extended when the entity adopts ROV ($st.rov = \top$) or when the AS hosts an RPKI repository.

When ROV is enabled, the following components are added:

- $st.TALs$: TAL files for the five RIRs.
- $st.cache$: Local cached RPKI data (e.g., certificates and ROAs).
- $st.VRPs$: Validated ROA Payloads, defined as tuples (AS, prefix, maxlen).
- $st.IP_{RP}$: IP address of the local Relying Party.

If the entity also hosts an RPKI repository:

- $st.IP_{RP}$: The IP address where the repository is hosted.
- $st.repoCache$: The local repository data, holds as a dictionary mapping from each issuer CA to its set of data (e.g., certificates and ROAs).

G.2 Simplifications

DNS Dependency in Repository Reachability

When a RP synchronizes with the global RPKI repository, it traverses the certificate hierarchy starting from the TALs. Each certificate in this chain, from the root RIR certificate downwards, contains a SIA extension that specifies the URI for the publication point of CA-issued objects. Consequently, the RP must rely on the DNS to resolve the repository host-names into IP addresses before a connection can be established.

This architectural dependency introduces a critical point of failure. If the repository’s authoritative nameservers are unreachable, or if the RP’s local DNS resolution fails, the RP cannot retrieve updated ROAs. Due to the “fail-open” design of RPKI, the eventual expiration of cached data causes routes to revert to a *NotFound* state. As a result, a DNS-induced disruption effectively strips the network of the security guarantees provided by Route Origin Validation. Hlavacek et al. [30] analyzed RPKI’s dependency on DNS, finding that 82.9% of relying parties use resolvers within a single AS and only 52% validate DNSSEC which introduces significant availability and integrity risks. The study also highlighted a lack of ROA coverage for critical DNS infrastructure: as of 2022, 10 of the 13 DNS root servers were announced from prefixes without ROAs. Ultimately, only 3 out of 50 publication points could be “securely reached”, defined as having their entire nameserver resolution chain protected by ROAs.

We use the modular design feature of CMoSS, that assumes DNS functionality as *blackboxes*. The BGP_{RPKI} protocol assumes a protocol that guarantees the DNS correctness requirement $\mathcal{R}_{\Delta[DNS]}^{DNS-COR}$ (defined in Appendix I), i.e., ensures correct resolution of a domain name in a bounded time $\Delta[DNS]$.

Router to RP communication. BGP routers do not validate RPKI cryptographic data; rather, they are configured to trust a RP that fetches and validates the data. This produces a final set of verified VRPs mapping prefixes to origins, allowing a router to effectively filter out bogus announcements without the cryptographic overhead. Communication between routers and the RP is conducted over the RTR protocol [8]. While the standard allows for different deployment strategies, it is highly recommended to place RPs close to the routers to prevent potential routing attacks and connection errors [7]. Beyond RFC recommendations, a recent study [50] shows that most networks locally operate an RP. Thus, in our analysis, we assume a locally hosted RP.

Certificates and CMS

We follow the work of Wrótniak et al. [84] when referring to certificates and their fields, denoting a certificate as ψ . We use dot notation to refer to a specific field in a certificate; for example, $\psi.issuer$ refers to the entity that issued ψ . Since resource certificates extend the standard PKIX profile, we rely on the work of Wrótniak et al. [84]. Therefore, the algorithms described below use the PKI scheme defined in their work.

RPKI signed objects, such as ROAs and Manifests, are encoded using the *Cryptographic Message Syntax* (CMS) format, a standardized structure for digitally signing and encrypting data defined in [32]. Similar to resource certificates, there exists a standard defining the base format of an RPKI Signed Object [40]. This base profile is then extended by further RFCs for each specific object type, such as [42] for ROAs and [3] for Manifests. The CMS structure contains the signed object content, such as an ASN and IP prefixes in a ROA, along with the End-Entity (EE) certificate used to sign it, and a *SignerInfo* field containing the digital signature over the content. EE certificates are resource certificates that are not CA certificates ($\psi.is\_CA = \perp$). They are issued on a one-to-one relationship: a unique certificate is generated for each signed object and contains a key used exclusively to sign and verify that specific object [33]. Consequently, validation is a two-step process: the verifier must first extract the EE certificate from the CMS wrapper and validate it against the issuing CA, and then use the EE public key to cryptographically verify the signed object itself.

While CMS objects differ from traditional X.509 certificates, they serve a similar purpose by encoding structured data with a digital signature to assert authenticity. Therefore, we can make use of the certificate notation defined by Wrótniak et al. [84], denoting both RPKI signed objects and resource certificates as ψ . Objects can be distinguished by the value of the $\psi.type$ field (e.g., ROA, manifest, or resource certificates). In the case of resource certificates, we follow the RPKI profile extension defined in [33], which defines both the *asResources* and *ipResources* fields, each of which binds the right of use to the associated public key.

To achieve the CMS encapsulation format, we define two new fields: $\psi.payload$ and $\psi.EE$. The $\psi.payload$ field contains a modular payload based on the certificate type; for example, when $\psi.type = \text{'ROA'}$, it contains the fields $\psi.payload.ipPrefixes$ and $\psi.payload.asID$, which authorize the association of the IP prefixes with the specified AS. The $\psi.EE$ field contains the EE resource certificate whose key signs the payload. The values of the outer certificate are the same as those in $\psi.EE$. For example, if we refer to the $\psi.issuer$ of a ψ with type ROA, its value is equal to $\psi.EE.issuer$. Table 3 includes a description of the fields present in each payload type, as well as all added fields and their references within the RPKI system.

BGP_{RPKI} use the generic certificate scheme $C^{S,\Theta}$ where the encoding function Θ is type-dependent. Specifically, Θ selects the appropriate encoding scheme based on the certificate type, as specified in the corresponding RFCs. For example:

$$\Theta(tbs) = \begin{cases} \Theta_{Cert}(tbs) & \text{if } tbs.type = \\ & \text{Resource certificate} \\ \Theta_{ROA}(tbs) & \text{if } tbs.type = \text{ROA} \\ \Theta_{MFT}(tbs) & \text{if } tbs.type = \text{Manifest} \\ \dots & \dots \end{cases}$$

Each Θ_τ corresponds to a concrete encoding format defined in RPKI RFCs [3, 33, 42]. The fields used in each RPKI object type are summarized in Table 3.

ROV

ROV is the functional application of RPKI data to the BGP decision process. When receiving a new route, the router must determine whether the origin AS claiming to announce a prefix is actually authorized to do so. This process is defined in RFC 6811 [54] and involves comparing the BGP route data against a local database of VRPs. Following validation, the router applies its routing policy to determine how to handle the route. In particular, routes classified as *Invalid* may be filtered or assigned a lower preference. The Best Current Practice in RFC 7115 [7] recommends discarding invalid routes while generally continuing to accept *Unknown* routes, due to the incremental and still-incomplete deployment of RPKI.

G.3 Scheme Implementation

We now present the BGP_{RPKI} implementation. It follows the API defined in Definition 2 and the definition of the PKI scheme detailed in Wrótniak et al. [84].

$BGP_{RPKI}.INIT$ Alg. 22 Initializes the local state based on the entity's assigned roles (Line 5). For CAs, it prepares local storage and invokes the underlying PKI for the root certificate (Lines 7–14; the certificate is received in Alg. 40, Line 7 based on the ASK model); RIRs additionally export TAL

file (Alg. 40, Line 10). For BGP, initializes the standard routing state (Line 17). If ROV is enabled, the router bootstraps: it calls `FetchRootCert` (Alg. 23, Line 22), which fetch the root certificate based on the preloaded TAL files, and initializes empty caches and VRP sets (Lines 19–26). It then calls `UpdateVRPs` (Line 27), which populates the VRP set from any preloaded certificates. Finally, it configures any hosted repository or TAL endpoints (Lines 29–35) and schedules the next wakeup (Line 38).

$BGP_{RPKI}.Certify$ Alg. 25 The certify operation manages the issuance of RCs and signed RPKI objects. It begins by validating the requested values, aborting if the requested IP or ASN allocations exceed the entity's holdings or if the object payloads are invalid (Lines 2–3). For signed objects, it follows the specification by generating a “one-time-use” [33] EE certificate (Lines 5–8). For both signed objects and RCs, `Certify` then invokes the underlying PKI (Line 10) and returns. The RPKI object encapsulation completes once the PKI answers. In the composition operation $BGP_{RPKI}.PKI-CERTIFY-RESP$ (Alg. 40), it wraps the payload together with the returned EE certificate (Line 16), recovers the matching private key from $st.EEKeys$ (Line 17), signs the wrapped object with that key (Line 18), and stores the result in $st.certs$.

$BGP_{RPKI}.Valid$ Alg. 29 The valid operation is stateless, the goal of this operation is to ensure that any given certificate or signed RPKI object is cryptographically valid and adheres to the resource hierarchy, based on object type (Line 2). For RC, EE and CRL certificates, `ValidChain` ensures a valid path to a trust anchor (Lines 8, 12 and 21), enforcing RPKI resource containment (Lines 19 and 26) alongside standard PKIX checks. For signed RPKI objects, we first performs explicit checks based on object type and validates the enclosed EE certificate. Specifically, we validates the signature and enforces explicit EE constraints, for example ROAs must define bounded IP prefixes without the ‘INHERIT’ flag (RFC 9582) (Lines 31–33).

$BGP_{RPKI}.WAKEUP$ Alg. 27 Executed periodically, the Wakeup function operate based on the roles of the entity. For CAs, it ensures repository freshness. It begins by revoking the previously published Manifest (Line 4) and generating a new CRL (Line 6), which is stored once the PKI subprotocol answers (Alg. 40, Line 24). The algorithm generate a new Manifest (Alg. 40, Line 25); this Manifest, along with the CRL, is then given as an output (Alg. 40, Line 26). Alternatively, if the entity is a router enforcing ROV (Line 8), it synchronize its local repository cache, fetching new objects and removing expired ones (Line 9). Finally, to ensure continuous operation, it schedules the next wake-up to occur within $\Delta[\text{'wake'}]$ (Line 12).

We use the BGP algorithms implemented in Section 6,

incorporating two internal logic updates. First, the decision logic is modified to support ROV. Second, the packetRouter function is extended to process RPKI information.

DECISION Alg. 28 The DECISION function extends standard BGP logic by integrating ROV. After gathering available route candidates, the algorithm verify each candidate's origin AS and prefix length against the VRP table (Line 4). Any route evaluated as *Invalid* is immediately dropped. This ensures that the best-path (Line 15) is either valid or unknown.

PACKETROUTER Alg. 26 This function acts as a traffic director for incoming network packets either for BGP or RPKI (Line 3). BGP messages are simply passed to the standard BGP process (Lines 4–5). For RPKI packets, the function routes the traffic based on the destination IP address (Line 7). Requests for RPKI files sent to IP_{PP} or IP_{TAL} are forwarded to a handler that serves these files to the sender (Lines 8–9). On the other hand, data arriving for the local RP (IP_{RP}) is processed to update the router's own cache (Lines 10–11). Finally, any unknown or unsupported traffic is dropped (Line 14).

UpdateRepoStore Alg. 24 This function allow the adversary to publish objects to a PP. It first verifies the received batch is non-empty (Line 2). The function enforces that all objects within the batch originate from the exact same issuer (Lines 6–7), if so its overwrites this exact CA cache with the new batch (Line 10).

H RPKI Sync Failure under DNS Hijack

Figure 9 illustrates how DNS hijack can cause RPKI synchronization failure.

I DNS Scheme and Correctness Requirement

In this section, we define a DNS scheme and the blackbox correctness requirement. This blackbox requirement allows us to perform our analysis under the assumption that we have a lower-level protocol that satisfies specific requirements, without going into the implementation details, providing a top-down modular approach that can be extended later.

DEFINITION 7 (Domain Name System Scheme). A Domain Name System scheme DNS is a set containing (at least) the following PPT algorithms, where each algorithm receives a state st , outputs an updated state st' , and is defined as follows:

- $\text{ResolveQuery}(d, \text{type}) \rightarrow (out)$: Initiates the resolution process for a given domain name d and query type. It returns an updated state st' alongside an output out containing a query identifier qid . This operation does not return the resolution result.

Algorithm 22 BGP_{RPKI}.INIT

```

1: operation BGPRPKI.INIT( $st, clk, params$ )
2:   if  $st \neq \perp$  then return ( $st, \perp$ ) ▷ Abort if initialized
3:    $out \leftarrow \{\}$ 
4:    $st.\lambda \leftarrow params.\lambda$ 
5:    $st.roles \leftarrow params.roles$ 
6:   if 'CA'  $\in st.roles$  then
7:      $st.certs \leftarrow \perp$ 
8:      $st.manifest \leftarrow \perp$ 
9:      $st.manifestNum \leftarrow 0$ 
10:     $st.EEKeys \leftarrow \emptyset$  ▷ Keys of the one-time-use EE certs
11:     $st.\Delta_{man} \leftarrow params.\Delta_{man}$  ▷ Validity interval of an issued manifest
12:    ( $st.IPs, st.ASNs$ )  $\leftarrow (params.IPs, params.ASNs)$ 
    ▷ Resources this CA may delegate
13:     $tbc \leftarrow \left\{ \begin{array}{l} ('AIA', params.AIA), \\ ('SIA', params.SIA), \\ ('ipResources', st.IPs), \\ ('asResources', st.ASNs) \end{array} \right\}$  ▷ Parent CA url /  $\perp$  if RIR  
▷ Publication point url  
▷ IP block delegations  
▷ ASN block delegations
14:     $out \leftarrow out \cup \{('INVOKE', ('PKI', 'NEW', INIT, (params, tbc)))\}$  ▷ The new instance is identified by  $st.PKI$  instance
15:  end if
16:  if 'Router'  $\in st.roles$  then
17:     $st.bgp \leftarrow BGP.INIT(st.bgp, params)$  ▷ Init BGP router state, See Alg 9
18:     $st.rov \leftarrow params.rov$ 
19:    if  $st.rov$  then
20:       $out \leftarrow out \cup \{('INVOKE', ('DNS', 'NEW', INIT, (params)))\}$  ▷ The new instance is identified by  $st.DNS$  instance
21:       $st.TALs \leftarrow params.tals$ 
22:      ( $st, Fout$ )  $\leftarrow \text{FetchRootCert}(st)$  ▷ see Alg 23
23:       $out \leftarrow out \cup Fout$ 
24:       $st.cache \leftarrow \emptyset \cup params.preLoadedCerts$  ▷ Pre-load cache, based on threat model
25:       $st.VRPs \leftarrow \emptyset$  ▷ Init VRP DB
26:       $st.IPRP \leftarrow params.IPRP$ 
27:       $st \leftarrow \text{UpdateVRPs}(st, clk)$  ▷ See Algorithm 35. Only has effect if exist any preLoadedCerts
28:    end if
29:    if  $params.IPP \neq \perp$  then
30:      if  $\exists b \in st.bgp.ipBlocks : params.IPP \subseteq b$  then return  $\perp$ 
31:       $st.IPP \leftarrow params.IPP; st.repoCache \leftarrow \emptyset$ 
32:    end if
33:    if  $params.IPTAL \neq \perp$  then
34:      if  $\exists b \in st.bgp.ipBlocks : params.IPTAL \subseteq b$  then return  $\perp$ 
35:       $st.IPTAL \leftarrow params.IPTAL; st.rootCert \leftarrow params.\Psi_{root}$ 
36:    end if
37:  end if
38:   $out \leftarrow out \cup \{('WakeAt', clk)\}$  ▷ Schedule the next execution, waking up prior to  $clk + \Delta['wake']$ 
39:  return ( $st, out$ )
40: end operation

```

Algorithm 23 FETCHROOTCERT: Root certificate fetch per TAL

```

1: procedure FetchRootCert( $st$ )
2:    $out \leftarrow \emptyset$  ▷ Prepare outgoing network actions
3:   for each  $tal \in st.TALs$  do ▷ one query per preloaded TAL
4:      $payload \leftarrow \{('Type', \text{FetchRootCert})\}$ 
5:      $pkt \leftarrow \left\{ \begin{array}{l} ('src\_ip', st.IPRP), \\ ('protocol', RPKI), \\ ('payload', payload), \\ ('id', \text{GetRandomPktId}()) \end{array} \right\}$  ▷ Builds the fetch pkt
6:      $out \leftarrow out \cup \{pkt, ('INVOKE', ('DNS', \text{ResolveQuery}, st.DNSInstance, tal.URL, A))\}$ 
7:   return  $st, out$  ▷ Returns the pkt and the formal ask instruction
8: end procedure

```

- $\text{ResolveQueryAnswer}(qid) \rightarrow (out)$: Returns the result of the resolution identified by qid . It returns

Table 3: Fields RPKI adds to the certificate abstraction of [84]. The inherited X.509 fields are as defined there; $\psi.type$ ranges over RC, EE, ROA, MFT and CRL.

Field	Description	Encoding in RPKI
<i>Resource certificate, RFC 6487 [33] profile</i>		
$\psi.ipResources, \psi.asResources$	Delegated IP blocks and ASNs, or INHERIT.	Defined in the <i>IP Address Delegation and AS Identifier Delegation Extension</i> per RFC 3779 [47]. Used to constrain the scope of subordinate certificates and ROAs.
$\psi.SIA$	Set of publication point locations.	Contains publication points for the CA (repository URI, manifest URI, CRL distribution point)
$\psi.AIA$	Publication point of the parent CA.	Contains URI of the parent CA's certificate and repository publication location.
<i>Added for the CMS encapsulation of signed objects</i>		
$\psi.EE$	End-Entity certificate whose key signs the object.	RPKI objects are CMS encapsulated, we added the EE field to represent it. More details are in Appendix G.2.
$\psi.payload$	Signed content; its shape is fixed by $\psi.type$. This is the content $\psi.pk$ verifies.	The CMS [32] <i>encapContentInfo</i> . The shapes are below, one per signed-object type.
<i>ROA payload, $\psi.type = 'ROA'$ (RFC 9582 [76])</i>		
$payload.ipPrefixes$	Non-empty set of pairs $(prefix, maxLen)$: an authorized prefix and the most specific length authorized under it.	The <i>ipAddrBlocks</i> field part of the cms content in <i>RouteOriginAttestation</i> . Every <i>prefix</i> must lie in $\psi.ipResources$. <i>maxLen</i> caps how specific a covered announcement may be, and must be at least $len(prefix)$.
$payload.asID$	AS authorized to originate every prefix in the ROA.	The <i>asID</i> field of the same <i>RouteOriginAttestation</i> .
<i>Manifest payload, $\psi.type = 'MFT'$ (RFC 9286 [3])</i>		
$payload.fileList$	Published file names mapped to their hashes.	The <i>fileList</i> field part of the cms content in <i>Manifest</i> , a sequence of <i>FileAndHash</i> pairs. An object is accepted only if its hash matches the listed one.
$payload.manifestNumber$	Manifest number.	The <i>manifestNumber</i> field of the same <i>Manifest</i> . Raised each publication cycle.
$payload.thisUpdate, payload.nextUpdate$	Freshness window the manifest claims.	The <i>thisUpdate</i> and <i>nextUpdate</i> fields of the same <i>Manifest</i> .

Algorithm 24 BGP_{RPKI}.UpdateRepoStore: Update Repository State

```

1: procedure BGPRPKI.UpdateRepoStore( $st, certs$ )
2:   if  $st = \perp \vee certs = \emptyset$  then
3:     return  $st$ 
4:   end if
5:   let  $\psi_{sample} \in certs$ 
6:   issuer  $\leftarrow \psi_{sample}.issuer$ 
7:   if  $\exists \psi \in certs$  s.t.  $\psi.issuer \neq issuer$  then
8:     return  $st$  ▷ Mixed issuers
9:   end if
10:   $st.repoCache[issuer] \leftarrow certs$  ▷ Update hosted files
11:  return  $st$  ▷ Return updated state
12: end procedure

```

Algorithm 25 BGP_{RPKI}.Certify: issue RCs and signed objects

```

1: operation BGPRPKI.Certify( $st, clk, tbc$ )
2:   if  $\left[ \begin{array}{l} tbc.ipResources \not\subseteq st.IPs \vee \\ tbc.asResources \not\subseteq st.ASNs \vee \\ (tbc.type = 'ROA' \wedge \text{VERIFYROATBC}(tbc) = \perp) \vee \\ (tbc.type = 'MFT' \wedge \text{VERIFYMFTTBC}(st, clk, tbc) = \perp) \end{array} \right]$  ▷ Check IP bounds
▷ Check ASN bounds
▷ See Alg. 38
▷ See Alg. 39
3:   return  $(st, \perp)$  ▷ Validation failure
4:   end if
5:   if  $tbc.type = 'EE'$  then
6:      $(EE.sk, EE.pk) \leftarrow C.KeyGen(params, 1^K)$ 
7:      $st.EEKeys \leftarrow (EE.sk, EE.pk)$  ▷ Generate and save EE keys
8:      $tbc \leftarrow \left\{ \begin{array}{l} ('pk', EE.pk), ('type', 'EE'), \\ ('subject', RandomUUID()) \end{array} \right\}$  ▷ Bind generated PK
▷ Random subject
9:   end if
10:   $out \leftarrow ('INVOKE', ('PKI', Certify, st.PKInstance, \{clk, tbc\}))$ 
11:  return  $(st, out)$ 
12: end operation

```

an updated state st' alongside an output $out = ('ResolveAnswer', d, type, value)$, where $value$ is either an IP address or $\perp$ in the event of a failure.

Resolution is not instantaneous: a resolver reaches its name-servers over the same IP layer we analyze, so its delay is network dependent, showing the complexity of analyzing in-

Algorithm 26 BGP_{RPKI}.PACKETROUTER

```

1: procedure BGPRPKI.PACKETROUTER( $st, pkt, sender$ )
2:   out  $\leftarrow \emptyset$ 
3:   switch ( $pkt.protocol$ ) do
4:     case BGP:
5:        $(st.bgp, out) \leftarrow \text{Update}(st.bgp, pkt.payload, sender)$  ▷ Pass routing up-  
dates to BGP
6:     case RPKI:
7:       switch ( $pkt.dst_ip$ ) do ▷ Route to internal RPKI subsystems
8:         case  $IP_{pp}, IP_{tal}$ :
9:            $(st, out) \leftarrow \text{HandleFetchRequest}(st, pkt)$  ▷ See Alg 33
10:        case  $IP_{rp}$ :
11:           $(st, out) \leftarrow \text{ProcessRepoResponse}(st, pkt, sender)$  ▷ See Alg 34
12:        default:
13:          out  $\leftarrow \perp$  ▷ Drop unrecognized des-  
tination
14:      end switch
15:    default:
16:      out  $\leftarrow \perp$ 
17:  end procedure

```

Algorithm 27 BGP_{RPKI}.WAKEUP: periodic maintenance

```

procedure BGPRPKI.WAKEUP( $st, clk$ )
1:   out  $\leftarrow \emptyset$ 
2:   if 'CA'  $\in st.roles$  then
3:     if  $st.manifest \neq \perp$  then
4:       out  $\leftarrow out \cup \{('INVOKE', ('PKI', Revoke, st.PKInstance, st.manifest))\}$ 
5:     end if
6:     out  $\leftarrow out \cup \{('INVOKE', ('PKI', Aux, st.PKInstance))\}$ 
▷ Invoke CRL generation
7:   end if
8:   if 'Router'  $\in st.roles \wedge st.rov$  then ▷ ROV adopter
9:      $(st, fetchOut) \leftarrow \text{UpdateCache}(st, clk)$  ▷ See Algorithm 30
10:    out  $\leftarrow out \cup fetchOut$ 
11:   end if
12:   out  $\leftarrow out \cup \{('WakeAt', clk)\}$  ▷ Schedule the next execution, waking  
up prior to  $clk + \Delta['wake']$ 
13:   return  $(st, out)$ 
end procedure

```

terdependent protocols such as BGP with RPKI and DNS. We therefore split resolution into a query event and a later answer event, and bound the delay between them by $\Delta['DNS']$.

Algorithm 28 BGP_{RPKI}.DECISION: Best Route Selection with ROV

```

procedure BGPRPKI.DECISION( $st, p$ )
1:   $out \leftarrow \emptyset$   $\triangleright$  Initialize message buffer
2:   $C \leftarrow \text{GATHERCANDIDATES}(st.bgp, p)$   $\triangleright$  See Algorithm 19
3:  if  $st.rov$  then
4:     $C \leftarrow \{r \in C \mid \text{CHECKROV}(st, p, r) \neq \text{'Invalid'}\}$   $\triangleright$  Drop RPKI-Invalid
5:  end if
6:  if  $C = \emptyset$  then
7:    if  $st.bgp.locRIB[p] \neq \emptyset$  then
8:       $st.bgp.locRIB[p] \leftarrow \perp$   $\triangleright$  Remove from local RIB
9:       $st.bgp.fib[p] \leftarrow \perp$   $\triangleright$  Remove route from FIB
10:      $(st.bgp.out) \leftarrow \text{PROPAGATE}(st.bgp, p, \perp)$   $\triangleright$  Announce route loss
11:     return  $(st, out)$ 
12:   end if
13:   return  $(st, \perp)$ 
14: end if
15:  $BestRoute \leftarrow \text{FINDBESTROUTE}(p, C)$   $\triangleright$  See Algorithm 20
16: if  $BestRoute = st.bgp.locRIB[p]$  then
17:   return  $(st, \perp)$   $\triangleright$  No change  $\rightarrow$  exit early
18: end if
19:  $st.bgp.locRIB[p] \leftarrow BestRoute$ 
20:  $st.bgp.fib[p] \leftarrow BestRoute.nextHop$   $\triangleright$  Install new best route
21:  $(st.bgp.out) \leftarrow \text{PROPAGATE}(st.bgp, p, BestRoute)$   $\triangleright$  Announce to neighbors
22: return  $(st, out)$ 
end procedure

23: procedure CHECKROV( $st, p, r$ )
24:   $V \leftarrow \{v \in st.VRPs \mid p \subseteq v.prefix\}$   $\triangleright$  Find covering VRPs
25:  if  $V = \emptyset$  then
26:    return 'NotFound'  $\triangleright$  No covering VRP exists
27:  end if
28:  if  $\exists v \in V$  s.t.  $r.asPath[|r.asPath| - 1] = v.asn \wedge \text{len}(p) \leq v.maxLen$  then
29:    return 'Valid'  $\triangleright$  Origin ASN (last-prepended aspath element) and length match
30:  end if
31:  return 'Invalid'  $\triangleright$  RFC 6811 validation failure
32: end procedure

```

J RIR Trust Anchor and Publication Point Deployment

This appendix motivates the RIR deployment model assumed in Section 7.4. To check our modeling assumptions against real deployments rather than take them for granted, we surveyed where and how the five RIRs deploy their trust anchors and PPs, covering both the RRDP and rsync PPs of each RIR. We set out to answer two questions. First, is the root certificate published on the same host as the RIR's PP? Second, who issues the ROAs covering those hosts, does each RIR certify its own infrastructure IPs, and are the resulting ROAs correctly configured?

J.1 Measurement and Validation

We surveyed all five RIRs. Each TAL names the root certificate by two URIs, one rsync and one HTTPS, both pointing at the same certificate file and the PP hosts named by the caRepository (rsync) and rpkiNotify (RRDP) URIs in the SIA extension of each TA certificate. Hostnames were resolved using two independent resolvers, and mapped to BGP origins and registries via RIPEstat [68] and Registration Data Access Protocol (RDAP). The covering ROAs were confirmed by

Algorithm 29 BGP_{RPKI}.Valid: Validity Check

```

1: operation BGPRPKI.Valid( $\psi, t, store, aux$ )
2:  switch ( $\psi.type$ )
3:    case 'ROA':
4:      return  $\left[ \begin{array}{l} \text{VALIDATEROA}(\psi) \wedge \\ \text{BGP}_{\text{RPKI}}.\text{Valid}(\psi.EE, t, store, aux) \end{array} \right]$   $\triangleright$  Verify ROA content and EE cert
5:    case 'MFT':
6:      return  $\left[ \begin{array}{l} \text{VALIDATEMANIFEST}(\psi, t) \wedge \\ \text{BGP}_{\text{RPKI}}.\text{Valid}(\psi.EE, t, store, aux) \end{array} \right]$   $\triangleright$  Verify MFT content and EE cert
7:    case 'EE', 'RC', 'CRL':
8:      return  $\exists \psi_{root} \in store$  s.t.  $\text{VALIDCHAIN}(\psi, t, \psi_{root}, aux) \neq \emptyset$   $\triangleright$  Chain to a trust anchor.
9:    default:
10:     return  $\perp$   $\triangleright$  Unknown object type
11:  end operation

12: procedure VALIDCHAIN( $\psi', t, \psi_{root}, aux$ )
13:  if  $\exists \xi \in aux, i \leq j \leq |\xi|$  s.t. then  $\triangleright$  Find valid path  $\xi$ 
14:     $\xi[i] = \psi'$   $\triangleright$  Path starts with  $\psi'$ 
15:     $\xi[j].issuer = \psi_{root}.subject$   $\triangleright$  Path ends at Root
16:     $\text{PKIX}_C.\text{Valid}(\xi[j], t, \psi_{root}, aux) \wedge$   $\triangleright$  Standard X.509 valid
17:     $\text{CHECKRESOURCECONTAINMENT}(\xi[j], \psi_{root}) \wedge$   $\triangleright$  Root resource check
18:     $\forall k, i \leq k < j :$   $\triangleright$  For all links in chain
19:      $\text{CHECKRESOURCECONTAINMENT}(\xi[k], \xi[k + 1]) \wedge$   $\triangleright$  RFC 3779 containment
20:      $\text{PKIX}_C.\text{Valid}(\xi[k], t, \psi_{root}, aux) \wedge$   $\triangleright$  Standard X.509 valid
21:      $\text{CERTISNOTREVOKED}(\xi[k], t, \psi_{root}, aux)$  then  $\triangleright$  Check CRL
22:     return  $\xi$ 
23:  end if
24:  return  $\emptyset$ 
25: end procedure

26: procedure CHECKRESOURCECONTAINMENT( $\psi_{child}, \psi_{parent}$ )
27:   $IP\_Valid \leftarrow (\psi_{child}.ipResources = \text{'INHERIT'}) \vee (\psi_{child}.ipResources \subseteq \psi_{parent}.ipResources)$ 
28:   $AS\_Valid \leftarrow (\psi_{child}.asResources = \text{'INHERIT'}) \vee (\psi_{child}.asResources \subseteq \psi_{parent}.asResources)$ 
29:  return  $IP\_Valid \wedge AS\_Valid$ 
30: end procedure

31: procedure VALIDATEROA( $\psi$ )
32:   $\psi_{EE} \leftarrow \psi.EE$ 
33:  return  $\left[ \begin{array}{l} C.\text{Verify}_{\psi_{EE}.pk}(\psi) \wedge \\ \psi_{EE}.is\_CA = \perp \wedge \\ \psi_{EE}.ipResources = \text{'INHERIT'} \wedge \\ \psi.payload.asID \neq \perp \wedge \\ \psi.payload.ipPrefixes \neq \emptyset \wedge \\ \forall (pfx, maxLen) \in \psi.payload.ipPrefixes : \\ (\exists b \in \psi_{EE}.ipResources : pfx \subseteq b) \wedge \text{len}(pfx) \leq maxLen \end{array} \right]$ 

 $\triangleright$  Verify CMS signature  

 $\triangleright$  EE cannot be a CA  

 $\triangleright$  RFC 9582: No IP inheritance  

 $\triangleright$  Payload names an origin AS  

 $\triangleright$  At least one prefix  

 $\triangleright$  Each prefix in EE scope, maxLen at least its length


34: end procedure
35: procedure VALIDATEMANIFEST( $\psi, t$ )
36:   $\psi_{EE} \leftarrow \psi.EE$ 
37:  return  $\left[ \begin{array}{l} C.\text{Verify}_{\psi_{EE}.pk}(\psi) \wedge \\ \psi_{EE}.is\_CA = \perp \wedge \\ \psi_{EE}.ipResources = \text{'INHERIT'} \wedge \\ \psi_{EE}.asResources = \text{'INHERIT'} \wedge \\ \psi.payload.thisUptime \leq t \leq \psi.payload.nextUptime \end{array} \right]$ 

 $\triangleright$  Verify CMS signature  

 $\triangleright$  EE cannot be a CA  

 $\triangleright$  RFC 9286: MUST inherit IP  

 $\triangleright$  RFC 9286: MUST inherit ASN  

 $\triangleright$  RFC 9286: t inside the claimed freshness window


38: end procedure
39: procedure CERTINMANIFEST( $\psi_{mft}, \psi$ )
40:  return  $\exists (fn, h) \in \psi_{mft}.payload.fileList : fn = \text{COMPUTEFILENAME}(\psi) \wedge h = \text{SHA256}(\psi)$ 
41: end procedure

```

Algorithm 30 Periodic Cache Synchronization

```

1: procedure UpdateCache(st, clk)
2:   out ← ∅ ▷ Initialize outgoing actions
3:   st.cache.certs ← {ψ ∈ st.cache.certs | ψ.to > clk} ▷ Remove stale certificates
4:   st.cache.TAs ← {ψ ∈ st.cache.TAs | ψ.to > clk} ▷ Remove stale root certificates
5:   (st.vrpOut) ← UpdateVRPs(st, clk) ▷ See Algorithm 35
6:   out ← out ∪ vrpOut
7:   for all ψ ∈ st.cache.certs do
8:     if ψ.type = 'RC' ∧ ψ.SIA ≠ ⊥ then ▷ Identify CA certificates that host repositories
9:       payload ← { ('Type', FetchRepo) }
10:      pkt ← { ('src_ip', st.IPKP),
              ('protocol', RPKI),
              ('payload', payload) } ▷ Build standard fetch packet
11:      for all url ∈ ψ.SIA do Query every publication point listed in the SIA (mirrors, RFC 6487)
12:        dnsQuery ← ('DNS', ResolveQuery, st.DNSInstance, url, A)
13:        out ← out ∪ {(pkt, ('INVOKE', dnsQuery))}
14:      end for
15:    end if
16:  end for
17:  return st, out ▷ Return updated state and queued actions
18: end procedure

```

Algorithm 31 UpdateValidationCache

```

procedure UpdateValidationCache(st, clk, repoData)
1: store ← st.cache.TAs, certs ← st.cache.certs
2: for all issuer, certs ∈ repoData do
3:   if ∃ ψmft, ∃ ψcrl ∈ certs s.t. then ▷ Locate MFT & CRL
4:     ψmft.type = 'MFT' ∧ ▷ Is Manifest
5:     ψcrl.type = 'CRL' ∧ ▷ Is CRL
6:     (ξmft ← CONSTRUCTCHAIN(ψmft, store, certs)) ≠ ∅ ∧ ▷ See Algorithm 36
7:     CERTINMANIFEST(ψmft, ψcrl) ∧ ▷ CRL listed in MFT, Line 39
8:     (ξcrl ← CONSTRUCTCHAIN(ψcrl, store, certs)) ≠ ∅ ∧
9:     BGP_RPKI.Valid(ψcrl, clk, st.TAs, ξcrl) ≠ ⊥ ∧ ▷ See Algorithm 25
10:    BGP_RPKI.Valid(ψmft, clk, st.TAs, (ξcrl, ξmft)) ≠ ⊥ ∧
11:    ∀ ψ' ∈ certs \ {ψmft, ψcrl} : ▷ Verify other files
12:      CERTINMANIFEST(ψmft, ψ') ∧
13:      (ξ ← CONSTRUCTCHAIN(ψ', store, certs)) ≠ ∅ ∧
14:      BGP_RPKI.Valid(ψ', clk, st.TAs, (ξ, ξcrl)) ≠ ⊥ then
15:        st.cache.certs ← st.cache.certs ∪ certs ▷ Commit strictly validated bundle
16:      end if
17:    end for
18:  return st
end procedure

```

two independent sources, console.rpki-client.org VRP snapshot [70] and a local Routinator 0.15.2 [61] run.

J.2 Co-location and Hosting Style

We began by comparing the TA certificate hosts to the RIR PP hosts. We found that for every RIR, the TA certificate host is one of that RIR's PP hosts. This means that the root certificate is always co-published with the PP at the same hostname. AFRINIC, APNIC and RIPE serve it from their rsync PP host, while for ARIN and LACNIC the two TAL URIs span both PP hosts.

We next resolve each hostname and identified three distinct deployment styles (Table 4):

1. *Fully self-hosted (ARIN)*: Root certificate host, rsync

Algorithm 32 CreateManifestTBC

```

procedure CreateManifestTBC(st, clk)
1: fileHashes ← ∅ ▷ Initialize file hash list
2: for each ψ' ∈ st.certs do
3:   fn ← COMPUTEFILENAME(ψ')
4:   h ← SHA256(ψ')
5:   fileHashes ← fileHashes ∪ {(fn, h)} ▷ Map filename to current hash
6:   st.manifestNum ← st.manifestNum + 1
7:   payload ← { ('manifestNumber', st.manifestNum), ▷ Manifest version number
               ('thisUpdate', clk), ▷ Issuance time
               ('nextUpdate', clk + st.Δman), ▷ Next scheduled update
               ('fileList', fileHashes) } ▷ Repository inventory
8:   tbc ← { ('type', 'MFT'), ▷ Manifest object flag
            ('issuer', st.i), ▷ Entity ID of the CA
            ('from', clk), ▷ Issuance time
            ('to', clk + st.Δman), ▷ Expiry time
            ('ipResources', 'INHERIT'), ▷ RFC 3779 resource rule
            ('asResources', 'INHERIT'), ▷ RFC 3779 resource rule
            ('payload', payload) } ▷ CMS content
9:   return tbc
end procedure

```

Algorithm 33 HANDLEFETCHREQUEST: serve repository objects

```

1: procedure HandleFetchRequest(st, pkt)
2:   out ← ∅; data ← ⊥; type ← ⊥ ▷ Default to fail-safe drop
3:   switch (pkt.dst_ip) ▷ Route by destination IP
4:     case IPPP: data ← st.repoCache; type ← 'RepoData' ▷ Serve ROAs and Manifests
5:     case IPTAL: data ← st.rootCert; type ← 'RootCertificate' ▷ Serve Trust Anchor (TAL)
6:   if data ≠ ⊥ then
7:     payload ← { ('Type', type), ('Data', data) }
8:     pkt ← { ('src_ip', pkt.dst_ip), ('dst_ip', pkt.src_ip),
              ('protocol', RPKI), ('payload', payload),
              ('id', GetRandomPktId()) }
▷ Reply addressed back to the requester
9:     (st.bgp, outsend) ← BGP.SENDPACKET(st.bgp, pkt)
10:    out ← out ∪ outsend ▷ Send response back to requester
11:  end if
12:  return (st, out)
13: end procedure

```

Algorithm 34 PROCESSREPORESPONSE: processes a repository response

```

1: procedure ProcessRepoResponse(st, clk, pkt, sender)
2:   out ← ∅
3:   payloadType ← pkt.payload.Type ▷ Identify incoming data phase
4:   if payloadType = 'RootCertificate' then ▷ Bootstrapping
5:     ψ ← pkt.payload.Data
6:     isValid ← PKI.Valid(ψ, clk, ∅, ∅) ▷ Verify expiry and self-signature
7:     if ∃ tal ∈ st.TALS s.t. ψ.pk = tal.pk ∧ isValid then ▷ Validate against configured TALs
8:       st.cache.TAs ← st.cache.TAs ∪ {ψ} ▷ Install verified root of trust
9:     end if
10:   else if payloadType = 'RepoData' then ▷ Bulk Sync
11:     repos ← pkt.payload.Data
12:     for each repo ∈ repos do
13:       st ← UpdateValidationCache(st, clk, repo) ▷ See Algorithm 31
14:     (st, out) ← UpdateVRPs(st) ▷ See Algorithm 35
15:   end if
16:   return (st, out) ▷ Return updated validation state
17: end procedure

```

PP and RRDP PP all run on ARIN-operated, ARIN-registered address space, all announced by three ARIN

Algorithm 35 BGP_{RPKI}.UpdateVRPs: apply cache changes and trigger BGP

```

1: procedure UpdateVRPs(st, clk)
2:   out ← ∅
3:   activeVRPs ← ∅ ▷ Fresh set for current valid world-view
4:   store ← st.cache.TAs, certs ← st.cache.certs
5:   for each  $\psi \in st.cache.certs$  do
6:     isValid ← ConstructAndValidate( $\psi, clk, store, certs$ ) ▷ See Algorithm 37
7:     if  $\psi.type = 'ROA' \wedge isValid \neq \perp$  then
8:       activeVRPs ← activeVRPs  $\cup$  ExtractVRPs( $\psi$ ) ▷ Gather all currently valid mappings
9:     end if
10:    if activeVRPs  $\neq st.VRPs$  then ▷ Detect both additions AND stale deletions
11:       $P_{eval} \leftarrow \{v.prefix \mid v \in (activeVRPs \cup st.VRPs)\}$  ▷ Gather all affected prefixes
12:       $st.VRPs \leftarrow activeVRPs$  ▷ Overwrite state to purge stale VRPs
13:      for each  $p \in P_{eval}$  do
14:        ( $st.bgpOut$ ) ← BGPRPKI.DECISION( $st, p$ ) ▷ See Algorithm 28
15:        out ← out  $\cup$   $st.bgpOut$  ▷ Queue any routing updates
16:      end if
17:    return ( $st, out$ )
18: end procedure

```

Algorithm 36 ConstructChain

```

procedure ConstructChain( $\psi_{target}, store, certs$ )
1: if  $\psi_{target} \in store$  then
2:   return [ $\psi_{target}$ ] ▷ Found Root, start chain
3: end if
4: if  $\exists \psi_{parent} \in certs.s.t.$ 
5:    $\left[ \begin{array}{l} \psi_{target}.issuer = \psi_{parent}.subject \wedge \\ C.Verify(\psi_{parent}.pk(\psi_{target})) \end{array} \right]$  ▷ Issuer/Subject Match ▷ Signature Valid
6: then:
7:    $\xi_{parent} \leftarrow CONSTRUCTCHAIN(st, \psi_{parent})$  ▷ Recurse up
8:   if  $\xi_{parent} \neq \emptyset$  then
9:     return  $\xi_{parent} ++ [\psi_{target}]$ 
10:   end if
11:   return ∅ ▷ Chain broken
end procedure

```

Algorithm 37 ConstructAndValidate

```

procedure ConstructAndValidate( $\psi_{target}, clk, store, certs$ )
1:  $\xi_{target} \leftarrow CONSTRUCTCHAIN(\psi_{target}, store, certs)$ 
2: if  $\xi_{target} = \emptyset$  then
3:   return  $\perp$  ▷ Path building failed
4: end if
5:  $\psi_{CRL} \leftarrow \{\psi \in certs \mid \psi.type = 'CRL' \wedge \psi.issuer = \psi_{target}.issuer\}$ 
6:  $\xi_{CRL} \leftarrow CONSTRUCTCHAIN(\psi_{CRL}, store, certs)$ 
7: if  $\xi_{CRL} = \emptyset$  then
8:   return  $\perp$  ▷ Path building failed
9: end if
10: return BGPRPKI.Valid( $\psi_{target}, clk, store, (\xi_{target}, \xi_{CRL})$ ) ▷ See Algorithm 25
end procedure

```

ASes (AS393220, AS393225, and AS394018) across three sites, with no CDN.

2. *Self-hosted root, CDN-fronted RRDP (RIPE, APNIC, AFRINIC)*: The root certificate host and rsync PP run on the RIR's own AS and address space (AS3333 for RIPE, AS4608 for APNIC, AS33764 for AFRINIC), but the RRDP PP is a CNAME to a CDN: Cloudflare (AS13335) for APNIC and AFRINIC, the Akamai CDN for RIPE. The CDN-fronted RRDP host therefore leaves the RIR's own address space and, because CDN edge selection is

Algorithm 38 VerifyROAtbc

Description: Validates a ROA EE profile per RFC 9582. The payload naming one AS and at least one prefix, every prefix covered by an EE resource block, and maxLen $\geq$ prefix length.

```

procedure VerifyROAtbc( $tbc$ )
1: ( $tbc, payload$ ) ← ( $tbc \setminus \{ 'payload' \}, tbc.payload$ ) ▷ Split the EE template from the ROA payload
2: if  $\neg \left[ \begin{array}{l} (tbc.asResources \in \{\perp, 'INHERIT'\}) \wedge \\ (tbc.ipResources \neq 'INHERIT') \wedge \\ (tbc.isCA = \perp) \end{array} \right]$  then
3:   return  $\perp$  ▷ EE has AS resources, inherits IP, or is a CA
4: end if
5: if  $payload.asID = \perp \vee payload.ipPrefixes = \emptyset$  then
6:   return  $\perp$  ▷ A ROA names one AS and a non-empty prefix set
7: end if
8: for all ( $pfx, maxLen$ )  $\in payload.ipPrefixes$  do
9:   if  $\nexists b \in tbc.ipResources : pfx \subseteq b$  then
10:    return  $\perp$  ▷ Prefix not covered by any EE resource block
11:   end if
12:   if  $maxLen < len(pfx)$  then
13:    return  $\perp$  ▷ RFC 9582 requires maxLen  $\geq len(prefix)$ 
14:   end if
15: end for
16: return  $\top$ 
end procedure

```

Algorithm 39 VerifyMFTtbc

Description: Validates a manifest based on RFC 9286.

```

procedure VerifyMFTtbc( $st, clk, tbc$ )
1:  $p \leftarrow tbc.payload$  ▷ Extract payload
2: return  $\left[ \begin{array}{l} (tbc.ipResources = 'INHERIT') \wedge \\ (tbc.asResources = 'INHERIT') \wedge \\ (tbc.isCA = \perp) \wedge \\ (clk \leq p.nextUpdate) \wedge \\ (p.thisUpdate < p.nextUpdate) \wedge \\ (p.manifestNumber = st.manifestNum) \wedge \\ \forall \psi' \in st.certs : CERTINMANIFEST(tbc, \psi') \end{array} \right]$ 

```

▷ inherits IP
 ▷ inherits ASN
 ▷ EE cannot be a CA
 ▷ Window is well formed
 ▷ This CA's current number
 ▷ Lists every published RPKI object, Line 39

resolver-dependent, has no single global (prefix, origin-AS) pair.

3. *Fully third-party cloud (LACNIC)*: The root certificate host and both PP services run entirely on Akamai Cloud (AS63949), this is a different platform from the Akamai CDN that fronts RIPE's RRDP.

The hosting style determines which trust-anchor tree protects each RIR, since a prefix is protected by whichever tree issues its covering ROA: cloud- and CDN-hosted addresses inherit the hosting provider's ROAs, which often descend from a *different* RIR's tree than the one being served. The right-most column of Table 4 gives the resulting dependency matrix. Every covering ROA for LACNIC's infrastructure descends from the ARIN and RIPE trees (Akamai prefixes), so the LACNIC tree protects none of it own, and the CDN-fronted RRDP endpoints of AFRINIC and APNIC likewise depend on the ARIN tree through the Cloudflare address space. ARIN

Algorithm 40 BGP_{RPKI}: composition callbacks

```

1: procedure BGPRPKI.DNS-RESOLVEQUERY-RESP(ip_address, pkt)
2:   pkt.dst_ip ← ip_address           ▷ Set the new packet's destination to the resolved IP
3:   (st.bgp, out) ← BGP.SENDPACKET(st, pkt)
4:   return (st, out)
5: end procedure
6: procedure BGPRPKI.PKI-INIT-RESP(ψ)
7:   st.rootCert ← ψ
8:   if st.roles ∈ {'RIR'} then
9:     TAL ← { ('PublicKey', st.rootCert.pk),
              ('URL', "params.IP_TAL") }
10:    out += TAL           ▷ Export TAL
11:  end if
12:  return (st, out, ψ)
13: end procedure
14: procedure BGPRPKI.PKI-CERTIFY-RESP(ψ, tbc)
15:   if ψ.type = 'EE' then
16:     tbcMS ← { ('payload', tbc.payload),
                 ('EE', ψ), ('type', tbc.type) }
17:     Find (sk, pk) ∈ st.EEKeys such that pk == ψ.pk
18:     ψ ← C.Signsk(tbcMS)           ▷ Encapsulate payload
19:   end if
20:   st.certs += ψ
21:   return (st, ψ)
22: end procedure
23: procedure BGPRPKI.PKI-AUX-RESP(ψcrit, clk)
24:   st.certs ← st.certs ∪ {ψcrit}     ▷ Store CRL for manifest generation
25:   tbcman ← CreateManifestTBC(st, clk)   ▷ See Algorithm 32
26:   out ← ('INVOKE', ('PKI', Certify, st.PKIinstance, {clk, tbcman}))
27:   return (st, out)
28: end procedure

```

on the other hand is fully self served. ARIN therefore protects the infrastructure of four of the five RIRs (its own, LACNIC's, and the RRDP side of AFRINIC's and APNIC's), giving it a structurally central role in the reachability of the whole system.

J.3 Evaluating ROA Coverage and Conflicts

We further examined the covering ROAs, and were interested in two specific questions. First, does their maxLen attribute follow the guidance of RFC 9319 [23]? Second, are there any conflicting ROAs that could cause invalidations? Across the five RIRs, the covering ROAs fall into two categories:

- *Owned-ROAs*: For the self-hosted infrastructure, nearly all of these are minimal in the sense of RFC 9319 [23], the ROA prefix equals the announced prefix and the maxLen equals the prefix length. These are the cleanest cases and admit no conflict. The one exception is AFRINIC's IPv4 ROA 'AS33764 196.216.2.0/23' with a maxLen of 24; this extra bit of slack leaves it open to a forged-origin subprefix hijack of rpki.afrinic.net, which is both AFRINIC's TA-certificate host and its rsync PP.
- *Provider aggregates ROAs*: In cases where RIRs utilize cloud- or CDN-hosted infrastructure, ROA coverage differs. For example, Cloudflare, which fronts the RRDP endpoints of APNIC and AFRINIC, uses exact-match ROAs published in the ARIN tree, multiple overlapping ROA exists but are published in the same PP. The Aka-

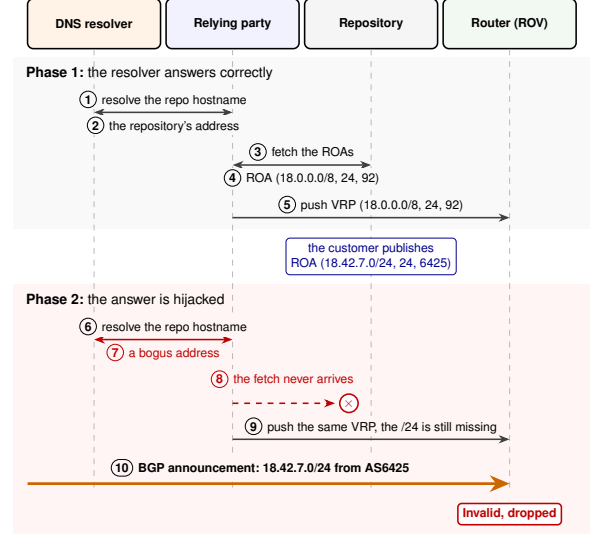


Figure 9: RPKI sync failure under a DNS hijack. AS92 holds 18.0.0.0/8 and delegates 18.42.7.0/24 to AS6425. **Phase 1:** the relying party resolves the repository hostname in ① and ②, fetches the ROAs in ③ and ④, and pushes the /8 VRP to the router in ⑤. **Phase 2:** the /24 ROA is published, but the resolver returns a bogus address in ⑥ and ⑦, so the fetch fails and the new /24 ROA is never learned in ⑧. The relying party keeps pushing only the covering /8 VRP in ⑨, so AS6425's announcement matches it by prefix (maxLen 24) but not by origin in ⑩, and the router drops it as Invalid. The announcement in ⑩ arrives from a BGP neighbor outside the figure.

mai CDN that fronts the RIPE RRDP is geo-dependent, meaning that different vantage points resolve it to different addresses. Liu et al. [45] report that in 12 countries the resolved addresses lack ROA coverage altogether. The Akamai Cloud carries LACNIC's root certificate and both of its PPs, there exist three valid signed ROAs that covers the IP addresses, all published in ARIN's PP. The first 'AS63949 172.233.0.0/19' with a maxLen of 24 authorizes the announcement used in practice. There exist two more ROAs: AS20940 172.232.0.0/13 and 172.224.0.0/12, that cover the same used IP but the maxLen that they use stop at their own aggregate length and will evaluate the used announcement as *invalid* if the first ROA was not shipped with them. That maxLen of 24 makes 172.233.0.0/19 susceptible to a global, wide-range, forged-origin subprefix hijack, and LACNIC can neither issue nor tighten any of these ROAs.

J.4 The RIR Deployment Model

The survey above motivates the RIR deployment model, formalized by the $\mu^{\text{RIRDeploy}}$ predicate (Algorithm 69). We as-

Algorithm 41 Δ -DNS resolution correctness requirement $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}} = \{(\rho_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}, 0)\}$

Input: Transcript T .

Output: $\perp$ if the adversary demonstrates that an honest entity output a record mapping a domain and type to a value, and a subsequent resolution query (after at least Δ time, but within the record's ttl) for the same domain and type by an honest entity was not answered with that value within $\Delta[\text{'DNS'}]$; otherwise, T .

```

1: requirement predicate  $\rho_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}(T)$ 
2:   return  $\nexists e_1, e_2 \in \{1, \dots, T.e\} \text{ s.t.}$   $\triangleright$  Adversary fails to find an announcement and a query contradicting it
3:      $T.ent[e_1].sp, T.ent[e_2].sp == \text{DNS} \wedge$   $\triangleright$  Both of DNS process
4:      $T.ent[e_1], T.ent[e_2] \in T.N - T.F \wedge$   $\triangleright$  Both are honest
5:      $T.\tau[e_1] + \Delta < T.\tau[e_2] \wedge$   $\triangleright$  Event 1 happens at least  $\Delta$  before Event 2
6:      $\exists d, type, ttl, value \text{ s.t.}$ 
7:        $T.out[e_1] = (\text{'AnnounceRecord'}, d, type, ttl, value) \wedge$   $\triangleright$  Event 1: Asserts domain record leads to value for duration  $ttl$ 
8:        $T.opr[e_2] = \text{ResolveQuery} \wedge$ 
9:        $T.inp[e_2] = (d, type) \wedge$   $\triangleright$  Event 2: Issues a query for the same domain and type
10:       $T.\tau[e_2] \leq T.\tau[e_1] + ttl \wedge$   $\triangleright$  Event 2 happens before the announced record expires
11:       $\neg \text{ANSWEREDWITH}(T, e_2, d, type, value)$   $\triangleright$  The query is not answered with the announced value in time
12: end requirement predicate

13: procedure  $\text{ANSWEREDWITH}(T, e_q, d, type, value)$ 
    $\triangleright$  True iff the resolution issued at event  $e_q$  is completed with value, at the same entity, no later than  $\Delta[\text{'DNS'}]$  after the query was issued
14:   return  $\exists e_3 \in \{1, \dots, T.e\} \text{ s.t.}$ 
15:      $T.opr[e_3] = \text{ResolveQueryAnswer} \wedge$ 
16:      $T.ent[e_3] = T.ent[e_q] \wedge$   $\triangleright$  Answer event of the same resolver
17:      $T.inp[e_3] = T.out[e_q].qid \wedge$   $\triangleright$  Answering that very query, and not another one
18:      $T.\tau[e_q] \leq T.\tau[e_3] \leq T.\tau[e_q] + \Delta[\text{'DNS'}] \wedge$   $\triangleright$  The answer arrives within  $\Delta[\text{'DNS'}]$  of the query
19:      $T.out[e_3] = (\text{'ResolveAnswer'}, d, type, value)$   $\triangleright$  And it carries the announced value
20: end procedure

```

Table 4: Deployment of each RIR’s fetch surface (root-certificate and PP hosts) and the trust-anchor tree(s) whose ROAs protect its critical IPs, as of July 2026. CDN and cloud placement is evidenced by the CNAME chain and the HTTP fingerprint. Note the two distinct Akamai products: RIPE’s RRDP is fronted by the Akamai *CDN* (akamaized.net), whereas LACNIC’s root and PP run on Akamai *Cloud* (IaaS).

RIR	Root + rsync PP	RRDP PP	Address-space reg-istry	Deployment style	Protecting trust-anchor tree(s)
ARIN	self-hosted (3 sites)	self-hosted, same pre-fixes	ARIN (own)	fully self-hosted	ARIN only (fully self-referential)
RIPE NCC	self-hosted	Akamai CDN (akamaized.net)	RIPE (own); edge varies	self-host root, CDN RRDP	RIPE only (RRDP edge ROA belongs to a third party at this vantage)
APNIC	self-hosted	Cloudflare CDN	APNIC (own); Cloudflare in ARIN space	self-host root, CDN RRDP	APNIC + ARIN (ARIN for the Cloudflare RRDP edge)
AFRINIC	self-hosted	Cloudflare CDN	AFRINIC (own); Cloudflare in ARIN space	self-host root, CDN RRDP	AFRINIC + ARIN (ARIN for the Cloudflare RRDP edge)
LACNIC	Akamai Cloud (IaaS)	Akamai Cloud (IaaS)	ARIN + RIPE (<i>not</i> LACNIC)	fully third-party cloud	ARIN + RIPE; the LACNIC tree protects none of it

sume that every RIR hosts its own root certificate and its own PP on infrastructure it originates itself, so the trust anchor and its publication point are co-located in the same domain. While the measured deployments deviate from this idealization (three RIRs front their RRDP PP with a CDN and LACNIC runs entirely on third-party cloud, see Table 4), the model

abstracts these away: since our analysis does not distinguish between the two fetch protocols (rsync and RRDP), each retrieval attempt iterates over all URIs supplied in the certificate, and abstract away the exact retrieval mechanism. Furthermore, we assume that all critical infrastructure is protected by a valid and tight ROA, that is, an exact-match ROA whose prefix

equals the announced prefix and whose maxLength equals that prefix length, leaving no room for a more-specific hijack. For the RIR RPs, we assume the same conditions. While no public records on these RPs are available, we believe that RIRs know best practices and hold the same high standards for all RPKI infrastructure.

K Measurements Details

We crawled the global RPKI repository tree using Routinator [61], starting from the five RIRs. During the validation process, Routinator stores a local cache of all fetched repository objects organized by hostname, from which we extract a total of 83 unique PP hostnames. We resolve each hostname via DNS resulting in 110 unique IPv4 addresses. Of the 83 hostnames, 3 failed the DNS lookup. For each IP address, the ASNs for the PPs were obtained via an *ipwhois* [73] lookup.

For each resolved IP address and ASN, we check the set of Validated ROA Payloads (VRPs) produced by Routinator to determine whether a valid ROA exists for a prefix covering the repository IP address. For PPs with ROA coverage, we further evaluate whether the maxLength attribute is correctly configured per RFC 9319 [23]. Specifically, a ROA is considered misconfigured if the prefix is shorter than /24 but has a longer maxLength, as this permits forged-origin subprefix hijacks that would be considered valid by ROV.

By hostname, 42 (52.5%) PPs had all IP addresses correctly covered by a ROA, 26 (32.5%) had at least one addresses which is covered by a ROA with a misconfigured maxLength, and 12 (15.0%) had no ROA for any of their addresses.

To determine ROV adoption, we cross-referenced the set of PP ASNs with an ROV adoption dataset. ROV adoption data was collected using the *rov_collector* library [19], which aggregates measurements from a variety of independent datasets [12, 31, 38, 44, 67, 69, 75]. We found that 43 (53.8%) of PPs are hosted in an AS that at least one of the datasets collected by the *rov_collector* [19] lists as ROV adopting. Requiring unanimous consensus across the datasets instead lowers this to 40 (50.0%).

The above measurement results are for PPs. However, in practice, communication between PPs and RPs is bidirectional, as the receiver must acknowledge receipt of data. Since RPs can be hosted in any AS, we obtain some statistics for Internet-wide adoption of ROV and ROAs. From the *rov_collector*, 8,435 ASes were identified by at least one source as deploying ROV. Additionally, the RPKI-ROV analysis from the NIST RPKI Monitor [60] reports 68.08% of unique IPv4 prefix-origin pairs are currently valid by ROA as of July 2026. Therefore, to ensure connectivity, we should ensure that the ASes that host PPs or RPs perform ROV and possess the appropriate ROAs.

L Misrouting under Prefix and Subprefix Attacks

This appendix motivates the models used in Section 8. To decouple BGP from the RPKI retrieval mechanism, we begin our analysis under the assumption that every ROV-adopting AS holds a valid ROA for the prefix originated, which is captured by the $\mu_{\delta_{e2e}}^{\text{ROA-InCache}}$ predicate (Algorithm 63). Under this assumption, we present a new form of attack, which we refer to as a *misrouting attack*, and consider it under different sets of models.

L.1 Misrouting via Prefix Attacks

Theorem 2 establishes connectivity against a scheduler adversary. A *prefix adversary* has stronger capabilities than the scheduler adversary: it can place malicious ASes in the topology and may inject arbitrary BGP announcements or drop any traffic. When facing such an adversary, some models are required to ensure that the adversary does not gain an advantage by the AS graph topology. The $\mu_{\text{AS}_{\text{Dst}}}^{\text{HonestBestPaths}}$ model (Algorithm 44), ensures that the adversary does not control ASes on selected routes, because such an adversary can trivially intercept packets without needing to perform a BGP hijack.

We state the results of this subsection under the model $\mathcal{M}_{f,\Delta}^{\text{RPKI-Prefix}} \equiv (\mu_{f,\Delta}^{\text{RPKI-Prefix}}, 0)$, where:

$$\mu_{f,\Delta}^{\text{RPKI-Prefix}} \equiv \left\{ \mu_{f,\Delta}^{\text{BGP}} \wedge \mu_{\text{AS}_{\text{Dst}}}^{\text{HonestBestPaths}} \wedge \left(\mu_{\text{Prefix}}^{\text{HijackAdv}} \wedge \mu_{\delta_{e2e}}^{\text{ROA-InCache}} \right) \right\} \quad (5)$$

$\mu_{f,\Delta}^{\text{BGP}}$ (Equation (2)) captures the BGP protocol settings. The $\mu_{\text{Prefix}}^{\text{HijackAdv}}$ predicate (Algorithm 45) limits corrupted ASes to prefix hijacks, ruling out subprefix hijacks, route leaks, origin spoofing, and control of a CA.

We begin with a variant under partial ROV adoption, in which we require that every AS on the best paths from AS_{Src} to AS_{Dst} adopts ROV, as formalized by the $\mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$ predicate (Algorithm 48). Prior work [24, 55] identified non-adopting ASes on the best path as the weak point that leaves RPKI susceptible to hijacks, so we thought that $\mu_{f,\Delta}^{\text{RPKI-Prefix}} \wedge \mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$ suffices for connectivity. It does not, due to *misrouting attacks*, in which the adversary steers traffic away from a secure path and diverts it toward a plausible path that contains a malicious AS. Figure 10 shows an example. AS99 originates 1.2/16, and the best path from AS77 is 77-9-10-12-99, made up of adopters as $\mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$ requires. AS666 announces 1.2/16 to its provider AS44, which does not adopt ROV and therefore prefers 44-666 and stops exporting 44-55-66-99 to AS92. AS92 rejects 44-666 by ROV, but that leaves it with no customer route, so it falls back to 92-56-99, learned from its peer AS56. AS77 prefers the shorter 77-92-56-99, leaves its best path, and its packets reach AS56, a passive malicious AS that drops them.

RP-cache extension discussed in §9.2.1 and the operational recommendation discussed in §9.2.3 that RPs be covered by ROAs and hosted in ROV-adopting ASes. The $\mu_{\Delta[\text{'mirror'}]}^{\text{SharedMirror}}$ predicate (Algorithm 72), enforces the redundancy proposal of Section 9.2.2, together with the recommendation to protect the critical RPKI infrastructure with ROA and ROV. We assume a stronger prefix attacker via $\mu_{\text{Prefix}}^{\text{HijackAdv}}$. We refer to the resulting model as $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Retrieval}} \equiv \{(\mu_{f,\Delta}^{\text{Hybrid-Retrieval}}, 0)\}$.

We now prove that under the retrieval model, a ROA published at an RIR PP is actually retrieved by the benign adopters in time.

LEMMA 5. *Let $G = (V, E, R)$ be the AS topology graph, let $v \in V$ be a benign ROV-adopting AS, and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Under the $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Retrieval}}$ model, assuming black-box protocols that satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, if a ROA ψ is published at an RIR PP by some time $t \geq t^{\text{sync}}$, then the RP of v holds ψ , validated in its cache, by time $t + \Delta[\text{'mirror'}] + (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}})$, except with negligible probability, where t^{init} is the time by which every benign RIR and v have initialized and provided that*

$$\begin{aligned} t^{\text{sync}} &= t^{\text{init}} + \Delta[\text{'orig'}] + \Delta[\text{'mirror'}] + c_{\text{prop}} \\ &\quad + (\text{diam}_{\text{VF}}(G) + 1) \\ &\quad (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}}) \end{aligned}$$

Proof. See Appendix N.1. $\square$

The full *Robust* model adds the destination-side topological conditions of $\mathcal{M}_{f,\Delta}^{\text{HonestROVPref}}$ (Section 8) together with $\mu_{\Delta[\text{'dest'}]}^{\text{DestROAReady}}$, which we motivate and define below:

$$\mu_{f,\Delta}^{\text{Hybrid-Robust}} \equiv \left\{ \begin{array}{l} \mu_{f,\Delta}^{\text{Hybrid-Retrieval}} \wedge \mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}} \wedge \\ \mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{HonestPaths}} \wedge \mu_{\Delta[\text{'dest'}]}^{\text{DestROAReady}} \end{array} \right\} \quad (8)$$

The $\mu_{\Delta[\text{'dest'}]}^{\text{DestROAReady}}$ predicate (Algorithm 64) requires the ROA authorizing AS_{Dst} for p to be published at an RIR PP by $t - \Delta[\text{'dest'}]$. We prove the theorem below under $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Robust}} \equiv \{(\mu_{f,\Delta}^{\text{Hybrid-Robust}}, 0)\}$.

THEOREM 4 (BGP_{RPKI} satisfies $(\delta_{e2e}, \delta_{\text{boot}})$ -SR-IP connectivity under the Robust model). *Let $G = (V, E, R)$ be an AS topology graph and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$, $c_{\text{fetch}} \equiv \Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2 \cdot c_{\text{prop}}$. The BGP_{RPKI} protocol asymptotically satisfies the $\mathcal{R}_{\delta_{e2e}, \delta_{\text{boot}}}^{\text{SR-IP-CON}}$ requirement assuming the $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Robust}}$ model holds asymptotically and blackbox protocols that asymptoti-*

cally satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, provided that:

$$\begin{aligned} \delta_{e2e} &\geq c_{\text{prop}} \\ \Delta[\text{'conf'}] &\geq \Delta[\text{'mirror'}] \\ \Delta[\text{'dest'}] &\geq \delta_{e2e} + \Delta[\text{'mirror'}] + c_{\text{fetch}} \\ \delta_{\text{boot}} &\geq \Delta[\text{'orig'}] + \Delta[\text{'mirror'}] \\ &\quad + \Delta[\text{'dest'}] + (\text{diam}_{\text{VF}}(G) + 1) \cdot c_{\text{fetch}} \end{aligned}$$

Proof sketch. $\mu_{f,\Delta}^{\text{Hybrid-Robust}}$ contains every model of $\mu_{f,\Delta}^{\text{HonestROVPref}}$ except $\mu_{\delta_{e2e}}^{\text{ROA-InCache}}$, so it suffices to show that the ROA covering the destination can be retrieved by $t - \delta_{e2e}$, which is exactly what $\mu_{\delta_{e2e}}^{\text{ROA-InCache}}$ ensures.

$\mu_{f,\Delta}^{\text{Hybrid-Robust}}$ contains $\mu_{f,\Delta}^{\text{Hybrid-Retrieval}}$, so Lemma 5 applies. It guarantees retrieval only for ROAs published at or after its bootstrap instant t^{sync} . The requirement gives $t^{\text{init}} \leq t - \delta_{e2e} - \delta_{\text{boot}}$, and the stated δ_{boot} then yields $t^{\text{sync}} \leq t - \Delta[\text{'dest'}]$:

$$\begin{aligned} t^{\text{sync}} &= t^{\text{init}} + \Delta[\text{'orig'}] + \Delta[\text{'mirror'}] + c_{\text{prop}} \\ &\quad + (\text{diam}_{\text{VF}}(G) + 1) \\ &\quad (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2 \cdot c_{\text{prop}}) \\ &\leq (t - \delta_{e2e} - \delta_{\text{boot}}) + \Delta[\text{'orig'}] + \Delta[\text{'mirror'}] + c_{\text{prop}} \\ &\quad + (\text{diam}_{\text{VF}}(G) + 1) \\ &\quad (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2 \cdot c_{\text{prop}}) \\ &\leq t - \delta_{e2e} + c_{\text{prop}} - \Delta[\text{'dest'}] \\ &\leq t - \Delta[\text{'dest'}] \end{aligned}$$

The first line is the definition of t^{sync} in Lemma 5, while the second and third inequalities follow from the provided δ_{boot} and $c_{\text{prop}} \leq \delta_{e2e}$.

$\mu_{\Delta[\text{'dest'}]}^{\text{DestROAReady}}$ gives a ROA ψ authorizing AS_{Dst} for p , published at an RIR PP by $t - \Delta[\text{'dest'}]$ and valid throughout the requirement window. Its publication time is therefore at least t^{sync} , meaning the lemma applies to it. Ensuring that every benign adopter holds ψ , validated, within $\Delta[\text{'mirror'}] + (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}})$ of publication. By the choice of $\Delta[\text{'dest'}]$ that is by $t - \delta_{e2e}$, which is what $\mu_{\delta_{e2e}}^{\text{ROA-InCache}}$ guarantees.

The remaining models of $\mu_{f,\Delta}^{\text{Hybrid-Robust}}$, namely $\mu_{f,\Delta}^{\text{BGP}}$, $\mu_{\text{Prefix}}^{\text{HijackAdv}}$, $\mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{AdoptedBP}}$, and $\mu_{\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}}^{\text{HonestPaths}}$, complete $\mu_{f,\Delta}^{\text{HonestROVPref}}$, so Theorem 3 applies unchanged and delivers the packet by $t + \delta_{e2e}$. $\square$

For the *shared mirroring* model, we define well-connected ASes.

DEFINITION 8 (Well-connected ASes). *Let G be an AS topology graph and let A and B be ASes. We say that A and B are well-connected if, in both directions, every best path between them consists solely of ROV-adopting ASes ($\mu_{A,B}^{\text{AdoptedBP}}$ and $\mu_{B,A}^{\text{AdoptedBP}}$) and every valley-free path between them consists solely of benign ASes ($\mu_{A,B}^{\text{HonestPaths}}$ and $\mu_{B,A}^{\text{HonestPaths}}$).*

All RIRs publish into one common set of mirrors $\mathcal{M}$. On top of that common set, the model imposes three conditions. First, every mirror serves at time t every object published at any mirror at $t - \Delta[\text{'mirror'}]$, where $\Delta[\text{'mirror'}]$ bounds the replication lag ($\Delta[\text{'mirror'}] = 0$ for direct publication to every location), so one successful fetch yields the whole RPKI dataset. Second, every benign host adopts ROV. Third, every benign ROV-adopting AS v has a mirror host AS_M that is well-connected (Definition 8) not only to v but to every AS on the best path from AS_M to v , which keeps the whole path benign and ROV-adopting. Because the hybrid mode co-locates all PPs at the RIRs (Section 7.4), we consider only RIR-level certificates; the delegated hierarchy is left to future work. Appendix M provides further details and justification for the model.

N.1 Proof of Lemma 5

LEMMA 5. *Let $G = (V, E, R)$ be the AS topology graph, let $v \in V$ be a benign ROV-adopting AS, and denote $c_{\text{prop}} \equiv \text{diam}_{\text{VF}}(G) \cdot (\Delta[\text{'com'}] + \Delta[\text{'clk'}])$. Under the $\mathcal{M}_{f,\Delta}^{\text{Hybrid-Retrieval}}$ model, assuming black-box protocols that satisfy $\mathcal{R}^{\text{PKI}}$ and $\mathcal{R}_{\Delta[\text{'DNS'}]}^{\text{DNS-COR}}$, if a ROA ψ is published at an RIR PP by some time $t \geq t^{\text{sync}}$, then the RP of v holds ψ , validated in its cache, by time $t + \Delta[\text{'mirror'}] + (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}})$, except with negligible probability, where t^{init} is the time by which every benign RIR and v have initialized and provided that*

$$\begin{aligned} t^{\text{sync}} &= t^{\text{init}} + \Delta[\text{'orig'}] + \Delta[\text{'mirror'}] + c_{\text{prop}} \\ &\quad + (\text{diam}_{\text{VF}}(G) + 1) \\ &\quad (\Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}}) \end{aligned}$$

Proof of Lemma 5. Denote $c_{\text{fetch}} = \Delta[\text{'wake'}] + \Delta[\text{'DNS'}] + 2\Delta[\text{'ask'}] + 2c_{\text{prop}}$. Fix a mirror host AS_M well-connected to v , which exists by $\mu_{\Delta[\text{'mirror'}]}^{\text{SharedMirror}}$. For an honest ROV adopter u write IP_u^{RP} for its IP_{RP} , the address of its relying party, p_u^{RP} for the prefix that u self-originates and that covers IP_u^{RP} , and ψ_u^{RP} for the ROA authorizing u to originate p_u^{RP} . Finally, let

$$t^c = t^{\text{init}} + \Delta[\text{'orig'}] + c_{\text{prop}} + \Delta[\text{'mirror'}].$$

In order to prove that v fetches the RPKI object served by AS_M , we need to show that v is able to send a fetch request and to receive the reply.

First, v is able to send the request, and from time t^c on that request reaches AS_M , based on the following:

1. *v knows where to send it, and needs no earlier fetch to find out.* By μ^{RobustRP} (Algorithm 70) every honest ROV adopter is preloaded with the RIR roots, which BGP_{RPKI}.INIT loads into its cache (Algorithm 22, Line 24), and by $\mu_{\Delta[\text{'mirror'}]}^{\text{SharedMirror}}$ their SIAs name every mirror. On each wakeup UpdateCache (Algorithm 30) queries every $url \in \psi.SIA$ (Line 11), so AS_M is among

the mirrors that v contacts and the other fetches may fail harmlessly.

2. *The request is delivered.* Let u be any honest ROV adopter whose best path to AS_M consists of benign adopters. Denote by p_M the prefix covering the repository address of AS_M , and by ψ_M the ROA authorizing AS_M to originate it. Lemma 4 holds with $q = p_M$, $\psi = \psi_M$, $AS_{\text{Dst}} = AS_M$ and destination u . $\mu_{\Delta[\text{'orig'}]}^{\text{RPKInit}}$ ensures that AS_M originates p_M by $t_M \leq t^{\text{init}} + \Delta[\text{'orig'}]$ and never withdraws it, and μ^{RobustRP} preloads ψ_M , validated, at every honest ROV adopter. The lemma therefore gives every AS on that path, u included, an installed route to p_M with origin AS_M by $t^{\text{init}} + \Delta[\text{'orig'}] + c_{\text{prop}}$. No competing announcement displaces it, $\mu^{\text{RIRDeploy}}$ and $\mu_{\Delta[\text{'conf'}]}^{\text{NoROACConflict}}$ forbid an honest CA from issuing a conflicting ROA, the adversary controls no CA under $\mu_{\text{Prefix}}^{\text{HijackAdv}}$, and $\mathcal{R}^{\text{PKI}}$ excludes forgery, so a prefix attacker announcing p_M is *Invalid* and filtered at every hop and at every time. Hence a request u sends at any $t \geq t^{\text{init}} + \Delta[\text{'orig'}] + c_{\text{prop}}$ reaches AS_M within c_{prop} by Lemma 2.

The reply is the other half, and it is where the difficulty lies. It is addressed to p_v^{RP} , so every AS that forwards it must have selected the route with origin v , since under $\mu_{\text{Prefix}}^{\text{HijackAdv}}$ an adversary announcing p_v^{RP} could otherwise hijack the prefix. Filtering that announcement require a ROA ψ_v^{RP} , which μ^{RobustRP} ensure it is published, so the ASes that carry the reply must have fetched it themselves.

μ^{RobustRP} ensures that ψ_v^{RP} is published at an RIR PP within $\Delta[\text{'orig'}]$ of the init of v , hence by $t^{\text{init}} + \Delta[\text{'orig'}]$, and stays valid over the window, so every mirror serves it by t^c ; resulting in the legitimate origination of p_v^{RP} as *Valid*.

What remains is to get them from AS_M into the caches along the return path, which is what the induction below prove.

The reply travels the best path from AS_M to v , which we write as

$$P = (q_0, q_1, \dots, q_m), \quad q_0 = AS_M, \quad q_m = v.$$

It is a simple valley-free path in G , so $m \leq \text{diam}_{\text{VF}}(G)$, and by $\mu_{AS_M,v}^{\text{AdoptedBP}}$ and $\mu_{AS_M,v}^{\text{HonestPaths}}$ of Definition 8, every q_i on the path is an honest ROV adopter.

We now show that every AS on P manages to fetch the repository of AS_M , working outward from the mirror towards v . Let

$$T_i = t^c + (i+1) \cdot c_{\text{fetch}}$$

We prove by induction on i that q_i completes a fetch of the repository of AS_M by time T_i , and from then on holds validated in its cache the whole store that AS_M served at that fetch. In particular q_i holds ψ_u^{RP} for every honest ROV adopter u .

Base case ($i = 0$). The AS $q_0 = AS_M$ hosts the mirror, so its relying party fetches from itself and needs no network exchange. The fetch it issues at its first wakeup at or after t^c is addressed

to its own repository address and is delivered locally (Algorithm 14, Line 4). The wakeup comes within $\Delta[\text{‘wake’}]$ and the name resolves within $\Delta[\text{‘DNS’}] + 2\Delta[\text{‘ask’}]$, so the fetch completes by $t^c + \Delta[\text{‘wake’}] + \Delta[\text{‘DNS’}] + 2\Delta[\text{‘ask’}]$, which is at most T_0 .

Inductive step ($i \rightarrow i + 1$). Assume each of $q_0, \dots, q_i$ has completed its fetch by T_i , and consider q_{i+1} . At its first wakeup at or after T_i , which comes within $\Delta[\text{‘wake’}]$, AS q_{i+1} fetches from every mirror; we follow the fetch addressed to AS_M , whose name resolves within $\Delta[\text{‘DNS’}] + 2\Delta[\text{‘ask’}]$. We show both steps of that fetch are delivered.

Since $\mu_{\Delta[\text{‘mirror’}]}^{\text{SharedMirror}}$ guarantees that q_{i+1} and AS_M are well connected, and that the ROA for the mirror is available from initialization, the request is delivered by items 1 and 2 above. The request arrives within c_{prop} .

The reply is addressed to $p_{q_{i+1}}^{\text{RP}}$. The sequence $(q_{i+1}, q_i, \dots, q_0)$ is a sub-sequence of the best path P , which carries one prefix, and since the μ^{VExport} policy cannot tell prefixes apart, P carries all of them, $p_{q_{i+1}}^{\text{RP}}$ included. Every AS on it lies on P and is therefore an honest ROV adopter. By the induction hypothesis, each of $q_0, \dots, q_i$ holds $\psi_{q_{i+1}}^{\text{RP}}$ validated, so each classifies the legitimate origination of $p_{q_{i+1}}^{\text{RP}}$ as *Valid*. Applying Lemma 4 with origin $a = q_{i+1}$, prefix $q = p_{q_{i+1}}^{\text{RP}}$, certificate $\psi = \psi_{q_{i+1}}^{\text{RP}}$ and path $(q_{i+1}, q_i, \dots, q_0)$, every q_j with $j \leq i$ installs a route to $p_{q_{i+1}}^{\text{RP}}$ with origin q_{i+1} within c_{prop} . No AS on the path diverts the reply, which arrives within a further c_{prop} by Lemma 2.

Waking, resolving, and the request and reply steps cost c_{fetch} in total, so the fetch of q_{i+1} completes by $T_i + c_{\text{fetch}} = T_{i+1}$, as required.

Since $m \leq \text{diam}_{\text{VF}}(G)$ we have $T_m \leq t^{\text{sync}}$, so by t^{sync} , every AS on P has fetched the repository of AS_M and holds ψ_v^{RP} , and from t^{sync} onward, every fetch v issues towards AS_M has both steps delivered and completes within c_{fetch} of the wakeup that triggers it. Let $t \geq t^{\text{sync}}$ and let ψ be published at an RIR PP by t , so by AS_M serves it from $t + \Delta[\text{‘mirror’}]$ on first fetch of v after $t + \Delta[\text{‘mirror’}]$ completes within c_{fetch} , and its reply left AS_M after the request arrived, hence after $t + \Delta[\text{‘mirror’}]$, so it carries ψ . The relying party of v therefore holds ψ , validated, by $t + \Delta[\text{‘mirror’}] + c_{\text{fetch}}$ as required. By $\mu^{\text{RPKIHybrid}}$, each reply carries every object with its whole issuing chain. So UpdateValidationCache (Algorithm 31) validates what it carries locally against the roots v has held since init. Except with negligible probability, being $\mathcal{R}^{\text{PKI}}$ in the applications of Lemma 4. $\square$

O Implementation and Evaluation of RobustRP

Next, we detail and evaluate our implementation of a RP that leverages the extensions from §9.2.1. Our implementation, termed *RobustRP*, is a fork of Routina-

tor [61], a popular open-source relying party. We add 241 lines of code (excluding tests). We make it available as open source at <https://github.com/shaycohen988/routinator-multi-resolver> [15].

We find that many major repository operators opt to use CDNs for their PPs, fronting RRDP connections to RPKI RPs. We identified 33 such PPs (out of 83 PPs in today’s RPKI deployment), including all PPs operated by the RIRs except ARIN, which operates its own CDN-like content delivery infrastructure (using 9 points of presence). The CDN replicates the PP across multiple servers at different ASes. However, today’s RPs only attempt to connect to one of them, which the CDN serves using DNS based on the RP’s location. We change this mechanism first by giving the RP more options for PP addresses, and then by extending it to fall back to connecting to other PPs in case of a connection failure.

Discovering PP addresses We replace Routinator’s default domain name resolver with a custom implementation that, for every PP name DNS lookup, queries in parallel 10 geographically distributed public DNS resolvers for that PP’s address. Table 5 lists the resolvers we use. Since each resolver is steered to a different CDN point of presence, RobustRP retrieves a much larger pool of IP addresses per PP than the baseline RP; we show the results in Table 1. Our implementation resolves the domain name to both IPv4 and IPv6 addresses to utilize all available servers.

Table 5: Resolvers queried in parallel by RobustRP. Resolvers span five geographic regions to maximize PoP diversity for DNS geo-routing CDNs.

Region	Resolver	Address
North America	Google	8.8.8.8
North America	OpenDNS	208.67.222.222
Global	Cloudflare	1.1.1.1
Europe	Quad9	9.9.9.9
Europe	AdGuard	94.140.14.14
Eastern Europe	Yandex	77.88.8.8
Asia-Pacific	114DNS	114.114.114.114
Asia-Pacific	AliDNS	223.5.5.5
Asia-Pacific	DNSPod	119.29.29.29
Asia-Pacific	TWNIC	101.101.101.101

PP connectivity recovery RobustRP exploits the IP address pool it obtains for each PP to implement a fallback mechanism. When a connection to a PP fails, it tries the next IP address for that PP until it either succeeds or exhausts the pool of IP addresses for that PP. We evaluate the benefit of using RobustRP in today’s RPKI deployment by measuring redundancy for each verified ROA payload, i.e., the number of PP servers serving that payload. Figure 12 shows the distribution.

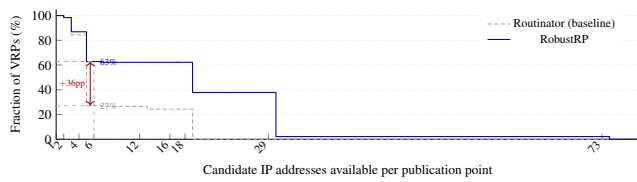


Figure 12: VRP-weighted CDF; fraction of VRPs served by a PP with at least N redundant IP addresses. At $N=6$, RobustRP covers **63%** of VRPs vs. **27%** for Routinator (baseline) (+36 percentage points), as marked. Total: 959,396 VRPs, as of June 11, 2026.

Algorithm 42 $\mu^{\text{WellFormedGraph}}$: Validate topology consistency and structure

Description: Ensures the AS topology is well formed: (i) every link is declared by *both* endpoints with complementary relationships (c2p $\leftrightarrow$ p2c, or p2p $\leftrightarrow$ p2p); and (ii) the provider-free ASes form a non-empty *full peering clique* that every AS can reach.

```

1: model  $\mu^{\text{WellFormedGraph}}(T)$ 
2:    $(V, E, R) \leftarrow \text{BUILDGRAPH}(T)$   $\triangleright R$  labels each edge c2p/p2c/p2p (Alg. 49)
3:    $\text{Valid} \leftarrow \{ (c2p, p2c), (p2c, c2p), (p2p, p2p) \}$   $\triangleright$  the complementary label pairs
4:   for all  $(u, v) \in E$  do
5:     if  $(v, u) \notin E \vee (R[(u, v)], R[(v, u)]) \notin \text{Valid}$  then
6:       return  $\perp$   $\triangleright$  link missing at one endpoint, or the two views disagree
7:     end if
8:   end for
9:    $\text{Tier1} \leftarrow \{ u \in V \mid \nexists v : (u, v) \in E \wedge R[(u, v)] = c2p \}$   $\triangleright$  ASes with no provider
10:   $\text{is\_clique} \leftarrow (\forall u, v \in \text{Tier1}, u \neq v \implies R[(u, v)] = p2p)$ 
 $\triangleright$  Tier-1 is a full peering mesh
11:   $\text{all\_reach} \leftarrow (\forall w \in V, \exists t \in \text{Tier1} : \text{REACHABLE}((V, E, R), w, t))$ 
 $\triangleright$  every AS can climb to a Tier-1
12:  return  $\text{Tier1} \neq \emptyset \wedge \text{is\_clique} \wedge \text{all\_reach}$ 
13: end model

```

Algorithm 43 $\mu^{\text{MaxPrefixLength}}$: no prefix longer than /24

Description: Models the RFC 7454 [17] operational practice that IPv4 prefixes more specific than /24 are filtered and hence not globally routable. No entity may originate or propagate a prefix longer than /24 (bound stated for IPv4).

```

1: model  $\mu^{\text{MaxPrefixLength}}(T)$ 
2:   return  $\forall e \in \{1, \dots, T.e\} \text{ s.t. } T.\text{opr}[e] \in \{\text{'ORIGINATE'}, \text{'Update'}\} : \text{len}(T.\text{inp}[e].p) \leq 24$ 
 $\triangleright$  Originated/propagated prefixes are no longer than /24
3: end model

```

Algorithm 44 $\mu^{\text{HonestBestPaths}}_{\text{AS}_{\text{Dst}}}$: every best path is all-honest

Description: ensures that all possible best paths from AS_{Src} to AS_{Dst} contain only honest ASes.

```

1: model  $\mu^{\text{HonestBestPaths}}_{\text{AS}_{\text{Dst}}}(T)$ 
2:    $(\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}, \text{ip}) \leftarrow T.\text{out}_{\mathcal{A}}$   $\triangleright$  Extract advertised routing info
3:    $\mathcal{P}_{\text{best}} \leftarrow \text{GETALLBESTPATHS}(T, \text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}})$   $\triangleright$  See Algorithm 55
4:   for all  $\text{path} \in \mathcal{P}_{\text{best}}$  do
5:     if  $\neg \text{ISPATHHONEST}(\text{path})$  then
6:       return  $\perp$   $\triangleright$  A possible best path is compromised
7:     end if
8:   end for
9:   return  $\top$   $\triangleright$  All best paths are fully honest
10: end model

```

P ADDITIONAL PREDICATE PROCEDURES

This section lists the model predicates used in the paper. They are composed into the BGP model of Equation (2), the RPKI hybrid-mode model of Equation (3), the prefix-adversary model of §8, and the robust-deployment model of §9.3.

Algorithm 45 $\mu^{\text{HijackAdv}}_{\text{type}}$: adversary limited to *type* hijacks

Description: This predicate ensure that the adversary is strictly limited to *type* hijacks (Prefix or Subprefix) attacks. Corrupted ASes forbidden from performing other types of attacks such as origin spoofing, route leaks, or operating as a CA in case of RPKI. Additionally, no honest CA certifies a ROA with a malicious origin for a prefix that was legitimately originated.

```

1: model  $\mu^{\text{HijackAdv}}_{\text{type}}(T)$ 
2:    $\text{IsCA} = \left[ \begin{array}{l} \nexists e \in \{1, \dots, T.e\} \text{ s.t.} \\ T.\text{ent}[e] \in T.F \wedge \\ T.\text{opr}[e] = \text{'Init'} \wedge \\ \text{'CA'} \in T.\text{inp}[e].\text{roles} \end{array} \right]$   $\triangleright$  No corrupted entity  

 $\triangleright$  may initialize with  

 $\triangleright$  the CA role
3:    $\text{IsPrefixHijack} = \left[ \begin{array}{l} \forall e \in \{1, \dots, T.e\} \text{ s.t.} \\ T.\text{ent}[e] \in T.F \wedge \\ (\text{'Send'}, \text{pkt}, \text{dest}) \in T.\text{out}[e] \wedge \\ (\text{pkt}, \text{'protocol'} = \text{'BGP'}) \implies \\ \text{IsAllowedPrefixHijack}(\text{pkt}, p, \text{type}, T.\text{ent}[e]) \end{array} \right]$ 
4:    $\text{HonestCA} = \left[ \begin{array}{l} \nexists e \in \{1, \dots, T.e\} \text{ s.t.} \\ T.\text{ent}[e] \notin T.F \wedge \\ \text{entity issues a ROA for an } \text{asn} \in T.F \wedge \\ \text{the prefix was legitimately originated} \end{array} \right]$ 
5:   return  $\text{IsCA} \wedge \text{IsPrefixHijack} \wedge \text{HonestCA}$ 
6: end model

```

Algorithm 46 μ^{FullROV} : every honest AS enables ROV

Description: This predicate asserts that the system exhibits Full ROV adoption. It verifies that no honest BGP entity is initialized with ROV validation disabled.

```

1: model  $\mu^{\text{FullROV}}(T)$ 
2:   return  $\nexists e \in \{1, \dots, T.e\} \text{ s.t.}$ 
3:    $T.\text{opr}[e] = \text{'Init'}$   $\triangleright$  Scan init events
4:    $T.\text{ent}[e] \in T.N - T.F$   $\triangleright$  by honest nodes
5:   Router  $\in T.\text{inp}[e].\text{roles}$   $\triangleright$  acting as BGP speakers
6:    $T.\text{inp}[e].\text{params.rov} = \perp$   $\triangleright$  with ROV disabled
7: end model

```

Algorithm 47 $\mu^{\text{HonestPaths}}_{\text{AS}_S, \text{AS}_D}$: every valley-free path is all-honest

Description: Evaluates whether all valley-free paths from the sender to the destination consist entirely of honest ASes.

```

1: model  $\mu^{\text{HonestPaths}}_{\text{AS}_S, \text{AS}_D}(T)$ 
2:    $\mathcal{P}_{\text{plausible}} \leftarrow \text{GETALLPLAUSIBLEPATHS}(T, \text{AS}_S, \text{AS}_D)$   $\triangleright$  See Algorithm 52
3:   for all  $\text{path} \in \mathcal{P}_{\text{plausible}}$  do
4:     if  $\neg \text{ISPATHHONEST}(\text{path})$  then  $\triangleright$  See Algorithm 53
5:       return  $\perp$   $\triangleright$  A plausible path is compromised
6:     end if
7:   end for
8:   return  $\top$ 
9: end model

```

Algorithm 48 $\mu^{\text{AdoptedBP}}_{\text{AS}_S, \text{AS}_D}$: every best route is all-ROV

Description: Verifies that every “best” route selected by the BGP decision process from the sender to the destination consists only ROV-adopting ASes. Under the Gao-Rexford model, best routes are evaluated first by economic preference (Customer > Peer > Provider) and then by shortest AS-path length.

```

1: model  $\mu^{\text{AdoptedBP}}_{\text{AS}_S, \text{AS}_D}(T)$ 
2:    $\mathcal{P}_{\text{best}} \leftarrow \text{GETALLBESTPATHS}(T, \text{AS}_S, \text{AS}_D)$   $\triangleright$  See Algorithm 55
3:   for all  $\text{path} \in \mathcal{P}_{\text{best}}$  do
4:     if  $\neg \text{ISPATHHONEST}(\text{path})$  then  $\triangleright$  See Algorithm 54
5:       return  $\perp$   $\triangleright$  A best path is not fully adopted
6:     end if
7:   end for
8:   return  $\top$ 
9: end model

```

Algorithm 49 BUILDGRAPH: Construct AS graph and relationship function

Description: Builds relationship-labeled AS graph from the ‘Init’ declarations. *Vertices are entity ids* (each AS is a single entity with a single ASN); the label $R[(u, v)] \in \{c2p, p2c, p2p\}$ records the link from u ’s point of view, the reverse label being supplied by v ’s own ‘Init’.

```

1: procedure BUILDGRAPH( $T$ )
2:    $V \leftarrow \emptyset$ ;  $E \leftarrow \emptyset$ ;  $R \leftarrow \emptyset$ 
3:    $lbl \leftarrow \{\text{provider:c2p, customer:p2c, peer:p2p}\}$   $\triangleright rel \mapsto label \text{ (partial map)}$ 
4:   for all  $\hat{e}$  s.t.  $T.opr[\hat{e}] = \text{'Init'}$  do
5:      $params \leftarrow T.inp[\hat{e}]$ ;  $u \leftarrow T.ent[\hat{e}]$ ;  $V \leftarrow V \cup \{u\}$   $\triangleright vertex = entity$ 
6:     for all  $(j, rel, \dots) \in params.\mathcal{N}$  do
7:        $V \leftarrow V \cup \{j\}$ ;  $E \leftarrow E \cup \{(u, j)\}$ ;  $R[(u, j)] \leftarrow lbl[rel]$   $\triangleright unknown \ rel \ is \ unset$ 
8:     end for
9:   end for
10:  return  $(V, E, R)$ 
11: end procedure

```

Algorithm 50 REACHABLE: Check reachability via $c2p$ edges

```

1: procedure REACHABLE( $(V, E, R)$ ,  $src$ ,  $dst$ )
2:    $Visited \leftarrow \{src\}$ ;  $Q \leftarrow [src]$   $\triangleright Initialize \ visited \ set \ and \ BFS \ queue$ 
3:   while  $Q \neq \emptyset$  do
4:      $v \leftarrow Q.pop()$   $\triangleright Dequeue \ next \ vertex$ 
5:     if  $v = dst$  then
6:       return  $\top$   $\triangleright Destination \ reached$ 
7:     end if
8:     for all  $u \in V$  s.t.  $(v, u) \in E \wedge R[(v, u)] = c2p$  do  $\triangleright Follow \ only \ customer \rightarrow provider \ edges$ 
9:       if  $u \notin Visited$  then
10:         $Visited \leftarrow Visited \cup \{u\}$ 
11:         $Q.push(u)$ 
12:      end if
13:    end for
14:  end while
15:  return  $\perp$   $\triangleright No \ c2p \ path \ exists \ from \ src \ to \ dst$ 
16: end procedure

```

Algorithm 51 CollectPolicies: Gather inbound/outbound policies of honest ASes

```

1: procedure COLLECTPOLICIES( $T$ ,  $direction$ )
  Description: Collects every policy configured for an honest AS in the requested direction, both from initialization and from every later SetPolicy update; a predicate built over the result therefore requires ALL historically-configured policy versions to comply, not just the latest one.
2:    $Policies \leftarrow \emptyset$   $\triangleright Initialize \ collection \ of \ (policy, \ entity) \ pairs$ 
3:   for all  $\hat{e} \in \{1, \dots, T.e\}$  do
4:     if  $T.ent[\hat{e}] \in T.N - T.F$  then
5:       if  $T.opr[\hat{e}] = \text{'Init'}$  then
6:         if  $direction = \text{'in'}$  then
7:            $Policies \leftarrow Policies \cup \{(T.inp[\hat{e}].params.policy_{in}, T.ent[\hat{e}])\}$ 
8:         else
9:            $Policies \leftarrow Policies \cup \{(T.inp[\hat{e}].params.policy_{out}, T.ent[\hat{e}])\}$ 
10:        end if
11:      else if  $T.opr[\hat{e}] = \text{SetPolicy}$  then
12:        if  $direction = \text{'in'}$  then
13:           $Policies \leftarrow Policies \cup \{(T.inp[\hat{e}].policy_{in}, T.ent[\hat{e}])\}$ 
14:        else
15:           $Policies \leftarrow Policies \cup \{(T.inp[\hat{e}].policy_{out}, T.ent[\hat{e}])\}$ 
16:        end if
17:      end if
18:    end if
19:  end for
20:  return  $Policies$ 
21: end procedure

```

Algorithm 52 GETALLPLAUSIBLEPATHS: Find all valley-free paths between two ASes

Description: Computes all valid, valley-free paths between the source and destination ASes based on established commercial routing relationships.

```

1: procedure GETALLPLAUSIBLEPATHS( $T, AS_{src}, AS_{dst}$ )
2:    $(V, E, R) \leftarrow BUILDGRAPH(T)$   $\triangleright Retrieve \ graph \ and \ relationships$ 
3:    $\mathcal{P}_{all} \leftarrow \{p \mid p \text{ is a simple path from } AS_{src} \text{ to } AS_{dst} \text{ in } G(V, E)\}$ 
4:    $\mathcal{P}_{plausible} \leftarrow \emptyset$ 
5:   for all  $path \in \mathcal{P}_{all}$  do
6:     if ISVALLEYFREE( $path, R$ ) then
7:        $\mathcal{P}_{plausible} \leftarrow \mathcal{P}_{plausible} \cup \{path\}$ 
8:     end if
9:   end for
10:  return  $\mathcal{P}_{plausible}$ 
11: end procedure

12: procedure ISVALLEYFREE( $path, R$ )
13:    $has\_peaked \leftarrow \perp$   $\triangleright Tracks \ if \ path \ has \ shifted \ from \ 'Up' \ to \ 'Down'$ 
14:   for all edge  $e = (u, v) \in path$  do
15:      $rel \leftarrow R[e]$ 
16:     if  $rel = \text{'c2p'}$  then
17:       if  $has\_peaked = \top$  then
18:         return  $\perp$   $\triangleright Cannot \ traverse \ c2p \ after \ peaking$ 
19:       end if
20:     else if  $rel = \text{'p2p'}$  then
21:       if  $has\_peaked = \top$  then
22:         return  $\perp$   $\triangleright Only \ one \ p2p \ link \ allowed, \ at \ the \ peak$ 
23:       end if
24:        $has\_peaked \leftarrow \top$ 
25:     else if  $rel = \text{'p2c'}$  then
26:        $has\_peaked \leftarrow \top$   $\triangleright Path \ is \ now \ moving \ downwards$ 
27:     end if
28:   end for
29:  return  $\top$ 
30: end procedure

```

Algorithm 53 ISHONESTPATH: Check whether every AS on a path is honest

Description: Returns true if every AS appearing on $path$ are benign (not corrupted).

```

1: procedure ISHONESTPATH( $path$ )
2:   return  $\forall as \in path : as \in T.N - T.F$ 
3: end procedure

```

Algorithm 54 ISPATHROVADOPTERS: Check whether every AS on a path adopts ROV

Description: Returns true if every AS appearing on $path$ initialized with ROV enabled ($params.rov \neq \perp$).

```

1: procedure ISPATHROVADOPTERS( $T, path$ )
2:   return  $\forall as \in path : \exists \hat{e} \in \{1, \dots, T.e\} \text{ s.t. } T.opr[\hat{e}] = \text{'Init'} \wedge T.ent[\hat{e}] = as \wedge T.inp[\hat{e}].params.rov \neq \perp$ 
3: end procedure

```

Algorithm 55 GETALLBESTPATHS: Select highest-ranked paths via Gao-Rexford rules

Description: Filters valley-free paths to find the highest-ranked routes based on maximizing economic local preference ($p2c > p2p > c2p$) and minimizing AS-path length. If multiple paths tie perfectly, all of them are returned.

```

1: procedure GETALLBESTPATHS( $T, AS_{src}, AS_{dst}$ )
2:    $\mathcal{P}_{plausible} \leftarrow \text{GETALLPLAUSIBLEPATHS}(T, AS_{src}, AS_{dst})$   $\triangleright$  See Algorithm 52
3:    $(V, E, R) \leftarrow \text{BUILDDGRAPH}(T)$ 
4:   if  $\mathcal{P}_{plausible} = \emptyset$  then
5:     return  $\emptyset$   $\triangleright$  No valid paths exist
6:   end if
7:    $highest\_pref \leftarrow 0$ 
8:    $\mathcal{P}_{pref} \leftarrow \emptyset$ 
9:   for all  $path \in \mathcal{P}_{plausible}$  do
10:     $first\_edge \leftarrow path[1]$ 
11:     $pref \leftarrow \text{GETPREFERENCE}(R[first\_edge])$ 
12:    if  $pref > highest\_pref$  then
13:       $highest\_pref \leftarrow pref$ 
14:       $\mathcal{P}_{pref} \leftarrow \{path\}$ 
15:    else if  $pref = highest\_pref$  then
16:       $\mathcal{P}_{pref} \leftarrow \mathcal{P}_{pref} \cup \{path\}$ 
17:    end if
18:  end for
19:   $min\_length \leftarrow \min_{p \in \mathcal{P}_{pref}} |p|$ 
20:   $\mathcal{P}_{best} \leftarrow \{p \in \mathcal{P}_{pref} \mid |p| = min\_length\}$ 
21:  return  $\mathcal{P}_{best}$ 
22: end procedure

23: procedure GETPREFERENCE( $rel$ )
24:   if  $rel = 'p2c'$  then
25:     return 3  $\triangleright$  Customer route (Highest)
26:   else if  $rel = 'p2p'$  then
27:     return 2  $\triangleright$  Peer route
28:   else
29:     return 1  $\triangleright$  Provider route ('c2p')
30:   end if
31: end procedure

```

Algorithm 56 $\mu^{VF_{export}}$: Ensure honest ASes' export policies are Gao-Rexford compliant

Description: Enforces the Gao-Rexford export rule on every honest AS's outbound policy, which must consist of exactly two rules: r_1 discards routes learned from a peer or provider when the receiver is a peer or provider (no free transit); r_2 is a catch-all that accepts everything else, so customer-learned and self-originated routes are exported to all neighbors (*export-to-all*).

```

1: model  $\mu^{VF_{export}}(T)$ 
2:    $Policies \leftarrow \text{CollectPolicies}(T, "out")$   $\triangleright$  Collect all outbound policies of honest ASes (Alg. 51)
3:   for each  $(policy, \iota) \in Policies$  do
4:     if  $\left[ \begin{array}{l} |policy| \neq 2 \vee \\ policy[1].C \neq [(senderRelationships, \{Provider, Peer\}), \\ (receiverRelationships, \{Provider, Peer\})] \vee \\ policy[1].A \neq [reject] \vee \\ policy[2].C \neq [(true)] \vee \\ policy[2].A \neq [accept] \end{array} \right]$ 
 $\triangleright$  exactly two rules
 $\triangleright r1$ : discard peer/prov  $\rightarrow$  peer/prov
 $\triangleright r2$ : accept the rest (cust/self  $\rightarrow$  all)
5:       return  $\perp$   $\triangleright$  Outbound policy at  $\iota$  is not exactly  $[r_1, r_2]$ 
6:     end for
7:   return  $\top$   $\triangleright$  All outbound policies are Gao-Rexford compliant
8: end model

```

Algorithm 57 $\mu^{VF_{pref}}$: Ensure inbound policies of honest ASes prefer customer routes

Description: Enforces the Gao-Rexford preference rule. Every honest AS must apply inbound policies where: r_1 gives highest preference to customer-learned routes, r_2 assigns lower preference to peer-learned routes, r_3 assigns lowest preference to provider-learned routes. Additionally, no later inbound rule may reject or override preferences.

```

1: model  $\mu^{VF_{pref}}(T)$ 
2:    $(customer\_range, peer\_range, provider\_range) \leftarrow T.out_{A.aux}$   $\triangleright$  Preference ranges announced by adversary
3:   if  $\left[ \begin{array}{l} \min(customer\_range) \leq \max(peer\_range) \vee \\ \min(peer\_range) \leq \max(provider\_range) \end{array} \right]$   $\triangleright$  Validate global hierarchy
4:     return  $\perp$ 
5:    $Policies \leftarrow \text{CollectPolicies}(T, "in")$   $\triangleright$  Collect inbound policies of honest ASes (Alg. 51)
6:   for each  $(policy, \iota) \in Policies$  do
7:      $(r_1, r_2, r_3) \leftarrow policy[1..3]$ 
 $\left[ \begin{array}{l} r_1.C \neq [(senderRelationships, \{Customer\})] \vee \\ r_1.A \neq [(setLocalPref, x), continue] \vee \\ x \notin customer\_range \vee \\ r_2.C \neq [(senderRelationships, \{Peer\})] \vee \\ r_2.A \neq [(setLocalPref, y), continue] \vee \\ y \notin peer\_range \vee \\ r_3.C \neq [(senderRelationships, \{Provider\})] \vee \\ r_3.A \neq [(setLocalPref, z), continue] \vee \\ z \notin provider\_range \vee \\ \exists i \geq 3 : \{reject, setLocalPref\} \in policy[i].A \end{array} \right]$   $\triangleright r1$ : prefer customers (highest localPref)  $\triangleright r2$ : prefer peers (lower than customers)  $\triangleright r3$ : prefer providers (lower than peers)  $\triangleright$  no later rejects/overrides
8:     if
9:       return  $\perp$   $\triangleright$  At least one inbound rule invalid at entity  $\iota$ 
10:    end for
11:  return  $\top$   $\triangleright$  All inbound policies are Gao-Rexford compliant
12: end model

```

Algorithm 58 $\mu^{NoProviderCustomerCycle}$: Provider to Customer subgraph is acyclic

Description: Accepts iff the provider $\rightarrow$ customer subgraph is acyclic (a Gao-Rexford topology condition): no directed cycle exists whose every edge is labeled p2c.

```

1: model  $\mu^{NoProviderCustomerCycle}(T)$ 
2:    $(V, E, R) \leftarrow \text{BUILDDGRAPH}(T)$   $\triangleright$  R labels each edge (Alg. 49)
3:   return  $\neg \exists u_0, u_1, \dots, u_k \in V, k \geq 1$  s.t.  $\triangleright$  no directed p2c cycle
 $u_0 = u_k \wedge$ 
 $\forall i \in \{0, \dots, k-1\} : (u_i, u_{i+1}) \in E \wedge R[(u_i, u_{i+1})] = p2c$ 
4: end model

```

Algorithm 59 $\mu^{PreferSelf}$: self-originated routes always preferred

Description: Ensures the first policy rule prioritizes self-originated routes and verifies that the rest of the rules do not contain overrides.

```

1: model  $\mu^{PreferSelf}(T)$ 
2:    $Policies \leftarrow \text{COLLECTPOLICIES}(T, "in")$   $\triangleright$  Gather inbound policies of honest ASes
3:   for each  $(policy, \iota) \in Policies$  do
4:      $r_0 \leftarrow policy[0]$   $\triangleright$  Inspect the primary rule
5:     if  $\left[ \begin{array}{l} r_0.C \neq [(selfOriginated)] \vee \\ r_0.A \neq [(setLocalPref, \infty), continue] \vee \\ \exists i \geq 3 : \{reject, setLocalPref\} \in policy[i].A \end{array} \right]$   $\triangleright$  First rule must be self-orig  $\triangleright$  Action: Max preference  $\triangleright$  No late overrides allowed
6:       return  $\perp$   $\triangleright$  Violation: AS  $\iota$  does not prioritize or maintain self-originated routes
7:     end for
8:   return  $\top$   $\triangleright$  All ASes correctly prioritize self-originated routes
9: end model

```

Algorithm 60 GETRIRCERTS: collect RIR root certificates

Description: Collects RIR root certs from the transcript.

```

1: procedure GETRIRCERTS( $T$ )
2:    $roots \leftarrow \emptyset$ 
3:   for all  $e \in \{1, \dots, T.e\}$  do
4:      $\left[ \begin{array}{l} T.opr[e] = \text{'BGP\_RPKI.PKI-INIT-RESP'} \wedge \\ \exists \hat{e}_i \in \{1, \dots, T.e\} \text{ s.t.} \\ T.opr[\hat{e}_i] = \text{'Init'} \wedge \\ T.ent[\hat{e}_i] = T.ent[e] \wedge \\ \text{'RIR'} \in T.inp[\hat{e}_i].roles \end{array} \right]$ 
5:   then:
6:      $roots \leftarrow roots \cup \{T.inp[e].\Psi\}$ 
7:   end for
8:   return  $roots$ 
9: end procedure

```

$\triangleright$ Operation is the PKI init-response callback
 $\triangleright$ Same entity's Init had role RIR

Algorithm 61 $\mu^{BgpInit}$ Model Predicate

Description: At every Init event, each physical peering link must be a /31 point-to-point subnet disjoint from the AS's owned IP blocks.

```

1: model  $\mu^{BgpInit}(T)$ 
2:   return  $\forall e \text{ s.t. } T.opr[e] = \text{'Init'} :$ 
3:    $\left[ \begin{array}{l} \forall (j, rel, asn_j, ip_{src}, ip_{dst}, p_{link}) \in \\ T.inp[e].params.\mathcal{N} : \\ \text{prefix\_len}(p_{link}) = 31 \wedge \\ ip_{src} \in p_{link} \wedge ip_{dst} \in p_{link} \wedge \\ ip_{src} \neq ip_{dst} \wedge \\ (\forall b \in T.inp[e].params.ipBlocks : p_{link} \cap b = \emptyset) \end{array} \right]$ 
4: end model

```

$\triangleright$ In every 'Init' event
 $\triangleright$ Peer subnets adhere to secure here to secure
 $\triangleright$ 31 topology and overlap no owned block

Algorithm 62 μ^{SHO} : one benign owner per prefix

Description: Ensures AS_{Dsts} is the unique benign owner of the prefix.

```

1: model  $\mu^{SHO}(T)$ 
2:    $(AS_{Src}, AS_{Dsts}, ip, p, pkt\_id, t) \leftarrow T.out_A$ 
3:   return
4:    $\left[ \begin{array}{l} |AS_{Dsts}| = 1 \wedge \\ \exists e \in \{1, \dots, T.e\} \text{ s.t.} \\ T.opr[e] = \text{'Init'} \wedge \\ T.ent[e] = AS_{Dsts} \wedge \\ T.ent[e] \in T.N - T.F \wedge \\ \exists b \in T.inp[e].params.ipBlocks : p \subseteq b \wedge \\ (\nexists e' \in \{1, \dots, T.e\} \text{ s.t.} \\ T.opr[e'] = \text{'Init'} \wedge \\ T.ent[e'] \in T.N - T.F \wedge \\ T.ent[e'] \neq AS_{Dsts} \wedge \\ Router \in T.inp[e'].roles \wedge \\ \exists b \in T.inp[e'].params.ipBlocks : b \cap p \neq \emptyset) \end{array} \right]$ 

```

$\triangleright$ Single destination AS
 $\triangleright$ Operation is Init
 $\triangleright$ Entity is AS_{Dsts}
 $\triangleright$ Entity is benign
 $\triangleright$ Allocation covers prefix
 $\triangleright$ No conflicting allocation:
 $\triangleright$ By any other
 $\triangleright$ honest entity
 $\triangleright$ that is a BGP speaker
 $\triangleright$ overlapping the prefix

Algorithm 63 $\mu_{\delta_{e2e}}^{ROA-InCache}$: honest ROV adopters hold the destination ROA

Description: Ensures that every honest ROV adopter has been given a ROA covering p and its validating chain prior to origination, and that those certificates stay valid throughout $[t - \delta_{e2e}, t + \delta_{e2e}]$.

```

1: model  $\mu_{\delta_{e2e}}^{ROA-InCache}(T)$ 
2:    $(AS_{Src}, AS_{Dsts}, ip, p, pkt\_id, t, label, aux) \leftarrow T.out_A$ 
3:   if  $label \neq \text{'IP-CON'}$  then
4:     return  $\perp$ 
5:   end if
6:    $roots \leftarrow GETRIRCERTS(T)$ 
7:   let  $adopters$  be the honest entities whose 'Init' sets  $params.rov = \top$ 
8:   return  $\exists AS_{Dst} \in AS_{Dsts}, \forall v \in adopters \text{ with } \mathcal{D}_v = \text{DELIVERED}(T, v, t - \delta_{e2e}) :$ 
9:    $roots \subseteq \mathcal{D}_v \wedge$ 
10:   $\exists \Psi_{roa} \in \mathcal{D}_v : \text{VALIDATES}(\Psi_{roa}, p, AS_{Dst}) \wedge$ 
11:   $\forall t' \in [t - \delta_{e2e}, t + \delta_{e2e}] : \text{CONSTRUCTANDVALIDATE}(\Psi_{roa}, t', roots, \mathcal{D}_v) = \top$ 
12: end model
13: procedure DELIVERED( $T, v, t'$ )
14:   Every RPKI object handed to v no later than t'.
15:   return  $\bigcup \{ \text{CERTSIN}(T.inp[\hat{e}]) \mid T.ent[\hat{e}] = v \wedge T.\tau[\hat{e}] \leq t' \}$ 
16: end procedure
17: procedure CERTSIN( $in$ )
18:   The RPKI objects carried by one step's input, reading an absent field as  $\emptyset$ .
19:   return  $in.params.preLoadedCerts \cup \text{OBJECTS}(in.pkt.payload.\text{'Data'})$ 
20: end procedure

```

$\triangleright$ Trust anchors
 $\triangleright$ p within $maxLen$, else ROV reads it Invalid; Algorithm 66
 $\triangleright$ valid across the whole window; Algorithm 37
 $\triangleright$ shipped at 'Init', or arriving in a packet
 $\triangleright$ Returns RPKI objects from a fetch payload.
 $\triangleright$ $\emptyset$ otherwise

Algorithm 64 $\mu_{\Delta[\text{'dest'}]}^{DestROAReady}$: destination ROA published before the send

Description: Ensures that a ROA authorizing AS_{Dst} to originate p is published at an RIR PP by $t - \Delta[\text{'dest'}]$.

```

1: model  $\mu_{\Delta[\text{'dest'}]}^{DestROAReady}(T)$ 
2:    $(AS_{Src}, AS_{Dsts}, ip, p, pkt\_id, t, label, aux) \leftarrow T.out_A$ 
3:   if  $label \neq \text{'IP-CON'}$  then
4:     return  $\perp$ 
5:   end if
6:    $roots \leftarrow GETRIRCERTS(T)$ 
7:    $rir\_pps \leftarrow \bigcup_{\Psi \in roots} \Psi.SIA$ 
8:    $pp\_ents \leftarrow \text{GETPPENTS}(T, rir\_pps)$ 
9:    $\mathcal{P} \leftarrow \text{PUBLISHED}(T, pp\_ents, t - \Delta[\text{'dest'}])$ 
10:  return  $\exists AS_{Dst} \in AS_{Dsts}, \Psi_d \in \mathcal{P} : \text{VALIDATES}(\Psi_d, p, AS_{Dst}) \wedge$ 
11:   $\forall \tau \in [t - \Delta[\text{'dest'}], t + \delta_{e2e}] : \text{CONSTRUCTANDVALIDATE}(\Psi_d, \tau, roots, \mathcal{P}) = \top$ 
12: end model

```

$\triangleright$ Algorithm 60
 $\triangleright$ honest hosts of those PPs; Algorithm 66
 $\triangleright$ fetchable a full cycle before the send
 $\triangleright$ valid across the whole window;
 $\triangleright$ not only at publication; Algorithm 37

Algorithm 65 $\mu_{\Delta[\text{conf}]}^{\text{NoROAConflict}}$: conflicting ROAs never appear alone

Description: A ROA *conflicts* with a pair (q, a) if it covers q without authorizing $(a, \text{len}(q))$. For $(p, \text{AS}_{\text{Dst}})$ and every benign RP's self-originated prefix covering its RP address, conflicting ROAs are not forbidden, but we require that whenever such a conflict is published and valid, a validating ROA published at least $\Delta[\text{conf}]$ earlier is valid too.

```

1: model  $\mu_{\Delta[\text{conf}]}^{\text{NoROAConflict}}(T)$ 
2:    $(\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}, \text{ip}, p, \text{pkt\_id}, t, \text{label}, \text{aux}) \leftarrow T.\text{out}_A$ 
3:    $\text{roots} \leftarrow \text{GETRIRCERTS}(T)$   $\triangleright$  See Algorithm 60
4:    $\text{rir\_pps} \leftarrow \bigcup_{\psi \in \text{roots}} \psi.\text{SIA}$ 
5:    $\text{pp\_ents} \leftarrow \text{GETPPENTS}(T, \text{rir\_pps})$   $\triangleright$  See Algorithm 66
6:    $\text{mitigation\_pairs} \leftarrow \text{GETMITIGATIONPAIRS}(T, p, \text{AS}_{\text{Dst}})$   $\triangleright$  See Algorithm 66
7:   return  $\text{MITIGATEDCONFLICT}(T, \text{mitigation\_pairs}, t, \text{roots}, \text{pp\_ents})$ 
8: end model
9: procedure  $\text{MITIGATEDCONFLICT}(T, \text{pairs}, t, \text{roots}, \text{pp\_ents})$ 
10:  return  $\forall (q, a) \in \text{pairs}, \forall t' \leq t + \Delta[\text{conf}]$ :
11:   $\mathcal{P} \leftarrow \text{PUBLISHED}(T, \text{pp\_ents}, t')$ ;  $\mathcal{P}' \leftarrow \text{PUBLISHED}(T, \text{pp\_ents}, t' - \Delta[\text{conf}])$ 
12:   $\left[ \begin{array}{l} \exists \psi_c \in \mathcal{P} : \text{CONFLICTS}(\psi_c, q, a) \wedge \\ \text{CONSTRUCTANDVALIDATE}(\psi_c, t', \text{roots}, \mathcal{P}) = \top \end{array} \right] \triangleright \begin{array}{l} \text{a valid conflict} \\ \text{published by } t'; \\ \text{Algorithm 66} \end{array}$ 
13:   $\Rightarrow \left[ \begin{array}{l} \exists \psi_v \in \mathcal{P}' : \text{VALIDATES}(\psi_v, q, a) \wedge \\ \text{CONSTRUCTANDVALIDATE}(\psi_v, t', \text{roots}, \mathcal{P}') = \top \end{array} \right] \triangleright \begin{array}{l} \text{a validating ROA} \\ \text{published by } t' - \Delta[\text{conf}], \text{ valid throughout } [t' - \Delta[\text{conf}], t'] \end{array}$ 
14: end procedure
```

Algorithm 66 Utilities procedures

```

1: utility  $\text{GETCRITIPS}(T, \text{rir\_pps})$ 
2:    $\text{rc\_ips} \leftarrow \left\{ T.\text{inp}[\hat{e}].\text{params}.\text{IP}_{\text{TAL}} \mid \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'Init'} \wedge \\ T.\text{inp}[\hat{e}].\text{params}.\text{role} = \text{'RIR'} \end{array} \right\}$ 
    $\triangleright$  root cert locations
3:    $\text{rir\_rp\_ips} \leftarrow \left\{ T.\text{inp}[\hat{e}].\text{params}.\text{IP}_{\text{RP}} \mid \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'Init'} \wedge T.\text{ent}[\hat{e}] \in T.N - T.F \wedge \\ T.\text{inp}[\hat{e}].\text{params}.\text{IP}_{\text{PP}} \in \text{rir\_pps} \wedge \\ T.\text{inp}[\hat{e}].\text{params}.\text{rov} \neq \perp \end{array} \right\}$ 
    $\triangleright$  RIR PP holders' RP addresses
4:   return  $\text{rir\_pps} \cup \text{rc\_ips} \cup \text{rir\_rp\_ips}$ 
5: end utility

6: utility  $\text{GETPPENTS}(T, \text{rir\_pps})$ 
7:   return  $\left\{ T.\text{ent}[\hat{e}] \mid \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'Init'} \wedge T.\text{ent}[\hat{e}] \in T.N - T.F \wedge \\ T.\text{inp}[\hat{e}].\text{params}.\text{IP}_{\text{PP}} \in \text{rir\_pps} \end{array} \right\}$ 
8: end utility

9: utility  $\text{GETMITIGATIONPAIRS}(T, p, \text{AS}_{\text{Dst}})$ 
10:   $\text{rp\_pairs} \leftarrow \left\{ (q, v) \mid \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'Init'} \wedge T.\text{ent}[\hat{e}] = v \wedge v \in T.N - T.F \wedge \\ T.\text{inp}[\hat{e}].\text{params}.\text{rov} \neq \perp \wedge \\ T.\text{opr}[\hat{e}] = \text{'Originate'} \wedge T.\text{ent}[\hat{e}] = v \wedge \\ q = T.\text{inp}[\hat{e}].p \wedge T.\text{inp}[\hat{e}].\text{params}.\text{IP}_{\text{RP}} \subseteq q \end{array} \right\}$ 
    $\triangleright$  every honest RP's self-originated prefix covering its RP endpoint
11:  return  $\{(p, \text{AS}_{\text{Dst}})\} \cup \text{rp\_pairs}$ 
12: end utility
13: utility  $\text{CONFLICTS}(\psi, q, a)$ 
14:   return  $\psi.\text{type} = \text{'ROA'} \wedge (\exists (pfx, x) \in \psi.\text{payload}.\text{ipPrefixes} : q \subseteq pfx) \wedge$ 
15:    $\neg \text{VALIDATES}(\psi, q, a)$ 
16: end utility
17: utility  $\text{VALIDATES}(\psi, q, a)$ 
18:   return  $\psi.\text{type} = \text{'ROA'} \wedge \psi.\text{payload}.\text{asID} = a \wedge$ 
19:    $\exists (pfx, \text{maxLen}) \in \psi.\text{payload}.\text{ipPrefixes} : q \subseteq pfx \wedge \text{len}(q) \leq \text{maxLen}$ 
20: end utility
21: utility  $\text{PUBLISHED}(T, \text{pp\_ents}, t)$ 
22:   return  $\bigcup \left\{ T.\text{inp}[\hat{e}].\text{certs} \mid \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'BGRPKI.UpdateRepoStore'} \wedge \\ T.\text{ent}[\hat{e}] \in \text{pp\_ents} \wedge T.\tau[\hat{e}] \leq t \end{array} \right\}$ 
23: end utility
```

Algorithm 67 $\mu^{\text{RPKIHybrid}}$: all objects published at an RIR PP

Description: This predicate captures the hybrid-mode publication assumption: every RC and ROA created by an honest entity has its SIA pointing to an RIR PP.

```

1: model  $\mu^{\text{RPKIHybrid}}(T)$ 
2:    $\text{rir\_certs} \leftarrow \text{GETRIRCERTS}(T)$   $\triangleright$  See Algorithm 60
3:    $\text{rir\_pps} \leftarrow \{\psi.\text{sia} \mid \psi \in \text{rir\_certs}\}$   $\triangleright$  RIR PP repository IPs
4:   return  $\forall \hat{e}_c \in \{1, \dots, T.e\} \text{ s.t. } T.\text{opr}[\hat{e}_c] = \text{'Certify'} \wedge T.\text{ent}[\hat{e}_c] \in T.N - T.F$  :
    $\triangleright$  honest certs published at an RIR PP
5:    $\forall \psi \in T.\text{out}[\hat{e}_c] : \psi.\text{sia} \in \text{rir\_pps}$   $\triangleright$  SIA at an RIR PP
6: end model
```

Algorithm 68 $\mu_{\Delta[\text{orig}]}^{\text{RPKInit}}$: hosts originate their RPKI prefixes at init

Description: Ensures that every benign entity that hosts RPKI infrastructure (a TAL host IP_{TAL} , a PP IP_{PP} , or an RP IP_{RP}) originates a BGP prefix covering that address within $\Delta[\text{orig}]$ of its 'Init' event, and never withdraws it.

```

1: model  $\mu_{\Delta[\text{orig}]}^{\text{RPKInit}}(T)$ 
2:   return  $\forall \hat{e}_i \text{ s.t. } T.\text{opr}[\hat{e}_i] = \text{'Init'} \wedge T.\text{ent}[\hat{e}_i] \in T.N - T.F$ , writing  $e = T.\text{ent}[\hat{e}_i]$  and  $s = T.\text{inp}[\hat{e}_i].\text{params}$  :
    $\triangleright$  each benign INIT event for each infrastructure address
3:    $\forall ip \in \{s.\text{IP}_{\text{TAL}}, s.\text{IP}_{\text{PP}}, s.\text{IP}_{\text{RP}}\} \setminus \{\perp\}, \exists \hat{e}'$  :
    $\triangleright$  e hosts
4:    $T.\text{opr}[\hat{e}'] = \text{'Originate'} \wedge T.\text{ent}[\hat{e}'] = e \wedge ip \subseteq T.\text{inp}[\hat{e}'].p \wedge T.\tau[\hat{e}'] \leq T.\tau[\hat{e}_i] + e \text{ originates } a$ 
    $\triangleright$  covering prefix within  $\Delta[\text{orig}]$ 
5:    $\nexists \hat{e}'' : T.\text{opr}[\hat{e}'] = \text{'Withdraw'} \wedge T.\text{ent}[\hat{e}'] = e \wedge T.\text{inp}[\hat{e}''].p = T.\text{inp}[\hat{e}'].p$ 
    $\triangleright$  and never withdraws it
6: end model
```

Algorithm 69 $\mu^{\text{RIRDeploy}}$: each RIR self-hosts TA, PP and RP

Description: The RIR deployment measured in Appendix J. Each honest RIR runs its trust anchor, its PP, and its relying party, and its PP is one of the locations its root certificate advertises. Every critical infrastructure is covered by an exact-match ROA valid at the required time t .

```

1: model  $\mu^{\text{RIRDeploy}}(T)$ 
2:    $(\text{AS}_{\text{Src}}, \text{AS}_{\text{Dst}}, \text{ip}, p, \text{pkt\_id}, t, \text{label}, \text{aux}) \leftarrow T.\text{out}_A$ 
3:    $\text{roots} \leftarrow \text{GETRIRCERTS}(T)$ 
4:    $\text{rir\_pps} \leftarrow \bigcup_{\psi \in \text{roots}} \psi.\text{SIA}$ 
5:    $\text{crit} \leftarrow \text{GETCRITIPS}(T, \text{rir\_pps})$   $\triangleright$  TAL hosts, PPs and RP endpoints; Algorithm 66
6:    $\text{pp\_ents} \leftarrow \text{GETPPENTS}(T, \text{rir\_pps})$ 
7:    $\mathcal{P} \leftarrow \text{PUBLISHED}(T, \text{pp\_ents}, t)$ 
8:   return  $\text{RIRCOLOCATION}(T, \text{rir\_pps}) \wedge$ 
9:    $\text{CRITICALTIGHTROA}(T, \text{crit}, t, \text{roots}, \mathcal{P}) \wedge$ 
10:   $\text{CRITICALNOCONFLICT}(T, \text{crit})$ 
11: end model

12: procedure  $\text{RIRCOLOCATION}(T, \text{rir\_pps})$ 
13:  return  $\forall \hat{e}_i \in \{1, \dots, T.e\} \text{ s.t. } T.\text{opr}[\hat{e}_i] = \text{'Init'} \wedge T.\text{ent}[\hat{e}_i] \in T.N - T.F \wedge \text{'RIR'} \in T.\text{inp}[\hat{e}_i].\text{roles}$ ,
   writing  $s = T.\text{inp}[\hat{e}_i].\text{params}$  :
14:    $s.\text{IP}_{\text{TAL}} \neq \perp \wedge s.\text{IP}_{\text{PP}} \neq \perp \wedge s.\text{IP}_{\text{RP}} \neq \perp$   $\triangleright$  one AS runs all three
15:    $s.\text{IP}_{\text{PP}} \in \text{rir\_pps}$ 
16: end procedure

17: procedure  $\text{CRITICALTIGHTROA}(T, \text{crit}, t, \text{roots}, \mathcal{P})$ 
18:  return  $\forall ip_c \in \text{crit}, \exists \hat{e}_o \in \{1, \dots, T.e\}$ , writing  $q = T.\text{inp}[\hat{e}_o].p, a = T.\text{ent}[\hat{e}_o]$  :
    $T.\text{opr}[\hat{e}_o] = \text{'Originate'} \wedge T.\text{ent}[\hat{e}_o] \in T.N - T.F \wedge ip_c \subseteq q \wedge$ 
19:   $\exists \psi \in \mathcal{P} : \left[ \begin{array}{l} \text{VALIDATES}(\psi, q, a) \wedge \\ \psi.\text{ipPrefix} = q \wedge \psi.\text{maxLen} = \text{len}(q) \wedge \\ \text{CONSTRUCTANDVALIDATE}(\psi, t, \text{roots}, \mathcal{P}) = \top \end{array} \right]$ 
    $\triangleright$  tight, no more-specific can be crafted. Algorithm 37
20: end procedure

21: procedure  $\text{CRITICALNOCONFLICT}(T, \text{crit})$ 
22:  return  $\nexists \hat{e}_o, \hat{e}_c \in \{1, \dots, T.e\} \text{ s.t.}, \text{ writing } q = T.\text{inp}[\hat{e}_o].p, a = T.\text{ent}[\hat{e}_o] :$ 
23:   $T.\text{opr}[\hat{e}_o] = \text{'Originate'} \wedge T.\text{ent}[\hat{e}_o] \in T.N - T.F \wedge \exists ip_c \in \text{crit} : ip_c \subseteq q \wedge$ 
24:   $T.\text{opr}[\hat{e}_c] = \text{'Certify'} \wedge \exists \psi \in T.\text{out}[\hat{e}_c] : \text{CONFLICTS}(\psi, q, a)$ 
25: end procedure
```

Algorithm 70 μ^{RobustRP} : RPs ship with a complete bootstrap cache

Description: This ensures that the initial cache of RP is expended to support a safe bootstrap. Every RP is shipped with the ROAs and certificates for all the RIR PPs, TAL locations and RIR RPs. Every benign RP's is originated and covered with a ROA, published at an RIR PP within $\Delta[\text{'orig'}]$ of that RP's init. Each of those ROAs stays valid from the 'Init' that holds it through $t + \delta_{\text{e2e}}$.

Notation: $\hat{e}$ is an benign init abbreviates $T.\text{opr}[\hat{e}] = \text{'Init'}$ and $T.\text{ent}[\hat{e}] \in T.N - T.F$; $s_{\hat{e}}$ is its $T.\text{inp}[\hat{e}].\text{params}$ and $t_{\hat{e}}$ its $T.\tau[\hat{e}]$. $\hat{e}_o$ originates q for a abbreviates $T.\text{opr}[\hat{e}_o] = \text{'Originate'}$, $T.\text{ent}[\hat{e}_o] = a \in T.N - T.F$ and $q = T.\text{inp}[\hat{e}_o].p$.

```

1: model  $\mu^{\text{RobustRP}}(T)$ 
2:    $(AS_{\text{src}}, AS_{\text{dsts}}, ip, p, pkt\_id, t, label, aux) \leftarrow T.out_{\mathcal{A}}$ 
3:   if  $label \neq \text{'IP-CON'}$  then
4:     return  $\perp$ 
5:   end if
6:    $roots \leftarrow \text{GETRIRCERTS}(T), \quad rir\_pps \leftarrow \bigcup_{\psi \in roots} \psi.SIA$ 
7:    $crit \leftarrow \text{GETCRITIPS}(T, rir\_pps), \quad pp\_ents \leftarrow \text{GETPPENTS}(T, rir\_pps)$ 
8:   return  $\text{CRITICALHOSTSADOPTROV}(T, rir\_pps) \wedge$ 
9:    $\text{RPPREFIXCOVERED}(T, pp\_ents, roots, t) \wedge$ 
10:   $\text{PRELOADEDCRITROAS}(T, crit, roots, t)$ 
11: end model
12: procedure  $\text{CRITICALHOSTSADOPTROV}(T, rir\_pps)$ 
  Every benign host of an SIA, advertised PP or a TAL adopts ROV.
13:   return every benign init  $\hat{e}$  with  $s_{\hat{e}}.IP_{\text{PP}} \in rir\_pps, s_{\hat{e}}.IP_{\text{TAL}} \neq \perp$ 
  or  $s_{\hat{e}}.IP_{\text{RP}} \neq \perp$  has  $s_{\hat{e}}.rov = \top$ 
14: end procedure
15: procedure  $\text{RPPREFIXCOVERED}(T, pp\_ents, roots, t)$ 
16:   return for every benign init  $\hat{e}_i$  with  $ip_v = s_{\hat{e}_i}.IP_{\text{RP}} \neq \perp$ ,
  writing  $\mathcal{P}_i = \text{PUBLISHED}(T, pp\_ents, t_{\hat{e}_i} + \Delta[\text{'orig'}])$ :
17:     some  $\hat{e}_o$  originates  $q \supseteq ip_v$  for  $a = T.\text{ent}[\hat{e}_i]$ ,
18:     and some  $\psi_{rp} \in \mathcal{P}_i$  has  $\text{EXACTROA}(\psi_{rp}, q, a, roots, \mathcal{P}_i, [t_{\hat{e}_i}, t + \delta_{\text{e2e}}]) = \top$ 
19: end procedure
20: procedure  $\text{PRELOADEDCRITROAS}(T, crit, roots, t)$ 
  Every benign adopter starts with the roots and a matching ROA for each critical infrastructure.
21:   return for every benign init  $\hat{e}$  with  $s_{\hat{e}}.rov = \top$ ,
  writing  $\mathcal{P} = s_{\hat{e}}.\text{preLoadedCerts}$ :
22:      $roots \subseteq \mathcal{P}$ , and
23:     for every  $ip_c \in crit$ , some  $\hat{e}_o$  originates  $q \supseteq ip_c$  for some  $a$ ,
24:     and some  $\psi_{roa} \in \mathcal{P}$  has  $\text{EXACTROA}(\psi_{roa}, q, a, roots, \mathcal{P}, [t_{\hat{e}}, t + \delta_{\text{e2e}}]) = \top$ 
25: end procedure
26: procedure  $\text{EXACTROA}(\psi, q, a, roots, \mathcal{S}, W)$ 
27:   return  $\text{VALIDATES}(\psi, q, a) \wedge \psi.ipPrefix = q \wedge \psi.maxLen = \text{len}(q) \wedge$ 
28:    $\text{CONSTRUCTANDVALIDATE}(\psi, \tau, roots, \mathcal{S}) = \top$  for every  $\tau \in W$ 
29: end procedure

```

Algorithm 71 Utilities for $\mu^{\text{SharedMirror}}_{\Delta[\text{'mirror'}]}$

```

1: utility  $\text{MIRRORHOSTSAREADOPTERS}(T, urls)$ 
2:   return  $\forall \hat{e} \in \{1, \dots, T.e\} :$ 
   
$$\left[ \begin{array}{l} T.\text{opr}[\hat{e}] = \text{'Init'} \wedge T.\text{ent}[\hat{e}] \in T.N - T.F \wedge \\ T.\text{inp}[\hat{e}].\text{params}.IP_{\text{PP}} \in \text{DNSresolve}(urls) \\ T.\text{inp}[\hat{e}].\text{params}.IP_{\text{RP}} \neq \perp \wedge T.\text{inp}[\hat{e}].\text{params}.rov = \top \end{array} \right] \Rightarrow$$

3: end utility

4: utility  $\text{SHAREDADVERTISEMENT}(roots, \mathcal{M})$ 
5:   return  $\forall \psi \in roots : \psi.SIA = \mathcal{M}$   $\triangleright$  No root points outside  $\mathcal{M}$ 
6: end utility

7: utility  $\text{REPLICATED}(T, urls)$ 
8:    $hosts \leftarrow \text{GETPPENTS}(T, urls)$ 
9:   return  $\forall$  real time  $t, \forall url \in urls$  with  $AS_{url} = \text{GETPPENTS}(T, \{url\}) \neq \emptyset$ :
    $\text{PUBLISHED}(T, hosts, t - \Delta[\text{'mirror'}]) \subseteq \text{PUBLISHED}(T, AS_{url}, t)$ 
10: end utility

11: utility  $\text{UNIFORMWELLCONNECTED}(T, adopters, \mathcal{M})$ 
12:   return  $\forall v \in adopters : \exists url \in \mathcal{M} :$ 
   
$$\exists AS_M \in \text{GETPPENTS}(T, \{url\}) :$$

   
$$\forall path \in \text{GETALLBESTPATHS}(T, AS_M, v) : \forall u \in path :$$

    $\text{WELLCONNECTED}(T, u, AS_M)$ 
13: end utility

14: utility  $\text{WELLCONNECTED}(T, A, B)$   $\triangleright$  Formal Definition 8
15:   return  $\left[ \begin{array}{l} \mu_{A,B}^{\text{AdoptedBP}}(T) \wedge \mu_{B,A}^{\text{AdoptedBP}}(T) \wedge \\ \mu_{A,B}^{\text{HonestPaths}}(T) \wedge \mu_{B,A}^{\text{HonestPaths}}(T) \end{array} \right]$ 
16: end utility

```

Algorithm 72 $\mu^{\text{SharedMirror}}_{\Delta[\text{'mirror'}]}$: all RIRs share one set of mirrors

Description: The *shared* mirroring model, every benign host of a mirror runs its own RP. All RIRs advertise one common set of mirrors $\mathcal{M}$, so a single mirror serves an adopter for every RIR at once. Each mirror serves at time t everything published at any mirror at $t - \Delta[\text{'mirror'}]$. Every honest ROV adopter v has a mirror host AS_M that is well-connected (Definition 8), as well as the ASes on the best path from v to AS_M .

```

1: model  $\mu^{\text{SharedMirror}}_{\Delta[\text{'mirror'}]}(T)$ 
2:    $roots \leftarrow \text{GETRIRCERTS}(T)$ 
3:    $\mathcal{M} \leftarrow \bigcup_{\psi \in roots} \psi.SIA$ 
4:   let  $adopters$  be the honest entities whose 'Init' sets  $params.rov = \top$ 
5:   return  $roots \neq \emptyset \wedge$ 
6:    $\text{MIRRORHOSTSAREADOPTERS}(T, \mathcal{M}) \wedge$ 
    $\triangleright$  All mirror hosts are adopters
7:    $\text{SHAREDADVERTISEMENT}(roots, \mathcal{M}) \wedge$ 
8:    $\text{REPLICATED}(T, \mathcal{M}) \wedge$ 
9:    $\text{UNIFORMWELLCONNECTED}(T, adopters, \mathcal{M})$   $\triangleright$  Every adopter  $v$  and its path to  $AS_M$  are well-connected.
10: end model

```
