RHINE: Robust and High-performance Internet Naming with E2E Authenticity

Huayi Duan, Rubén Fischer, Jie Lou, Si Liu, David Basin, and Adrian Perrig, *ETH Zürich*

https://www.usenix.org/conference/nsdi23/presentation/duan

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

RHINE: Robust and High-performance Internet Naming with E2E Authenticity

Huayi Duan, Rubén Fischer, Jie Lou, Si Liu, David Basin, and Adrian Perrig
ETH Zürich

Abstract

The variety and severity of recent DNS-based attacks underscore the importance of a secure naming system. Although DNSSEC provides data authenticity in theory, practical deployments unfortunately are fragile, costly, and typically lacks end-to-end (E2E) guarantees. This motivates us to rethink authentication in DNS fundamentally and introduce RHINE, a secure-by-design Internet naming system.

RHINE offloads the authentication of *zone delegation* to an end-entity PKI and tames the operational complexity in an *offline* manner, allowing the efficient E2E authentication of *zone data* during *online* name resolution. With a novel logging mechanism, Delegation Transparency, RHINE achieves a highly robust trust model that can tolerate the compromise of all but one trusted entities and, for the first time, counters threats from superordinate zones. We formally verify RHINE's security properties using the Tamarin prover. We also demonstrate its practicality and performance advantages with a prototype implementation.

1 Introduction

The importance of DNS as an integral part of the Internet cannot be overstated. If DNS is corrupted, so would be all relying Internet services [33]. Yet, this critical system has no built-in protection for data at rest or in transit. The infamous Kaminsky attack [57] raised worldwide awareness of the severity of DNS cache poisoning and thereafter spurred the deployment of several protocol-level defense mechanisms. Recent years have, however, witnessed a flurry of new vulnerabilities [17, 66, 67, 90] that revive the threat of cache poisoning and DNS hijacking in general [50].

The implications of these attacks are profound: they enable the sabotage of a wide spectrum of online systems, ranging from web applications and email to time synchronization and cryptocurrencies [33]. One of most alarming facts is that DNS plays an essential role in bootstrapping the Internet's security. In the modern web PKI, certificate issuance relies on DNS-based channels for domain validation. If such channels are unauthenticated, attackers can manage to acquire fraudulent TLS certificates and impersonate domains [25, 27, 81]. Hence, an end-to-end (E2E) authenticated naming system is necessary for E2E secure communication.

DNS Security Today. Strengthening plain DNS with security guarantees has been a decades-long but still largely ongoing endeavor. DNSSEC [18] is by far the most important security extension to DNS. It allows a zone owner to cryptographically sign DNS records which, at least in theory, averts the threat of DNS hijacking. However, the deployment of DNSSEC is still far from complete (e.g., it is estimated that only 25% of DNS responses worldwide are validated as of mid-2022 [15]), and years' of practical experience indicates that it is highly fragile and fraught with problems.

The complexity of DNSSEC makes its operation an error-prone and expensive process. It requires each zone to synchronize its keying materials with its parent. Any inconsistency in an authentication chain will cause validation and hence resolution failure. This has caused frequent outages at all levels of the DNS hierarchy [54]. Validation failure can incur severe overhead to DNS servers and the name resolution process [53]. Partly because of these factors, and partly by design [88], end hosts rarely validate signed records by themselves but rely on validating recursive resolvers at best [64]. As a result, DNSSEC fails to provide E2E data authentication in practice, despite pervasive DNS interception [65, 71, 77].

The trust model of DNSSEC is also controversial. DNS is not designed for security, and mismanagement of DNSSEC by DNS operators is commonplace [29, 82]. Compromising a zone's secret key implies the control of all its subzones. This raises the concern that DNSSEC consolidates the power of the few Internet governance bodies and state governments over the DNS namespace [83]; in fact, large-scale DNS hijacking campaigns sponsored by state agencies have already been observed in the wild [46]. DNSSEC requires a validating entity to trust all zones on an authentication chain; any one of them can provide correctly signed yet bogus data [2].

These issues have their root in DNSSEC's underlying architecture, which mirrors the hierarchical namespace, and therefore they cannot be resolved within DNS. This poses the question: *Is it possible to build a DNS-compatible yet robust naming system that enables efficient E2E authentication?*

Introducing RHINE. We provide an affirmative answer to this question with the design, verification, implementation, and evaluation of a system called RHINE. Our key insight is that the authentication of *zone data* and *zone delegation* in DNS, while treated identically by DNSSEC, should be decoupled. The latter form of authentication, which is more delicate and costly, can be performed by external trusted entities in an *offline* manner. Specifically, we employ certificate authorities (CAs) from the web PKI to certify zone delegation, allowing clients that already rely on these CAs to efficiently validate zone data during *online* name resolution.

Despite its promising opportunities, this architecture also raises unique challenges. Certifying a zone's authority with CAs creates a *circular dependency*, because, as mentioned earlier, secure certificate issuance hinges on a secure naming system in the first place. On a different front, the corruption of a single CA may put the entire DNS namespace at risk. Moreover, malicious DNS and PKI authorities can interact in subtle ways to subvert a zone's authenticity.

What we strive for is a system of *checks and balances* where the parties involved (zone owners, CAs, and loggers) watch over each other so that no single party or partial collusion between them can undermine a zone's authority. RHINE systematically addresses security threats arising from the envisioned architecture, offering a set of protocols for secure zone management and E2E-authenticated name resolution. At its core is Delegation Transparency (DT), a novel public logging mechanism to maintain global zone delegation status.

It is essential to rigorously establish the expected security properties for our design. Using a state-of-the-art security protocol verifier, Tamarin [68], we have formally proved that RHINE guarantees E2E data authenticity for legitimately delegated zones in a highly robust trust model.

Our evaluation with a prototype implementation shows that RHINE can cope with real-world certificate issuance rates (millions per day) and, compared with DNSSEC, achieve lower resolution latency and higher resolver performance.

2 Problem Statement

We start by introducing the basic concepts of DNS. Afterwards, we contextualize the data authentication problem and analyze the intrinsic weaknesses of DNSSEC.

2.1 Name Resolution Basics

The global DNS namespace is organized as a tree structure, where each node is a *zone* that manages *resource records* mapping names to IP addresses and other data. Delegating a portion of a zone creates another node in the tree and hence a (sub)zone. Below the root zone lie top-level domains (TLDs) such as `.com` and `.org`, second-level domains (SLDs) such as `a.com`, and so forth. A zone should be authoritative for all names under it except those under its *delegated* subzones. For example, assuming the zone `b.a.com` exists but `c.a.com` does not, then the zone `a.com` is authoritative for `c.a.com` and `d.c.a.com` but not `b.a.com` or `d.b.a.com`. A zone's apex is the name identifying the zone itself.

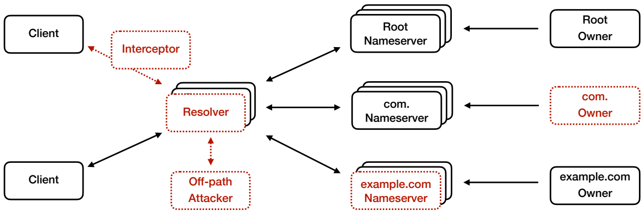

Figure 1: A simplified DNS infrastructure. Components in red and dotted lines indicate various threats to data authenticity.

DNS runs on a distributed infrastructure. We consider a simplified infrastructure with four types of entities depicted in Figure 1. The *owner* of a zone is a logical entity with legitimate authority over it. When the context is clear, we extend the term "zone" to also indicate its owner. A zone hosts its data on multiple *(authoritative) nameservers*, which in many cases are not under the control of the zone owner [58, 76]. In the name resolution process, a *(recursive) resolver* handles name lookup queries from *clients* (aka stub resolvers), by iteratively asking nameservers for matching record(s) in a top-down manner. Caching at resolvers reduces the overall lookup costs and helps DNS operate at Internet scale.

2.2 Authentication in DNS

Plain DNS offers no authentication of resource records. They can be corrupted anywhere before reaching a client, as highlighted in Figure 1. Any on-path network node can access, modify, and fabricate DNS messages. An off-path adversary can also intervene in the resolution process and inject bogus data, as demonstrated by the Kaminsky attack and its variants [66, 67]. Nameservers and resolvers may deviate from their expected behavior due to domain hijacking [85], malware infection [32], business incentives [87], or regulatory pressure [72]. Less obvious threats are posed by malicious zone owners themselves, who can surreptitiously (and somewhat rightfully) manipulate their subzones [2, 83].

Network attackers can be thwarted by secure channels. DNSCurve [23] was proposed to protect the communication between a resolver and nameservers using an in-band key exchange scheme. More recent and widely deployed protocols include DNS over TLS (DoT) [52] and DNS over HTTPS (DoH) [47], which focus on securing the last-mile communication between a client and a resolver. These solutions fail to mitigate risks arising from nameservers, resolvers, and any intermediary servers on the resolution path.

As the most prominent security addition, DNSSEC enhances DNS with data integrity and origin authentication.

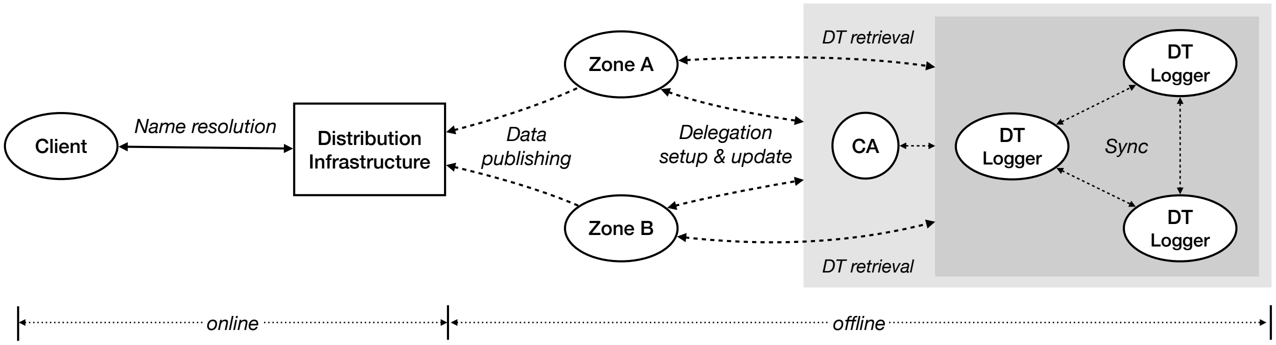

Figure 2: The high-level architecture of RHINE, where operational complexity (e.g., the authentication and management of zone authority) is pushed towards an *offline* phase. The arrows represent interactions between (groups of) entities.

It allows a zone to cryptographically sign its records using a secret key, with the corresponding public key signed by the parent zone. A security-aware resolver can verify a signed record by following an authentication chain all the way up to the root zone (key), without trusting any on-path servers.

2.3 Problems with DNSSEC

Since the signing of the root zone's key in 2010, DNSSEC has seen gradual uptake, but the deployments are not all smooth. It is often cited by practitioners as overly complicated and not worth the costs it exacts [54]. We analyze its drawbacks in practical operation and from a security perspective.

Fragile Operation. DNSSEC requires synchronization between each pair of adjacent nodes in an authentication chain. Any inconsistency (e.g., missing or mismatching keys or security parameters) between a zone and its parent will cause validation and hence resolution failure, blocking not only the failed zone but also all its subzones. It is thus unsurprising that Internet outages caused by DNSSEC happen frequently *at all levels* including the root, TLDs and SLDs, and *across various organizations* including DNS governance bodies themselves (e.g., ICANN and RIPE) as well as large service providers (e.g., Verisign, Dyn, and Google) [54].

While DNSSEC already imposes significant performance overhead with respect to plain DNS resolution, validation failure can further boost its costs. It is estimated that with failure factored in, the authoritative nameservers of a DNSSEC-signed zone should be prepared to handle 10 times the query traffic volume and 100 times the response traffic volume of their unsigned counterparts for an Internet-wide deployment [53]. The potential of abusing DNSSEC for denial of service (DoS) is well-recognized and many real-world attacks have been reported [1].

The operational complexity, high failure rate, and performance overhead all contribute to the fact that end hosts rarely validate DNSSEC-signed records [64]. It is actually by design that end hosts should rely on validating recursive resolvers to verify records [88]. As a result, DNSSEC almost never provides E2E data authentication in practice.

Fragile Security. The security of DNSSEC rests on DNS itself. However, unlike PKIs, DNS is not designed for security; and unlike CAs, zone owners and operators may not be security-savvy. Real-world measurements have revealed widespread mismanagement of DNSSEC with flawed security practices (using weak keys, reusing keys for multiple zones, etc.) [29, 82]. The compromise of a zone's secret key endangers not only the zone itself but also all its subzones.

A common criticism of DNSSEC is that it consolidates the Internet's governance [83]. The root zone is governed by ICANN, the most important TLD `.com` is managed by Verisign under the jurisdiction of US law, and each country-code TLD is ultimately controlled by the corresponding sovereign state. Large-scale DNS hijacking campaigns sponsored by state agencies have been observed in real world [46].

While the governance model of DNS remains a subject of controversy, from a technical point of view, DNSSEC's trust model is fundamentally fragile in that it provides clients with no option but to trust all zones on a delegation chain. A malicious zone can surreptitiously claim and serve authenticated data for names belonging to any subzone. This problem has just begun to gain attention from the Internet community, and there is a proposal to mark the root zone and TLDs as delegation-only so that their ability to serve authoritative data is limited [2]. However, implemented within DNS, this mechanism cannot solve the inherent limitations of DNSSEC.

2.4 Desired Properties

Our analysis of DNSSEC reveals the following properties desired by an ideal authenticated Internet naming system.

- **P1: End-to-end (E2E) data authenticity.** A validating client must be assured that any verified resource record is indeed generated by the genuine authoritative zone.
- **P2: Authentication efficiency.** The computation and communication costs of authenticating resource records, especially in case of failure, are lower than DNSSEC.
- **P3: Operational robustness.** The authentication of a zone's data is unaffected by any superordinate zone's op-

erational faults in managing security, e.g., misconfigured security policies or keying materials.

- **P4: Robust trust model.** If a zone relies on a group of entities (including its parent and any external trusted parties) to establish its authority, then no single entity or partial collusion between them can claim authority over the zone.

3 RHINE Overview

RHINE is a naming system with built-in security, satisfying all the properties listed in Section 2.4. Our starting point is the observation that the authentication of a DNS zone consists of two parts: authenticating resource records during name resolution and, when the zone is created, authenticating the delegation's legitimacy. The latter can be offloaded from clients to external trusted entities: in particular, the CAs in today's web PKI that billions of clients already rely on. Once a zone is delegated and certified, it can serve authenticated data and manage its security independently, without synchronizing with its parent as in the case of DNSSEC. This isolates the failures caused by a zone's security mismanagement from its subzones. The reduction in validation failure and authentication chain length also improves name resolution performance.

This new security architecture simultaneously achieves the desired properties **P1**, **P2**, and **P3**. Yet, it introduces both unprecedented opportunities and challenges to meet **P4**, without which the system can be broken in many ways. This is the main focus of our design (Section 3.3).

RHINE Architecture. We depict our architecture in Figure 2. It consists of two parts. In the *offline* part, zone owners establish new delegations by acquiring publicly logged RHINE certificates (RCert) from a CA and loggers (Section 5.1). An existing zone can update its RCert or delegation status (Section 5.2). It also periodically retrieves delegation status proofs (DSP) (Section 5.3) from a public transparency log (Section 4). A zone signs its resource records using its RCert and publishes them to a distribution infrastructure. During *online* name resolution, a client who already relies on the web PKI can easily verify an answer's authenticity using the associated RCert and DSP (Section 5.4).

This architecture shifts much of DNSSEC's complexity to offline operations, minimizing the risk of failure during the name resolution process. It also clearly separates the distribution and authentication of DNS data. While we intend to reuse the existing DNS infrastructure consisting of authoritative nameservers, recursive resolvers, forwarders, etc., RHINE can be instantiated with other distribution architectures such as a peer-to-peer network [14], or enable client authentication of records received through DoT or DoH.

3.1 Notation and Primitives

We use *uppercase* letters (e.g., X) to identify entities (zone owners, CAs, and loggers) that run RHINE protocols, and *lowercase* letters in the subscript to identify zones (e.g., $\texttt{ZN}_x$) and their associated data (e.g., $\texttt{RCrt}_x$). For brevity, we sometimes refer to the pair of zones related by delegation and their corresponding owners simply as the *parent* and *child*.

Table 1: Summary of Notation.

Notation	Definition
pk_X, sk_X	The key pair of entity X (in *uppercase*)
$\texttt{ZN}_x$	A zone identified by x (in *lowercase*)
$\texttt{RCrt}_x$, zpk_x, zsk_x	The RCert and associated key pair of $\texttt{ZN}_x$
$:=$	Definition/assignment operator
$(a,b,\dots)$	A tuple of values encoded as a string
$\texttt{H}(\cdot)$	A secure hash function
$\langle m\rangle_X$ or $\langle m\rangle_x$	A message signed with sk_X or zsk_x
$\Sigma.\texttt{Vf}(k,m)$	Verify a signed message m with a key/cert k
$\texttt{Acc.Vf}(ac,p)$	Verify a membership proof p with digest $\texttt{ac}$

We use standard cryptographic primitives including secure hash functions and digital signatures. The public keys of CAs and loggers are known to all entities. To design succinct data structures, we also use cryptographic accumulators [34] that can commit sets of values into small digests and generate compact membership proofs. Classic constructions include the Merkle hash tree (MHT) [69] and its variants. Table 1 summarizes our notation.

3.2 Threat Model

Table 2 summarizes the adversaries we consider in the design of RHINE and the expected security properties.

$\mathcal{A}_1$ is a conventional Dolev-Yao network attacker [37] (who can eavesdrop, modify, and inject messages transmitted over the network) augmented with the ability to control the entire DNS distribution infrastructure.

The next two types of adversaries pertain to today's web PKI ecosystem: $\mathcal{A}_2$ can issue arbitrary certificates by compromising a CA; $\mathcal{A}_3$ can compromise some loggers and provide fake or inconsistent log data to users. Since we repurpose the web PKI to authenticate delegated zone authority, RHINE must also deal with these adversaries.

For the first time, we systematically address an adversary ($\mathcal{A}_4$) that controls a zone and attempts to subvert its subzones by declaring authoritative data for them. This capability is inherent in the hierarchical naming structure of DNS, i.e., a name under a zone is also under its parent zone. For an authenticated naming system where zone data is cryptographically signed, $\mathcal{A}_4$ has access to the private key of the zone it controls. As an example, an $\mathcal{A}_4$ attacker compromising the TLD `xyz` can generate valid records for `abc.example.xyz`, despite that the SLD `example.xyz` has been legitimately delegated.

Overall, we consider attackers that seek to, given the stated capabilities, break the naming system's *data authenticity but not availability*—that is, tricking clients into accepting malicious data rather than preventing clients from receiving any answer. We assume that the attackers cannot break the cryptography primitives used by RHINE, and that privacy aspects of DNS are outside this paper's scope.

Table 2: Summary of adversaries considered in DNS and the web PKI. We also list the corresponding security properties and representative defense mechanisms to achieve them.

Adversary	Capability	Security property (informal)	Defense mechanisms
Dolev-Yao	controls communication networks	channel security	DoT/DoH/DoQ, DNSCurve
$\mathcal{A}_1$	+ DNS distribution infrastructure	data authenticity	DNSSEC, GNS [14]
$\mathcal{A}_2$	controls some CA(s)	certificate misissuance prevention	ARPKI [21], F-PKI [28]
$\mathcal{A}_3$	controls some logger(s)	tolerating all but one compromises	LogPicker [36], CTng [63]
$\mathcal{A}_4$	controls a DNS zone (e.g., a TLD)	authority independence (of subzones)	-
$\mathcal{A}_1 + \mathcal{A}_2 + \mathcal{A}_3 + \mathcal{A}_4$ (strongest possible adversary)		**E2E authenticity with robust trust**	**RHINE**

3.3 Design Rationale

RHINE strives to counter the strongest possible adversary that combines all capabilities as shown in Table 2. Before fleshing out RHINE's design, we discuss the main aspects to be considered, analyze why existing approaches fail, and highlight the intuitions behind our solutions.

3.3.1 Validating Zone Ownership ($\mathcal{A}_1$)

Secure delegation in RHINE requires the expected zone owner to request an RCert from a CA. The issuing CA must verify that a requesting entity indeed controls the zone to be certified. Commonly known as domain validation (DV), this process is mandatory for the issuance of TLS certificates. In standard practice [20], the requester proves its ownership of a domain by publishing a challenge token specified by the contacted CA. A network attacker can exploit an insecure channel in this process to obtain a fraudulent certificate. Unfortunately, all practical DV channels hinge on DNS and are therefore exploitable by an $\mathcal{A}_1$ attacker [24, 27, 81]. Applying these standard DV methods to our case will lead to a ***circular dependency****: the CA depends on an authenticated zone for ownership validation and RCert issuance, but meanwhile, the zone needs an RCert to authenticate its data in the first place.*

RHINE solves this dilemma by engaging the parent to approve the delegation. This is indeed necessary, as the parent still legitimately controls the child before it is established. Specifically, the parent must sign a delegation request using its own RCert. The CA can then verify that the *current* owner of the child zone approves the delegation. In doing so, RHINE creates an *implicit offline* authentication chain of delegated authority, as opposed to what is explicitly constructed by DNSSEC, and shifts the heavy authentication workload away from the client side of DNS.

3.3.2 Preventing Certificate Misissuance ($\mathcal{A}_2$ & $\mathcal{A}_3$)

Security breaches of CAs [49] spurred the deployment of Certificate Transparency (CT) [61], which employs public logs to make misissued certificates detectable. Mainstream browsers have mandated public logging for TLS certificates to be valid [3, 6]. One limitation of CT is that it provides *deterrence* rather than *prevention*. Fraudulent certificates may still be used before being detected and revoked. CT loggers passively accept certificates that meet basic validity criteria (properly formatted and signed, non-expired, etc.) but never validate domain ownership as CAs do. Also, the compromise of loggers has already occurred in practice [79].

RCerts are more critical than TLS certificates in terms of security, because the naming service is one of the weakest links in many Internet systems including the web PKI itself [33]. In addition, the detection of fraudulent RCerts is more involved in that it requires investigating delegation chains rather than individual domains. Therefore, we need preventive measures to foil the misissuance and logging of unauthorized RCerts.

There are proposals to make today's web PKI more resilient to the compromise of CAs and loggers [21, 28, 36, 63]. Yet, they are not applicable to the new security architecture we envision for RHINE. This is because: (1) they are designed for TLS certificates and so they will suffer from the bootstrapping dilemma discussed earlier (Section 3.3.1), and (2) their log data models, either reusing or building upon CT, do not meet our security and performance requirements (Section 3.3.3).

We address the $\mathcal{A}_2$ and $\mathcal{A}_3$ adversaries from several aspects: (1) integrating loggers into the certificate issuance process for proactive verification of the data to be logged, (2) enabling a zone to choose its own trusted loggers rather than relying on whichever loggers are chosen by CAs (as in the case of CT), and (3) enforcing loggers and CAs to crosscheck each other throughout the certificate issuance and logging process. This allows RHINE to defeat attackers that can compromise multiple trusted entities designated by a zone.

3.3.3 Countering Parental Attacks ($\mathcal{A}_4$)

A zone gains *authority independence* if its ancestors cannot claim authoritative data under its authority. The cryptography of DNSSEC makes the situation even worse than in regular DNS. Our new security architecture does not immediately address this challenge. In particular, a malicious parent can still serve authentic records for delegated children, using its own RCert or alternative child RCerts acquired by it. In order to counter such parental attacks, we must enable a dependent entity (client or CA) to verify the status of the delegations in question without trusting zone owners themselves.

Since delegation status can be inferred from logged RCerts, it seems plausible to design a solution atop CT. A closer

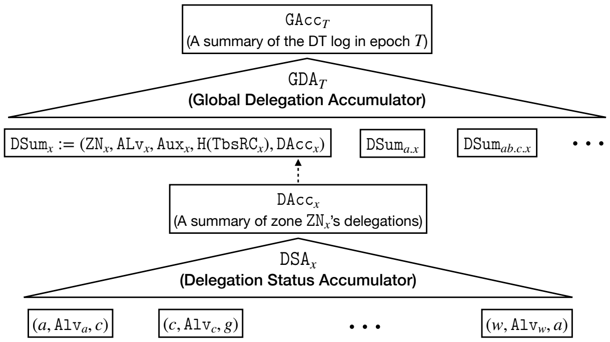
$\texttt{GAcc}_T$
(A summary of the DT log in epoch T)
$\texttt{GDA}_T$
(Global Delegation Accumulator)
$\texttt{DSum}_x := (\texttt{ZN}_x, \texttt{ALv}_x, \texttt{Aux}_x, \texttt{H}(\texttt{TbsRC}_x), \texttt{DAcc}_x)$
$\texttt{DSum}_{a.x}$
$\texttt{DSum}_{ab.c.x}$
$\cdots$
$\texttt{DAcc}_x$
(A summary of zone $\texttt{ZN}_x$'s delegations)
$\texttt{DSA}_x$
(Delegation Status Accumulator)
$(a, \texttt{Alv}_a, c)$
$(c, \texttt{Alv}_c, g)$
$\cdots$
$(w, \texttt{Alv}_w, a)$

Figure 3: The data structures used by DT.

look reveals several pitfalls. First, a dependent entity needs to ascertain that a zone in question does *not* exist, but CT has no native support for *absence proofs*. Second, such proofs must have global coverage, but CT loggers operate independently and maintain only partial views of all issued certificates. It would be onerous to assemble and synchronize data from all CT logs with correctness and performance guarantees. Third, the data structures used by CT are too heavy to represent and authenticate global delegation status.

These inefficiencies motivate us to create a more efficient transparency mechanism dedicated to keeping track of the entire namespace's delegation structure.

4 Delegation Transparency

At the heart of RHINE is Delegation Transparency (DT), a lightweight verifiable log design. In contrast to CT, which maintains the *history* of all certificates ever issued, DT offers an up-to-date *snapshot* of global zone delegation status. A single DT log is replicated to a consortium of loggers. The loggers receive requests to update delegation status and periodically synchronize with each other to maintain a consistent log. They also provide publicly verifiable delegation information. Below we introduce the basics of DT. Its operation as an integral part of RHINE is described in Section 5.

Log Data. Figure 3 depicts DT's data model. We define the delegation status of a zone $\texttt{ZN}_x$ as a tuple $(\texttt{ALv}_x, \texttt{Aux}_x, \texttt{CSet}_x)$, where the first item is the *authority level* of $\texttt{ZN}_x$ (explained below), the second is auxiliary information for $\texttt{ZN}_x$ (e.g., expiration time or revocation status of the delegation), and the third is a set representing $\texttt{ZN}_x$'s child zones and their authority levels: $\texttt{CSet}_x := \{(c_1, \texttt{ALv}_{c_1}), (c_2, \texttt{ALv}_{c_2}) \dots, \}$.

We encode the delegation status of $\texttt{ZN}_x$ into a data structure called $\texttt{DSum}_x$ (delegation summary). $\texttt{DSum}_x$ contains a cryptographic digest of the zone's RCert. This ensures that at any time there is only one valid RCert per zone, capturing that the authority over a zone should be unique. Since a zone may have many delegations, $\texttt{DSum}_x$ stores the digest ($\texttt{DAcc}_x$) of an accumulator $\texttt{DSA}_x$ over $\texttt{CSet}_x$ rather than $\texttt{CSet}_x$ itself. This reduces the cost of authenticating a specific child's (non)existence. Each input element of $\texttt{DSA}_x$ contains the label and authority level of one child as well as the label of the next child in a *canonical order* [19]. This allows a single membership proof from the accumulator to prove either the presence or absence of a child zone.

	TER	¬TER ∧ ¬DOL	DOL	
IND	(1, 0)*	(1, 1)*	(0, 1)*	EOI
		(1, 2)	(0, 2)*	¬EOI
¬IND	(1, 0)*	(1, 1)*	(0, 1)*	

Figure 4: The authority level matrix derived from the interaction of constraint flags. The shaded area indicates the division caused by the EOI flag. In each pair (a, b), $a \in \{0, 1\}$ encodes a zone's ability to serve authoritative data for all its names excluding those of its independent subzones; $b \in \{0, 1, 2\}$ encodes a zone's delegation capability (0: not allowed; 1: non-independent child only; 2: any child). The cases marked with * permit fast data validation (see Section 5.4).

For efficient synchronization and auditing of the DT log, we introduce a global accumulator GDA over all $\texttt{DSum}_x$s. Loggers can commit GDA's digest (GAcc), along with the necessary data to replay logged changes, into an authenticated data structure that supports succinct consistency proofs [62].

Authority Level. While we envision that authority independence is desired by many zones (including all TLDs and SLDs), this may not always be the case, for instance when the parent and child are managed by the same entity. To enable fine-grained control over zone authority, we introduce the concept of *authority level*, which places constraints on what a zone can do to its data and delegation. We define authority levels using *constraint flags*, as depicted in Figure 4.

The flag IND indicates a zone's authority independence. By definition, an independent zone has the sole authority over its names, whereas a non-independent (¬IND) zone's names are also under the authority of its parent. The data served by a zone comes in two types: authoritative and delegation. A *terminating* (TER) zone can serve only authoritative data; all leaf zones are by default terminating. A *delegation-only* (DOL) zone can serve only delegation data (i.e., NS records in DNS); all TLDs are supposed to be delegation-only. Note that these two flags cannot be set simultaneously as this would lead to an empty and useless zone. Since a non-independent zone can never delegate to an independent child, authority independence can end at some non-leaf zone on a delegation path; such a zone is marked as *end-of-independence* (EOI).

Delegation Status Proof (DSP). Clients make use of the DT log in the form of DSP, which consists of a timestamped $\texttt{DSum}_x$ signed by loggers and, if necessary, a membership proof from $\texttt{DSA}_x$ for some child zone $\texttt{ZN}_y$ of $\texttt{ZN}_x$. A DSP enables clients to determine a zone's realm of authority and hence whether to accept an answer signed with the corresponding RCert. A malicious parent may use an outdated

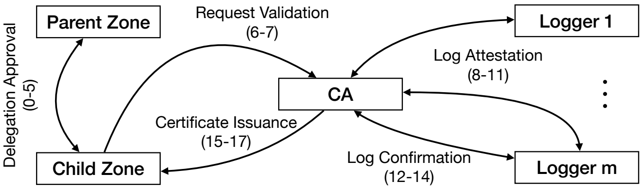

Figure 5: Overview of the delegation setup protocol.

DSP to trick clients into accepting fraudulent data for the names belonging to a delegated child. Prudent clients should accept only DSPs that are recent enough.

5 RHINE Protocols

We specify RHINE's core functions with a set of protocols, including the secure management of zone delegation, the maintenance and usage of DT, and E2E-authenticated name resolution. The entire system operates in *epochs*, which are consecutive time windows of a predetermined length. This is necessary to keep DT loggers in synchrony and to establish the system's security. In each epoch, zone owners can securely set up new delegations or update existing ones until a cut-off time. The resulting changes in these zones' delegation status will be applied to the DT log within the same epoch and take effect from the next epoch. Zone owners can actively monitor the log for unexpected events like attacks or operational faults, and take action accordingly. They also regularly retrieve signed log entries to prove their authority over answers served during name resolution.

5.1 Secure Delegation Setup

In RHINE, delegating a zone $\mathtt{ZN}_c$ begins with the intended owner C negotiating the delegation with P, the owner of the parent zone $\mathtt{ZN}_p$. This follows standard DNS practices, e.g., domain registration. Afterwards, C should run the secure delegation setup protocol specified in Figure 6 to obtain an RCert. This protocol follows the design intuitions presented in Section 3.3. An overview of its flow is depicted in Figure 5.

In the initial phase (Steps 0-5), C asks for a signed approval (apv) for its delegation request (sdr) from P. The request encodes the trusted entities selected by C, the delegation parameters negotiated with P, and most importantly, the public key to be certified. The corresponding private key is also used to sign the request. There must be a way for P to authenticate the association between C and the key. This is done using an initial secure out-of-band key registration procedure (Step 0), for example, via a secure web portal with account-based client authentication when P is a domain name registrar.

Next, C sends the request to a CA for validation (Steps 6-7). In addition to verifying the parent's approval, the CA checks the delegation's legitimacy using the DT log. If everything is correct, the CA sends a pre-logging request (prl), which includes a to-be-signed certificate, to the designated loggers for crosschecking (Steps 8-11). After assembling the loggers' attestations, the CA randomly picks one of them to store the logging request ($lreq$) as an input to the later aggregation process (Steps 12-14). Finally, C receives an RCert accompanied by attestations and a confirmation that the zone $\mathtt{ZN}_c$'s delegation status will be added to the DT log (Steps 15-17).

Our design ensures that the entire delegation setup process is witnessed by multiple parties and any misbehaving party will be held accountable for the messages it signs. Any verification failure will cause the protocol execution to abort, broadcasting a failure message to all the involved parties. It is impossible to obtain a valid logged RCert without faithfully following the protocol. Even in the presence of an omnipotent attacker whose capabilities go beyond our threat model, RHINE still allows a zone owner to detect and counter attack attempts before harm is caused (see Section 6).

Secure Bootstrapping. The delegation setup protocol assumes the parent's RCert already exists. A bootstrapping problem arises at the top of the namespace. Representing a critical Internet authority in itself, the root zone should not depend on another CA. Therefore, we treat the root zone as a root CA that signs its own RCert. Similarly, TLDs resemble intermediate CAs with their RCerts signed by the root RCert. This allows the root zone and TLDs to retain their innate power over the namespace, effectively restricting an external CA's influence over the namespace to SLDs and below.

5.2 Secure Delegation Update

Once delegated, a zone can manage itself mostly independently of its parent. This includes updating its RCert and other delegation parameters. Similarly to delegation setup, processing an update request involves some CA and loggers as witnesses. The parent's involvement is required only for a request to extend the delegation's validity period or to change the child's authority level from non-independent to independent. RHINE has built-in support for certificate revocation. An updated RCert automatically revokes the old one, because by design a zone can only have one valid RCert at any time; a zone can also request for explicit revocation.

The update protocol is similar to the delegation setup protocol for the message flow and verification procedures. The major difference is that a zone should now sign the update request using its own RCert (instead of the parent's) to prove its authority. We provide further details in Appendix A.2.

5.3 DT Aggregation and Retrieval

In each epoch, loggers will receive disjoint sets of requests to update the DT log. To ensure the log's global consistency, they must aggregate all requests by synchronizing with each other. Wanner et al. formalized this problem as *secure log replication*—a special case of state machine replication—and proposed Logres, a formally verified log replication protocol with Byzantine fault tolerance that is optimal in terms of round complexity and the number of tolerable faults [86].

```
0. $C$ generates a key pair $(zpk_c, zsk_c)$ and register $zpk_c$ to
   $P$ via a secure out-of-band channel.
1. $C$ : select $A$ (a CA) and $\mathcal{L}_c$ (a set of loggers)
   // $t_0$ is a timestamp within the current epoch $\mathtt{T}$, $al$ is the
   requested authority level, $aux$ is auxiliary information.
       : $rid := \mathtt{H}(\mathtt{ZN}_c, zpk_c, A, \mathcal{L}_c, t_0, al, aux)$
   // $rid$ is implicitly included in all subsequent messages
2. $C \rightarrow P$ : $sdr := \langle rid, \mathtt{SDReq}(\mathtt{ZN}_c, zpk_c, A, \mathcal{L}_c, al, aux)\rangle_c$
3. $P$ : Verify $\Sigma.\mathtt{Vf}(zpk_c, sdr)$ and whether $al, aux$
       :     match what are agreed upon with $C$.
4. $P \rightarrow C$ : $\mathtt{RCrt}_p$, $apv := \langle \mathtt{SDApprvl}(\mathtt{H}(sdr))\rangle_p$
5. $C$ : Verify $\Sigma.\mathtt{Vf}(\mathtt{RCrt}_p, apv) \wedge \mathtt{Match}(apv, sdr)$
6. $C \rightarrow A$ : $sdr$, $apv$, $\mathtt{RCrt}_p$
7. $A$ : Verify $\Sigma.\mathtt{Vf}(\mathtt{RCrt}_p, apv) \wedge \Sigma.\mathtt{Vf}(zpk_c, sdr)$
       :     $\wedge\ \mathtt{Match}(apv, sdr)$
       : Retrieve $dsp := (\langle \mathtt{DSum}_p, \mathtt{T}'\rangle_{\mathcal{L}_c}, mem)$
       :     from local cache or the loggers $\mathcal{L}_c$.
   // Check if DSP is valid and the delegation is legit
       : Verify $\Sigma.\mathtt{Vf}(pk_{\mathcal{L}_c}, \langle \mathtt{DSum}_p, \mathtt{T}'\rangle_{\mathcal{L}_c})$
       :     $\wedge\ \mathtt{Match}(\mathtt{RCrt}_p, \mathtt{DSum}_p) \wedge \mathtt{T}' = \mathtt{T}\ \wedge$
       :     $\mathtt{Acc.Vf}(\mathtt{DAcc}_p, mem) \wedge \mathtt{Legal}(\mathtt{ALv}_p, al)$
       : $tbsrc := \mathtt{TBSCert}(\mathtt{ZN}_c, zpk_c, A)$
   // Pre-logging requests to all designated loggers
8. $A \rightarrow \mathcal{L}_c$ : $prl := \langle \mathtt{PreLog}(sdr, apv, tbsrc)\rangle_A$, $\mathtt{RCrt}_p$
9. $L_i$ : Verify $\Sigma.\mathtt{Vf}(pk_A, prl) \wedge L_i \in \mathcal{L}_c$
       :     $\wedge\ \Sigma.\mathtt{Vf}(\mathtt{RCrt}_p, apv) \wedge \Sigma.\mathtt{Vf}(zpk_c, sdr)$
   // Check if the to-be-signed cert matches the requested
       :     $\wedge\ \mathtt{Match}(apv, sdr) \wedge \mathtt{Match}(sdr, tbsrc)$
   // Check the delegation's legitimacy using local DT log
       :     $\wedge\ \mathtt{ZN}_c$ not delegated $\wedge\ \mathtt{Legal}(\mathtt{ALv}_p, al)$
       : $nds := (\mathtt{T}, A, \mathcal{L}_c, \mathtt{ZN}_c, al, aux, \mathtt{H}(tbsrc))$
10. $L_i \rightarrow A$ : $att_i := \langle \mathtt{LogAttest}(L_i, \mathtt{H}(nds))\rangle_{L_i}$
11. $A$ : Verify $\Sigma.\mathtt{Vf}(pk_{\mathcal{L}_c}, \{att_i\}) \wedge \mathtt{Match}(prl, \{att_i\})$
   // $L$ is randomly selected from $\mathcal{L}_c$ by $A$
12. $A \rightarrow L$ : $lreq := \langle \mathtt{LogReq}(L, nds, \{att_i\}_{L_i \in \mathcal{L}_c})\rangle_A$
13. $L$ : Verify $\Sigma.\mathtt{Vf}(pk_A, lreq) \wedge \mathtt{Match}(nds, \{att_i\})$
       :     $\wedge\ \Sigma.\mathtt{Vf}(pk_{\mathcal{L}_c}, \{att_i\}) \wedge L \in \mathcal{L}_c$
       : Add $lreq$ to a pending pool for aggregation
14. $L \rightarrow A$ : $lc := \langle \mathtt{LogCfm}(L, \mathtt{H}(nds))\rangle_L$
15. $A$ : Verify $\Sigma.\mathtt{Vf}(pk_L, lc) \wedge \mathtt{Match}(lc, lreq)$
       : $\mathtt{RCrt}_c := \langle \mathtt{FinalRCert}(\mathtt{TbsRC}_c, \mathcal{L}_c)\rangle_A$
16. $A \rightarrow C$ : $\mathtt{RCrt}_c$, $\{att_i\}_{L_i \in \mathcal{L}_c}$, $lc$
17. $C$ : Verify $\Sigma.\mathtt{Vf}(pk_A, \mathtt{RCrt}_c) \wedge \mathtt{Match}(sdr, \mathtt{RCrt}_c)$
       :     $\wedge\ \Sigma.\mathtt{Vf}(pk_{\mathcal{L}_c}, \{att_i\}) \wedge \mathtt{Match}(sdr, \{att_i\})$
       :     $\wedge\ L \in \mathcal{L}_c \wedge \Sigma.\mathtt{Vf}(pk_L, lc) \wedge \mathtt{Match}(sdr, lc)$
```

Figure 6: The secure delegation setup protocol. A party stores the messages it sends and receives whenever necessary. The function Match() checks the consistency between two data objects and Legal() checks a delegation's legitimacy. Other functions, such as SDReq() and TBSCert(), construct proper data objects from the input parameters.

This protocol however cannot be directly applied to our case, because it is agnostic to the validity of inputs: a malicious logger that participates in the consensus process faithfully can still inject arbitrary bogus data into the log.

To this end, we enhanced Logres with input validation (among other technicalities), requiring each logging request to be attested by the specified trusted entities (Steps 10-13, Figure 6). Using the modified version as a core consensus routine, we designed a secure aggregation protocol (Appendix A.3) that allows a majority of honest DT loggers to efficiently maintain a consistent log even in case of Byzantine faults.

Within an epoch, DT loggers can run the aggregation protocol multiple times according to some system-wide policies, e.g., at regular intervals or whenever their pools of pending request become filled up. Pipelining log aggregation with delegation setup and update improves overall system efficiency. Loggers should stop accepting new requests when the number of pending requests is estimated to exceed what they can aggregate after the cut-off time. This ensures that all requests confirmed in an epoch (Step 15, Figure 6) can be applied to the DT log by the end of the epoch.

After a successful execution of the delegation setup or update protocol in epoch $\mathtt{T}$, the owner of a zone $\mathtt{ZN}_x$ should actively monitor the DT log. Once the change to its delegation status has been admitted, it can retrieve from its designated loggers $\mathcal{L}_x$ a signed log entry $\langle \mathtt{DSum}_x, \mathtt{T}+1\rangle_{\mathcal{L}_x}$ (and $\mathtt{DSA}_x$ as well if it is updated), which will be used to generate DSPs in epoch $\mathtt{T}+1$. Each zone should retrieve its (re-)signed log entry once per epoch, even if its delegation status is not changed. Note that using the epoch counter, instead of higher-precision time units, to timestamp log requests and DSPs effectively guarantees RHINE's synchrony while reducing the system's reliance on secure global time synchronization (e.g., [40]).

Parameter Selection. Epoch length is an important system-wide parameter and its selection comes with trade-offs. A large value means long waiting time for zones' delegation status changes to take effect. A small value leads to frequent retrieval of the DT log and thereby performance issues. On balance, we suggest a practical epoch length of 48 hours and a cut-off time 24 hours before the end of an epoch. This is based on our evaluation results as well as CT's Maximum Merge Delay of 24 hours [62]—the longest time period within which CT loggers must add promised certificates to their logs. We consider doubling the waiting time in DT acceptable because the administration of zone delegation happens less frequently than the management of TLS certificates for domains in already established zones. With the suggested parameters, it takes 24–48 hours to set up an operational zone.

5.4 Authenticated Name Resolution

With only minor changes, RHINE can augment the plain name resolution of unprotected DNS (or any other distribution infrastructure) with E2E data authentication. A zone owner

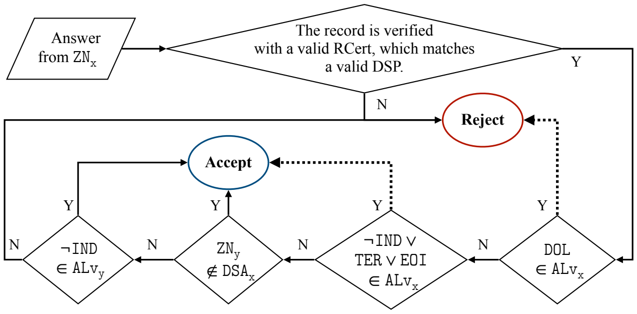

Figure 7: The flow of validating an answer received from zone $\mathtt{ZN}_x$, which has a potential child $\mathtt{ZN}_y$ that encloses the queried name. Dashed arrows indicate shortcuts for verification.

needs to sign its resource records using the zone's RCert before publishing them to nameservers. Whenever an authoritative answer is to be provided, a nameserver will also return the corresponding RCert and DSP to the querying client.

Figure 7 depicts the data validation flow. It starts with the verification of the signed records using RCert, similarly to DNSSEC. The client then additionally verifies whether the RCert matches the cryptographic digest contained in the DSP. Afterwards, the client decides whether the queried name falls within $\mathtt{ZN}_x$'s realm of authority by checking its authority level $\mathtt{ALv}_x$. In most cases (Figure 4), a shortcut can be taken to make a quick decision: an answer for a non-apex name from a delegation-only zone is always rejected by definition; an answer from a non-independent, end-of-independence, or terminating zone is always accepted because there exists no further independent subzone. If none of these applies, the client will examine $\mathtt{ZN}_x$'s potential child zone $\mathtt{ZN}_y$ that encloses the queried name, which involves verifying a membership proof from $\mathtt{DSA}_x$, and accepts the answer only if $\mathtt{ZN}_y$ does not exist or is non-independent.

6 Formal Security Analysis

The overall security goal of RHINE is to preserve a zone's data authenticity against powerful adversaries. This can be broken down into two concrete objectives: (1) preventing attackers from obtaining a valid RCert to take over a zone that is *not yet delegated*, and (2) preventing the forgery of authoritative data from an *already delegated* zone. In the first case, a victim zone is still under the legitimate control of its parent, and therefore the parent must be assumed trusted for a meaningful notion of security. In the second case, the additional protection of a zone from its malicious parent leads to the notion of authority independence.

We define these two objectives with the following theorems (presented informally). Table 3 summarizes the main security parameters in RHINE. The constraint $f < n/2$ is required by Logres [86]. We additionally require that $m \leq f+1$ holds for any zone, as otherwise an attacker with the $\mathcal{A}_3$ capability can inject arbitrary data into the DT log.

Table 3: Main Security Parameters in RHINE

	Definition	Constraint
n	Number of loggers in a global DT setup	-
f	Number of tolerable faulty loggers	$f < n/2$
m	Number of loggers chosen by a zone	$m \leq f+1$

Theorem 1 *If a zone $\mathtt{ZN}_x$ is delegated with $\mathtt{RCrt}_x$ issued and logged in epoch $\mathtt{T}$, then a corresponding secure delegation request must have been approved by the parent earlier in epoch $\mathtt{T}$, even if an $\mathcal{A}_1+\mathcal{A}_2+\mathcal{A}_3$ attacker formed by the entities specified in $\mathtt{RCrt}_x$ is present throughout epoch $\mathtt{T}$.*

The security guarantee provided by Theorem 1 resembles that of the ACME protocol, which also concerns unauthorized certificate issuance [24], but RHINE deals with much stronger attackers. In fact, RHINE allows even better security than is promised by this theorem. We discuss the following situations where the threat assumptions are violated.

It may happen that an adversarial parent, despite having approved a legitimate delegation for a child, front-runs the delegation setup protocol for the child zone with different parameters (in particular the key to be certified) in the current epoch. Yet, the expected owner of the child zone can detect from the DT log the misissued RCert, which will remain unusable until the associated DSP becomes valid in the next epoch (see Section 5.3). The owner being impersonated can then request to revoke the illegitimate delegation before it takes effect, by presenting the parent's approval as evidence to the relevant CA and loggers.

An even worse, though unlikely, case is that an attacker manages to control a CA and m loggers. This enables it to issue an RCert for a target zone and log the corresponding entry to DT. Still, the zone's real owner can actively monitor the log and take action to defeat such attack attempts.

Theorem 2 *For a DT-logged zone $\mathtt{ZN}_x$ with $\mathtt{RCrt}_x$ in epoch $\mathtt{T}$ and its delegation status not updated between $\mathtt{T}$ and $\mathtt{T}+k$ $(k > 0)$, if a client accepts an answer for a name under $\mathtt{ZN}_x$ using $\mathtt{RCrt}_y$ in epoch $\mathtt{T}+k$, then it must be that $\mathtt{RCrt}_x$=$\mathtt{RCrt}_y$, even if an $\mathcal{A}_1+\mathcal{A}_2+\mathcal{A}_3+\mathcal{A}_4$ attacker formed by the entities specified in $\mathtt{RCrt}_y$ is present between epoch $\mathtt{T}$ and $\mathtt{T}+k$.*

There are several technicalities in the definition above. First, between epoch $\mathtt{T}$ and $\mathtt{T}+k$ $(k > 0)$, $\mathtt{RCrt}_x$ is zone $\mathtt{ZN}_x$'s only valid certificate whose secure digest is logged in DT; $\mathtt{ZN}_x$ may have been delegated and updated before $\mathtt{T}$. Second, because of the hierarchical naming structure, $\mathtt{ZN}_y$ is either $\mathtt{ZN}_x$ or its ancestor, but not an arbitrary zone. Third, the attacker is defined with respect to the $\mathtt{RCrt}_y$ received by the client instead of $\mathtt{RCrt}_x$. This captures the reality that a client has no prior knowledge of an RCert's validity.

Theorem 2 formalizes data authenticity for established zones. It covers various scenarios where an attacker may acquire and use invalid, fraudulent or outdated RCerts to trick clients into accepting bogus data. RHINE maintains security

Table 4: Summary of our implementation in Go.

Component	LoC	Supporting System	Used by
`librhine`	2.2K	`gRPC`, `CBOR` [26]	Common
`rmanager`	0.6K	`BadgerDB` [4]	Zone owner
`rhine-ca`	0.68K	`BadgerDB`	CA
`dt-log`	0.76K	`BadgerDB`, `SMT` [10]	Log operator
`rserver`	0.35K	`CoreDNS` [7]	DNS operator
`rresolv`	0.7K	`SDNS` [16]	DNS operator
`rdig`	1K	`miekg/dns` [41]	End user

as long as one of the involved loggers stays non-compromised. This assumption is much weaker than that of DNSSEC, which rests on every node on the chain being honest.

Formal Verification. An informal argument or even pen-and-paper proof of these theorems can hardly lead to high assurance of RHINE's security guarantees. We have formally proved them using the Tamarin prover [68], an advanced tool for the verification of security protocols [22, 31, 42]. This approach helped us identify many subtle flaws in our early designs. We have modeled all the core protocols of RHINE, covering the secure setup and update of a zone delegation, the aggregation and retrieval of the DT log, as well as the authenticated distribution and resolution of the zone's data. This amounts to around 1500 lines of formal specification. We refer the reader to Appendix B for further details.

7 Implementation

We developed a prototype of RHINE. Table 4 summarizes our implementation efforts and the system's dependencies. The lines of code (LoC) reported do not count the supporting systems. Our prototype provides two software suites.

Offline Management. This suite includes four components. The library `librhine` defines common data structures and utilities. `rmanager` is intended to be an all-in-one toolbox for zone owners: key registration and delegation approval (in parent mode), request generation and validation (in child mode), log data retrieval, etc. `rhine-ca` offers all functions needed by a CA. `dt-agg` realizes a DT logger. It implements a self-balancing MHT for `DSA` (the per-zone accumulator) and a sparse MHT for `GDA` (the global accumulator).

These components operate in synchronous mode. For every protocol instance, they each create a `goroutine` that blocks itself after sending out a request and resumes upon receiving a response. All components can handle concurrent requests up to the available computing, memory, and bandwidth resources.

Name Resolution. We implement our nameserver (`rserver`) and recursive resolver (`rresolv`) with existing DNS frameworks. RHINE introduces new data types, RCert (encoded in the `X.509` format) and DSP. We store them using `TXT` records encoded as base64 strings and call them RoA (realm-of-authority) records. For DSP we store `DSum` and the membership proofs from `DSA` separately, as the latter are not required in most cases. Each zone has just one `DSum`, whereas the number of membership proofs equals the number of delegated child zones. Below are example records for zone `eg.com`.

```
_rcert.eg.com 60 IN TXT "Ed25519 MIIBITCB..."
_dsum.eg.com  60 IN TXT "rZXAwGzEZMBcGA1U..."
eg.com        60 IN DNSKEY 257 3 15 8fcCpq...
eg.com        60 IN RRSIG DNSKEY 15 2 60 2...
abc.eg.com    60 IN TXT "DSAPf u+EuVu6xX+..."
xyz.eg.com    60 IN TXT "DSAPf t9GsbAeavK..."
```

We do not use the private key of an RCert to directly sign regular records but instead treat this key as an equivalent of DNSSEC's key signing key (KSK). A KSK authenticates a zone signing key (ZSK), which in turn signs regular records. The record `eg.com` of type `DNSKEY` in the example above is a ZSK, followed by a `RRSIG` record authenticating it using the RCert. This layer of indirection in key usage allows a zone owner to securely store an RCert's private key offline and fetch it on demand, reducing the risk of security breaches.

We modified two built-in plugins of `CoreDNS` for `rserver` to serve RHINE-related data: the `file` plugin parses RoA records loaded from a zone file and, when processing a query, places them in the additional section of a response message; the `sign` plugin provides signing functions using RCerts.

`rresolv` augments `SDNS` with the functions to query, cache, validate and serve RoA records, reusing most of its codebase for the resolution of regular DNS records. `rresolv` always validates authoritative answers received from nameservers using RoA records and caches only verified data. It also always attaches the corresponding RoA records in the response message to a client, enabling E2E data authentication by default.

On the client side, we developed `rdig`, a `dig`-like tool for DNS-style name lookup with mandatory data validation.

8 Performance Evaluation

We evaluated our RHINE prototype in a private cloud network with 2Gbps bandwidth, using cloud servers with dedicated 8-core CPU (2.6GHz) and 16GB RAM running Ubuntu 22. Unless otherwise specified, the round-trip time (RTT) as reported by `ping` between any pair of servers is expanded to 100 ms using the `tc` utility. For the cryptographic algorithms in both RHINE and DNSSEC, we use Ed25519 for digital signatures and SHA256 for secure hash functions. In line with RHINE's architecture, the evaluation consists of two independent parts for offline and online operations.

8.1 Offline Management Performance

The first part of our evaluation aims to answer two questions. (1) *Can RHINE's offline protocols cope with real-world certificate issuance rates?* (2) *Is DT practical and scalable in terms of computation, communication, and storage cost?*

RCert Issuance. We measured the throughout of the secure delegation setup protocol, using two servers to run `rmanager`

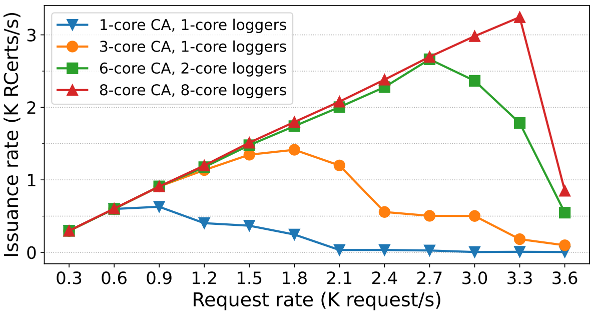

Figure 8: RCert issuance throughput.

(one for the child and the other for the parent), one server to run `rhine-ca`, and two servers to run `dt-log`. The child server generates delegation setup requests for predefined child zones whose keys have been registered at the parent server. The log maintained at the logger servers is initialized with an entry for the parent zone without any child. We limit the number of CPU cores used by the most critical CA and logger servers to understand their scaling behavior.

Figure 8 reports the results. With one core, the CA is the slowest server capping the issuance rate at around 600 RCerts per second. Using three cores, it can catch up with the loggers for a throughput of 1.4K RCerts per second. Doubling this configuration also doubles the achievable throughput. The decay in throughput with increasing request rates is due to the child server overwhelming itself with too many pending requests. Overall, our setup with these 8-core servers can issue a maximum of 3.3K RCerts per second.

To put this number in context, we consider the performance requirement of Let's Encrypt, the largest ACME-backed CA that accounts for around 80% (5M) of all daily logged CT entries [11]. Our test servers with moderate resources can already issue nearly 12M RCerts per hour. This indicates that our design can easily cope with real-life certificate issuance workloads. Such a performance is explained by RHINE's streamlined issuance process, which unlike ACME does not involve a time-consuming challenge-response procedure.

DT Consensus. We evaluate the performance-critical DT consensus process with different numbers of loggers (each on a separate server). We consider only delegation setup requests as the main consensus routine is agnostic to the type of requests. We limit the bandwidth between each pair of loggers to 1Gbps to simulate a common network setup. Each server pre-loads 50K requests into its memory before the protocol starts. With $n = 5$ and $f = 2$, it takes merely 54 seconds for the honest loggers to achieve consensus. The time increases slightly to 71 seconds with two more honest loggers. When considering one more faulty node ($n = 7, f = 3$), the consensus process finishes after 208 seconds. This trend in performance is expected as more faulty nodes mean more rounds of message exchanges in the consensus routine. Assuming the loggers run instances of the protocol consecutively with input batches of size 50K, it will take roughly 2.5 hours to process 2M requests. This meets the above-mentioned requirement for daily certificate issuance.

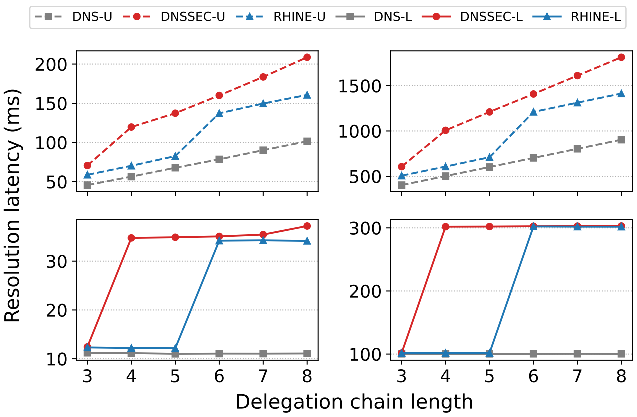

Figure 9: Upper and lower bounds of resolution latency under two network settings (left: RTT=10 ms; right: RTT=100 ms). The similar trends on the left and right plots suggest that network delay dominates the overall resolution latency.

We observe that bandwidth is the determining factor for the overall performance, as the protocol runs n consensus routines in parallel. Yet, the bottleneck in our experiment setup is the memory size of individual servers, each of which needs to cache the input from all others. With more memory, the servers can exchange larger batches of requests. We can thus expect higher performance in a production environment with more powerful hardware.

Log Size. We estimate the DT log's overall size by taking into account the `DSum`s and `DSA`s of all zones as well as the `GDA`. The size is determined by the distribution of zone delegations. The number of children of each TLD is publicly known from domain registries (e.g., 159M for `.com`) [13], but zone enumeration is required to learn the exact number of children for SLDs and further subzones. Developing an accurate view of the global DNS delegation tree is a challenging task in itself, and we leave it for future work.

Our estimation uses the statistics collected by enumerating sample zones from the Tranco list [74] and assumes an exponential decay in delegation: on average, x% of all SLDs have 4 children (and the rest of them have no child), x% of all third-level domains have 3 children, and so forth. The DT log's size is estimated to be 48GB with $x = 1$, 75GB with $x = 10$, and 779GB with $x = 50$ (which is likely an overestimation). This is only a fraction of the space requirement of CT logs, each of which can consume TBs of storage [5].

8.2 Name Resolution Performance

The second part of our evaluation investigates how E2E authentication affects name resolution performance from the perspectives of end users and naming service operators.

We compare RHINE with plain DNS and DNSSEC. For the

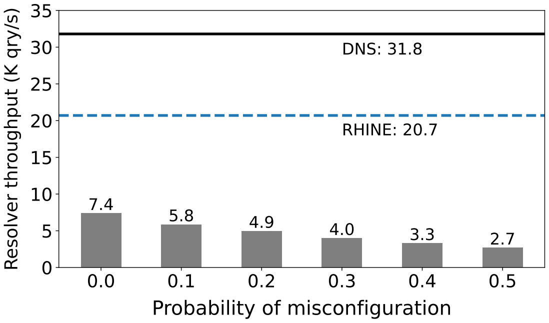

Figure 10: The resolver's query processing capacity in different systems. For DNSSEC (data shown as bars), we vary the probability of inconsistency between a zone and its children.

latter two, we use the unmodified `CoreDNS` as the nameserver and `SDNS` as the resolver. For DNSSEC, we implement a validating client and modify `SDNS` to always return a complete authentication chain. For RHINE, we consider fast validation without membership proof as this covers the vast majority of cases (Section 5.4). The resolver is preloaded with a root zone key for DNSSEC and a CA certificate that is used to sign RCerts. The experiments used synthetic zone files populated with random resource records. Each record's name contains one more label than its residing zone. All labels have a fixed length of 6, the average calculated from the Tranco list [74].

Resolution Latency. End users are sensitive to the latency of name lookup queries. We inspect the bounds of resolution latency as determined by the resolver's cache. The upper bound is obtained when a resolver iteratively queries all the relevant nameservers for a non-cached record. The lower bound is obtained when a resolver returns an answer directly from its cache. We set up eight nameservers, which host a delegation chain of zones, one client, and one resolver. We run the experiments with two network settings: one with the RTT between the cloud servers expanded to 10 ms, and the other to 100 ms. The data reported for each experiment is averaged over 100 trials. Figure 9 depicts the results.

As can be seen, RHINE constantly outperforms DNSSEC, and its performance edge comes mainly from the savings in network communication. If a UDP message carrying a DNS response exceeds the size limit (512 bytes by default [35]), the client will retry the query using TCP. Since RHINE has fewer data authenticating records than DNSSEC, retransmission is triggered less frequently. This advantage is most pronounced in case of cache hits. RHINE can sustain its negligible cost over plain DNS for longer delegation chains than DNSSEC. The performance gap will further increase should more expensive cryptographic algorithms be used. In fact, most DNSSEC-signed zones still use RSA signatures [70], which are much larger than the Ed25519 signatures used in our evaluation.

Resolver Overhead. Since augmenting regular name resolution with E2E data authentication requires the most changes to a resolver's behavior, we focus on analyzing the performance overhead of a validating resolver. Our experiments use separate cloud servers for one client (running `dnsperf` [8] as the load generator), four nameservers each hosting a level of zones from the root to third-level zones (aka subdomains, which are common in modern cloud-based web services [45]), and one resolver under test. More specifically, we generate 15 TLDs each with 8K SLDs; each SLD further delegates to one third-level zone with one `A` record for a terminal name. The client's query trace contains 480K names randomly sampled from the 120K terminal names. The resolver's cache size is set as 100K[1] and the cache is warmed up by querying all terminal names once before each experiment. With this setup, we manage to maintain a typical cache hit ratio of around 80% for client queries [30, 56]. For DNSSEC, we simulate common security mismanagement that causes inconsistencies in authentication chains and hence validation failure [54], by programming the nameservers to return incorrect `DS` records with a given probability.

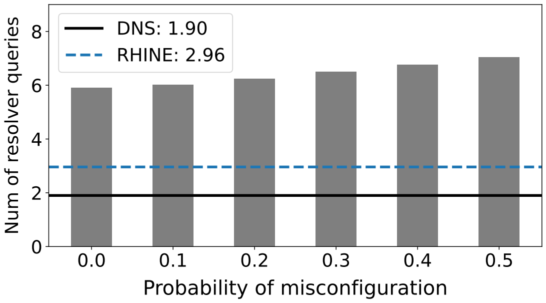

Figure 11: The average number of resolver queries per client request in case of cache miss. DNSSEC data is shown in bars.

Figure 10 reports the resolver's throughput in terms of the number of client queries processed per second. The results indicate that RHINE has a moderate impact on the resolver's processing capacity, with a reduction of 34.9% in the overall throughput. Suffering from higher costs to retrieve and validate authentication chains, DNSSEC already reduces the throughput by 76.7% with successful validation and by even wider margins as the failure rate rises.

Figure 11 reports the average number of queries that the resolver sends to authoritative nameservers in case a client query cannot be directly answered from the cache. RHINE introduces roughly one additional query per client request (35.8% increase); this is attributed to the retrieval of RoA records (Section 7). DNSSEC increases the resolver's query load by 2× even in the absence of validation failure. Such overhead is higher than expected and can be explained by the

[1] `SDNS` has several cache instances for different purposes. What matter in our experiments are the primary `PCache` for `A` and `RRSIG` records as well as the `NSCache` for `NS` and `DS` records; for RHINE we have another `RoACache`. All their sizes are set as 100K. The only exception is that for the evaluation of DNSSEC we set the size of `PCache` as 190K, in order to maintain the 80% client query cache hit ratio for fair comparison. This is because `SDNS` also stores DNSSEC's `DNSKEY` records in `PCache`. With this setup, our measurements show that for DNSSEC, half of all the attempts to look up `DNSKEY` records in `PCache` fail and thereby trigger extra queries.

contention between DNSKEY and A records in the cache.

Our measurement results suggest that a fine-grained cache design, which separates security-related records from regular records, is crucial to a validating resolver's performance.

9 Related Work

Authenticated Naming Services. Building a decentralized naming service over a peer-to-peer (P2P) network was first attempted by CoDoNS [75]. It adopts DNSSEC for data authentication and thus suffers from the same drawbacks. The GNU Name System (GNS) [14] is a modern incarnation of this idea. It allows one to define a zone using a unique key and create its own namespace rooted at the zone. However, when used for a global consistent namespace, GNS will establish a chain of trust similarly to DNSSEC with the same fragility problems. Deploying these radical systems is well recognized as a practical challenge [43]. In contrast, RHINE can be deployed on the existing DNS infrastructure.

Several projects use blockchain to design tamper-proof naming systems [9, 12, 55], but whether they can achieve the same level of performance and scalability as DNS remains open. Donovan and Feamster [38] propose to reduce the overhead of DNSSEC by letting resolvers trust each other for signed records, but this does not provide E2E authentication.

In an early position paper [39], Fetzer et al. re-purpose SSL certificates to sign DNS records, each with a separate certificate. Yet, the authors only sketched a preliminary scheme, without thoroughly exploring the challenges and large design space of authenticating DNS with the web PKI as we do.

DNS and PKI. The interplay between DNS and PKIs has a long history. DANE allows a DNS domain to certify its own certificates using DNSSEC [48]. To reduce the risk of users accepting misissued certificates, a domain can also specify the CA authorized by it using the CAA record [44]. A fundamental problem with these designs is that they move users' trust from CAs to DNS authorities. It is unlikely that the latter are more trustworthy, since they are not even in the security business as CAs are. RHINE does not simply reverse the flow of trust, i.e., relying on CAs for the authentication of name data, but rather creates a robust system where all authorities counterbalance each other's power over the namespace.

Transparency Logs. With the widespread deployment of CT, transparency logs have proven to be an integral part of modern PKIs. Many enhancements to their functionality, security, and performance have been proposed. CIRT [80] extends CT to store certificate revocation information. CTng [63] and LogPicker [36] aim to relax the trust assumptions in CT by having multiple entities (loggers or monitors) attest log entries. DT's design also follows this generic approach. F-PKI [28] introduces a map server to provide a global view of all certificates and associated policies; this role is akin to a DT logger. New authenticated data structures are proposed to improve the efficiency of transparency logs [51, 84]. They can be potentially incorporated into DT for performance improvement.

Formal Analysis of PKI. Bhargavan et al. formally model an early draft of the ACME protocol and discovered several attacks [24]. Our formal approach has also helped us identify many subtle flaws in RHINE's early designs. ARPKI [21] and DTKI [89] are PKI designs formally verified with Tamarin. Compared with them, RHINE involves more subtly interacting entities and hence is more challenging to model and verify.

10 Conclusion and Discussion

After much recent activity in the DNS space (e.g., discovery of new attacks, frequent service outages, and growing concerns about privacy), a window of opportunity is opening up for a fundamental re-design of DNS to achieve high levels of security. We revisit the existing security architecture of DNS through a modern lens, pinpointing the intrinsic limitations therein and proposing RHINE as our solution to these long-standing problems. It offloads the heavy and error-prone authentication task away from the client-facing side of DNS, enabling efficient authenticated name resolution all the way to end hosts. The deployment of RHINE can bootstrap the security of today's web PKI and Internet at large.

Deployment. We briefly discuss the incentives and costs for different entities to deploy RHINE. There is no doubt that the global Internet community shares a common interest for E2E-authenticated name resolution.

From the perspective of DNS zone owners and operators, RHINE can offer better security, lower failure rate and operational costs, and more robust naming services than DNSSEC; RHINE indeed requires fewer changes to their existing hardware and software, because it obviates the need to frequently maintain, distribute, and validate DNSSEC-style authentication chains in regular operation. The extra investments in operating the offline part of RHINE, which can be fully automated with our well-defined protocols similarly to ACME, are comparable to what is already required by the web PKI.

CAs are likely among the most enthusiastic proponents of RHINE, because the issuance of RCerts will significantly expand and consolidate their security services. The operators of transparency logs have the same motivations; DT loggers will be a select subset of CT loggers.

For end users, RHINE is easier to adopt than DNSSEC because they already rely heavily on the web PKI. For instance, the RCert-based data validation function can be easily integrated into web browsers, which have built-in support for name lookup (e.g., DoT and DoH) and certificate validation.

Acknowledgment

We would like to thank the reviewers and our shepherd, Matthew Caesar, for their valuable comments that help us substantially improve the paper. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC).

References

[1] DNSSEC Targeted in DNS Reflection, Amplification DDoS Attacks. https://community.akamai.com/, 2016.

[2] The DELEGATION_ONLY DNSKEY flag. Internet-Draft draft-ietf-dnsop-delegation-only-02, Internet Engineering Task Force, 2021.

[3] Apple's Certificate Transparency Policy. https://support.apple.com/en-us/HT205280, 2022.

[4] BadgerDB: Fast key-value DB in Go. https://github.com/dgraph-io/badger, 2022.

[5] Cert Spotter Stats. https://sslmate.com/resources/certspotter_stats, 2022.

[6] Chrome Certificate Transparency Policy. https://googlechrome.github.io/CertificateTransparency/ct_policy.html, 2022.

[7] CoreDNS: DNS and Service Discovery. https://coredns.io, 2022.

[8] DNS-OARC/dnsperf: DNS Performance Testing Tools. https://github.com/DNS-OARC/dnsperf, 2022.

[9] Ethereum Name Service. https://ens.domains, 2022.

[10] FPKI/SMT Implementation. https://github.com/netsec-ethz/fpki/tree/smt, 2022.

[11] Merkle Town. https://ct.cloudflare.com, 2022.

[12] Namecoin. https://www.namecoin.org, 2022.

[13] Registrar Stats. https://www.domainstate.com/registrar-tld-breakup.html, 2022.

[14] The GNU Name System. https://www.gnunet.org/en/gns.html, 2022.

[15] Use of DNSSEC Validation for World. https://stats.labs.apnic.net/dnssec/, 2022.

[16] Yasar Alev. SDNS: Privacy important, fast, recursive dns resolver server with dnssec support. https://github.com/semihalev/sdns, 2022.

[17] Eihal Alowaisheq, Siyuan Tang, Zhihao Wang, Fatemah Alharbi, Xiaojing Liao, and XiaoFeng Wang. Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[18] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFCs 6014, 6840.

[19] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840, 6944, 9077.

[20] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. Automatic Certificate Management Environment (ACME). RFC 8555 (Proposed Standard), March 2019.

[21] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2017.

[22] David Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV Standard: Break, Fix, Verify. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[23] Daniel J. Bernstein. DNSCurve: Usable Security for DNS. https://dnscurve.org.

[24] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Nadim Kobeissi. Formal Modeling and Verification for Domain Validation and ACME. In *Processings of the International Conference on Financial Cryptography and Data Security (FC)*, 2017.

[25] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2018.

[26] C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). RFC 8949 (Internet Standard), December 2020.

[27] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. Domain Validation++ For MitM-Resilient PKI. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2018.

[28] Laurent Chuat, Cyrill Krähenbühl, Prateek Mittal, and Adrian Perrig. F-PKI: Enabling Innovation and Trust Flexibility in the HTTPS Public-Key Infrastructure. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2022.

[29] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *Proceedings of the USENIX Security Symposium*, 2017.

[30] Secure64 Software Corporation. Lies, Damn Lies and DNS Performance Statistics. White paper, 2017.

[31] Cas Cremers, Jaiden Fairoze, Benjamin Kiesl, and Aurora Naska. Clone Detection in Secure Messaging: Improving Post-Compromise Security in Practice. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[32] David Dagon, Chris Lee, Wenke Lee, and Niels Provos. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2008.

[33] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. From IP to Transport and beyond: Cross-Layer Attacks against Applications. In *Proceedings of the ACM SIGCOMM Conference*, 2021.

[34] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives. In *Topics in Cryptology — CT-RSA 2015*, 2015.

[35] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. DNS Transport over TCP - Implementation Requirements. RFC 7766 (Proposed Standard), March 2016. Updated by RFCs 8490, 9103.

[36] Alexandra Dirksen, David Klein, Robert Michael, Tilman Stehr, Konrad Rieck, and Martin Johns. Log-Picker: Strengthening Certificate Transparency Against Covert Adversaries. *Proceedings on Privacy Enhancing Technologies*, 2021(4):184–202, 2021.

[37] D. Dolev and A. Yao. On The Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[38] Sean Donovan and Nick Feamster. Alternative Trust Sources: Reducing DNSSEC Signature Verification Operations with TLS. *ACM SIGCOMM Computer Communication Review*, 45(4):353–354, 2015.

[39] Christof Fetzer, Gert Pfeifer, and Trevor Jim. Enhancing DNS security using the SSL trust infrastructure. In *Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, 2005.

[40] Marc Frei, Jonghoon Kwon, Seyedali Tabaeiaghdaei, Marc Wyss, Christoph Lenzen, and Adrian Perrig. G-sinc: Global synchronization infrastructure for network clocks. In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS)*, 2022.

[41] Miek Gieben. DNS library in Go. https://github.com/miekg/dns, 2022.

[42] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David A. Basin. A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols. In Srdjan Capkun and Franziska Roesner, editors, *Proceedings of the USENIX Security Symposium*, 2020.

[43] Christian Grothoff, Matthias Wachs, Monika Ermert, and Jacob Appelbaum. Toward Secure Name Resolution on The Internet. *Computers & Security*, 77:694–708, 2018.

[44] P. Hallam-Baker and R. Stradling. DNS Certification Authority Authorization (CAA) Resource Record. RFC 6844 (Proposed Standard), January 2013. Obsoleted by RFC 8659.

[45] Shuai Hao, Haining Wang, Angelos Stavrou, and Evgenia Smirni. On the DNS Deployment of Modern Web Services. In *Proceedings of the IEEE Conference on Network Protocols (ICNP)*, 2015.

[46] Muks Hirani, Sarah Jones, and Ben Read. Global DNS Hijacking Campaign: DNS Record Manipulation at Scale. https://www.fireeye.com/blog/threat-research/, 2019.

[47] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484 (Proposed Standard), October 2018.

[48] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012. Updated by RFCs 7218, 7671, 8749.

[49] Hans Hoogstraaten, Ronald Prins, Daniël Niggebrugge, Danny Heppener, Frank Groenewegen, Janna Wettink, Kevin Strooy, Pascal Arends, Paul Pols, Robbert Kouprie, Steffen Moorrees, Xander van Pelt, and Yun Zheng Hu. Black Tulip: Report of the Investigation into the DigiNotar Certificate Authority Breach. Technical report, August 2012.

[50] Rebekah Houser, Shuai Hao, Zhou Li, Daiping Liu, Chase Cotton, and Haining Wang. A Comprehensive Measurement-based Investigation of DNS Hijacking. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*, 2021.

[51] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. Merkle^2: A Low-Latency Transparency Log System. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[52] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858 (Proposed Standard), May 2016. Updated by RFC 8310.

[53] Geoff Huston. Measuring DNSSEC Performance. https://labs.apnic.net, 2013.

[54] IANIX. Major DNSSEC Outages and Validation Failures. https://ianix.com/pub/dnssec-outages.html, 2021.

[55] Lin Jin, Shuai Hao, Yan Huang, Haining Wang, and Chase Cotton. DNSonChain: Delegating Privacy-Preserved DNS Resolution to Blockchain. In *Proceedings of the IEEE Conference on Network Protocols (ICNP)*, 2021.

[56] Jaeyeon Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and The Effectiveness of Caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603, 2002.

[57] Dan Kaminsky. It's the end of the cache as we know it. Presented at Black Hat USA, 2008.

[58] Aqsa Kashaf, Vyas Sekar, and Yuvraj Agarwal. Analyzing Third Party Service Dependencies in Modern Web Services: Have We Learned from the Mirai-Dyn Incident? In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2020.

[59] Cyrill Krähenbühl, Seyedali Tabaeiaghdaei, Christelle Gloor, Jonghoon Kwon, Adrian Perrig, David Hausheer, and Dominik Roos. Deployment and scalability of an inter-domain multi-path routing infrastructure. In *Proceedings of the International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2021.

[60] Jonghoon Kwon, Juan A. García-Pardo, Markus Legner, François Wirz, Matthias Frei, David Hausheer, and Adrian Perrig. SCIONLab: A next-generation Internet testbed. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*.

[61] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962 (Experimental), June 2013. Obsoleted by RFC 9162.

[62] B. Laurie, E. Messeri, and R. Stradling. Certificate Transparency Version 2.0. RFC 9162 (Experimental), December 2021.

[63] Hemi Leibowitz, Haitham Ghalwash, Ewa Syta, and Amir Herzberg. CTng: Secure Certificate and Revocation Transparency. Cryptology ePrint Archive, Paper 2021/818, 2021.

[64] Wilson Lian, Eric Rescorla, Hovav Shacham, and Stefan Savage. Measuring the Practical Impact of DNSSEC Deployment. In *Proceedings of the USENIX Security Symposium*, 2013.

[65] Baojun Liu, Chaoyi Lu, Haixin Duan, Ying Liu, Zhou Li, Shuang Hao, and Min Yang. Who Is Answering My Queries: Understanding and Characterizing Interception of the DNS Resolution Path. In *Proceedings of the USENIX Security Symposium*, 2018.

[66] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[67] Keyu Man, Xin'an Zhou, and Zhiyun Qian. DNS Cache Poisoning Attack: Resurrections with Side Channels. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2021.

[68] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 2013.

[69] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Proceedings of Advances in Cryptology (CRYPTO)*, 1988.

[70] Moritz Müller, Willem Toorop, Taejoong Chung, Jelte Jansen, and Roland van Rijswijk-Deij. The Reality of Algorithm Agility: Studying the DNSSEC Algorithm Life-Cycle. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2020.

[71] Jeman Park, Aminollah Khormali, Manar Mohaisen, and Aziz Mohaisen. Where Are You Taking Me? Behavioral Analysis of Open DNS Resolvers. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[72] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. Global Measurement of DNS Manipulation. In *Proceedings of the USENIX Security Symposium*, 2017.

[73] Adrian Perrig, Peter Müller, Samuel Hitz, David Hausheer, David Basin, Markus Legner, and Laurent Chuat. *The Complete Guide to SCION*. Springer, 2022.

[74] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2019.

[75] Venugopalan Ramasubramanian and Emin Gün Sirer. The Design and Implementation of a next Generation Name Service for the Internet. In *Proceedings of the ACM SIGCOMM Conference*, 2004.

[76] Venugopalan Ramasubramanian and Emin Gün Sirer. Perils of transitive trust in the domain name system. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2005.

[77] Audrey Randall, Enze Liu, Gautam Akiwate, Geoffrey M Voelker, Stefan Savage, and Aaron Schulman. Home is Where the Hijacking is: Understanding DNS Interception by Residential Routers. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2021.

[78] rhine-team. `https://github.com/rhine-team/RHINE-Prototype`, 2022.

[79] Jeremy Rowley. CT2 Log Compromised via Salt Vulnerability. `https://groups.google.com/a/chromium.org/g/ct-policy/c/aKNbZuJzwfM`, 2020.

[80] Mark Dermot Ryan. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2014.

[81] Lorenz Schwittmann, Matthäus Wander, and Torben Weis. Domain Impersonation is Feasible: A Study of CA Domain Validation Vulnerabilities. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[82] Haya Shulman and Michael Waidner. One Key to Sign Them All Considered Vulnerable: Evaluation of DNSSEC in the Internet. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[83] Thomas and Erin. Against DNSSEC. `https://sockpuppet.org/blog/2015/01/15/against-dnssec/`, 2015.

[84] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. Transparency Logs via Append-Only Authenticated Dictionaries. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[85] Thomas Vissers, Timothy Barron, Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The Wolf of Name Street: Hijacking Domains Through Their Nameservers. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[86] Joel Wanner, Laurent Chuat, and Adrian Perrig. A Formally Verified Protocol for Log Replication with Byzantine Fault Tolerance. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, 2020.

[87] Nicholas Weaver, Christian Kreibich, and Vern Paxson. Redirecting DNS for Ads and Profit. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2011.

[88] B. Wellington and O. Gudmundsson. Redefinition of DNS Authenticated Data (AD) bit. RFC 3655 (Proposed Standard), November 2003. Obsoleted by RFCs 4033, 4034, 4035.

[89] Jiangshan Yu, Vincent Cheval, and Mark Ryan. DTKI: A New Formalized PKI with Verifiable Trusted Parties. *The Computer Journal*, 59(11):1695–1713, 2016.

[90] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices. In *Proceedings of the USENIX Security Symposium*, 2020.

```
0. $X$ (the owner of $\mathtt{ZN}_x$) publishes $\mathtt{RCrt}_x$, $\mathtt{DSum}_x$, $\mathtt{DSA}_x$
   to the infrastructure $D$ to be queried by client $U$.
1. $U \rightarrow D$ : $\mathtt{QUERY}(\mathtt{qname}, \mathtt{qtype})$
2. $D$      : $rec := \langle \mathtt{RRset}(\mathtt{qname}, \mathtt{qtype}) \rangle_x$
            : $dsp := \langle \mathtt{DSum}_x, T \rangle_{\mathcal{L}_x}$
   // ALv is encoded with four bits: DOL, IND, TER, EOI
   // The only case requiring a membership proof
            : If $\mathtt{ALv}_x$ = (IND, ¬DOL, ¬TER, ¬EOI):
   // $\mathtt{ZN}_y$ is a (potential) child zone enclosing qname
            :    $\mathtt{ZN}_y := \mathtt{GetChild}(\mathtt{ZN}_x, \mathtt{qname})$
   // Get the membership proof for $\mathtt{ZN}_y$'s (non-)existence
            :    $mem := \mathtt{Acc.GenPf}(\mathtt{DAcc}_x, \mathtt{ZN}_y)$
            :    $dsp := dsp \cup \{mem\}$
3. $D \rightarrow U$ : $\mathtt{ANSWER}(rec, \mathtt{RCrt}_x, dsp)$
   // $\mathtt{ZN}_x$ is extracted from $\mathtt{RCrt}_x$
4. $U$      : Verify $\mathtt{ZN}_x$ encloses qname
            :      $\wedge\ \Sigma.\mathtt{Vf}(\mathtt{RCrt}_x, rec)\ \wedge\ \Sigma.\mathtt{Vf}(pk_{\mathcal{L}_x}, dsp)$
            :      $\wedge$ $T$ is the current epoch
            :      $\wedge$ $\mathtt{ZN}_x$ is not revoked (from $\mathtt{Aux}_x$)
            :      $\wedge\ \mathtt{Match}(\mathtt{DSum}_x, \mathtt{RCrt}_x)$
            : // Short-cut cases:
            : If DOL in $\mathtt{ALv}_x$ and qname $\neq \mathtt{Apex}(\mathtt{ZN}_x)$:
            :    Reject the answer
            : Else if ¬IND or TER or EOI in $\mathtt{ALv}_x$:
            :    Accept the answer
            : Else : // Otherwise, check $\mathtt{ZN}_y$'s status
            :    If $mem$ is for absence :
            :       Accept if $\mathtt{Acc.Vf}(\mathtt{DAcc}_x, mem)$
            :       Else : // $\mathtt{ZN}_y$ already delegated
            :          Verify $\mathtt{Acc.Vf}(\mathtt{DAcc}_x, mem)$
            :          Accept the answer if ¬IND in $\mathtt{ALv}_y$
```

Figure 12: Authenticated name resolution protocol.

A Protocol Specifications

A.1 Authenticated Name Resolution

RHINE's name resolution protocol is presented in Figure 12. It is agnostic to the underlying distribution infrastructure D, which can be instantiated, for example, with the existing DNS infrastructure (consisting of authoritative nameservers, recursive resolvers, forwarders, etc.) or a P2P network such as GNUnet [14]. RHINE mandates that each answer to a client query contains, in addition to the authoritative resource records, the RCert and DSP of the zone that claims the authority, and that a client always validates answers by itself, thereby enforcing E2E authentication.

A.2 Secure Delegation Update

Figure 14 describes RHINE's secure delegation update protocol. An established zone uses its own RCert and the associated DSP to prove its realm of authority. The protocol's overall flow resembles the delegation setup process, except that the parent's involvement is needed only in a few cases.

```
0. Each $L_i \in \mathcal{G}$ has a set $\mathcal{X}_i$ of requests ($lreq$) as input.
1. Run $n$ rounds of $O_i := \mathtt{LogresConsensus+}(L_i, \mathcal{X}_i)$
   with each $L_i$ as the leader proposing input data in a
   round. After that, all loggers obtain the same set $O$
   $:= \bigcup_{i=1}^{n} O_i$ as output.
2. Each $L_i$ filters the requests in $O$ by keeping only the
   earliest one in case of conflicts, applies the resulting
   operations to its local $\mathtt{GDA}_T$, and computes a new
   digest $\mathtt{GAcc}_{T+1}$. Broadcast $\langle \mathtt{GAcc}_{T+1} \rangle_{L_i}$ to $\mathcal{L}$
3. Each $L_i$ accepts and finalizes the aggregation result
   if it receives $f$ valid signatures over $\mathtt{GAcc}_{T+1}$.
```

Figure 13: DT aggregation protocol.

A zone can freely update its RCert and DT entry within the validity period of the delegation. The parameter to update must not include a delegation expiration time beyond what is currently specified in the zone's DSum. To extend the validity period before the delegation expires, a zone must negotiate with its parent (e.g., renewing the business contract) and get the latter's approval. Another type of update that requires the parent's consent is changing a non-independent zone to an independent one, as this affects the parent's realm of authority. The update of authority level is subject to additional restrictions. For example, a zone cannot change itself to terminating unless all its existing delegations become invalid (expired or revoked); similarly, a zone cannot change itself to end-of-independence if it still has any independent child. A zone can only update its delegation status (except its DAcc, which is affected by the changes to its subzones' delegation status) once per epoch. For ease of presentation, we abstract away these checks of an update request's legitimacy in the protocol specification.

It is possible for a zone to update the CA and loggers it relies on, which is important after security breaches of these trusted entities. In this case, the zone will run the update protocol with a set of new trusted entities.

The revocation of a secure delegation comes in two forms. Since RHINE mandates one RCert per zone at any time, the issuance of a new RCert for a zone implicitly revokes the old one. A zone can also make an explicit revocation request through the update protocol. This will fail the validation of the zone's data signed with its current RCert. The operation is irreversible, meaning that a revoked zone can only be re-established through the secure delegation setup protocol.

One caveat in enforcing one RCert per zone is that the loss of a zone's private key may lock up the zone until the existing delegation expires. This conundrum can be addressed by having a zone pre-generate a signed revocation request, preferably immediately after the delegation setup. The zone

```
0.  $C$ prepares a data object $\texttt{Upd}$ that encodes the parameters to be updated.
    // $A$ and $\mathcal{L}_c$ may be different from those in $\texttt{RCrt}_c$
1.  $C$ : Select $A$ and $\mathcal{L}_c$
    // $rid$ is implicitly included in all subsequent messages
      : $rid := \texttt{H}(t_0, \texttt{ZN}_c, \texttt{Upd}, A, \mathcal{L}_c)$
      : $sur := \langle rid, \texttt{SUReq}(\texttt{ZN}_c, \texttt{Upd}, A, \mathcal{L}_c)\rangle_c$
      : [Get $apv := \langle \texttt{SUApprvl}(\texttt{H}(sur))\rangle_p$
      :     and $\texttt{RCrt}_p$ from the parent $P$].
2.  $C \rightarrow A$ : $sur$, $\texttt{RCrt}_c$, [$apv$, $\texttt{RCrt}_p$]
3.  $A$ : Retrieve $dsp_c$ [, $dsp_p$] from $\mathcal{L}_c$
      : Verify $sur$ with $\texttt{RCrt}_c$ and $dsp_c$
      :     [, $apv$ with $\texttt{RCrt}_p$ and $dsp_p$]
      : Verify the validity of $\texttt{Upd}$
      :     [, create a new $tbsrc$ according to $\texttt{Upd}$ ]
      : $prl := \langle \texttt{PreLog}(sur, [apv, tbsrc])\rangle_A$
4.  $A \rightarrow \mathcal{L}_c$ : $prl$, $\texttt{RCrt}_c$, [$apv$, $\texttt{RCrt}_p$]
5.  $L_i$ : Verify $prl$, $sur$, [and $apv$,] using the
      :     corresponding certificates and the local log
      : Verify the validity of $\texttt{Upd}$
      : $uds := (\texttt{T}, A, \mathcal{L}_c, \texttt{ZN}_c, \texttt{Upd}, [\texttt{H}(tbsrc)])$
6.  $L_i \rightarrow A$ : $att_i := \langle \texttt{LogAttest}(L_i, \texttt{H}(uds))\rangle_{L_i}$
7.  $A$ : Verify $\{att_i\}$ against $prl$
    // $L$ is randomly selected from $\mathcal{L}_c$ by $A$
8.  $A \rightarrow L$ : $lreq := \langle \texttt{LogReq}(L, uds, \{att_i\}_{L_i \in \mathcal{L}_c})\rangle_A$
9.  $L$ : Verify $lreq$ and $\{att_i\}$ and their consistency
      : Add $lreq$ to the pending pool for aggregation
10. $L \rightarrow A$ : $lc := \langle \texttt{LogCfm}(L, \texttt{H}(uds))\rangle_L$
11. $A$ : Verify $lc$ against $lreq$
      :     [, $\texttt{RCrt}'_c := \langle \texttt{FinalRCert}(\texttt{TbsRC}_c, \mathcal{L}_c)\rangle_A$]
12. $A \rightarrow C$ : $prl$, $\{att_i\}_{L_i \in \mathcal{L}_c}$, $lc$, [$\texttt{RCrt}'_c$]
13. $C$ : Verify all received data against $sur$
```

Figure 14: Secure delegation update protocol (simplified). Messages and operations in square brackets [m] are optional.

can then revoke the existing delegation in case of key loss. Clearly, the pre-generated revocation request itself should be stored separately from the private key in a secure place.

A.3 DT Aggregation with Modified Logres

Figure 13 presents the DT aggregation protocol, with the core modified Logres consensus routine depicted in Figure 15. Logres obviates the need for leader selection by having each participating node lead and run an instance of the consensus routine in parallel with all others. Each consensus instance contains up to $f+1$ rounds of message exchanges among the nodes, where f is the number of Byzantine faulty nodes tolerable by the system. In normal situations where the leader is honest and correctly operates, the consensus routine will terminate in just two rounds.

Our main modification of Logres is in lines 13–17 of Figure 15, which describes the additional data validation required by RHINE. Only valid input values will be added to the output set. Another important change is that we refine the algorithm to allow taking sets of values as input, as the original version abstracts the input data as a single value. This entails several technicalities including whether to have nodes' witness on valid values that may only constitute a subset of the input. For simplicity, we always treat the witnesses on the original input set even it may contain invalid data. This results in redundant data validation in each round. Optimizing performance in this regard is an interesting future work.

The final output set O from the consensus process may contain duplicate or conflicting operations as a result of attacks (when RHINE's threat assumptions are violated; see Section 6) or operational faults. For example, two requests may contain different parameters to create a new log entry for a just delegated zone. This is possible because there is a delay for the DT log to be synchronized across loggers. In such situations, the loggers will keep the earliest operation and discarding other conflicting operations for the zone in question; any potential attacks and faults will become detectable once the current execution of the aggregation protocol ends.

B Formal Verification of RHINE

This section introduces important aspects of our formal specification of RHINE and the security properties we verify. We refer the reader to the project repository [78] for full details.

Abstractions. The formalization of non-trivial protocols using Tamarin can run into the state explosion problem, which makes the analysis intractable. To this end, we use several abstractions to reduce the complexity of our model while still faithfully capturing the essence of RHINE protocols.

First, rather than modeling the entire namespace, we focus on a few zones represented symbolically that suffice to describe generic zone delegation. This includes a parent zone that can be malicious, a child zone in question (i.e., $\texttt{ZN}_x$ in Theorem 1 and 2), and another child zone (of the same parent) serving to validate our model's correctness. We consider all of them to be independent zones for meaningful a security analysis. This also obviates the need to model the processing of authority level.

For the time dimension, we model three epochs. In `T0`, the pre-established parent zone can publish data for name resolution and approve delegation requests. Only one child zone can be delegated in `T0` and the other in `T1`; the first child zone can also be updated in `T1`. Zone delegation or update is not permitted in the last epoch `T2`. These symbolic epochs are intended to enforce the sequence of events and they are not necessarily consecutive. This arrangement allows the model to capture security threats throughout a zone's life cycle.

For the DT aggregation process, we model only RHINE-specific input validation without specifying the Logres con-

```
This is one of the $|\mathcal{L}|$ parallel runs of the consensus
process with $L_i$ being the leader. The code is for $L_j$.
If $j = i$:
  broadcast $\langle \mathcal{X}_i, L_i \rangle_{L_i}$ to all other loggers
  return $O_i := \mathcal{X}_i$
Else:
  $\mathcal{W} := \emptyset$ // witnessed values
  $O_i := \emptyset$ // agreed-upon output
// Start $f+1$ rounds of message exchanging
  For round $r = 1, \dots, f+1$ :
// A timeout is needed to stop waiting for the messages from faulty nodes.
    $\mathcal{M}$ := received messages
    $\mathcal{V} := \emptyset$ // valid input values
    For $\langle \mathcal{X}, L_i \rangle_{\mathcal{L}_W} \in \mathcal{M}$
// Logres-specific checking
      If $|\mathcal{L}_W| \geq r \wedge \Sigma.\mathtt{Vf}(pk_{\mathcal{L}_w}, \langle \mathcal{X}, L_i \rangle_{\mathcal{L}_W})$
        $\mathcal{Y} := \emptyset$
// RHINE-specific data validation. $A$, $\mathcal{L}_c$ are from $op$
        For $lreq \in \mathcal{X}$ :
          $lreq := \langle \mathtt{LogReq}(L, op, attset) \rangle_A$
          If $\Sigma.\mathtt{Vf}(pk_A, lreq) \wedge L = L_i$
              $\wedge\, \Sigma.\mathtt{Vf}(pk_{\mathcal{L}_c}, attset)$
              $\wedge\, \mathtt{Match}(op, attset)$
            $\mathcal{Y} := \mathcal{Y} \cup \{lreq\}$
        $\mathcal{V} := \mathcal{V} \cup \{\mathcal{Y}\}$
// Continue Logres processing on valid RHINE input
    If $\mathcal{V} \setminus \mathcal{W} \neq \emptyset$:
      If $|\mathcal{V} \cup \mathcal{W}| = 1$ :
        $O_i := \mathcal{V}$ // Only one element in $\mathcal{V}$
      Else :
        $O_i := \emptyset$ // No agreed-upon value yet
      $\mathcal{N} := \emptyset$ // Messages for next round
// Add witness to the original input
      For $\mathcal{Y} \in \mathcal{V} \setminus \mathcal{W}$ :
        Find $\langle \mathcal{X}, L_i \rangle_{\mathcal{L}_W} \in \mathcal{M}$ s.t. $\mathcal{Y} \subset \mathcal{X}$
        $\mathcal{N} := \mathcal{N} \cup \{\langle \mathcal{X}, L_i \rangle_{\mathcal{L}_W \cup \{L_j\}}\}$
      multicast $\mathcal{N}$ to all non-leader loggers
// Update witnessed value
    $\mathcal{W} := \mathcal{V}$
// In the end of this protocol run, all honest loggers
// will have the same $O_i$, which contain all valid log
// operations as a subset of $\mathcal{X}_i$.
  return $O_i$
```

Figure 15: The LogresConsensus+$(L_i, \mathcal{X}_i)$ protocol.

sensus routine, as its security has been formally verified by its authors [86] and is inherited by our enhanced version. For the update protocol, we consider updating the certified key as well as the designated CA and loggers without the need for parental approval. This reflects a zone owner's ability to independently manage the zone's security.

We refrain from modeling explicit servers and the name resolution algorithm, as this would result in a overcomplicated model. Instead, we create an abstract distribution infrastructure by taking advantage of Tamarin's underlying pattern matching mechanism. Moreover, we represent all data structures (RCerts, DT log, resource records, etc.) as sets of values and omit non-essential data such as `Aux`.

B.1 Protocol Specification

Tamarin models a security protocol as a labeled transition system (LTS) where a state of the LTS consists of the local states of the protocol participants, the adversary's knowledge, and messages on the network. States are modeled as a finite multiset of *facts*. The system's dynamics are specified by labeled multiset rewriting rules that transform the facts.

Protocol Roles. Our model introduces five roles: `P` is the owner of an established parent zone, `C` is an entity wishing to securely establish a child zone, `CA` is an RCert issuer, `L` is a DT logger, and `U` is an end user trying to resolve a name under a child zone. The state space of a protocol in a symbolic Tamarin model generally grows exponentially with the number of involved actors, which are instances of roles in interleaved protocol sessions. We model `P` as a singleton that is instantiated only once. No limitation is imposed on other roles. RHINE allows a zone to choose the number m of relying loggers. We set $m = 2$ for all zones in the model. This keeps the complexity of verification manageable without weakening the security properties we verify. The other parameters f and n are irrelevant in the model because of the simplified aggregation process.

Adversary. Tamarin provides a built-in network model with a Dolev-Yao adversary: any outbound message is added to the adversary's permanent knowledge; any inbound message is constructed by the adversary from its knowledge. We leverage this feature to create an adversary-controlled distribution infrastructure without any explicit servers: publishing a zone simply means sending its signed records to the network, and name resolution is realized by sending a query to and receiving the matching record (and associated RCert and DSP) from the network. To model the compromise of an entity, we reveal its private key to the network, which enables the adversary to impersonate the entity by forging its signatures. We also allow malicious child zone owners so that the adversary's capability is not limited by the model itself.

Protocol Rules. We use several example rules to explain our modeling style and choices. Figure 16 lists two rules modeling the CA's processing of an initial request in the secure delegation protocol. A rule is defined in the form of

```
[state facts] --[event facts]-> [state facts].
  (premise)                       (conclusion)
```

The lines in a `let ... in` block defines macros that are expanded in the respective rule.

```
rule CA_Preissuance_1:
  let
    sdr_data = <'SDReq', epoch, zone, $C,
                zpkC, $P, $CA, $L1, $L2>
    sdr = <sdr_data, sig>
    apv_data = <'SDApproval', h(sdr)>
  in
  [
    In(<$C, $CA, sdr, apv, rcP>)
  , !CA_St_0($CA, ~skCA)
  , !ZPk_P($P, zpkP)
  , Fr(~dsrid)
  ]
  --[
      NotEq($CA, $L1)
    , NotEq($CA, $L2)
    , NotEq($L1, $L2)
    , Eq(verify(apv, apv_data, zpkP), true)
    , Eq(verify(sig, sdr_data, zpkC), true)
    ]->
  [
    CA_St_1($CA, ~skCA, sdr, apv, rcP,
            ~dsrid, epoch)
  , DSPReq(~dsrid, epoch, $CA,
           zone('Parent'), $L1, $L2) ]
```

```
rule CA_Preissuance_2:
  let
    sdr_data = <'SDReq', epoch, zone, $C,
                zpkC, $P, $CA, $L1, $L2>
    sdr = <sdr_data, sdr_sig>
    rcP = <'RCert', <tbsP, $L1_P, $L2_P>, rcP_sig>
    dsum_P = <'DSum', zone('Parent'), htbsP,
              <'Delegations', dlgt1, dlgt2>>
    dsp_P = <'DSP', epoch, dsum_P, dsp_sig1, dsp_sig2>
    tbsrc = <'TBSCert', zone, $C, zpkC, $CA>
    prl_data = <'PreLog', sdr, apv, rcP, tbsrc>
    prl = <prl_data, sign(prl_data, ~skCA)>
  in
  [ DSPResp(~dsrid, $L1, $L2, $CA, dsp_P)
  , CA_St_1($CA, ~skCA, sdr, apv, rcP, ~dsrid, epoch)
  , !Pk($L1, pkL1), !Pk($L2, pkL2)
  ]
  --[ Eq(verify(dsp_sig1, <dsum_P, epoch>, pkL1), true)
    , Eq(verify(dsp_sig2, <dsum_P, epoch>, pkL2), true)
    , Eq(htbsP, h(tbsP))
    , NotEq(dlgt1, zone), NotEq(dlgt2, zone)
    , CAPreissued(epoch, $P, $C, zpkC, $CA, $L1, $L2)
    ]->
  [ CA_St_2($CA, ~skCA, sdr, tbsrc)
  , Out(<$CA, $L1, $L2, prl>) ]
```

Figure 16: Two rules from our model describing the CA's actions in Step 7, Figure 6

A fact `F(t1, t2, ...)` involves symbolic terms `t1, t2`, ... that contain variables, constants, functions, network messages, etc. The execution of a rule consumes facts in the LTS's current state that match the rule's premise, and produces new facts that are added to the state. Persistent facts of the form `!F(t1, t2, ...)` are never removed from the state, once added. A public variable `$t` (often used to identify an actor) or a constant `'t'` is always known to the adversary. A fresh variable `~t` is typically used to model random numbers such as keys. We use several Tamarin's built-in functions, including pair (`<t1, t2>`), hashing (`h(t)`), and signing (`sign(t1, t2)` and `verify(t1, t2, t3)`). We also defined our own functions including `zone(t)`, `name(t)`, and `epoch(t)`. Although we use them to simply record constants in our current model, it is possible to introduce equational theories for them to capture a hierarchical naming structure and unlimited epoch transition.

The rule `CA_Preissuance_1` models `$CA` receiving a secure delegation request from `$C` over the insecure network using Tamarin's built-in `In()` fact. Facts `!CA_St_0()`, `CA_St_1()` record `$CA`'s local state. `!ZPk_P()` models the access to the parent zone's public key. The event facts `Eq()` and `NotEq()` specify equality and inequality checks using Tamarin's *restriction* mechanism. We apply them to model the bulk of an actor's local processing of a message, including signature verification and consistency checking. According to the protocol specification (Figure 6), the CA needs to retrieve the parent zone's DSP from the designated loggers over an out-of-band secure channel. The facts `DSPReq()` and `DSPResp()` model such a channel. At the end of the rule `CA_Preissuance_1`, the CA makes a retrieval request with a random id generated using the built-in fact `Fr()`.

In the rule `CA_Preissuance_2`, the CA continues to verify the received DSP and send out a pre-logging message over the insecure network using the built-in `Out()` fact. Event facts such as `CAPreissued()` there facilitate the definition of properties in a model-independent way.

One of the most important event facts we consider is `ZoneDelegated()`, which signifies the successful establishment of a child zone. It should not be placed at the last step of the secure delegation protocol, but where the zone owner has verified the updated DT log (within the distribution window of an epoch). Our model precisely captures this consideration in the rule `Child_Accept_T0` shown in Figure 17.

A zone delegated in an epoch can publish its data in the subsequent epochs. To model this, we introduce a linear fact `ZonePublishable()` that allows a zone to publish at most once in an epoch. The rule `Child_Accept_T0` states that a zone delegated in `T0` can publish once in `T1` and once in `T2`. The parent zone is initialized in `T0` and so it can publish in all three epochs.

The reason why we model three epochs instead of two is to cover the scenario where an attacker attempts to acquire an RCert in `T1` for a zone delegated in `T0`. Such an attack sce-

```
rule Child_Accept_T0:
    let // The following are macros used to improve the specification’s readability
      sdr_data = <’SDReq’, epoch(’T0’), zone, $C, zpkC, $P, $CA, $L1, $L2>
      sdr = <sdr_data, sdr_sig>
      tbsrc = <’TBSCert’, zone, $C, zpkC, $CA>
      rcert_data = <tbsrc, $L1, $L2>
      rcert = <’RCert’, rcert_data, rcert_sig>
      lcfm_data = <’LogCfm’, $L1, hnds>
      lcfm = <lcfm_data, lcfm_sig>
      nds = <epoch(’T0’), $CA, $L1, $L2, zone, h(tbsrc)>
    in
    [ C_St_2($C, ~zskC, sdr)
    , In(<$CA, $C, rcert, att1, att2, lcfm>)
    , !Pk($CA, pkCA), !Pk($L1, pkL1), !Pk($L2, pkL2)
    , DTMonitor(epoch(’T0’), ’Setup’, logged_htbs) // Monitor the updated DT log
    ]
  --[ Eq(verify(rcert_sig, rcert_data, pkCA), true) // The cert is issued by the designated CA
    , Eq(verify(att1, <’LogAttest’, h($L1, nds)>, pkL1), true) // and attested by the loggers
    , Eq(verify(att2, <’LogAttest’, h($L2, nds)>, pkL2), true)
    , Eq(verify(lcfm_sig, lcfm_data, pkL1), true) // The logging operation is confirmed
    , Eq(h(nds), hnds)                            // and matches the previous logging request
    , Eq(h(tbsrc), logged_htbs) // The monitored log entry is correct
    , ZoneDelegated(epoch(’T0’), zone, $P, $C, ~zskC, $CA, $L1, $L2) // Successful delegation event
    ]->
    [ ZonePublishable(epoch(’T1’), zone, $C, ~zskC, rcert)
    , ZonePublishable(epoch(’T2’), zone, $C, ~zskC, rcert) ]
```

Figure 17: A rule modeling the child zone owner’s acceptance of an RCert in epoch `T0`.

nario is different from acquiring an RCert for a non-existent child zone of an existing zone. The former case is captured by Theorem 2 and the latter case by Theorem 1.

As mentioned, our model uses constants to encode zones and names. There must be a way to specify the relations between them. We employ a few hard-coded rules and restrictions to model and enforce a hierarchical name structure.

```
rule Zone_Record_Generator_PX:
  [ GenRecord(zone(’Parent’)) ] --[]->
  [ Record(zone(’Parent’), name(’NameX’)) ]

rule Zone_Record_Generator_CX:
  [ GenRecord(zone(’ChildX’)) ] --[]->
  [ Record(zone(’ChildX’), name(’NameX’)) ]

restriction Naming_Structure:
  "All z n #i.
    NameInZone(z, n)@i ==>
    (z = zone(’Parent’) & n = name(’NameX’)) |
    (z = zone(’ChildX’) & n = name(’NameX’)) "
```

These two rules state that the name `'NameX'` is under both zone `'Parent'` and zone `'ChildX'`, and both of them can publish records for the name. This allows the model to capture attack scenarios where a malicious parent zone serves bogus records for an existing child zone.

B.2 Property Specification

In Tamarin, the execution of a protocol generates a *trace*—a sequence of event facts, associated with timepoints, from rules triggered during the execution. A trace property is a set of traces defined using guarded first-order logic formulae over event facts and timepoints (denoted as terms of the form `#t`). We specify the security theorems introduced in Section 6 as trace property (defined using keyword `lemma`) shown in Figure 18. The formal specification is self-explanatory with the event facts serving as predicates that encode the informally presented theorems. We discuss a few technicalities.

Using the `Compromised()` fact, we can flexibly configure the adversary’s capabilities. The adversary by default has the $\mathcal{A}_1$ capability and can compromise any actor except an entity requesting the delegation for a child zone. Not allowing the compromise of the parent zone owner and at least one of the designated loggers leads to an $\mathcal{A}_1$+$\mathcal{A}_2$+$\mathcal{A}_3$ attacker. Imposing only the latter constraint gives the adversary the $\mathcal{A}_1$+$\mathcal{A}_2$+$\mathcal{A}_3$+$\mathcal{A}_4$ capabilities.

In the lemma `E2E_Authenticity`, we do not specify the order of the event `ZoneDelegated` and `UserAccept`, as the order is implied by the epochs they occur, i.e., the latter hap-

```
lemma Delegation_Security:
  "All epoch zone P C zskC
       CA L1 L2 #i1 #i2.
    ( RCertRequested(epoch, zone, P, C,
                     zskC, CA, L1, L2)@i1
    & ZoneDelegated(epoch, zone, P, C,
                    zskC, CA, L1, L2)@i2
    & not (Ex #j. Compromised(P)@j & j<i2)
    & ( not (Ex #j. Compromised(L1)@j & j<i2)
      | not (Ex #j. Compromised(L2)@j & j<i2)
      )
    ==>
    ( Ex #k. SDApproved(epoch, zone, P,
                 C, pk(zskC), CA, L1, L2)@k
      & k < i2 )"
```

```
lemma E2E_Authenticity:
  "All P czone C_0 epoch zone U qid qname #i1 #i2
         zskC_0 CA_0 L1_0 L2_0 zskC CA L1 L2.
    ( ZoneDelegated(epoch(’T0’), czone, P, C_0,
                    zskC_0, CA_0, L1_0, L2_0)@i1
    & UserAccept(epoch, zone, U, qid, qname,
                 pk(zskC), CA, L1, L2)@i2
    & (epoch = epoch(’T1’) | epoch = epoch(’T2’))
    & not (Ex #j. UpdateLogged(epoch(’T1’),
                               czone)@j)
    & ( not (Ex #j. Compromised(L1)@j & i1<j) |
        not (Ex #j. Compromised(L2)@j & i1<j) ) )
    ==>
    ( zone = czone & zskC_0 = zskC
    & CA_0 = CA & L1_0 = L1 & L2_0 = L2 )"
```

Figure 18: The specification of Theorem 1 (left) and Theorem 2 (right) we proved for RHINE.

pens only in T1 and T2. For the adversary, we do not limit its capabilities in epoch 0, which ends at i1 when the concerned zone czone is delegated. This does not affect security analysis, because our model allows only one zone to be delegated per epoch and disallows the adversary to obtain a child zone owner's private key.

To capture the update protocol's security, we also formalize the following property. It states that once a zone is delegated, its RCert (in particular, the certified key zskC_1 and trusted entities C1_1, L1_1, L2_1) can be updated only by the zone owner generating a signed request using the genuine key (zskC = zskC_0), even if an $\mathcal{A}_1$+$\mathcal{A}_2$+$\mathcal{A}_3$+$\mathcal{A}_4$ is present after the initial delegation setup.

```
lemma Update_Security:
  "All P C_0 zskC_0 CA_0 L1_0 L2_0 #i1
       C_1 zskC_1 CA_1 L1_1 L2_1 #i2
      zone zskC.
    ( ZoneDelegated(epoch(’T0’), zone, P,
          C_0, zskC_0, CA_0, L1_0, L2_0)@i1
    & ZoneUpdated(epoch(’T1’), zone,
        C_1, zskC, zskC_1, CA_1, L1_1, L2_1)@i2
   & (not (Ex #j. Compromised(L1_1)@j & i1<j) |
      not (Ex #j. Compromised(L2_1)@j & i1<j)))
    ==>
   ( Ex #i3. UpdateRequested(epoch(’T1’), zone,
        C_1, zskC, zskC_1, CA_1, L1_1, L2_1)@i3
      & zskC = zskC_0
      & i3 < i2 ) "
```

All these properties are defined over *all* traces. Tamarin also supports proving lemmas that hold when there *exists* a fulfilling trace. This is commonly used for sanity checks of the specification. We have defined multiple such lemmas to test whether our model implements the expected semantics. The following example checks whether the parent zone can legitimately serve records in T0 when no child is delegated.

```
lemma Normal_Resolution_Parent_T0:
  exists-trace
   "Ex P zpk CA L1 L2 U
         qid qname #i1 #i2 #i3.
      ParentInit(zone(’Parent’), P, zpk,
                 CA, L1, L2)@i1
    & UserSentQuery(U, qid, qname)@i2
    & UserAccept(epoch(’T0’), zone(’Parent’),
            U, qid, qname, zpk, CA, L1, L2)@i3
    // no compromise of any actor
    & not (Ex A #k. Compromised(A)@k)"
```

C Achieving High Availability

DNS is a frequent target of (distributed) DoS attacks [58]. A massive DNS outage can make a wide swath of online services unavailable, for example the historic Facebook outage in October 2021. By decoupling the authentication and distribution of a naming system's data (see Section 3), RHINE also separates the concerns of data authenticity and service availability, allowing them to be addressed independently.

One promising direction to ensure the naming service's availability, even amid large-scale DDoS attacks, is to protect the distribution infrastructure with SCION [73], a next-generation secure Internet architecture. With an array of orchestrated mechanisms, including high-speed packet (source) authentication, traffic monitoring and filtering, as well as lightweight bandwidth reservation, SCION can defend against all types of network-level DoS attacks that target network links and nodes and end hosts, offering guaranteed control-plane operation and data delivery. SCION has seen real-world deployments with proven scalability and performance [59].

We plan to deploy RHINE in SCIONLab [60], a full-fledged global Internet testbed, and thoroughly evaluate its practicality, usability, and availability against DDoS attacks.