Decoding DNSSEC Errors at Scale: An Automated DNSSEC Error Resolution Framework using Insights from DNSViz Logs

Md. Ishtiaq Ashiq
Virginia Tech
Blacksburg, Virginia, USA
iashiq5@vt.edu

Olivier Hureau
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
Grenoble, France
hureauo@univ-grenoble-alpes.fr

Casey Deccio
Brigham Young University
Provo, Utah, USA
casey@byu.edu

Taejoong Chung
Virginia Tech
Blacksburg, Virginia, USA
tijay@vt.edu

Abstract

Low adoption and high misconfiguration rates continue to blunt the security benefits of DNSSEC. Drawing on 1.1M historical diagnostic snapshots covering 319K second-level and their subdomains between 2020 and 2024 from the DNSViz service, this paper delivers the first longitudinal, data-driven taxonomy of real-world DNSSEC failures. The study shows that NSEC3 misconfigurations, delegation failures and missing/expired signatures account for more than 70% of all bogus states, and that 18% of such domains remain broken.

Guided by these insights, we introduce DFixer: an offline tool that (i) groups cascaded error codes into root causes, and (ii) auto-generates high-level instructions and corresponding concrete BIND command sequences to repair them. Evaluation with a purpose-built ZReplicator testbed demonstrates that DFixer remedies 99.99% of observed errors in seconds. The curated error-to-command mapping is openly released to foster wider, more reliable DNSSEC deployment.

CCS Concepts

• **Networks** → **Network measurement**; **Application layer protocols**; • **Security and privacy** → **Security protocols**.

Keywords

DNS, DNSSEC, DNS Diagnostics, Network Measurement

ACM Reference Format:
Md. Ishtiaq Ashiq, Olivier Hureau, Casey Deccio, and Taejoong Chung . 2025. Decoding DNSSEC Errors at Scale: An Automated DNSSEC Error Resolution Framework using Insights from DNSViz Logs. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25), October 28–31, 2025, Madison, WI, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3730567.3764428

This work is licensed under a Creative Commons Attribution 4.0 International License.

IMC '25, Madison, WI, USA

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1860-1/25/10
https://doi.org/10.1145/3730567.3764428

1 Introduction

DNS Security Extensions (DNSSEC) [6–8] were introduced in 1999 to prevent DNS spoofing and on-path tampering by adding cryptographic authenticity checks to DNS data. Each zone signs its records (`RRSIG`) using a private key and publishes the corresponding public key (`DNSKEY`), while the parent zone holds a `DS` record that attests to the child's key. A resolver thus constructs a "chain of trust" from the queried domain up to the root, rejecting records with missing or invalid signatures.

Despite these security benefits, DNSSEC adoption remains strikingly low. Only around 5% of `.com` and `.net` domains [1] are signed, with a similarly small fraction (7%) overall [16]. Moreover, misconfiguration rates are alarmingly high: more than 30% of "signed" domains are effectively unresolvable [23]. Administrators must contend with intricate tasks such as key generation and rotation, RRSIG updates, and DS synchronization at the parent. A single oversight can cascade into multiple validation failures.

Real-world outages underscore how serious these misconfigurations can be. At the TLD level, `.nz` and `.fj` both experienced DNSSEC-related interruptions that broke resolution for large swaths of users [18, 19]. Even major platforms like Slack [36] suffered downtime when their DNSSEC setup lapsed. These episodes highlight the fragility of DNSSEC and the difficulty of maintaining secure, signed zones at scale.

Administrators seeking help often turn to diagnostic tools such as `DNSViz` [20]. While these tools accurately flag specific errors (e.g., missing DNSKEY, expired RRSIG), they do not isolate the root cause or supply a direct path to remediation. In practice, an operator may see dozens of messages in `DNSViz` without knowing which one or two errors trigger them all. Many resort to turning off DNSSEC rather than face repeated breakages and troubleshooting [9, 24].

In this paper, we harness `DNSViz`'s historical logs to examine the full lifecycle of DNSSEC errors. We investigate which issues appear most often, how long they persist, and whether administrators eventually fix or abandon them. Our longitudinal analysis shows that certain errors—such as nonzero NSEC3 iteration count or missing signatures and non-existence proofs—appear more than others. These findings emphasize the need for a solution that goes beyond mere diagnosis and actually guides operators toward precise, actionable fixes.

Building on these insights, we introduce DFixer, a domain-specific tool that unifies error detection, root-cause identification, and automatic remediation. We curate a mapping from each commonly observed DNSSEC misconfiguration (e.g., missing or revoked key, invalid signature) to the series of `BIND` commands needed to restore a valid chain of trust. Unlike general-purpose large language models, which often neglect error dependencies and produce vague suggestions, our approach yields correct and reproducible results for real misconfigurations found in `DNSViz` logs.

Our main contributions are:

1. *Comprehensive Measurement of DNSSEC Errors.* Using `DNSViz`'s historical data, we provide a fine-grained analysis of which DNSSEC errors dominate, how they propagate, and how quickly administrators address them.
2. *Root-Cause Diagnostics & Automated Fixes.* We design DFixer, a pipeline that correlates multiple secondary errors into a few root cause(s) and then generates exact `BIND` commands (e.g., `dnssec-signzone`, `dnssec-keygen`) to fix them. Our evaluation with more than 296K erroneous zones shows that DFixer can fully resolve DNSSEC-related errors in 99.99% of them.
3. *Validation via Controlled Replication.* We replicate real misconfigurations in a local sandbox using our tool ZReplicator to ensure that DFixer's recommended fixes genuinely resolve the issues when rechecked with `DNSViz`. Out of 296K zones with diverse combinations of DNSSEC errors, we were able to exactly emulate 79% of them using ZReplicator.

We believe DFixer can help reduce administrator error rates and promote sustainable DNSSEC deployment. To facilitate further development and reproduction, we release the codes for both ZReplicator and DFixer at

https://dnssec-debugger.netsecurelab.org

2 Background

2.1 DNS and DNSSEC

The Domain Name System (DNS) underpins the Internet's mapping of *domain names* to specific *values*, such as the IP address of a server or the location of an email service. For example, an `A` record binds a domain name to an IPv4 address, while an `NS` record designates the authoritative name server for a domain.

However, the original DNS protocol offered no cryptographic safeguards, leaving it susceptible to attacks in which adversaries could forge responses. In response, the *DNS Security Extensions* (DNSSEC) [6–8, 22] were introduced to provide data authenticity and integrity. DNSSEC introduces the following new record types to secure the domain hierarchy:

- `DNSKEY` records: Public keys that authenticate DNS data. Typically, each zone publishes two `DNSKEY` records: one for routine zone signing (known as zone-signing key or ZSK) and another for key-signing tasks (known as key-signing key or KSK), thereby compartmentalizing cryptographic responsibilities.
- `RRSIG` (Resource Record Signature) records: Cryptographic signatures covering entire sets of DNS records—so-called "RRSets"—for a particular domain name and record type. For example, a single `RRSIG` secures all `A` records under `example.org`, ensuring they have not been tampered with. The `RRSIG` is generated with a private key that corresponds to the public key declared in `DNSKEY` records.
- `DS` (Delegation Signer) records: Hashes of `DNSKEY` records stored in the *parent* zone. These hashes establish a secure link from child to parent, continuing upward until reaching the DNS root. Like other record types, `DS` entries are authenticated via `RRSIG` records.
- `NSEC`/`NSEC3` (Next Secure) records: `NSEC` records are used to prove the non-existence of specific records of a domain. `NSEC3` was introduced later [28] as a variant of `NSEC` to prevent zone enumeration using hashing and reduce the size of the zone file via the opt-out flag.

By cryptographically chaining each layer of the DNS hierarchy to the root, DNSSEC fortifies the global naming system against forgery and manipulation. The carefully designed interplay of `DNSKEY`, `RRSIG`, and `DS` records ensures that DNS responses can be validated from the root down to individual subdomains.

2.2 DNSSEC Status

DNSSEC enables a resolver to classify DNS answers into distinct validation states. A security-aware resolver generally assigns one of three labels to a response:

- `Secure`, which indicates that the answer has been cryptographically verified. In this case, the resolver was able to build a complete chain of trust from a known trust anchor (e.g., the DNS root) down to the DNS record in question using all required DNSSEC records (e.g., `DS` and `RRSIG` records).
- `Insecure`, which indicates that at some delegation point, there exists a signed proof of the non-existence of a `DS` record. This situation typically arises when the queried domain lies in an unsigned zone (or beneath one).
- `Bogus`, which indicates a DNSSEC failure: the resolver expected the answer to be Secure (the zone is supposed to be signed), but the validation could not be successfully completed. In this case, the resolver believes there should have been a valid chain of trust, yet one or more checks failed; for example, a required signature was invalid or missing, or a `DNSKEY` did not match its `DS` record.

2.3 DNSSEC Deployment and Management

Despite over a decade of advocacy, DNSSEC's global adoption has remained remarkably low. Sources like the Internet Society [2, 17] and Verisign's DNSSEC Scoreboard [1] indicate that only around 5% of `.com` and `.net` domains are signed.

Operational Challenges. Registrar support for DNSSEC remains uneven; for example, only five registrars responsible for 50% of domain name registrations offered robust DNSSEC tooling in 2019 [11, 35]. Many organizations delegate DNS hosting to third parties (e.g., Cloudflare [12]), introducing further hurdles such as inconsistent `DS` record [11]. On the resolver side, Chung et al. [10] found that only 12% of security-aware resolvers actually validated DNSSEC records, undermining the protocol's potential impact.

Inherent Complexity. Alongside operational challenges, the inherent complexity in DNSSEC's fundamental design actively hinders its broader adoption [29]. The standard demands for meticulous precision and coordination between multiple entities which makes routine tasks like key or algorithm rollovers risky if any actor in the parent-child chain fails to follow best practices [31, 32]. For example, 8% of resolvers failed to adopt new root KSK even after a considerable period which led to the postponement of root KSK rollover in 2017 [31]. Deccio et al. [13] also showed that 24% of 2,634 measured zones exhibited configurations guaranteed to fail due to administrative complexity.

Real-World Failures. Misconfigurations have caused high-profile outages at the TLD level [18, 19] and for major platforms like Slack [36]. Recent measurements by Nosyk et al. [33] found 3.1M (1%) of 303M domains generating Extended DNS Errors (EDE) [26], suggesting a persistent gap between DNSSEC's theoretical security and practical deployment.

Overall, these studies underscore a recurring theme: DNSSEC demands precise multi-party coordination, making misconfigurations alarmingly common. Our work further examines *which* errors are most common, and *why* they arise (§3) and proposes an automated remedy (§4).

2.4 DNSViz

`DNSViz` [20] is a widely used DNS and DNSSEC diagnostic tool that can visualize, analyze, and help debug DNS configurations. Many administrators first encounter `DNSViz` through its interactive web interface [20] where they can input a domain name and view a graphical depiction of its DNS and DNSSEC status. This visualization highlights each step in the trust chain and flags any misconfigurations or validation failures.

DNSViz Command-Line Tool. Beyond the web interface, `DNSViz` also provides a command-line utility that underpins much of its functionality. This utility offers two primary commands that gather DNS data and then interpret it:

- `probe`*:* Systematically queries all the authoritative name servers of a domain to collect its DNS and DNSSEC data (e.g., `DNSKEY`, `DS`, and `RRSIG` records).
- `grok`*:* Interprets and "decodes" the raw DNS data gathered by `probe`, attempting to build a complete chain of trust from the root to the queried domain. If any link is missing or invalid, `grok` flags an error and annotates precisely where in the trust chain validation failed.

Error Codes. `DNSViz` assigns a rich set of error codes to describe potential DNSSEC misconfigurations. These codes capture both straightforward issues—such as missing or invalid signatures or expired `RRSIG` records—and more complex scenarios, including incomplete algorithm setup or delegation errors. At the time of writing, there are over 50 unique error codes, each accompanied by a human-readable message that clarifies the exact nature of the problem (e.g., *"The DS RRset for the zone included algorithm 5, but no RRSIG with algorithm 5 covering the RRset was returned in the response."*).

Category	Root	TLD	SLD+
Total Snapshots	6,234	356,136	747,455
Unique Domains	1	4,196	319,277
w/ at least Two Snapshots	1	2,349	84,962
`CD` **domains**	0	642 (27.3%)	21,734 (25.5%)
`SD` **domains**	1	1,707 (72.7%)	63,228 (74.4%)

Table 1: Overview of the `DNSViz` dataset. It contains 1.1M total snapshots spanning from 2020-03-11 to 2024-09-25. Second-level and lower (SLD+) domains are determined using the Public Suffix List (PSL). CD and SD domains are defined later in §3.2.2.

Relevance to DNSSEC Management. Many operators rely on `DNSViz`'s web interface for quick diagnostics. By mapping out the DNSSEC trust chain in detail, `DNSViz` pinpoints the invalid records and delegation points. However, DNSSEC errors often have interdependencies; for example, an expired `RRSIG` on the apex of a zone can cascade into multiple validation failures, or a malformed `NSEC` record may disrupt negative responses across the trust chain.

Yet, translating `DNSViz`*'s error messages into specific commands or configuration updates can be challenging for administrators.* An error like expired signature may simply require re-signing the zone in `BIND`, whereas a mismatch between `DNSKEY` and `RRSIG` algorithms might necessitate a more complex key rollover process.

In this work, we first leverage `DNSViz`'s extensive logs to gain insights into the prevalence and persistence of DNSSEC misconfigurations, revealing recurring pitfalls and highlighting opportunities for automated or semi-automated solutions. We then leverage `DNSViz`'s diagnostics to *automatically map error codes to actionable fixes in* `BIND`, streamlining the remediation process and reducing downtime.

3 Understanding DNSSEC Debug Patterns

3.1 Dataset

We obtained the complete historical database of `DNSViz` from DNS-OARC [37]. Our analysis focuses on data collected between March 2020 and September 2024, as summarized in Table 1.

Terminology and Dataset Scope. `DNSViz` operates under a query-driven model: when an administrator or user analyzes a Query Domain, `DNSViz` recursively traverses the domain's DNSSEC chain of trust, from the root down to the zone containing Query Domain (called Query Zone). Along the way, it queries various record types (e.g., `A`, `AAAA`, `MX`) and verifies `DS`/`DNSKEY` pairs in each delegation. To test negative responses (e.g., `NSEC` or `NSEC3`), `DNSViz` also queries random sub-labels that do not exist.

Every response—DNS data, `RRSIG` records, delegation details, negative proofs—undergoes validation against DNSSEC standards, and `DNSViz` assigns specific error (or warning) codes for detected failures. In this paper, we *only* consider DNSSEC-related errors that (1) can cause a validator to return `SERVFAIL` or (2) violate a DNSSEC "MUST" property from relevant RFCs [6–8], excluding advisory ("SHOULD"-level) warnings and non-DNSSEC issues.

Limitation. All analyses in the `DNSViz` dataset come from *user-initiated scans*, which may introduce self-selection bias, as the

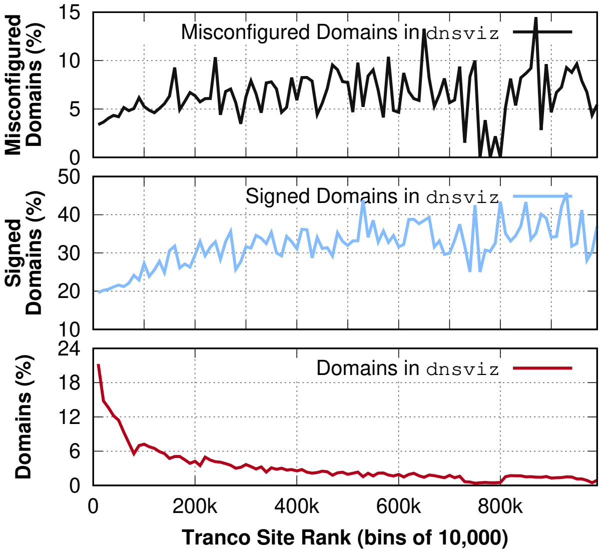


Figure 1: Percentage of domains from our `DNSViz` dataset appearing in each Tranco 1M bin [34], including the share of signed and misconfigured domains.

dataset might not reflect the broader DNS ecosystem. Even so, it captures a wide range of real-world DNSSEC configurations and offers a practical lens on how they behave *in practice*, especially for domains whose administrators *actively* sought diagnostic insight. While we understand that domains that never appear in `DNSViz` may behave differently, the collected data should be sufficiently representative to support the analysis presented in next sections.

DNSViz Prevalence. To assess the broader significance of this dataset, we compare the domains appearing in `DNSViz` logs to domain popularity rankings. Figure 1 highlights the percentage of Tranco top-1M domains [34] also present in our `DNSViz` dataset. Two observations stand out:

- *High Coverage of Popular Domains.* Among the top 10k domains, 20% appear in the `DNSViz` logs (red line, bottom). This indicates that well-known domains are frequently tested or monitored, either by their operators or by curious users, presumably because DNSSEC correctness is seen as especially critical for large or high-traffic sites; the top graph supports this, showing that DNSSEC misconfigurations are comparatively less common among top 20k domains.
- *Diverse Representation Among Signed Domains.* The blue line in Figure 1 (middle) shows that a substantial fraction of domains that have ever been DNSSEC-signed (at least once) appear across the entire popularity spectrum. In fact, more than 30% of these domains—even those consistently in the lower popularity bins—show up in the `DNSViz` logs. This finding indicates that smaller or niche domain owners also rely on `DNSViz` to troubleshoot or refine their DNSSEC configurations, suggesting that the tool's usage extends beyond mere curiosity. Rather, it meets the genuine operational needs of a diverse set of domain operators, regardless of their popularity ranking.

3.2 Validation and Category Definitions

In order to systematically examine `DNSViz`'s diagnostic data, we first establish a scheme for categorizing individual snapshots (§3.2.1) and then group domains based on how their DNSSEC statuses evolve over time (§3.2.2). This approach allows us to distinguish truly *static* configurations (e.g., consistently valid DNSSEC setups) from those experiencing configuration changes, either intentional (operator debugging, key rollover) or accidental (signature expiry, missing DS record)[1].

3.2.1 Snapshot Categorization. A *snapshot* corresponds to the state of a Query Domain at a specific time when `DNSViz` was invoked. We classify each into one of six categories:

- `sv` (*signed and valid*). No DNSSEC errors are detected: all signatures (`RRSIG`) are current and verifiable, and a DNSSEC-validating resolver would successfully authenticate the domain.
- `svm` (*signed and valid with misconfiguration*). For every query to each authoritative server, a valid authentication path up to root can be constructed; however, there exists at least one DNSSEC violation in the snapshot. Violations in this category typically do not cause validation failure but depends on the validating resolver. An example is exceeding NSEC3 iteration count [25], which most resolvers ignore [14][2].
- `sb` (*signed and bogus*). At least one query to at least one authoritative server of the domain fails cryptographic validation (e.g., an expired `RRSIG`). These errors produce a `SERVFAIL` response code, effectively breaking DNS resolution for validating resolvers.
- `is` (*insecure*). The domain is *explicitly unsigned* in a valid way. For instance, the parent zone has a confirmed *proof of non-existence* (no `DS` record) that resolvers recognize. Thus, the zone is treated as "plain DNS" rather than bogus.
- `lm` (*lame*). The name servers for the Query Zone do not respond correctly or cannot be resolved, resulting in a lame delegation. Both DNSSEC and standard DNS queries would fail.
- `ic` (*incomplete*). The `NS` records for the Query Zone appear only within the zone itself but not in the parent zone. This typically disrupts normal delegation, leading to inconsistent or failed lookups.

Since our analysis centers on DNSSEC status, we primarily focus on four DNSSEC-related categories: `sv`, `svm`, `sb`, and `is`. The other two categories, `lm` and `ic`, reflect fundamental issues with the zone's overall functionality rather than DNSSEC policy per se.

3.2.2 Domain Categorization. Domains can have multiple snapshots over time, each potentially falling into a different category. To highlight meaningful changes, we divide domains into two broad groups:

- *Changing Domains (CD):* Domains that exhibit at least two snapshots with different DNSSEC-related error codes or statuses. Such

---

[1] We assume that the DNS response messages `DNSViz` extracts from authoritative servers of a zone are always authentic and observed errors/misconfigurations are not because of a failed Man-in-the-Middle attack.

[2] A minority of resolvers treat nonzero NSEC3 iteration counts as fatal, as highlighted by Daniluk et al. [14]. However, this is implementation-dependent [25].

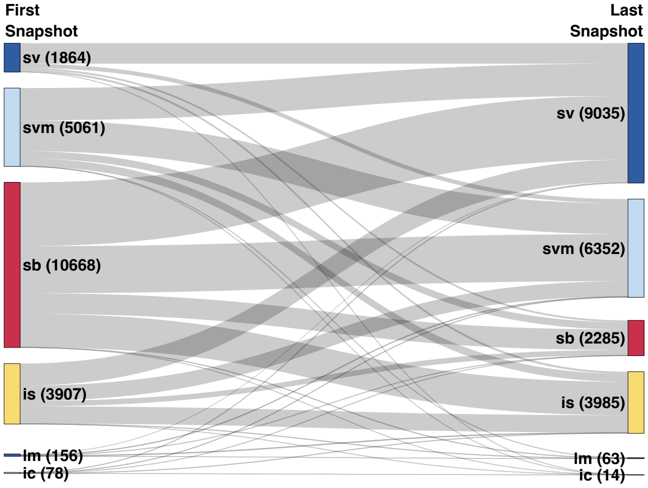


Figure 2: For domains in the CD category, comparing DNSSEC status between the first and last snapshots recorded by DNSViz. Note the moderate but significant fraction of domains that either enable or disable DNSSEC over time.

differences indicate that something in the DNSSEC configuration changed between observations, whether intentionally (e.g., key rollover, DS update) or unintentionally (e.g., expired signatures). We analyze these domains extensively to understand how operators fix (or fail to fix) misconfigurations.

- *Stable Domains (SD):* Domains whose snapshots all reside in the *same* DNSSEC category throughout the dataset. The rest of the domains with more than one snapshot are in this category as shown in Table 1.

3.3 How Useful Is `DNSViz`?

`DNSViz` is widely recognized for identifying DNSSEC misconfigurations, but the extent to which operators use it to *fix* errors (rather than just diagnose them) remains unclear. To gauge its practical impact, we examine the *Changing Domains (CD)* subset—domains with at least two snapshots and at least one shift in DNSSEC error codes.

Figure 2 illustrates how these domains transition from their *first* observed snapshot to their *last*:

- *Positive Trajectory.* Among the domains initially in `sb` (signed and bogus), 7,200 (67%, out of 10,668) eventually corrected their DNSSEC errors, reaching `sv` (signed and valid) or `svm` (signed and valid w/ misconfigurations). This indicates a meaningful fraction of operators do act on `DNSViz`'s diagnostics to remedy their zones.
- *Newly Signed.* Out of previously unsigned domains, 2,400 (62%, out of 3907) proceeded to enable DNSSEC by their final snapshot, suggesting `DNSViz` might also be used to guide the initial deployment of zone signing.

On the other hand, `DNSViz` logs reveal *reverse* trends as well: about 650 domains (9.4%, out of 6,925) that began in a valid state (`sv` or `svm`) reverted to insecure (no DS records), and 588 (8.4%, out of 6,925) transitioned to `sb` (bogus) by their last snapshot. We analyze these cases more closely in §3.4 to explore the potential reasons (e.g., key rollover mishaps, expired signatures, or explicit DNSSEC deactivation).

Previous State	Transitioned State	Cause	# of Domains
`sv`	`sb`	**Total**	4,064
		`NS` Update	272 (6.7%)
		Key Rollover	1,836 (45.2%)
		Algo. Rollover	1,230 (30.3%)
`sv`	`is`	**Total**	804
		`NS` Update	56 (7%)
		Key Rollover	241 (30%)
		Algo. Rollover	145 (18%)

Table 2: Causes of negative transitions from a valid (`sv`) DNSSEC state to either bogus (`sb`) or insecure (`is`). Key rollovers and algorithm changes together account for roughly two-thirds of `sv` → `sb` transitions, while a smaller fraction stems from nameserver (`NS`) updates. We observe a similar pattern for `sv` → `is` transitions as well.

Overall, Figure 2 confirms that the dataset is quite invaluable in understanding the evolution of DNSSEC configuration on a large scale.

3.4 Exploring Negative Transitions

While many domains improve their DNSSEC configurations over time, a notable fraction transition from an `sv` (signed and valid) state to `sb` (signed and bogus) or `is` (insecure). To understand these "negative transitions" in more depth, we manually investigated a subset of such cases.

Key Triggers for Bogus States. Out of 4,064 instances where a domain shifted from `sv` to `sb` in any *two consecutive* snapshots, we identify three potential causes:

1. *Nameserver (`NS`) Updates.* Operators sometimes switch their authoritative servers (e.g., migrating to a new DNS operator). In doing so, they occasionally reuse old zone files to sign the zone or forget to upload new `DS` records at the registrar, resulting in invalid `RRSIG` or mismatched DS entries. To determine this, we looked into the `NS` record of the Query Zone for each snapshot and counted the number of times a shift to bogus state coincides with a change in `NS` record. For instance, `apopo.website` maintained `sv` status for 28 snapshots but switched to a new nameserver on the 29th, accidentally resigning the zone with stale DNS content. That single oversight triggered an invalid `RRSIG` error, dropping the domain to `sb`.
2. *Key Rollovers.* Regularly updating cryptographic keys helps prevent key compromise and protects the integrity of the "chain of trust". However, if administrators fail to synchronize `DS` and `DNSKEY`s updates properly, a zone can briefly or indefinitely become bogus. To identify negative transitions due to key rollover, we examined the `DNSKEY` record of the Query Zone for each snapshot and counted how often a shift to `sb` coincided with a `DNSKEY` change. For example, `ciast.edu.my` had a valid DNSSEC configuration with one `KSK` and one `ZSK`, but an attempted ZSK rollover introduced four additional DNSKEYs, resulting in an inconsistent key set and a misconfigured DNSSEC

Category	Subcategory	# of snapshots (%)	# of domains (%)
Delegation	⑤ Missing KSK for Algorithm	63,004 (8.4%)	25,102 (7.9%)
	① Invalid Digest	1,103 (0.15%)	466 (0.15%)
Key	③ Inconsistent `DNSKEY` b/w Servers	19,330 (2.6%)	6,393 (2%)
	Revoked Key	302 (0.04%)	45 (0.014%)
	Bad Key Length	108 (0.01%)	21 (0.007%)
Algorithm	② Incomplete Algorithm Setup	6,859 (0.9%)	1,883 (0.5%)
Signature	Missing Signature	38,662 (5.2%)	18,306 (5.7%)
	④ Expired Signature	11,670 (1.6%)	4,494 (1.4%)
	⑥ Invalid Signature	10,336 (1.4%)	3,152 (1%)
	Incorrect Signer	1,961 (0.3%)	550 (0.2%)
	Not Yet Valid Signature	663 (0.09%)	125 (0.04%)
	Incorrect Signature Labels	99 (0.01%)	25 (0.008%)
	Bad Signature Length	42 (0.006%)	13 (0.004%)
TTL	⑧ Original TTL Exceeds RRSet TTL	4,999 (0.7%)	1,769 (0.6%)
	TTL Beyond Expiration	2,556 (0.3%)	864 (0.3%)
NSEC(3)	⑦ Missing Non-existence Proof	65,378 (8.7%)	17,768 (5.6%)
	Incorrect Type Bitmap	18,218 (2.4%)	4,070 (1.3%)
	Bad Non-existence Proof	9,678 (1.3%)	3,255 (1%)
NSEC(Only)	Incorrect Last `NSEC`	405 (0.05%)	214 (0.07%)
NSEC3(Only)	⑨ Nonzero Iteration Count (NZIC)	215,036 (28.8%)	62,870 (19.7%)
	Inconsistent Ancestor for `NXDOMAIN`	2,296 (0.3%)	1,410 (0.44%)
	Incorrect Closest Encloser Proof	1,278 (0.17%)	415 (0.13%)
	Invalid `NSEC3` Hash	456 (0.06%)	200 (0.06%)
	Invalid `NSEC3` Owner Name	301 (0.04%)	152 (0.05%)
	Incorrect Opt-out Flag	186 (0.02%)	64 (0.02%)
	Unsupported `NSEC3` Algorithm	74 (0.01%)	11 (0.003%)
-	w/ at least One DNSSEC Error	296,813 (39.7%)	81,805 (25.6%)

Table 3: Prevalence of various DNSSEC error types in our `DNSViz` dataset, covering 319,277 second-level and their lower-level domains (total 747,455 snapshots); for example, the "Nonzero Iteration Count" in NSEC3 appears in 215,036 snapshots (28.8%) spanning 62,870 domains (19.7%). The markers (①–⑨) highlight notable or especially frequent misconfigurations in each category, which are also presented in Figure 4. Last row represents the # of snapshots (and domains) with at least 1 error from above categories.

setup. This is a common mistake during rollovers, as reported in [30].

(3) *Key Algorithm Rollovers.* A more complex type of rollover where the algorithm associated with the `DNSKEY` record (e.g., from RSA to ECDSA) is changed as well, not just the key itself [27]. Operators can do this to switch to a more secure and/or shorter key and signatures or use multiple algorithms simultaneously to prevent validation failure due to unsupported algorithms. However, given DNSSEC algorithm rollover is a complicated process [38], this creates more window to break DNSSEC validation.
To identify whether any of the transitions to bogus state was caused by this, we looked into the `DNSKEY` algorithms of the Query Zone for each snapshot and counted the number of times a shift in state coincides with an update in `DNSKEY` algorithm. Out of 4,064 such transitions, we observed a key algorithm change in 1,230 (30.3%) instances.

Collectively, these three categories account for 81% of the observed negative transitions (see Table 2).

Switching to Insecure. Similarly, we identified 7% of cases where valid domains transitioned to `is` following nameserver updates, while in 30% and 18% cases, domains removed their `DS` record and turned off DNSSEC following key rollover and algorithm rollover respectively. Moreover, we found 610 domains that first became bogus from valid and then disabled DNSSEC permanently as per their latest snapshot. Although we cannot confirm the exact administrative intentions, these patterns highlight usability challenges and point to potential mismanagement during server migrations or key transitions.

3.5 Error Prevalence in DNSSEC

To identify which DNSSEC issues arise most frequently, we examine 47 distinct `DNSViz` error codes spanning our entire dataset. Table 3 groups them under 8 parent categories (e.g., "Delegation," "Key," "Signature," etc.) and 26 subcategories to facilitate broader analysis.

- `NSEC`/`NSEC3` *Errors.* `NSEC` and `NSEC3` issues are the single most prominent category. In particular, the "Nonzero Iteration Count" code appears in 215,036 snapshots (covering 62,870 domains)—by far the largest single error. Another 65,378 snapshots (17,768 domains) exhibit a "Missing Non-existence Proof" error, indicating that negative responses (`NXDOMAIN`, `NODATA`) lack the appropriate `NSEC`/`NSEC3` records. These two alone underscore that negative-proof mechanics remain a persistent challenge for administrators.
- *Delegation Errors.* Errors in establishing a valid chain of trust also loom large. For instance, having a "Missing KSK for Algorithm" (i.e., a DS record referencing a key algorithm not actually

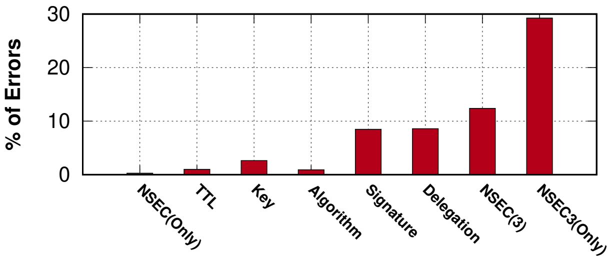


Figure 3: Percentage of DNSSEC error categories out of 747,455 snapshots for second-level and lower domains. NSEC(3)-related errors, including nonzero iteration counts and missing negative proofs, account for a significant fraction. We also see many delegation failures and signature (`RRSIG`) issues.

present) appears in 63,004 snapshots and affects 25,102 domains. Meanwhile, “Invalid Digest” (a DS record whose hash does not match any `DNSKEY`) occurs in 1,103 snapshots across 466 domains. Although smaller in absolute terms, such mismatches directly trigger `SERVFAIL` and are often among the most critical to fix.

- *Signature Anomalies.* Signature-related issues are widespread: “Missing Signature” arises in 38,662 snapshots (18,306 domains), while “Expired Signature” affects 11,670 snapshots (4,494 domains), and “Invalid Signature” appears 10,336 times (3,152 domains). Collectively, these reflect a broad set of operational lapses, from forgetting to re-sign the zone before signatures expire to incorrectly applying the private key.
- *Key-Related Inconsistencies.* Among the errors in the “Key” category, the most common is having inconsistent `DNSKEY` sets across different authoritative servers (19,330 snapshots, 6,393 domains). Although less prevalent than major `NSEC` (3) or signature issues, this inconsistency can break validation and lead to intermittent `SERVFAIL`. Other key-level errors (e.g., “Bad Key Length,” “Revoked Key”) are relatively rare but still appear in hundreds of snapshots.
- *Algorithmic Gaps.* A noteworthy 6,859 snapshots (1,883 domains) exhibit “Incomplete Algorithm Setup,” where an algorithm referenced by the zone is not consistently present in all `RRSIG` records or does not align with the parent zone's `DS` records. This often arises when an operator attempts to upgrade or change algorithms but omits essential steps (e.g., forgets to sign the zone with all unique algorithms present in the zone [39]).

Figure 3 provides a visual summary, highlighting the disproportionate frequency of `NSEC3` iteration count violations and missing `RRSIG` sets. These findings suggest that some DNSSEC intricacies such as negative proof management remain persistent trouble spots.

3.6 Understanding Error Resolution Patterns

While identifying the *kinds* of DNSSEC errors is useful, understanding *how quickly* they are addressed is equally crucial. Table 4 shows the number of transitions in the `CD` set between any two consecutive snapshots and median time for these transitions. As we can see, transition from bogus to valid has a median time of only 0.7 hours. We further identified 1,856 domains (out of 4,064) that transitioned from a valid to a bogus state and subsequently recovered (back to valid). For this subset, the median time for the valid to bogus transition is 238.6 hours, whereas the median for bogus to valid is 0.6 hours. This clearly indicates that administrators tend to respond much more promptly to errors that cause resolution failures.

From \ To	sv	svm	sb	is
sv	–	1310 34.2h	4064 133.7h	804 58.6h
svm	3132 73.4h	–	5573 104.2h	1486 71.8h
sb	8052 0.7h	8065 0.87h	–	3922 1.6h
is	2150 2.7h	2097 3.3h	2001 1.8h	–

Table 4: Adjacency matrix of state transitions. Each cell shows # of transitions (top) and median time in hours (bottom). A dash (–) means no observed transition. We can observe administrators reacting promptly when their domain becomes bogus.

To further understand which individual error code is easier to fix or needs quick operator attention, we now focus only on domains that eventually corrected their DNSSEC status, transitioning from `sb` or `svm` to `sv`. For each error code with at least 100 *fix* instances, we define two timestamps:

- t_1: The first snapshot with `sb` (signed-bogus) or `svm` (signed-valid w/ misconfig) category where the error is present. If this snapshot is `sb`, the error is deemed *critical* (leading to `SERVFAIL`); under `svm`, it is *non-critical.*
- t_2: The first snapshot in which the error is no longer detected and the domain is fully valid (`sv`).

We compute the time difference between snapshots t_1 and t_2 for each instance and plot the distributions in Figure 4. We note two *limitations*:

Overestimation in fix time. Since `DNSViz` rescans are triggered by users, a domain might fix an error shortly after t_1 yet go unrecorded until the next user-initiated scan. Hence, we could overestimate problem durations (or miss short-lived fixes that revert). Nevertheless, as shown in Figure 5, 65% of the domains have a median time difference of less than a day between their consecutive two snapshots which suggests that these scanning gaps are not very common, so the overall trends remain informative.

Underestimation of critical errors. Some error codes in `svm` category may also lead to `SERVFAIL`. It is impossible to identify them deterministically without complete validation context. For example, “Incomplete Algorithm Setup” error code may or may not produce `SERVFAIL` depending on whether the validating resolver supports one of the algorithms that can form a valid chain of trust up to root. Therefore, our estimation of critical errors may potentially be an underestimation of actual number of critical errors.

Regardless of the caveats, we make the following useful observations from this analysis:

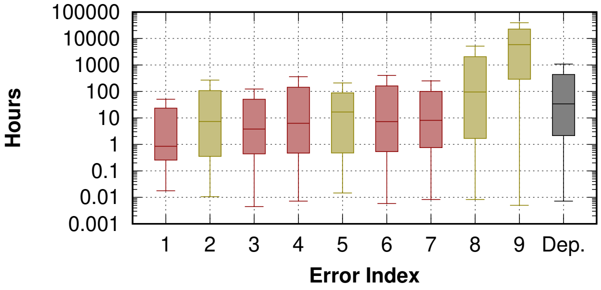


Figure 4: Resolution times for critical (red boxes) vs. non-critical (orange boxes) DNSSEC errors. The numbers on the x-axis (1–9) are indices that map directly to the numbered error categories detailed in Table 3. Note that the median time to deploy DNSSEC (black box) is more than a day; this is likely attributed to the operational challenges in DNSSEC identified by prior works [11].

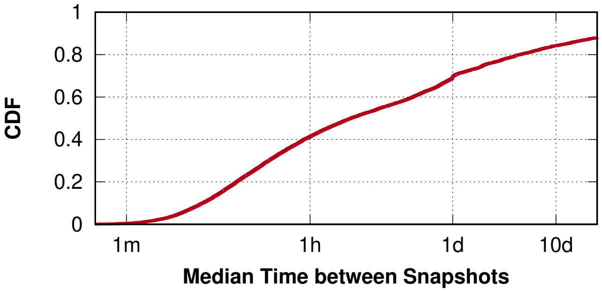


Figure 5: 65% of the domains have a median time difference of less than a day between their consecutive two snapshots

- *Critical Errors Are Fixed Sooner.* Delegation problems such as DS and DNSKEY mismatches that break trust chains are often resolved within 2–3 days in 80% of cases. Similarly, in 80% cases, inconsistent DNSKEY sets take about four days while expired or invalid signatures take a bit longer (10 days). This is somewhat surprising, since re-signing can be automated, suggesting that some operators may still use manual signing procedure.
- *Non-Blocking Errors Are Overlooked.* Errors typically not causing SERVFAIL—such as TTL mismatch with Original TTL field in RRSIG record or a nonzero NSEC iteration count—persist for weeks or months (60 and 250 days, respectively, in 80% of occurrences). Because they do not break resolution, administrators often de-prioritize them indefinitely.

Finally, Table 5 shows a large fraction of domains *never* remedied their DNSSEC errors during our observation period. Roughly 18% of domains once in `sb` remained in that status, and 36.5% of those that once turned off DNSSEC (i.e., reverted to `is`) never re-enabled signing. These persistent shortcomings highlight the operational complexities of DNSSEC and the tendency for domains to abandon signing when errors become unmanageable.

Category	**# of Domains**	
	w/ at least One Snapshot	**Not Resolved**
`sb` (signed & bogus)	15,209	2,731 (18%)
`svm` (signed & valid w/ misconfig.)	9,052	5,603 (61.9%)
`is` (insecure)	7,149	2,611 (36.5%)

Table 5: For each DNSSEC state of interest, we list (i) the total number of domains that appear in that state at least once, and (ii) how many never rectified the issue or re-enabled DNSSEC according to their latest snapshot in our dataset.

Implications. Overall, despite `DNSViz`'s thorough diagnostics, the path from "error found" to "error fixed" often remains lengthy or incomplete. Critical errors do tend to be addressed (albeit not instantly), but non-critical issues remain in place for long durations, and many domains drop DNSSEC altogether rather than maintain it. This underscores the need for a more *automated*, domain-aware remediation mechanism. We explore such a solution in the next section.

4 DFixer: A Comprehensive DNSSEC Fixing Framework

We introduce DFixer, a framework that pinpoints DNSSEC misconfigurations and translates these diagnoses into actionable `BIND` commands. Compared to purely diagnostic tools (e.g., `DNSViz`) or naive Large Language Model (LLM) suggestions, DFixer provides concrete, verifiably correct fixes for the *root causes* of DNSSEC errors.

4.1 Rationale and Key Observations

The design of DFixer rests on two central insights:

(1) *Common Patterns of DNSSEC Errors.* Despite DNSSEC's complexity, many domains exhibit the same recurring mistakes: stale DS records, incomplete algorithm transitions, expired RRSIG signatures etc. Consequently, the distinct *root causes* are relatively few, even if individual domains show multiple secondary errors.

(2) *Small Repertoire of BIND Commands.* Although DNSSEC provisioning can be intricate, `BIND` ultimately relies on a limited set of commands (e.g., `dnssec-signzone` for signing the zone, `dnssec-keygen` for generating keys, `dnssec-dsfromkey` for generating DS records). These suffice for most scenarios if one knows how to apply them correctly and *in the right order.*

Leveraging these observations, DFixer pairs real-world `DNSViz` misconfiguration data with a manually curated knowledge base of *error-to-command* mappings. The end goal: for each snapshot of a domain, DFixer identifies the root cause(s), devises a minimal remediation plan, and then presents precise commands to achieve a valid DNSSEC state.

4.2 Why Existing Tools and Naive LLMs Fall Short

Diagnostic-Only Tools. Well-known utilities, such as `DNSViz` [20], Zonemaster [40], DNSSEC Debugger [15] excel at detecting DNSSEC anomalies. However, they typically produce extensive lists of error

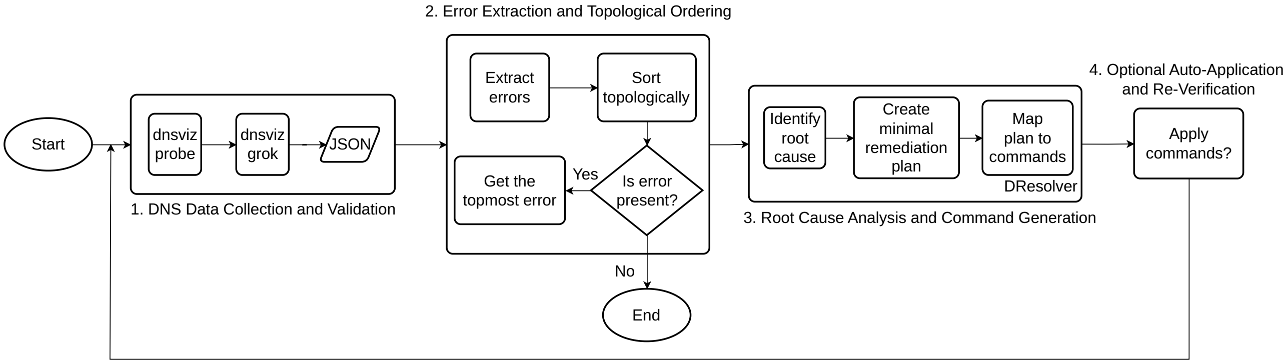

2. Error Extraction and Topological Ordering
Extract errors
Sort topologically
Is error present?
Yes
Get the topmost error
No
End
Start
dnsviz probe
dnsviz grok
JSON
1. DNS Data Collection and Validation
Identify root cause
Create minimal remediation plan
Map plan to commands
DResolver
3. Root Cause Analysis and Command Generation
4. Optional Auto-Application and Re-Verification
Apply commands?

Figure 6: Overview of the DFixer pipeline. Each iteration collects DNSSEC data with `probe` and `grok` filters relevant error codes, resolves root causes via DResolver, and produces fix commands. The process repeats until no blocking errors remain.

messages that do not isolate the few root causes. This can overwhelm novice administrators in understanding *which* errors directly break the chain of trust and *which* are mere side effects. For instance, a simple extraneous `DS` record can trigger over a dozen distinct error messages. Without further guidance, operators may struggle to fix the *correct* underlying issue.

Naive LLM-Based Approaches. Large Language Models (e.g., GPT-4o) show promise in synthesizing text, and are able to provide highly dynamic and interactive solution. Thus, we wanted to verify how naive LLM-based prompt engineering techniques perform in our problem domain. As an experiment (Appendix §7.3), we used GPT-4o to help us diagnose and fix common DNSSEC-related problems and observed that it often:

- *Produced generic suggestions* (e.g., "Verify your DS record or check whether your zone is signed with the correct key").
- *Hallucinated details* such as fictitious key tags, algorithms, or flags when given partial logs or images from `DNSViz`.
- *Missed interdependencies* among multiple errors, suggesting partial fixes that fail to address the root cause.

Meanwhile, existing natural-language-to-shell-command tools [5] from NLP research fail to capture DNSSEC-specific semantics. Thus, a carefully curated mapping from *diagnosed error* to *BIND command sequence* remains indispensable.

Limitation. Our findings about naive LLMs should be read as illustrative rather than definitive; in Appendix §7.3, the examples show generic advice, hallucinated key tags or algorithms, and confusion between parent and child issues, but they do not constitute a comprehensive cross-model benchmark.

First, the baseline uses a single general-purpose model and a small prompt family, without any knowledge retrieval, or fine-tuning mechanism. Second, LLM outputs are non-deterministic and models evolve; we did not control seeds, compare vendors, or sweep sampling parameters, so results should not be over-generalized. Finally, several pitfalls we observe reflect *intrinsic DNSSEC operational complexity*, including multi-phase key or algorithm rollovers, not merely LLM shortcomings [27, 38]. As future work, we plan to evaluate whether DFixer can be extended into an agentic AI system to further enhance its usability.

4.3 Design and Workflow of DFixer

DFixer follows an *iterative* pipeline, illustrated in Figure 6. Below, we highlight the major steps:

1. *DNS Data Collection and Validation:* DFixer invokes the `dnsviz probe` and `dnsviz grok` commands on the target domain.
2. *Error Extraction and Topological Ordering:* DFixer parses the `grok` JSON, isolates DNSSEC-related error codes, and constructs a dependency graph. Many errors are *cascading* (e.g., a missing key can trigger numerous signature or delegation errors). By topologically sorting these errors, DFixer ensures that it addresses the root cause first.
3. *Root-Cause Analysis and Command Generation:* A module called DResolver first identifies the root cause from zone context i.e., topmost error code, presence or absence of some companion errors, and state of the zone. For example, if the topmost error picked is "Revoked Key", DResolver checks whether:
   - a "No secure entry point" error is also present. This indicates that there exists a `DS` record that is linked to the revoked key which needs to be purged.
   - there are other valid `KSK` present. If yes, just removing any associated `DS` record and deactivating the revoked key is fine. Otherwise, a new `KSK` key pair needs to be generated along with the deactivation of revoked key.

   In this way, DResolver then synthesizes a remediation plan with a set of instructions (e.g., "Generate a new KSK, upload new DS, remove the incorrect DS, re-sign the zone"). Finally, it translates each instruction in the plan to `BIND` *commands.*[3] The associated parameters for these commands are populated from zone context.
4. *Optional Auto-Application and Re-Verification:* The operator can run DFixer in two modes:

---

[3] `dnssec-keygen`, `dnssec-signzone`, `dnssec-settime`, `dnssec-dsfromkey`

- *Suggest Only (dry-run)*: DFixer prints out the exact commands (with placeholders for key file or zone file paths).
- *Auto-Apply*: If the operator grants the necessary privileges (e.g., shell access) and local data paths, DFixer can directly execute these commands (except the ones that require manual interaction), re-running `probe` and `grok` after each fix.

This iterative re-verification continues until no DNSSEC errors remain. Figure 8 in Appendix shows a sample fix plan for a domain when its only KSK has the REVOKED flag on and is linked to a `DS` record.

4.4 System Model and Assumptions

In parallel with these pipeline stages, DFixer makes certain assumptions about the operator and environment.

Actor and Capabilities. We assume a *DNS operator* with `rwx` privileges on an authoritative server (running `BIND`). This operator has basic DNSSEC knowledge (record types, signing logic) and wants an automated solution to *pinpoint* errors and *fix* them.

Objectives. DFixer aims for:

- *Correctness*: Precisely identify root causes and map them to valid BIND commands.
- *Comprehensiveness*: Cover a broad range of DNSSEC failures, from stale DS records to incorrect algorithm rollovers.
- *Extensibility*: Although focused on BIND, our approach can be adapted for other servers like `NSD` or `PowerDNS` by creating analogous error-to-command mappings, which will be shown in §5.6.

4.5 ZReplicator: Replicating Real Errors in a Local Environment

Although `DNSViz` data reveals myriad real DNSSEC errors, DFixer cannot directly correct zones owned by third parties. To *fully* validate whether DFixer's recommended fixes actually work, we developed ZReplicator, a custom local replication module.

Motivation. We want to recreate real-world DNSSEC errors in a sandbox where we can:

1. *Inject* the exact misconfigurations (e.g., stale DS, invalid signatures) from `DNSViz` logs.
2. *Run DFixer*, apply suggested commands, and confirm that the zone becomes valid after re-checking.

This offers a true "test-fix-verify" cycle, unlike passively analyzing logs.

Replication Process. ZReplicator proceeds as follows:

1. *Create a Base Zone:* We first set up a fully signed `BIND` zone (e.g., `a.com`) to serve as the root for our local environment. This enables us to automate communication with parent zone (upload `DS` record) during evaluation.
2. *Emulate Parent and Child Zones:* We create sub-zones (e.g., `par.a.com` and `inv-chd.par.a.com`) mirroring the parameters (# of `DNSKEY` records and their flags, algorithms, and key size, # of `DS` records, and their digest type, usage of `NSEC` vs. `NSEC3`, etc.) extracted from the real domain's `DNSViz` logs.
3. *Inject DNSSEC Errors:* By tweaking the zone files (e.g., setting an RRSIG expiration in the past, removing a DNSKEY on one nameserver), we replicate the target error scenario. This can also include multi-server inconsistencies or algorithm mismatches.

This local replication ensures we can *empirically* verify DFixer's effectiveness and correctness. In §5, we detail how we use ZReplicator to rigorously evaluate DFixer across thousands of misconfiguration scenarios extracted from `DNSViz` logs.

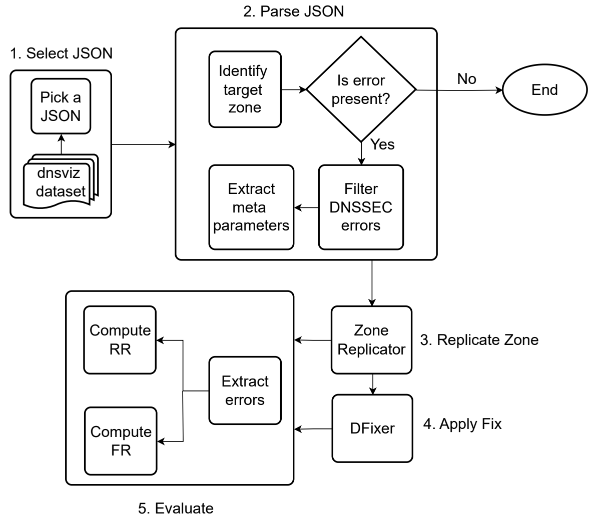


Figure 7: High-level illustration of our evaluation pipeline. Each JSON file from the `DNSViz` dataset is parsed, replicated locally via ZReplicator, and then processed by DFixer.

5 `DFixer`: Evaluation

We now present a comprehensive evaluation of DFixer, leveraging the large-scale `DNSViz` dataset described in §3.1. Our primary goals are to assess whether DFixer (1) accurately reproduces real-world DNSSEC errors through the ZReplicator module (§4.5), and (2) effectively fixes these replicated misconfigurations in an automated manner.

5.1 Evaluation Pipeline

As summarized in Table 1, our `DNSViz` dataset comprises 747K snapshots (JSON files) for SLD and lower domains.

Figure 7 illustrates our experimental pipeline, which proceeds as follows:

1. *Select JSON Snapshot.* We iterate over the `DNSViz` dataset (747K snapshots for SLD+ domains), taking one JSON file at a time.
2. *Parse JSON.* For each snapshot, we extract all *DNSSEC-related* errors present in the leaf zone ("target zone") such as invalid `RRSIG` or missing `DNSKEY`. We also record *zone meta-parameters* that are critical for accurate replication and subsequent fixes:
    - *DNSKEY properties:* Number of keys, their algorithms, and key sizes. These details are essential for generating correct `BIND` commands (e.g., `dnssec-keygen` arguments) so that we can match the zone's actual cryptographic setup.

Dataset	# of Snapshots	$GE \neq \phi$	$IE \subseteq GE$ & $IE \neq \phi$	$AE = \phi$ & $IE \subseteq GE$	RR	FR
NZIC Only ($S1$)	168,482	166,591	166,470	166,470	98.81%	100.0%
Remaining ($S2$)	128,331	119,362	101,003	100,998	78.71%	99.99%
Total	296,813	285,953	267,473	267,468	90.11%	99.99%

Table 6: Performance of ZReplicator and DFixer among the snapshots with IE $\neq \emptyset$.

- *Delegation settings:* Parent-zone data, such as `DS` record digest types and algorithms, needed for consistent DS creation or removal.
- *NSEC vs. NSEC3 usage:* Whether the zone uses `NSEC` or `NSEC3` (and if `NSEC3`, value of iteration count, salt, and flags). This information ensures that commands like `dnssec-signzone` reflect the same negative-proof mechanism as in the original zone.

(3) *Replicate Zone via ZReplicator.* We feed the extracted error codes and zone parameters to our ZReplicator module, which locally reproduces the misconfigurations using two authoritative `BIND` name servers to mimic multi-server inconsistencies.

(4) *Apply DFixer.* DFixer runs on the replicated zone, iteratively applying fixes.

(5) *Evaluate.* Finally, we *re-invoke* `probe`/`grok` on both the replicated (pre-fix) and rectified (post-fix) zones to verify: (1) whether ZReplicator successfully replicated the original errors, and (2) whether DFixer's recommended commands eliminated those errors.

5.2 Evaluation Metrics

We first define three sets of error codes:

- *Intended Errors (IE):* Set of DNSSEC errors present in the target zone
- *Generated Errors (GE):* Set of DNSSEC errors present in the corresponding replicated zone by ZReplicator
- *After-fix Errors (AE):* Set of DNSSEC errors present after applying DFixer on the replicated zone

Given the above sets, we define two core metrics:

- Replication Rate (RR). We measure how often ZReplicator successfully reproduces *all* intended errors from the original snapshot[4].

$$\text{RR} = \frac{\text{\# of cases where IE} \subseteq \text{GE and IE} \neq \phi}{\text{\# of JSON where IE} \neq \phi}$$

  A higher RR implies comprehensive coverage of real-world misconfigurations.

- Fix Rate (FR). We also measure how often DFixer eliminates all generated errors in the replicated zone:

$$\text{FR} = \frac{\text{\# of cases where IE} \subseteq \text{GE and GE} \neq \phi \text{ and AE} = \phi}{\text{\# of cases where IE} \subseteq \text{GE and GE} \neq \phi}$$

  A higher FR indicates that DFixer's prescribed commands are both accurate and complete for the replicated scenarios.

5.3 Implementation and Environment:

We implement the entire pipeline inside Docker containers running Ubuntu 22.04. Each container hosts two `BIND` (v9.18) instances to emulate multi-server authoritative zones. We run these containers on a server with 38 CPU cores and 187 GB of RAM, processing 747K `DNSViz` JSON files within 36 hours. We will open-source our container images and orchestration scripts to encourage reproducibility and future DNSSEC research.

5.4 Experiment Results

We evaluate DFixer using 747K snapshots for SLD and lower level subdomains in our dataset; we summarize the result in Table 6. Among these, 296,813 contain at least one DNSSEC-related error. As shown in Table 3, the "Nonzero Iteration Count" (NZIC) error is disproportionately common compared to other issues. Because fixing NZIC is relatively straightforward (mostly involves with re-signing the zone with zero iterations), we divide the dataset into two subsets to give a more representative view:

- *S1*: Snapshots where NZIC is the *only* DNSSEC error.
- *S2*: All other snapshots, which contain either NZIC alongside other errors or completely different errors.

Replication Rate: From the 296,813 snapshots that have at least one DNSSEC-related error, we observe 2,058 unique error combinations. Our ZReplicator module successfully replicates 90.11% of these. In subset S1 (NZIC-only), ZReplicator achieves a near-perfect replication ratio of 98.81%. In contrast, S2 (which excludes NZIC-only scenarios) reflects more realistic conditions and attains a slightly lower replication ratio of 78.71%. For the remaining 21.29% $S2$ snapshots where ZReplicator failed to replicate all the intended errors ($IE \not\subseteq GE$), it was able to generate a subset of intended errors ($GE \subset IE$ and $GE \neq \phi$) in 67.18% cases, while for the remaining 32.82% cases, it could not generate any of the intended ones ($GE = \phi$). We discussed the reasons behind this in §5.5.

Fix Rate: DFixer achieves an overall fix rate of 99.99% across all snapshots, indicating that once a misconfiguration is reproduced, DFixer effectively resolves it. Even in the more complex S2 subset, DFixer's fix rate remains near-perfect; for the small fraction of S2 snapshots (5) that DFixer failed to resolve, we manually inspected them and found the parent zone to have a bogus state with `DS` records but no `DNSKEY`. In such scenarios, parent zone operators

---

[4] In certain cases, ZReplicator may produce benign companion errors alongside the errors in *IE* that does not have any bearing in DFixer's corrective instructions. For instance, to simulate a "Missing KSK for algorithm" error, ZReplicator creates a `DS` record referencing a non-existent key with an algorithm 'a' that is not present in the zone, rather than precisely matching the algorithm found in the DS record of the target zone. This can trigger "Missing Signature for Algorithm" error when no signatures for 'a' exist.

Instruction	1st **iteration**	2nd **iteration**	3rd **iteration**	4th **iteration**
Sign the zone	62,406 (41.67%)	13,845 (89.98%)	1,148 (62.19%)	7 (19.44%)
Remove the incorrect `DS` record	46,242 (30.87%)	1,319 (8.57%)	668 (36.19%)	29 (80.56%)
Upload the `DS` record	14,066 (9.39%)	117 (0.76%)	12 (0.65%)	-
Generate a `KSK`	13,148 (8.78%)	83 (0.54%)	-	-
Synchronize the DNS authoritative server	11,391 (7.61%)	-	-	-
Generate `ZSK`	1,491 (1.0%)	-	-	-
Reduce TTL of a specific record	947 (0.63%)	1 (0.01%)	-	-
Remove the revoked key	82 (0.05%)	22 (0.14%)	18 (0.98%)	-

Table 7: Instructions issued by DFixer during its iterative remediation process in the S2 subset (i.e., zones with more complex DNSSEC errors). This highlights how DFixer repeatedly removes incorrect DS records, re-signs zones, or updates keys until all misconfigurations are resolved.

must address their errors before the child zone administrators can proceed with DFixer.

Command Distribution in DFixer: DFixer uses an iterative remediation strategy, issuing multiple rounds of `BIND` commands until all DNSSEC errors are resolved (Figure 6). In our experiments, we find that *no zone required more than four iterations to become fully compliant*, and most errors were cleared by the second iteration-highlighting DFixer's efficiency even for interdependent misconfigurations. Table 7 details the most frequently applied instructions in the $S2$ subset (zones with non-trivial DNSSEC problems). Several insights stand out:

- *Single-Iteration vs Multi-Iteration Fix.* A single iteration is sufficient when all observed errors share one root cause. For example, a zone with a revoked KSK, missing DNSKEY signature, and an invalid `DS` referencing a non-existent KSK can be repaired in one pass by 1) generating a new KSK pair, 2) re-signing with the correct keys, 3) publishing the correct `DS`, 4) deactivating the bad `DNSKEY`, and 5) removing the incorrect `DS`. In this case, a single iteration suffices because all the observed errors share the same underlying cause: faulty KSK setup.
  On the other hand, zones with multiple independent errors typically require multiple iterations to achieve full resolution. For example, if a zone simultaneously exhibits a nonzero NSEC3 iteration count error and an extraneous `DS` record, DFixer will proceed incrementally: in the first iteration, it removes the extraneous `DS` record, and in the second, it re-signs the zone with an NSEC3 iteration count of zero.
- *Recurring Need for Zone Signing.* Unsurprisingly, "Sign the zone" dominates across iterations, accounting for 41.67% of all commands in the first pass, 89.98% in the second, and 62.19% in the third. Zone signing is inherently necessary whenever cryptographic material changes (e.g., after removing or adding keys, regenerating signatures).
- `DS` *Record Management is Second Most Common.* Removing incorrect `DS` records (30.87% in the first iteration and 8.57% in the second) is critical for fixing common misconfigurations, such as stale or mismatched DS entries. This finding corroborates prior work showing that nearly 30% of DNSSEC-signed domains struggle with DS-related errors [10]-often because DS maintenance must be coordinated through registrars [11].
- *Rapid Convergence in Most Cases.* Only a small fraction of domains requires a third or fourth iteration, typically when administrators must introduce a new key, retire an old one, or wait out TTLs. Even then, the majority of these multi-iteration scenarios still rely on the same recurring command set: DS updates, zone signing, and key generation or removal. Once the correct sequence is applied, the zone swiftly transitions to full compliance.

Overall, it illustrates that while DNSSEC debugging can appear daunting, particularly given the diverse error codes, a finite set of well-ordered, repeatable commands is sufficient to address the vast majority of misconfigurations.

5.5 Limitations of ZReplicator and DFixer

While ZReplicator and DFixer cover a broad range of DNSSEC errors, few constraints limit their scope:

5.5.1 ZReplicator.

- *Some zone-file errors remain unreproducible:* Among the 47 distinct DNSSEC error codes present in our dataset, a few could not be consistently replicated; for example, some negative-proof anomalies or a `DNSKEY` with an impossible bit length permitted by a buggy nameserver defy replication in our local environment, often because `BIND` or our signing utilities *refuse* to load blatantly invalid records. However, the proportion of these errors are relatively very low; in our dataset, only 2% snapshots have these errors. Because these errors are often intermingled with other replicable errors, they reduce the overall efficacy of ZReplicator.
- *Only leaf-zone replication is supported:* ZReplicator recreates a target zone and (optionally) its immediate parent, assuming full control of both. However, a domain may contain `CNAME` or `DNAME` records pointing to external zones beyond an operator's authority. For example, if `c.example.com` has a `CNAME` to `example.org`, ZReplicator treats `example.org` as out of scope. This design choice ensures we faithfully replicate *local* errors a zone owner can fix, without attempting to mirror unrelated external dependencies.
- *Algorithm-distribution constraints:* We attempt to preserve the DNSSEC algorithm usage from each original snapshot. In practice, some algorithms (e.g., DSA-NSEC3-SHA1) are no longer supported by `BIND`. When encountering such algorithms, we substitute them with available alternatives (e.g., RSASHA256, ECDSAP256SHA256), provided they are not already in use. A

small fraction of zones, however, exhaust all supported algorithms, making exact replication or even generation impossible.

5.5.2 DFixer.

- *Requires manual update of* `DS` *records:* In our evaluation of DFixer, we were able to automate everything including interaction with the parent zone by setting up a base zone under our control. In real-world deployments, however, such interactions would need to be performed manually. Although `CDNSKEY` and `CDS` [21] records could have been used for this purpose, their adoption remains very limited, so we chose not to rely on them. Additionally, some registrars use different formats for submitting `DS` records; for instance, AWS Route 53 requires uploading the public key instead. These edge cases are not covered in our instructions, as we follow the common patterns used by most registrars.
- *Optimality not guaranteed:* DFixer's command order is derived from a manually crafted dependency graph. While this typically yields few-iteration fixes, it is *not guaranteed to be globally minimal in every scenario.*
- *Unreplicated errors and the evaluation gap:* ZReplicator cannot reproduce a small set of anomalies (e.g., negative proofs and buggy-nameserver artifacts), so we could not empirically test DFixer on those cases. Nevertheless, our DFixer implementation addresses all error codes currently defined in `DNSViz`. For instance, all the unreproducible negative proof anomalies can be resolved through a straightforward zone re-signing while a DNSKEY with an invalid bit length can be handled by generating a valid key, removing the invalid one, and re-signing the zone. Based on this coverage, we argue that DFixer demonstrates a high degree of generalizability. However, we acknowledge the need for further empirical validation, which we plan to pursue by integrating DFixer with live `DNSViz` in future work.

Overall, these limitations reflect our primary goal of assisting domain owners in fixing *zone-level* DNSSEC errors. They do not diminish DFixer's utility for the majority of commonly observed misconfigurations, nor do they undercut ZReplicator's ability to reproduce most real-world scenarios where the zone file itself is at fault.

5.6 Testing Extensibility

Although our primary pipeline focuses on generating `BIND` commands, we also explore DFixer's applicability to other authoritative server software. DNSSEC operations—such as key generation, zone signing, and DS management—share common principles across implementations, suggesting that a direct mapping from `BIND`-centric commands to other server CLIs is often feasible. Below, we discuss how our manually curated commands can be adapted to `NSD` and `PowerDNS`.

NSD:. `NSD` is a widely used, high-performance authoritative DNS server [4]. To validate our approach, we manually map the `BIND` commands produced by DFixer to equivalent `NSD` workflows using the `ldns` utilities (e.g., `ldns-keygen`, `ldns-signzone`). We replicate each DNSSEC error code on two `NSD` instances (v4.8.0) and confirm that *every replicated error could be resolved through these mapped commands.* This outcome demonstrates that our error-to-command logic can be extended to `NSD` with minimal effort.

PowerDNS:. `PowerDNS` is another open-source authoritative server supporting multiple backends (e.g., PostgreSQL, MySQL, `BIND` files) and various DNSSEC modes (live-signing, pre-signed). Its live-signing mode generates signatures on the fly, which thwarted our attempts to inject controlled misconfigurations. We therefore used the `BIND` backend in pre-signed mode, replicating all misconfigurations except involving `NSEC` or `NSEC3` records due to a known `PowerDNS` bug [41]. We also built a one-to-one mapping from `BIND` to `pdnsutil`, but `PowerDNS` does not permit fixing errors in a pre-signed zone via `pdnsutil`. Therefore, we could not validate the effectiveness of our mapped commands in `PowerDNS`. As a workaround, we used the `BIND` commands generated by DFixer to fix the zone locally, then imported the repaired zone back into `PowerDNS`. Using this mechanism, we confirmed that DFixer was also able to successfully repair `PowerDNS` zones as well.

Knot DNS:. `Knot DNS` is another open-source high-performance authoritative DNS server [3]. To evaluate the potential applicability of DFixer to Knot, we investigated the feasibility of mapping our existing BIND commands and parameters set to Knot DNS environment. Our analysis indicates that this mapping can be achieved without significant difficulty. Specifically, operations such as key generation with configurable algorithm and size, as well as key retirement and removal, are supported through the `keymgr` utility. Furthermore, parameters related to NSEC/NSEC3 handling and signature lifetime management can be configured within the `policy` section of the `Knot DNS` configuration.

Takeaway: Any authoritative software that exposes fundamental operations such as zone signing, key generation, key activation or deactivation with basic parameter customization can host DFixer's repair plan with a thin translation layer.

6 Conclusion

Our large-scale analysis of `DNSViz` logs reveals a pronounced gap between DNSSEC's theoretical security and its operational reality. We address this gap with DFixer, which resolves `DNSViz`-reported errors to specific `BIND` commands. In local replication tests using an internally developed tool ZReplicator, DFixer succeeds in quickly repairing misconfigurations that might otherwise cause persistent failures. While certain limitations remain in ZReplicator, DFixer performance highlights how automated, domain-specific tooling can significantly reduce DNSSEC downtime and discouragement. By focusing on actionable fixes rather than mere diagnostics, we hope to advance DNSSEC adoption and strengthen global DNS security.

Acknowledgments

We thank anonymous reviewers and our shepherd for their helpful comments. This research was supported in part by NSF grant CNS-2339378 and Comcast Innovation Fund. We also especially thank DNS-OARC for their generous support in providing the DNSViz datasets.

References

[1] DNSSEC SCOREBOARD. https://www.verisign.com/en_US/company-information/verisign-labs/internet-security-tools/dnssec-scoreboard/index.xhtml.

[2] Internet Society. https://www.internetsociety.org/.

[3] KnotDNS. https://www.knot-dns.cz/.

[4] NSD. https://www.nlnetlabs.nl/projects/nsd/about/.

[5] Shell-AI: let AI write your shell commands. https://github.com/ricklamers/shell-ai.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, IETF, 2005. http://www.ietf.org/rfc/rfc4033.txt.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, IETF, 2005. http://www.ietf.org/rfc/rfc4035.txt.

[8] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034, IETF, 2005. http://www.ietf.org/rfc/rfc4034.txt.

[9] M. Brown. Calling time on DNSSEC: The costs exceed the benefits. 2023. https://www.mattb.nz/w/2023/06/02/calling-time-on-dnssec/.

[10] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. *USENIX Security*, 2017.

[11] T. Chung, R. van Rijswijk-Deij, D. Choffnes, A. Mislove, C. Wilson, D. Levin, and B. M. Maggs. Understanding the Role of Registrars in DNSSEC Deployment. *IMC*, 2017.

[12] Cloudflare. http://www.cloudflare.com.

[13] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra. Quantifying and Improving DNSSEC Availability. *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011.

[14] C. A. Daniluk, Y. Nosyk, A. Duda, and M. Korczynski. Zeros Are Heroes: NSEC3 Parameter Settings in the Wild. *IMC*, 2024.

[15] DNSSEC Debugger. http://dnssec-debugger.verisignlabs.com.

[16] DNSSEC Deployment Report. https://rick.eng.br/dnssecstat/.

[17] DNSSEC Statistics. https://www.internetsociety.org/deploy360/dnssec/statistics/.

[18] DNSSEC chain validation issue: technical incident report. https://internetnz.nz/news-and-articles/dnssec-chain-validation-issue-technical-incident-report/.

[19] DNSSEC issues take Fiji domains offline. https://blog.cloudflare.com/dnssec-issues-fiji/.

[20] DNSViz. http://dnsviz.net.

[21] A. Eden. Announcing CDS/CDNSKEY Support. 2019. https://blog.dnsimple.com/2019/02/cds_cdnskey/.

[22] D. Eastlake. Domain Name System Security Extensions. IETF RFC 2535, IETF, 1999.

[23] G. Huston. Measuring the use of DNSSEC. 2023. https://blog.apnic.net/2023/09/18/measuring-the-use-of-dnssec/.

[24] G. Huston. Calling time on DNSSEC? 2024. https://blog.apnic.net/2024/05/28/calling-time-on-dnssec/.

[25] W. Hardaker and V. Dukhovni. Guidance for NSEC3 Parameter Settings. RFC 9276, 9276, RFC Editor, 2022.

[26] W. Kumari, E. Hunt, R. Arends, W. Hardaker, and D. C. Lawrence. Extended DNS Errors. RFC 8914, RFC Editor, 2020.

[27] Key Management, DNSSEC Guide : Chapter 6. Advanced Discussions. https://dnsinstitute.com/documentation/dnssec-guide/ch06s04.html.

[28] B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155, IETF, 2008.

[29] E. Lewis. Where did DNSSEC go wrong? https://blog.apnic.net/2024/07/05/where-did-dnssec-go-wrong/.

[30] H. Lee, M. I. Ashiq, M. Muller, R. van Rijswijk-Deij, T. Kwon, and T. Chung. Under the Hood of DANE Mismanagement in SMTP. *USENIX Security*, 2022.

[31] M. Müller, M. Thomas, D. Wessels, W. Hardaker, T. Chung, W. Toorop, and R. van Rijswijk-Deij. Roll, Roll, Roll your Root: A Comprehensive Analysis of the First Ever DNSSEC Root KSK Rollover. *IMC*, 2019.

[32] M. Müller, W. Toorop, T. Chung, J. Janssen, and R. van Rijswijk-Deij. The Reality of Algorithm Agility: Studying the DNSSEC Algorithm Life-Cycle. *IMC*, 2020.

[33] Y. Nosyk, M. Korczyński, and A. Duda. Extended DNS Errors: Unlocking the Full Potential of DNS Troubleshooting. *IMC*, 2023.

[34] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. TRANCO: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. *NDSS*, 2019.

[35] S. Roth, R. van Rijswijk-Deij, and T. Chung. Tracking Registrar Support for DNSSEC: It's Slowly Getting Better. *IMC*, 2019.

[36] The Case of the Recursive Resolvers: What Happened During Slack's DNSSEC Rollout. https://slack.engineering/what-happened-during-slacks-dnssec-rollout/.

[37] The DNS Operations, Analysis, and Research Center (DNS-OARC). https://www.dns-oarc.net/oarc/programme.

[38] P. Wouters, O. Sury, and I. S. Consortium. Algorithm Implementation Requirements and Usage Guidance for DNSSEC. IETF, 2019.

[39] S. Weiler and D. Blacka. Clarifications and Implementation Notes for DNS Security (DNSSEC). RFC 6840, IETF Request for Comments 6840, IETF, Feb. 2013.

[40] Zonemaster. https://zonemaster.net/en/run-test.

[41] zone2sql and load-zone are not suitable for presigned zones; set-presigned is a useless button. https://github.com/PowerDNS/pdns/issues/8892.

7 Appendix

7.1 Ethics

Our dataset only contains public DNS information for a zone and does not have any personally identifiable information. For ZReplicator, we configured our replicated zones inside a local containerized environment, thus not facing public DNS servers. Additionally, during our experimentation with `probe` utility, we ensured to not query root and TLD servers by leveraging the `-x` and `-a` flag in `DNSViz`.

7.2 Sample DFixer Output

Figure 8 presents a sample remediation workflow suggested by DFixer when the target zone has the `REVOKED` bit set in its only KSK.

7.3 Effectiveness of LLM-Based Approach

Although large language models (LLMs) can be helpful for troubleshooting, our experiments show that prompt engineering alone is insufficient for guiding administrators to fully correct DNSSEC misconfigurations. To demonstrate this, we created several invalid zones using ZReplicator, analyzed them with DNSViz, and queried `GPT-4o` using the formats shown in Figure 9 and Figure 10. We

Description

I'm encountering issues during the process of enabling DNSSEC for my domain. I ran a check with `dnsviz.net` for debugging, which highlighted several configuration errors, but the root cause of these problems is unclear to me. I have attached the debugging analysis data and visualization diagram from DNSViz in JSON and PNG format respectively. I have also copied and pasted the unique error messages from DNSViz.
<error messages>
<attachments>

Question

Given the files and messages from DNSViz, can you assist me in understanding the specific points of failure, and determining the root cause of the misconfigurations? Give me the identified root cause(s) in concise and short bullet points.

Directives

- Be as specific as possible with variable parameters such as DNSKEY algorithms, key_tags etc. For example, if your root cause is a DNSKEY with REVOKED flag on, please associate the key_tag of the DNSKEY you are referencing.
- Do not report any misconfigurations that belong to the ancestor zones.
- If there are multiple root causes, list them in order of importance, with the most critical issues appearing first.
- For privacy, please do not refer to the domain name directly.

Figure 9: Prompt format used for root cause identification

Question

For each of the identified root cause(s), please provide specific instructions on how to resolve them.

Directives

Extract the variable parameters such as DNSKEY algorithms, key_tags, NSEC or NSEC3 usage, DS digest type etc. from the current zone file settings when possible. For example, if I need to resign the zone, specify NSEC if the zone is currently using NSEC and NSEC3 otherwise.

Figure 10: Prompt formats used for devising remediation plan with commands

Root Cause: Your DS record is linked to a DNSKEY (key_tag=k) with REVOKED flag on.
Remediation Plan with BIND commands: Replace the path variables in angle brackets with values of your own environment. Parameters in braces and variables will be automatically populated by DFixer.

(1) Generate a new KSK key pair. Execute: `cd <key_dir> && dnssec-keygen -f KSK -a {algo} -b {key_size} -n ZONE {zone}`. This command should create two new key files inside your key directory; please note the name of the public key file (with `.key` extension).
(2) Generate the DS record from the generated public key file. Replace `public_key_file` with the name of the public key file from previous step and execute: `cd <key_dir> && dnssec-dsfromkey -2 <public_key_file>`. This command should show the contents of your DS record in standard output.
(3) Upload the DS record to the parent zone. This must be done manually via your registrar.
(4) Remove the DS record linked to the revoked DNSKEY (key_tag=k) from the parent zone. This also needs to be done manually via your registrar.
(5) Wait at least one full TTL (ttl) for the removed DS record to expire from the cache of any validator. Nothing to execute; in "auto-apply" mode, DFixer will automatically infer the TTL and wait out this period before executing the next command.
(6) After ttl seconds, delete the DNSKEY (key_tag=k, pre-revoked key_tag=pk) from your zone. Replace `key_file` with the name of the public key file associated with key_tag=k, and execute: `dnssec-settime -D {current_time} <key_dir/key_file>`
(7) Re-sign the zone. Execute: `cd <key_dir> && dnssec-signzone -N INCREMENT -3 {salt} -S -o {zone} -t <zone_dir/unsigned_zone_file>`

Figure 8: Sample remediation workflow produced by DFixer for a domain with its only KSK revoked and linked to a DS record

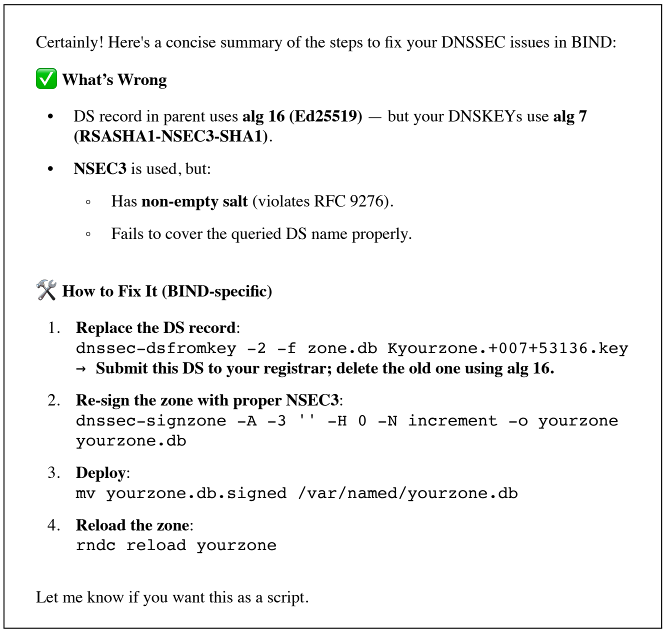
present representative results from one particularly problematic test case here.

In this test zone, we introduced two major errors:

(1) *Extraneous* DS *records* using an algorithm for which no DNSKEY exists.
(2) *A bad non-existence proof* in the parent zone, designed to test whether the LLM can distinguish child from parent zone issues.

Figure 11 shows GPT-4o's analysis response when prompted per Figure 9. While the LLM correctly identified the main problem, we observed several limitations:

(1) It did not specify *which* DS records were invalid.
(2) It conflated errors between the child and parent zones.
(3) It failed to rank or prioritize the issues.



Certainly! Here's a concise summary of the steps to fix your DNSSEC issues in BIND:

✅ **What's Wrong**

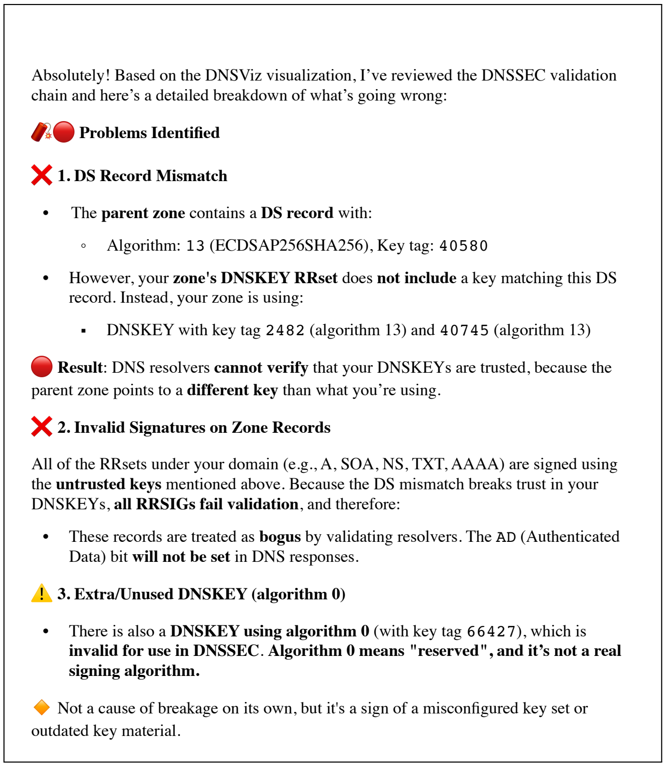
- DS record in parent uses **alg 16 (Ed25519)** — but your DNSKEYs use **alg 7 (RSASHA1-NSEC3-SHA1)**.
- **NSEC3** is used, but:
    - Has **non-empty salt** (violates RFC 9276).
    - Fails to cover the queried DS name properly.

🛠️ **How to Fix It (BIND-specific)**

1. **Replace the DS record**:
   ```
   dnssec-dsfromkey -2 -f zone.db Kyourzone.+007+53136.key
   ```
   → Submit this DS to your registrar; delete the old one using alg 16.
2. **Re-sign the zone with proper NSEC3**:
   ```
   dnssec-signzone -A -3 '' -H 0 -N increment -o yourzone
   yourzone.db
   ```
3. **Deploy**:
   ```
   mv yourzone.db.signed /var/named/yourzone.db
   ```
4. **Reload the zone**:
   ```
   rndc reload yourzone
   ```

Let me know if you want this as a script.

Figure 12: Response from `GPT-4o` when passed the prompt in Figure 10 for our invalid setup



Absolutely! Based on the DNSViz visualization, I've reviewed the DNSSEC validation chain and here's a detailed breakdown of what's going wrong:

🧨🔴 **Problems Identified**

❌ **1. DS Record Mismatch**

- The **parent zone** contains a **DS record** with:
    - Algorithm: `13` (ECDSAP256SHA256), Key tag: `40580`
- However, your **zone's DNSKEY RRset** does **not include** a key matching this DS record. Instead, your zone is using:
    - DNSKEY with key tag `2482` (algorithm 13) and `40745` (algorithm 13)

🔴 **Result**: DNS resolvers **cannot verify** that your DNSKEYs are trusted, because the parent zone points to a **different key** than what you're using.

❌ **2. Invalid Signatures on Zone Records**

All of the RRsets under your domain (e.g., A, SOA, NS, TXT, AAAA) are signed using the **untrusted keys** mentioned above. Because the DS mismatch breaks trust in your DNSKEYs, **all RRSIGs fail validation**, and therefore:

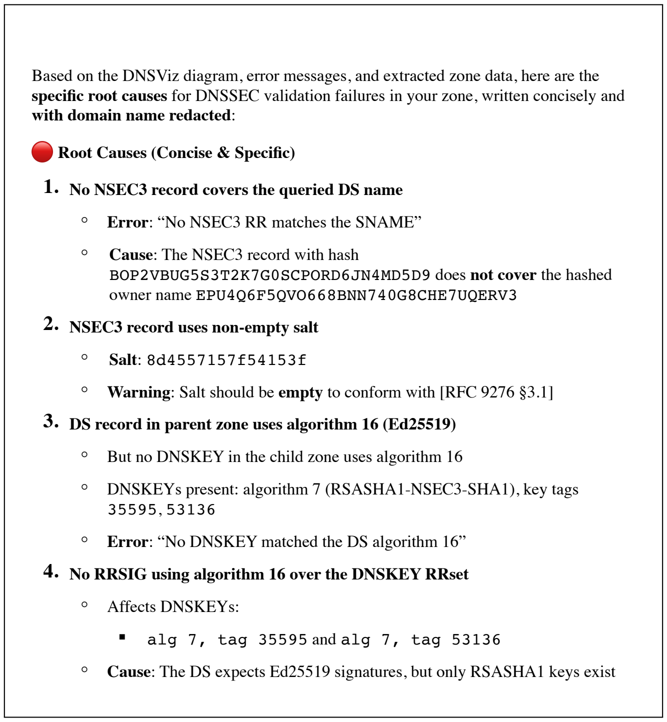
- These records are treated as **bogus** by validating resolvers. The `AD` (Authenticated Data) bit **will not be set** in DNS responses.

⚠️ **3. Extra/Unused DNSKEY (algorithm 0)**

- There is also a **DNSKEY using algorithm 0** (with key tag `66427`), which is **invalid for use in DNSSEC**. **Algorithm 0 means "reserved", and it's not a real signing algorithm.**

🔶 Not a cause of breakage on its own, but it's a sign of a misconfigured key set or outdated key material.

Figure 13: A hallucination example from `GPT-4o` when passed the DNSViz visualization for our invalid setup



Based on the DNSViz diagram, error messages, and extracted zone data, here are the **specific root causes** for DNSSEC validation failures in your zone, written concisely and **with domain name redacted**:

🔴 **Root Causes (Concise & Specific)**

1. **No NSEC3 record covers the queried DS name**
    - **Error**: "No NSEC3 RR matches the SNAME"
    - **Cause**: The NSEC3 record with hash `BOP2VBUG5S3T2K7G0SCPORD6JN4MD5D9` does **not cover** the hashed owner name `EPU4Q6F5QVO668BNN740G8CHE7UQERV3`
2. **NSEC3 record uses non-empty salt**
    - **Salt**: `8d4557157f54153f`
    - **Warning**: Salt should be **empty** to conform with [RFC 9276 §3.1]
3. **DS record in parent zone uses algorithm 16 (Ed25519)**
    - But no DNSKEY in the child zone uses algorithm 16
    - DNSKEYs present: algorithm 7 (RSASHA1-NSEC3-SHA1), key tags `35595, 53136`
    - **Error**: "No DNSKEY matched the DS algorithm 16"
4. **No RRSIG using algorithm 16 over the DNSKEY RRset**
    - Affects DNSKEYs:
        - `alg 7, tag 35595` and `alg 7, tag 53136`
    - **Cause**: The DS expects Ed25519 signatures, but only RSASHA1 keys exist

Figure 11: Response from `GPT-4o` when passed the prompt in Figure 9 for our invalid setup

(4) It presented additional warnings, contrary to our request for concise root-cause analysis.

Figure 12 shows the model's suggested remediation plan (following Figure 10). Unfortunately, the guidance and commands were partially incorrect:

(1) It advised "replacing" the `DS` record, even though valid one was already in place; the correct and minimal fix was just to remove the extraneous DS entries at the parent zone.

(2) It recommended re-signing the zone, which was irrelevant since the primary issue lay in the parent's NSEC3 records.

(3) The proposed `signzone` commands lacked essential parameters.

Moreover, when we presented `GPT-4o` with a screenshot of the DNSViz output (Figure 13), the model *hallucinated* specific key_tags and algorithms, further undermining its reliability. These observations underscore that while LLMs can offer partial assistance, they often struggle with nuanced DNSSEC errors and require additional domain-specific logic to be truly effective.