

# The Design of Metrics for Quantifying the DNSSEC Deployment

Eric Osterweil, Dan Massey, Lixia Zhang,

**Abstract**—This paper examines the deployment of the DNS Security Extensions (DNSSEC), which adds cryptographic protection to DNS, one of the core components in the Internet infrastructure. We analyze the data collected from the initial DNSSEC deployment which started in 2005, and identify three critical metrics to gauge the deployment: availability, verifiability, and validity. Our results provide the first comprehensive look at DNSSEC’s deployment and reveal a number of challenges that were not anticipated in the design but have become evident in the deployment. First, obstacles such as middle-boxes (firewalls, NATs, etc.) that exist in today’s Internet infrastructure have proven to be problematic and have resulted in unforeseen *availability* problems. Second, the public-key delegation system of DNSSEC has not evolved as it was hoped and it currently leaves over 98.5% of DNSSEC zones isolated and *unverifiable*, unless some external key authentication mechanism is added. Third, our results show that cryptographic verification is not equivalent to *validation*; a piece of verified data can still contain the wrong value. Finally, our results demonstrate the essential role of monitoring and measurement in the DNSSEC deployment. We believe that the observations and lessons from the DNSSEC deployment can provide insights into measuring future Internet-scale cryptographic systems.

## I. INTRODUCTION

Security is a fundamental challenge facing the Internet, and cryptographic technologies are generally viewed as a powerful tool-set for addressing security challenges. There have been a number of efforts to retrofit existing protocols with cryptographic protection [4], [6], [5], [9], [14], [20], and one clear lesson that has emerged from these efforts is that adding cryptographic protection to existing systems tends to be difficult. This is especially true for *Internet-scale* systems. Internet-scale systems are large in size as measured by the number of their components, which belong to a large number of *independent* administrative authorities without any central control. Yet deploying a cryptographic protection in an Internet-scale system means that the mechanism needs to be deployed across the entire Internet and can be used by all desired parties in a cohesive manner.

In this paper we examine the DNS Security Extensions (DNSSEC)[4], [6], [5]. The DNSSEC protocol set is considered mature and its global deployment efforts started

E. Osterweil and L. Zhang are with the Department of Computer Science, University of California, Los Angeles, Los Angeles, CA 90095 USA.

D. Massey is with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523 USA

in 2005. The SecSpider monitoring project[3] has been tracking the DNSSEC deployment since shortly after the rollout began. Its public site tracks the number of secured DNS zones as viewed from diverse locations around the globe. It allows one to determine whether a particular zone has turned on DNSSEC and also tracks the evolution of zone-specific operational decisions, such as the choice of public keys and signature lifetimes. Live data has been available for a few years and historical data dating back to the first few months of DNSSEC deployment is also available.

To quantify both the effectiveness of cryptographic protection that early DNSSEC adopters may gain and the obstacles in DNSSEC’s deployment, we analyze the collected DNSSEC monitoring data using three measurement metrics: *availability*, *verifiability*, and *validity*. Our measurement and analysis show that there are a number of challenges that were not anticipated in the design but have become evident in the deployment. First, middleboxes, such as firewalls and NATs, that exist in today’s Internet infrastructure have proven to be obstacles in DNSSEC rollout and have resulted in unforeseen *availability* problems. Second, the public-key delegation system in the DNSSEC design has not evolved as it was hoped and it currently leaves more than 98.5% of DNSSEC-enabled zones isolated and *unverifiable*, unless some external key authentication mechanism is added. Third, our results show that cryptographic protection has its own limitations. That is, cryptographic verification is not equivalent to *validation*; a piece of cryptographically verified data can still contain incorrect value.

Our contributions in this paper are three-fold. First, based on our observations of the current DNSSEC deployment, we derive three basic metrics to quantify the effectiveness of DNSSEC’s deployment. Earlier measurement results reported in [16] provided some basic observations including the number of existing DNSSEC zones, the operational practice in managing cryptographic deployment, and the existence of DNSSEC data that is vulnerable to replay attacks; this paper not only reports more recent measurement data, but most importantly, the newly defined metrics enable us to *quantify* the observed problems in a meaningful way. Second, our measurement results expose previously undocumented open issues in the DNSSEC deployment. Third, our results demonstrated the essential role of monitoring and measurement in the DNSSEC deployment. We believe

that the observations and lessons reported in this paper can provide insights into the challenges in developing future Internet cryptographic systems.

The remainder of this paper is organized as follows. Section II discusses the general design of DNSSEC. Next, in Section III we describe our basic approach to monitoring and quantifying the deployment of DNSSEC. In Section IV we present the quantitative results of analyzing DNSSEC. Lastly, we discuss our findings and conclusions in Section V.

## II. BACKGROUND

The Domain Name System (DNS) [12] maps hostnames such as `www.ucla.edu` to IP addresses and provides a wide range of other mapping services ranging from email to geographic location. Virtually every Internet application relies on looking up certain DNS data. In this section we introduce a basic set of DNS terminology which is used throughout the text, including resource records (RRs), resource record sets (RRsets), and zones, followed by an overview of the DNS Security Extensions.

All DNS data is stored in the same data structure called *Resource Records* (RRs), and each RR has an associated name, class, and type. For example, an IPv4 address for `www.ucla.edu` is stored in an RR with name `www.ucla.edu`, class IN (Internet), and type A (IPv4 address). A host with several IPv4 addresses will have several RRs, each with the same name, class, and type but its own IPv4 address. The set of *all* resource records associated with the same name, class, and type is called an *Resource Record Set* (RRset). DNS resolvers query for RRsets. For example, when a browser queries for  $\langle \text{www.ucla.edu}, \text{IN}, \text{A} \rangle$ , the reply will be the RRset for `www.ucla.edu` with all the IPv4 addresses for that name. Note that the smallest unit that can be requested in a query is an RRset, and all DNS actions including cryptographic signatures, discussed later, apply to RRsets instead of individual RRs.

The DNS is a distributed database organized in a tree structure. At the top of the tree, the root zone delegates authority to *top level domains* like `com.`, `net.`, `org.`, and `edu.`. The zone `com.` then delegates authority to create `google.com.`, `edu.` delegates authority to create `ucla.edu.`, and so forth. In the resulting DNS tree structure, each node corresponds to a *zone*. Each zone belongs to a single administrative authority and is served by multiple *authoritative nameservers* to provide name resolution services for all the names in the zone. Every RRset in the DNS belongs to a specific zone and stored at the nameservers of that zone. For example, the RRset for  $\langle \text{www.ucla.edu}, \text{IN}, \text{A} \rangle$  belongs to the `ucla.edu` zone and stored in the `ucla.edu` nameservers; while the RRset for  $\langle \text{www.colostate.edu}, \text{IN}, \text{A} \rangle$  belongs to the `colostate.edu` zone and stored in the `colostate.edu` nameservers.

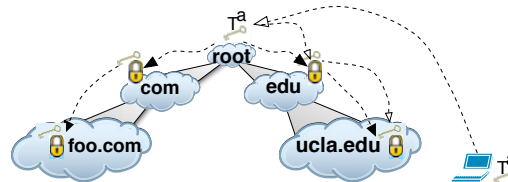


Fig. 1. Resolvers preconfigure the root zone’s public key as a trust anchor ( $T^a$ ) and can then trace a “chain of trust” from that key down the DNSSEC hierarchy to any zone’s key that they have encountered.

### A. DNSSEC Overview

Security was not a primary objective when the DNS was designed in mid 1980’s and a number of well known vulnerabilities have been identified [8], [7]. DNSSEC provides a cryptographic solution to the problem, which seems pretty simple and intuitive. To prove that data in a DNS reply is authentic, each zone creates public/private key pairs and then uses the private portions to sign data. Its public keys are stored in a new type of RR called DNSKEY, and all the signatures are stored in another new type of RR called RRSIG. In response to a query, an authoritative server returns both the requested data and its associated RRSIG RRset. A resolver that has learned the DNSKEY of the requested zone can verify the *origin authenticity* and integrity of the reply data. To resist replay attacks, each signature carries a definitive expiration time.

Let  $N_i$  define the set of authoritative name servers for zone  $z_i$  which are online and reachable. When the zone is operating correctly, it does not matter which of the servers in  $N_i$  handles a query and the answer can be cached. However, the design does assume that the resolver can obtain and authenticate the zone’s DNSKEY.

In order to authenticate the DNSKEY for a given zone, say `www.ucla.edu`, the resolver needs to construct a *chain of trust* that follows the DNS hierarchy from a trusted root zone key down to the key of the zone in question (this is shown in Figure 1). In the ideal case, the public key of the DNS root zone would be obtained offline in a secure way and stored at the resolver, so that the resolver can use it to authenticate the public key of `edu.`; the public key of `edu.` would then be used to authenticate the public key of `ucla.edu.`

There are two challenges in building the chain of trust. First, a parent zone must encode the authentication of each of its child zone’s public keys in the DNS. To accomplish this, the parent zone creates and signs a Delegation Signer (DS) RR that corresponds to a DNSKEY RR at the child zone, and creates an authentication link from the parent to child. It is the child zone’s responsibility to request an update to the DS RR every time the child’s DNSKEY changes. Although all the above procedures seem simple and straightforward, one must keep in mind that they are performed manually, and people inevitably make errors, especially when handling large zones that have hundreds

Symbol	Description
$r$	A resource record
$k$	A DNSKEY
$s^k$	An RRSIG (verifiable by $k$ )
$R$	An RRset defined as $(r_0, \dots, r_i, s_0, \dots, s_j)$
$R^s$	A secure $R$ with one or more $s^k$
$K$	A DNSKEY RRset defined as $(k_0, \dots, k_i, s_0^{k_0}, \dots, s_j^{k_i})$
$T^a$	A $k$ trust anchor
$z$	A secure zone defined as $(R_0^s, \dots, R_i^s, K)$
$N_i$	The set of online nameservers for a zone $z_i$
$Z^s$	The set of all secure zones

TABLE I  
DEFINITION OF TERMS

or thousands of children zones.

Moreover, the parent and child zones belong to *different* administrative authorities, each may decide independently is and when they turn on DNSSEC. This leads to the second and more problematic challenge. If the parent zone is not signed, there is no chain of trust leading to the child zone's DNSKEY. This orphaned key effectively becomes an isolated trust anchor for its subtree in the DNS hierarchy. To verify the data in these isolated DNSSEC zones, one has to obtain the keys for such isolated trust anchors offline in a secure manner. DNSSEC resolvers maintain a set of well-known "trust-anchor" keys ( $T^a$ ) so that a chain of key sets + signatures (secure delegation chain) can be traced from some  $T^a$  to a DNSSEC key  $K$  lower in the tree. The original DNSSEC design envisioned that the its deployment would be rolled out in a top-down manner. Thus only the root zone's  $K$  would need to be configured in all resolvers'  $T^a$  sets and all secure delegations would follow the existing DNS hierarchy. However without the root and top level domains deploying DNSSEC (as is the case today) there could be potentially millions of isolated trust anchors. In fact various approaches have been proposed for securely obtaining these trust anchors.

**Secure Resolution Procedure:** For illustrative purposes, Algorithm 1 describes the verification process DNS-SEC resolvers follow when contacting and querying zones for DNSSEC data. Table I summarizes the notations used in describing a secure zone.

### III. MONITORING AND MEASUREMENT

Although the DNSSEC deployment is still in its early stages, our measurement results have already revealed some key challenges that face Internet cryptographic systems. The SecSpider monitoring project has been operating since major DNSSEC deployment efforts began in 2005. During that time, the number of monitored zones has grown from tens to tens of thousands. While this is a small fraction of the World's DNS zones, the set continues to grow rapidly.

---

#### Algorithm 1: Resolution algorithm for DNSSEC

---

```

Data: Given  $N_i$ 
Input: Query ( $Q$ ) for "www.foo.bar" from foo.bar ( $z_i$ )
begin
  if get  $K_i$  from  $N_i$  then
    if able to trace chain from known  $T^a$  then
      send  $Q$  to one of  $N_i$ 
      if  $\exists R^s \in z_i$  such that  $Q \in R^s$  AND verify signatures in  $R^s$  against  $K_i$  then
        Data is verified
      else
        Unable to verify data
      else
        Key cannot be verified as  $z_i$ 's
    else
      No data for  $z_i$  can be verified
  end

```

---

Despite the relatively small size of DNSSEC's initial rollout, it is quite complex and scaling the monitoring is already a major concern. The number of records in a zone can vary from tens to millions and each DNS zone is served by several authoritative servers. Thus, even a single zone is, essentially, a distributed view of a dataset. This fact raises interesting questions about what to monitor and how to quantify the results in a way that provides both insight to general trends and an ability to analyze specific issues.

In addition to the distributed nature of a zone, the view of its data also varies depending on the location that it is monitored from. In the simplest case, connectivity issues may prevent some monitoring points (called pollers) from reaching a zone's servers. Although one may assume that connectivity problems are rare, middleboxes (such as firewalls, NATs, or proxies) are pervasive today. Such middleboxes along the paths between pollers and the authoritative servers, and other qualities of the path itself, can dramatically change the view of a zone. Our data analysis in later sections will illustrate the impact of middleboxes; for the moment it suffices to say that the location of monitoring is an important factor. During the course of deploying our monitoring system, our set of pollers has expanded from a single poller to a collection of distributed pollers in different continents, and this has offered fundamentally different views of some DNSSEC datasets.

Our monitoring system gathers a vast volume of data on DNSSEC resource record sets (RRsets) as viewed from different locations over different times. The detailed data is publicly available and it allows people to investigate specific questions such as "was RRset  $X$  available from pollers in Asia on January 31, 2008?" But the vast volume of raw data is complex and even its volume can potentially mask any insight into how the overall system is performing.

The situation is analogous to monitoring BGP routing, another Internet-scale system. BGP monitoring projects such as Oregon RouteViews [15] provide invaluable raw data containing millions (if not billions) of BGP updates. Hidden in this data are important lessons about the overall system behavior, but simply looking at raw BGP update logs does not directly answer the question of how well BGP is performing. Similarly, simply presenting millions of DNS RRset query results does not directly answer the question of how well DNSSEC is performing, how effective it may be in providing cryptographic protections for the DNSSEC-enabled zones, and more importantly, how these measures may be changing over time.

In order to gain a *quantitative* assessment of DNSSEC as a whole, we derived the following three measures:

- **Availability:** This measures whether the system can provide all the data to the end-systems requesting it.
- **Verifiability:** This measures whether the end-system can cryptographically verify the data it receives.
- **Validity:** This measures whether the verified data is actually valid. Note that in an actual deployments, it does not necessarily follow that all *verified* data is indeed *valid*.

In the following three subsections, we formalize each measure into a metric and detail our approach for quantifying the results.

#### A. Availability

Intuitively, one would like to know whether a secure zone is “available.” This is an important 2-way street because zone operators need to be aware of any problems resolvers may have in receiving their data, and resolvers would like to know why they may be unable to get certain data from a zone. However, before one can know if a zone is “available,” one must define what that means. Thus, we define a measure that captures the intuitive notion of availability, but in quantifiable metric. In Section IV, we show how this metric is effective in quantifying DNSSEC’s non-uniformity. Such a quantification should facilitate an empirical way to measure improvements in DNSSEC’s deployment.

**Selecting RRsets and Nameservers:** A secure zone is comprised of a set of RRsets. The number of these sets in a zone can range from fewer than ten to over tens of millions, but the DNSKEY set (key-set) plays a special role in DNSSEC. The key-set holds the zone’s public key(s), and by definition, every secure zone must contain an instance of this set at its apex. The keys in the key-set are required to verify other RRsets and are also needed to verify authentication chains to descendant zones in the DNSSEC hierarchy (Figure 1). Without getting the key-set, even if a resolver can obtain other RRsets, it cannot verify them. We thus argue that for the purpose of DNSSEC, zone availability can be reasonably represented by the availability of the key-set itself.

Having identified a specific set to monitor, we next consider which of the many authoritative servers of a zone to query. Work in [18] has shown that, due to various configuration errors, different servers of the same zone may exhibit different behaviors. For example, some may be listed as authoritative but actually fail to answer queries, and some other servers may be authoritative but unreachable. In this paper we focus on whether a zone’s key-set is available via *any authoritative server*<sup>1</sup>.

**Availability Metric:** Given a set of *pollers* ( $P$ ) who send queries at a set of given *polling times* ( $T$ ), we denote the availability of zone  $z_j$  from poller  $p_i \in P$  at time  $t \in T$  as  $A(p_i, z_j, t)$ . In this paper,  $A(p_i, z_j, t)$  is either 1 to indicate the poller was able to obtain  $z_j$ ’s key-set ( $K_j$ ), or 0 to indicate it could not be retrieved. For example,  $A(p_1, z_1, t)$  is set to 1 if poller  $p_1$  could obtain  $K_1$  from zone  $z_1$  at time  $t$ . Similarly  $A(p_2, z_1, t)$  is set to 0 if poller  $p_2$  could not obtain the  $K_1$  from zone  $z_1$  at time  $t$ .

This metric is designed to allow a more nuanced definition of availability in which the value can vary *between* 0 and 1. For example, one might include representations for nameserver availability, a combination of multiple RRsets, the agreement ratio between a zone’s nameservers, or other facets. However in this paper, a simple definition of availability suffices and we set the value of  $A(p_i, z_j, t)$  to either 1 ( $K_j$  obtained) or 0 ( $K_j$  not obtained).

Having defined zone availability for a particular poller at a particular time  $t$ , we combine the results from multiple pollers to obtain a single for the zone availability at time  $t$ . Let  $A_{max}(z_j, t) = \max_{i=0, |P|} A(p_i, z_j, t)$  denote the highest availability metric obtained for any poller. This value represents the best observed view of availability for this zone at this time. We say a zone  $z_j$  is *available* at time  $t$  iff  $A_{max}(z_j, t) > 0$ . Our later results show that a vast majority of DNSSEC zones were “available” during polling times  $\in T$ .

**Availability Dispersion Metric:** While the above definition of availability focuses on whether *some* resolver (represented by  $p_i$ ) can reach a zone, we are also interested in describing *how many* resolvers can reach the zone. Our later results show that in many cases, even though some resolvers can reach a zone, others cannot. If a zone is available, the variance in availability is quantified as *availability dispersion*. More precisely, we denote the zone  $z_j$ ’s availability dispersion at time  $t$  as:

$$disp(z_j, t) = \frac{\sum_{i=0}^{|P|} A_{max}(z_j, t) - A(p_i, z_j, t)}{|P|}$$

The intuition for the dispersion metric first considers the zone’s  $A_{max}(z_j, t)$ . All other pollers are compared against this best case and pollers with lower availability increases the dispersion. For example, if all pollers see a zone as

<sup>1</sup>SecSpider is able to detect the problems reported in [18]. However, this is beyond the scope of this paper.

available the dispersion will be 0. Furthermore, if we take the limit as the number of polling locations approaches the total number of resolvers on the Internet, we can see that the availability dispersion approaches the mean behavior for resolvers.

There is a clear difference between the polling failures that stem from persistent availability issues and those that represent transient network problems. The availability dispersion metric is designed to address the former (persistent problems). In the case of the latter (transient problems), the timeout/retry strategy of the monitoring apparatus is useful in attempting to overcome failures. The specific timeout/retry strategy used is described in more detail in Section IV.

Recall our metric is designed for  $A(p_i, z_j, t)$  values that range between 0 and 1, but this paper considers only values of 0 and 1 and thus  $disp()$  can be simplified. Since dispersion is only calculated if at least one poller can reach the zone ( $A_{max}(z_j, t) = 1$ ), any other poller that can reach the zone will not contribute to the numerator in dispersion ( $A_{max}(z_j, t) - A(p_i, z_j, t) = 0$ ), but  $A(p_i, z_j, t) = 0$  will contribute 1 to the numerator ( $A_{max}(z_j, t) - A(p_i, z_j, t) = 1$ ). Therefore, our dispersion calculation simplifies to the average number of failed pollers.

Next, we take the instantaneous metrics and apply an Exponentially Weighted Moving Average (EWMA) to obtain:

$$\overline{disp}(z_j) = (\alpha \times disp(z_j)) \times \left( (1 - \alpha) \times \overline{disp}(z_j) \right)$$

EWMA incorporates the history of dispersion without over-penalizing zones who are normally available but were not at the time of a recent poll and without being overly charitable to zones that are normally unavailable, but who were available at the time of the last poll. Thus, while the timeout/retry strategy helps to overcome some transient problems, the actual dispersion metric also reduces their effect (if they are indeed only transient).

Because high-dispersion indicates potential problems, while low or no dispersion represents that the effect on availability is uniform, we take the *complement* of the average availability dispersion to reflect the Internet's effect on availability:

$$avail_{disp}(z_j) = (1 - \overline{disp}(z_j))$$

## B. Verifiability

The previous section presented a metric for assessing the *availability* of DNSKEY RRsets (key-sets) and by extension the zones that serve them. But simply accepting key-sets without any verification defeats the underlying purpose of adding cryptography. DNSSEC was introduced because resolvers may receive incorrect responses caused by unintentional errors or intentional attacks. Even using key-sets can leave a resolver vulnerable if a man-in-the-middle attack allows an adversary to give a resolver a bad

key [8], [17]. Thus a resolver must be able to verify key-sets and this section introduces a metric that captures the intuitive notion of whether this can be done.

To verify any data, a resolver must be configured with some initial set of keys from trusted zones, referred to as *trust anchors*. Figure 1 illustrates this process.

If all zones were secure and each secure zone coordinated with its parent in the DNS tree, then resolvers would only need to be configured with a single trust anchor, corresponding to the root public key. However, not all zones are secure and not all secure zones coordinate with their parents in the DNS tree. The result is that there are gaps in the authentication chain and these gaps must be bridged by adding additional trust anchors. In the worst case, there could be no authentication chains and a resolver would need to be configured with a trust anchor corresponding to each zone (which would be tens of millions for a full deployment). In the ideal case, resolvers are configured with a single trust anchor. To quantify where the current deployment stands, we introduce a *verifiability* metric that captures the amount of configuration needed to verify key-sets.

**Verifiability Metric:** Let  $T^a$  denote a trust anchor. We say the key-set ( $K_i$ ) for zone  $z_i$  is covered by trust anchor  $T^a$  iff there is an authentication chain leading from  $T^a$  to  $z_i$ . If  $|Z^s|$  denotes the number of total secure zones and  $|T^a|$  is the minimum number of trust anchors needed to cover all secure zones then we say the overall verifiability of the system is:

$$V^f = 1 - \frac{|T^a| - 1}{|Z^s|}$$

The intuition for this expression comes from DNSSEC's goal of a single trust anchor. Thus, the expression accepts a single trust anchor as optimal (hence the  $-1$  term), and penalizes all instances above 1. Note that if no authentication chains had been established between any secure zones, a resolver would need to configure  $K_i$  for each zone  $z_i \in Z^s$  as a trust anchor, and  $V^f \rightarrow 0$ . If DNSSEC is deployed in a contiguous region of the DNS tree and all zones in this region establish authentication chains with their direct parent, then we will only need a single  $T^a$  and  $V^f \rightarrow 1$ .

## C. Validity

The previous sections considered whether zones' critical DNSKEY RRsets (key-sets) were *available* to resolvers and how much configuration was needed to *verify* these key-sets. This section considers whether data is actually *valid* and illustrates that there are key differences between verified data and valid data. More specifically, verification refers the cryptographic process in which a data unit is either verified or not. Validity, on the other hand, refers to whether the data actually corresponds to what the zone administrator *intended* (ground truth) and a data unit is either valid or invalid. Based on the overlapping intents

	Verified	Unverified
Valid	Ideal Behavior	False Negative
Invalid	False Positive	Intended Defense

TABLE II  
VERIFICATION VS VALIDITY MATRIX.

of verification and validity there are four possible combinations, which are shown in Table II. Our validity metric ( $V^d$ ) focuses on the validity of DNSSEC data.

Ideally, data obtained by a resolver is both valid and verified (upper left box in Table II). For example, if a zone administrator correctly enters and signs zone data, then a resolver should be able to obtain and verify this valid data. DNSSEC adds cryptographic checks in the hope of detecting invalid data by using cryptography alone (lower right box in Table II). For example, if someone modifies data in flight (after being signed), then the old values become invalid and one expects that the signature verification will fail. This is the intended behavior of DNSSEC but both operational errors and successful attacks can cause this to fail.

**False Negatives:** The case of false negatives (upper right box in Table II) occur when a resolver gets data that is actually valid, but is unable to verify it. The most trivial case of this is when a resolver receives plain DNS responses, but finds that there are *no* signatures attached. This, for example, was observed during the early DNSSEC development. Some sites could not obtain signed data from secure zones even though the server correctly attached the signatures. This problem was caused by intervening firewalls that blocked any response that contained signatures. From the firewall’s perspective: the resolver had made a simple request but the response *also included the signatures*. To the well intentioned firewall, these unknown signature records were clearly some sort of attack and the responses were dropped. Answers that did not include signatures were passed through, but could not be verified by the resolver.

Another important example of false negatives can occur when a zone unintentionally breaks its own secure delegation from its parent. This can happen if a zone creates a new key pair and re-signs all RRsets with the *new* key before updating the authentication chain with the parent. Specifically, this is when the child zone updates its key-set, but the parent has yet to update the corresponding delegation (DS) record(s). From the perspective of the parent zone and resolvers, the authentication chain points to the previous key but all signatures have been produced by the new key. This scenario arises due to the difficulty in coordinating operational practices (key rollovers) across administrative boundaries. Based on the anticipation of this particular scenario we track it as follows.

Let  $|R^{DS}|$  denote the total number of unique DS records seen by pollers and let  $|R_v^{DS}|$  denote the number of *verified*

DS records that match key-sets in the corresponding child zone. The ratio between these values reflects the validity of authentication chains, or *delegation validity*:

$$V_{deleg} = \frac{|R_v^{DS}|}{|R^{DS}|}$$

Ideally, every delegation record would verify a key-set of a corresponding child and  $V_{deleg} \rightarrow 1$ . Lack of verification indicates that: the child has removed a key-set too quickly and has broken the authentication chain, or that the parent has been slow in removing an obsolete delegation record, or the parent has added a new delegation record before the child was ready. Thus, a ratio value of less than one indicates that there are zones that have broken delegations leading to their child zones.

**False Positives:** In addition to the configuration errors described above, *False Positives* (lower left box in Table II) can also occur. To illustrate this we draw an analogy from BGP and then show a DNSSEC-specific example.

In BGP there is an infrequent occurrence of routing leak-outs [11]. In these cases, an Autonomous System (AS) accidentally announce routes to peers that it really can’t reach. Here the routers have sent routing announcement data (and often use MD5 checksums to make it verifiable), but the data is false. Even though the MD5 sums on these data streams are verifiable, the data is not *valid*. In DNSSEC, an attacker that has compromised a zone’s private key can generate and sign records that appear to come from the zone. These records are invalid (e.g. a record may contain the wrong IP address of a web server [19]). However, these records will still pass cryptographic verification checks. This type of compromise and other security breaches are hopefully rare, but administrative errors are inevitable in large scale deployments.

We show evidence in Section IV that operational practices combined with lack of revocation in the DNSSEC design allow a weak form of false positives to occur where an attacker can replay *stale RRsets* long after the RRsets have been removed from the zone’s authoritative servers (and have been flushed from caches).

An RRset is stale if an administrator has changed data *values* in new sets, but a signature covering the previous values has not expired. In this case, the stale set could be replayed by an attacker or malfunctioning cache. Figure 2 illustrates this scenario. At time  $t_0$ , RRset 1 is created and signed. The signature includes an expiration date of time  $t_2$  (indicated by the bottom bar). At time  $t_1$ , RRset 1’s value is modified. For example, the IP address of a host may have changed. The modified RRset is distributed to all authoritative servers and the previous value is flushed from caches after the TTL expires. However, an attacker can continue to replay the old record until the signature expires at time  $t_2$ . Resolvers that receive the stale (blue) RRset will verify the signatures and declare that the set is valid.

SecSpider is able to automatically detect stale RRsets by tracking zones over time. We let  $R^v$  denote the set of RRsets whose signatures have not expired and are still verifiable. Thus,  $R^v$  includes sets that are currently served by a zone and older sets whose signatures have not expired yet. We denote the number of RRsets in  $R^v$  as  $|R^v|$ . We now define  $R^{stale}$  such that  $R^{stale} \in R^v$  and the sets in  $R^{stale}$  have different values than those currently being served (as seen in Figure 2). We say that  $|R^{stale}|$  is the number of RRsets in  $R^{stale}$ . We pay close attention to these sets because they could be replayed by an attacker. The ratio of these two values yields the proportion of stale data that is observed, or the *data freshness*:

$$V_{fresh} = 1 - \frac{|R^{stale}|}{|R^v|}$$

A value of 1 indicates that no obsolete RRsets could be replayed while a value less than 1 indicates that a fraction of verifiable RRsets could be replayed and would allow *invalid* data to be verified.

Note that in the case of a stale RRset, the attacker is only replaying data that was previously valid. In many cases, this type of vulnerability will raise little or no concern. However, one problematic scenario occurs when an attacker has compromised a zone’s private key and the zone attempts an unplanned key rollover. At such a time, an attacker can replay the stale key-set in order to *verify* (but not validate) the compromised key. Using the compromised key, the attacker can then forge arbitrary data from the zone. Some authentication chains have lifetimes of weeks, months, and in some cases years. Thus, key compromises combined with stale RRset-replays pose serious challenges. A complete discussion of the vulnerabilities and possible mitigations is beyond the scope of this paper, but can be found in [17], [8].

We, thus, characterize the validity metric ( $V^d$ ) of DNSSEC as an n-dimensional tuple of measurable validity metrics. Other types of validity dimensions are plausible and worth investigation, but in this work we use our experience to identify 2 operationally relevant dimensions to characterize:

$$V^d = \langle V_{deleg}, V_{fresh} \rangle$$

Our selection of these 2 dimensions is based on observational evidence that they are existing problems and that there is also awareness of them in the operational community.

#### IV. DEPLOYMENT STATUS TODAY

DNSSEC deployment data was collected using the SecSpider monitoring project [3]. The revised DNSSEC RFCs [4], [6], [5] were published in March 2005, and SecSpider began shortly afterward in October 2005. The monitoring project uses a collection of *pollers* that send DNS queries

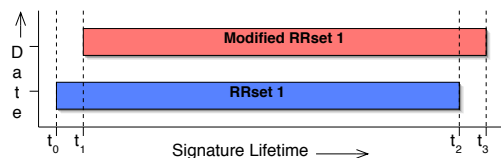


Fig. 2. If data changes, such as in “Modified RRset 1”, then the old RRset 1 will still be verified by the zone’s keys (even though the data is no longer valid).



Fig. 3. Polling locations

to the authoritative name servers of zones and use the DNS responses to form the raw data for this study. The polling locations are located in the United States, Europe, and Asia, and on networks comprised of universities, home access, and enterprises (Figure 3). This monitoring system uses a central server to control the pollers and schedule when queries should be sent and where to send them. All pollers are scheduled to execute the same queries at approximately the same time from their individual vantage points. In order to discover new zones, the monitoring system uses zone transfers (when possible) and exploits DNSSEC behaviors such as using the NSEC record to “walk” (e.g. definitively identify and retrieve all records in) a zone. The overall results provide a detailed history of each secure zone as viewed from the system’s poller locations.

The dataset discussed covers October 2005 through April 2009 and includes 15,982 secure zones. However, while this set of secure zones includes many well established DNS zones such as the `se ccTLD`, it also includes other secure zones that are clearly deployed only for testing. An example is:

`unknownalgorithm.nods.test.jelte.nl.netlabs.nl`

In this case, the actual name of the zone indicates that it is used for testing, and other zones in this same delegation (under `test.jelte.nl.netlabs.nl`) account for over 60% of all secure zones. To focus on how DNSSEC deployment is proceeding in “production” zones, our analysis began by pruning zones that appeared to be operating in a test-capacity. In order to classify zones as production we started by including all secure TLDs and all secure zones under the `arpa` TLD as production zones. Next, we added zones in other parts of the DNS

tree that pointed to an active web server or mail server as production. Thus, all zones considered in the study are zones that perform actions that suggest their operational status is important and taken seriously by operators. Though it can be argued that this test may have missed some legitimate zones, and may have included some test zones, it served as an automated way to identify reasonable candidates for measurements. The list of production zones is also posted on the project website and announced on DNSSEC deployment mailing lists. Zone administrators can use a web interface to change a zone’s status from testing to production or vice versa. This pruning process reduced the set of secure zones that were considered in this study from 15,982 to 5,053 secure “production” zones.

### A. Availability

Using the metric described in Section III-A, we begin our analysis with data as viewed from our polling locations. At regular intervals, all pollers query all secure zones. The polling system described herein is designed to mimic a generalized DNS resolver, with only minor differences. For instance, the pollers will each issue up to three queries with timeout thresholds set to a conservative 10 seconds<sup>2</sup>.

We set  $A_{max}(z_i, t) = 1$  if at least one poller could receive a response from zone  $z_i$  at time  $t$ . Our results found that  $A_{max}(z_i, t) = 1$  in 99.178% of our experiments. Out of the 5,053 zones in our study, only 75 zones ever encountered an instance where  $A_{max}(z_i, t) = 0$ . Because our preprocessing eliminated zones created purely for testing purposes, we theorized that the reliability of the remaining zones would be high due to the fact that they run production services and their outages would not be significant. Our results appear to confirm this. However, even though these zones are only considered when they were reachable, different pollers can have very different views of zone availability. Specifically, when requesting zone data, resolvers in some locations receive no answer, while others (at the same time, but from different locations) have no difficulty obtaining a response. The availability dispersion metric described in the previous section captures this difference between pollers. Figure 4 shows that roughly 20% of the monitored zones suffer availability dispersion. This means that some resolvers may not be able to receive critical data from a zone based solely on where they query from.

The reason why this dispersion exists was traced to Path Maximum Transmission Unit (PMTU [13]) problems. Recall each link along the path from poller to authoritative server has a Maximum Transmission Unit (MTU) that is the *largest* packet size it can support. The PMTU is the *smallest* MTU along a path. DNSSEC response messages can include public keys (DNSKEYs) and signatures (RRSIGs) which make them considerably larger than a typical DNS

<sup>2</sup>One popular DNS tool (DiG) uses 3 retries with a default timeout of 5 seconds.

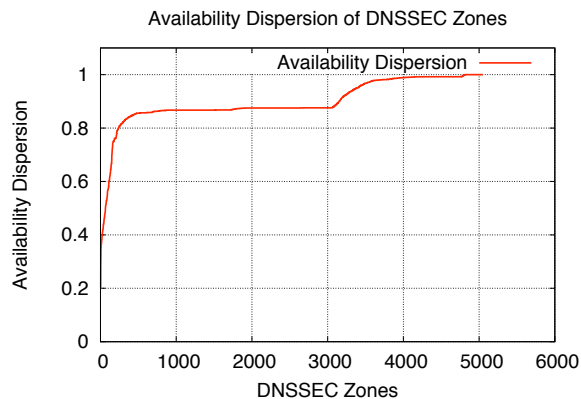


Fig. 4. The rank order of secure zones and their corresponding availability dispersion in ascending order.

response. As these responses travel along a path from the authoritative server to the resolver, these larger UDP packets may exceed the path’s maximum transmission unit (PMTU). As a result packets may be fragmented or dropped. In one specific case, the IP layer had fragmented the DNSSEC response messages and a local firewall was configured to disallow fragmented DNS packets. As result, the poller never received the responses.

To better understand the impact of response packet size, we modified our pollers to send queries with varying maximum response sizes. A DNSSEC query specifies a maximum response size that the resolver can support. The recommended maximum response size is 4,096 bytes and is set by default. If the query received no response, our pollers used a binary search to find the smallest maximum response size that would elicit a response. We call the process of sending queries with varying maximum response size *PMTU exploration*. During a PMTU exploration, problems manifested themselves in one of two ways: either zone data was received with a truncation bit (TC) set<sup>3</sup>, or the message was completely dropped (causing pollers to timeout). When data was received, the PMTU exploration was characterized as, “successful,” otherwise it was considered to have “failed.”

Figure 5 shows that while most pollers were able to retrieve zone data without encountering PMTU failures, poller #6 consistently had more trouble than the others. Figure 5 also indicates that in certain cases, the DNSKEY RRset size may reduce availability to the point that data is unavailable (via UDP) even after PMTU explorations. Figure 6 shows that certain pollers have significantly more trouble successfully getting data when a PMTU problem has been encountered. One can note that poller #6 attempts more than 10,000 more PMTU explorations than any other poller, and that almost 50% of the PMTU problems result in data that could not be retrieved via UDP (no matter what

<sup>3</sup>The TC bit indicates that the server wants to send more data but it won’t fit in the existing message.

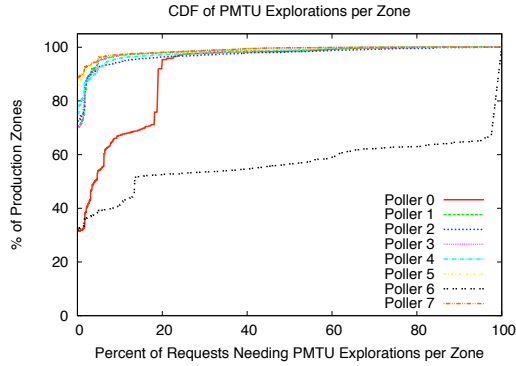


Fig. 5. This figure is a CDF of the distribution of how often zones need to be PMTU walked (broken down by pollers). One can see, for example, that from poller 0 roughly 70% of the zones only need PMTU walks about 20% of the time (or less). Of note is that from poller 6 roughly 60% of the zones need PMTU walks up to 90% of the time.

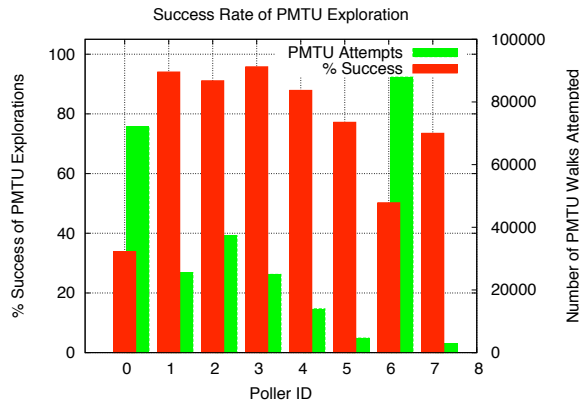


Fig. 6. This Figure uses cases in which pollers suffered PMTU failures. The figure shows the percent of times that pollers were able to successfully get data by adjusting the message-size of the request from an authoritative zone (finding a size that fits), as opposed to cases in which zones data cannot fit through the Internet to a poller.

size packet is specified).

A small set of zones suffer uniform PMTU exploration problems across all pollers. We conjecture that the link with the MTU problem happens to be close to their source (perhaps the first hop).

It is important to note that modern DNS and DNSSEC resolvers are encouraged by RFCs to initially request data using UDP. If a failure (i.e. no response) occurs resolvers will generally give up. Thus, a PMTU failure may not even prompt a resolver to *try* TCP. However, if a TC bit is received, resolvers may try smaller message sizes (PMTU exploration) or retry their query using TCP. Our results indicate that TCP is a reliable fall-back mechanism. However, we also note various opinions in the operational community decry TCP for DNS [1] and some locations may disallow TCP queries and/or the TCP query behavior

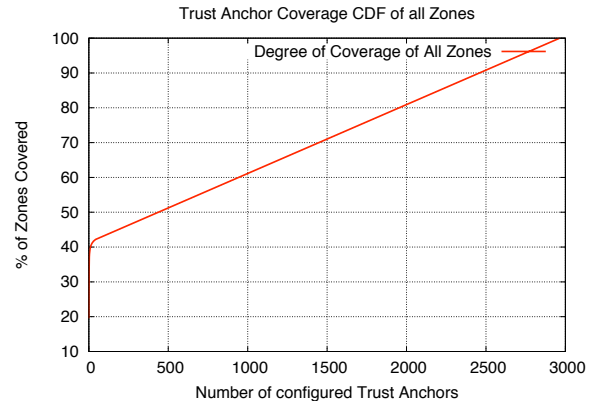


Fig. 7. This Figure is a CDF of the percent of secure zones that are reachable from a secure delegation or a trust anchor as one adds zones' key-sets to a trust-anchor list.

may raise other problems.

Compounding the PMTU problem faced by DNSSEC is the fact that some network-infrastructure components assume DNS traffic will not diverge from a very vanilla specification. Components such as firewalls may detect DNSSEC messages as malformed *DNS* messages and drop them, fearing that they may be dangerous. Examples of firewall compatibility issues include Cisco's PIX firewall before version 6.0. In addition, [2] is a broad examination of several types of home firewall/routers that found a significant proportion of these devices simply failed to process DNSSEC properly and led some zone operators to discontinue the use of DNSSEC.

### B. Verification

DNSSEC envisioned a top-down deployment where authentication chains would lead from the DNS root to most, if not all, zones. In stark contrast to this vision, the secure hierarchy of DNSSEC today is quite fragmented. Of the 5,053 secure zones in our study, fully 2,922 of these have *no* authentication chain leading to them. Ideally, a resolver would only need to configure one single trust anchor (corresponding to the DNS root) but today a resolver would need to manually configure 2,922 trust anchors in order to verify all existing signed DNSSEC data. Today, manually configuring 2,922 trust anchors and updating these trust anchors is tedious at best, but this clearly becomes infeasible as the number of secure zones moves from hundreds to millions.

A resolver may choose to only configure some of the 2,922 trust anchors and Figure 7 shows the percentage of secure zones that can be verified if the resolver configures trust anchors in a greedy manner. A resolver that configures the top 10 trust anchors can verify data in 40% of the secure zones. This is because a small number of zones participate in authentication chains. By configuring the trust anchor for some zone  $z_j$ , a resolver may also be able to verify

data from other secure descendants of  $z_j$ . Unfortunately most zones are not part of an authentication chain and configuring the trust anchor for  $z_i$  allows the resolver to verify data from only  $z_i$ .

It is also important to note that configuring a trust anchor is not a one-time operation. Whenever a key-set that exists in a trust anchor list is changed, the trust anchor list must be updated. The churn in large trust anchor lists increases operational and configuration overhead.

Our verification metric from the previous section captures the added configuration challenge. There are currently 2,922 trust anchors for 5,053 zones resulting in:

$$V^f = 1 - \frac{2,922 - 1}{5,053} = 0.578$$

In an ideal deployment, there would be a single trust anchor and we would have a score  $V^f = 1$ . Earlier monitoring results seemed to suggest that  $V^f$  was improving over time and some longer authentication chains were formed. But unfortunately this improvement in  $V^f$  proved to be an artifact of testing. Several large collections of test zones were deployed and connected via authentication chains. These test zones help operators experiment with managing authentication chains, but don't reflect production use and many of these test configurations are operated by a single organization, so true large scale *inter-administration* testing is still needed.

After removing the test zones, there has been a noticeable change in the  $V^f$  value. For example, in October 10<sup>th</sup>, 2007  $V^f = 1 - \frac{634-1}{815} = 0.223$ . Specifically, a number of ccTLDs (notably *se*, *bg*, *br*, *pr*) have deployed DNSSEC and are becoming trust anchors for large numbers of zones.

We define an *island of security* as a zone  $z$  and all secure descendants of  $z$  that can be reached by authentication chain starting at  $z$ . Thus the size of an island is the number of secure zones in the island<sup>4</sup>. A single zone that deploys DNSSEC but does not coordinate authentication chains with its parent or any of its children forms an island of size 1. Today there are 2,922 distinct islands of security in our study and 98.5% of them have size 1.

Figure 8 shows the current size of the largest islands. In addition to island size, the number of distinct administrative domains within the island is also important. We believe Internet cryptographic systems are interesting due to both their large size and their large number of independent administrative authorities. For example, an island of security that includes 60 zones operated by 60 different organizations requires coordinating authentication chains across different organization boundaries and, in our view, is more interesting than an island operated by a single administrative domain.

To infer whether an island includes multiple administrative domains, we analyzed the number of unique sets

<sup>4</sup>If DNSSEC were fully deployed, there would be a single island of security with the root zone as its trust anchor and its size would be the total number of DNS zones.

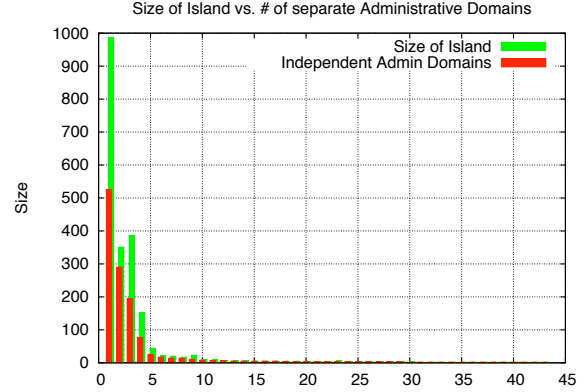


Fig. 8. This Figure shows the rank order of the largest observed Islands of security. The bars indicate that many of the zones in these islands are served by the same nameservers.

of nameservers serving the zones in each island. If two zones are served by the same set of name servers, we assume these zones are operated by the same administrative domain. Figure 8 shows that among the largest observable islands, many still consist of a relatively small number of administrative domains. The largest island of security includes 987 secure zones, but only 526 administrative domain.

Reducing the number of required trust anchors and creating large diverse islands of security are perhaps the most fundamental challenges facing DNSSEC deployment.

### C. Validity

As discussed in Section III-C, *validity* is distinct from verification. Due to operator errors, design flaws, implementation bugs, or intentional attacks, invalid data may be verified by a resolver (false positives). Similarly, valid data may fail a verification check (false negatives). Although, our monitoring did not detect intentional attacks, the lack of active attacks was not surprising given the current state of deployment. Instead, our results focus on two areas where operational practices lead to false negatives via broken authentication chains and false positives where stale data can be replayed.

**False Negatives:** In order to authenticate zone data, a resolver must be able to obtain the zone's public key. The discussion above shows most of these public keys need to be manually configured as trust anchors. For the other public keys that can be reached via authentication chains, we consider how well these authentication chains are maintained. In particular, a secure delegation (DS) record stored at the parent zone must match a DNSKEY stored at the child zone. As of April 1st, 2009, our pollers had observed 62,598 DS records and 60,614 of these records matched DNSKEYs in the child.

$$V_{deleg} = \frac{60,614}{62,598} = 0.968$$

On October 10<sup>th</sup>, 2007, there were 1,518 DS records observed by our pollers and 1,356 of these records correctly verified the DNSKEYs in the child zone. Thus, the deployment measurements at that time evaluated to:  $V_{deleg} = \frac{1,356}{1,518} = 0.893$ .

If a zone stores only one DS record for a child, and this DS record fails to match a DNSKEY, then the authentication chain is broken. There is no way for a resolver to verify the child zone’s public key. The results above suggest that 3.17% of the authentication chains observed by our poller were broken and data verification would have failed for all data in the affected child zone and all its descendents.

On the other hand, if there are multiple DS records for a child stored at the parent, it may be the case that one authentication chain works and the other broken DS records are simply old data that the parent has been slow to remove. However, DNSSEC envisioned that a parent zone would have exactly one DS record for each child. Even during a key roll over (e.g. when the child zone changes its DNSKEY), there is still exactly one DS record at the parent at all times. This is accomplished by having multiple DNSKEY records at the child and rollover procedures are described in detail in [10]. Recent practices have begun to emerge such that zone operators often include two DS records so that both digest types of SHA-1 and SHA-256 can be used by resolvers. At the time of these measurements 1,046 zones had exactly one DS record at the parent, 969 had two DSes, and 18 had more than two.

**False Positives:** While we did not observe any active attacks against secure zones, we did observe operational practices that would allow a misconfigured cache or attacker to replay stale data. Our analysis was focused on infrastructure records used by resolvers to navigate the DNS tree hierarchy. Specifically, we considered whether an attacker or misconfigured cache might be able to replay stale DNSKEY, DS, SOA, NS, and associated A RRsets.

Due to the potential impact of replaying these records, we tracked changes in them and determined whether the stale value could be replayed as described in Section III-C. Figure 9 breaks the set of secure zones into buckets based on the number of stale RRsets that were associated with the zone. Each zone is quantized based on whether is has 0, 1-10, 11-100, or more than 100 stale RRsets on each day. The results show that for some time, zones tended to have quite a few stale sets associated with them. The graphs show that in December 2006 there as only 1 zone with over 100 stale infrastructure records that could be intentionally or unintentionally replayed.

This is primarily caused by zones selecting long signature lifetimes. For example if a DS record is signed using a one year signature lifetime and changes only a few days later, the stale DS record can be replayed until the year long signature expires. Since DNSSEC includes no revocation mechanism, selecting long signature lifetimes creates a long period where stale data may be replayed and verified by unsuspecting resolvers.

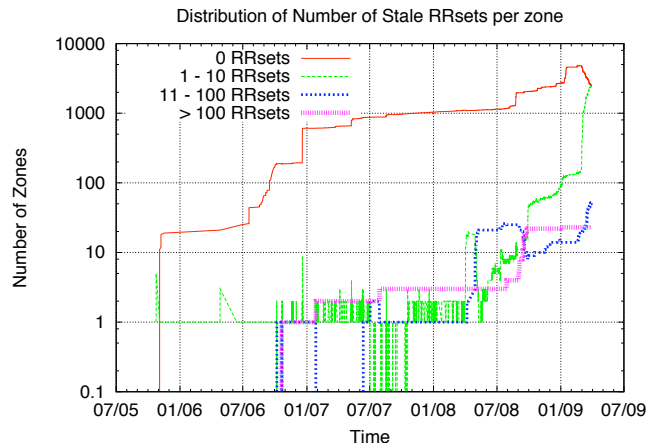


Fig. 9. This figure plots the number of zones that have either 0, 1-10, 11-100, or more than 100 stale RRsets associated with them over time. Note that the y-axis is in log-scale.

Early in 2007, the zones with more than 10  $R^{stale}$  began to decline in number. Unfortunately, currently, there are an increasing number of zones that have more than 100 stale RRsets. We note that there was a relatively large decline (in proportion to the total number of zones) of stale RRsets in 2007 that roughly corresponded in time with stale data monitoring results being available on our monitoring site and appearing on deployment mailing lists.

Based on the characterization in Section III-C we represent the state of DNSSEC from April 1<sup>st</sup>, 2009 as:

$$V_{fresh} = 1 - \frac{18,352}{100,752} = 0.818$$

The longitudinal evaluation of the  $V_{deleg}$  dimension, above, shows improvement when contrasted with  $V_{fresh}$  from October 10<sup>th</sup>, 2007:  $V_{fresh} = 1 - \frac{14,476}{27,196} = 0.468$ . Here we see an evident trend.

Overall, this calculates the validity as of October 10<sup>th</sup>, 2007, and then on April 1<sup>st</sup>, 2009 as a tuple:

$$\langle 0.893, 0.468 \rangle \rightarrow \langle 0.968, 0.818 \rangle$$

The merit of these absolute values is subject to debate. However, we present their relative values as systematic metrics that capture certain deployment specifics. In this regard, we note that there is a dramatic increase in freshness of DNSSEC’s validity. A qualitative interpretation of this would indicate that significantly fewer chances exist for resolvers to encounter stale, or misconfigured data in DNSSEC.

## V. DISCUSSION AND CONCLUSIONS

Since 2005, we have collected a vast amount of data on DNSSEC’s deployment. Our goals have consistently been to help inform operational practices with actual data. For example, the timing of our discovery and dissemination of

RRset staleness coincided with a large drop in its incidence. We posit that some operational groups became aware of the implications of rapid re-signing of their zones and adjusted this behavior. These simple changes help improve the overall DNSSEC system and demonstrate the value of distributed monitoring.

More generally, we have presented a set of metrics that quantify the DNSSEC deployment in ways that proved quite useful. These metrics have allowed us to collapse massive volumes of data into a few simple quantifiable values whose results helped shape further analysis and forensics surrounding operational failure modes. Our use of these metrics has revealed 3 fundamental challenges: First, data in Internet systems is not always universally available. Issues such as PMTU limitations, transient failures, and misconfigurations are a fact of life for these systems. Using our availability metric as an indicator, we have gauged the severity of this PMTU problem and can now design solutions.

Second, our verifiability metric clearly illustrates a fundamental challenge facing all cryptographic deployments in the Internet; how does one obtain the trust anchor information (e.g. its public key) in a secure, verified, and *robust* way? DNSSEC directly addressed this problem by designing a hierarchical PKI which minimizes the necessary trust anchor to one, however its design assumptions are not congruent with the common requirement that every party in the Internet tends to make their own decision about whether/when they may deploy new functions. From the facts that the Internet does not have a central authority and that not everyone trusts the same parties, one may conjecture that there may necessarily be multiple trust anchors, making the problem more difficult. How best to solve this cryptographic bootstrapping problem remains a critical open question.

The DNSSEC community has taken notice of some of the problems discussed herein and is exploring “look-aside validation” (DLVs [21]) where some central site or sites verify many public keys and become de facto authorities. We are also working on a novel solution that uses our monitoring apparatus as a diverse lookup infrastructure that can look for DNSKEY consistency and provide a repository of DNSKEYS which, although not cryptographically verified, form a consistent view from multiple diverse locations and whose correctness can be double-checked by individual key owners.

Finally, even early deployment shows DNSSEC is a highly dynamic and a continuously evolving system. Thus, its behaviors must be continuously monitored to capture new failures and challenges. By measuring one gets data and that can inform a system’s design, by quantifying data one can decipher its meaning and gauge the progress, and by monitoring one is able discover problems as they arise so that designs can be revisited.

## REFERENCES

- [1] DNS anomalies and their impacts on DNS cache servers. <http://www.nanog.org/mtg-0410/pdf/toyama.pdf>.
- [2] DNSSEC incident report, broadband routers. [http://www.dnssec-deployment.org/wg/materials/20071107/dnssec\\_incident\\_en.pdf](http://www.dnssec-deployment.org/wg/materials/20071107/dnssec_incident_en.pdf).
- [3] SecSpider. <http://secspider.cs.ucla.edu/>.
- [4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirement. RFC 4033, March 2005.
- [5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.
- [6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034, March 2005.
- [7] D. Atkins and D. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004.
- [8] S. M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the Fifth Usenix Unix Security Symposium*, pages 199–208, 1995.
- [9] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.
- [10] O. Kolkman and R. Gieben. DNSSEC Operational Practices. RFC 4641, NLnet Labs, September.
- [11] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *SIGCOMM 2002*, pages 3–16, New York, NY, USA, 2002. ACM.
- [12] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM '88*, pages 123–133, 1988.
- [13] J. Mogul and S. Deering. Path MTU Discovery. RFC 1191, DECWRL and Stanford University, November 1990.
- [14] J. Ng. Extensions to BGP to Support Secure Origin BGP (soBGP). Internet draft, Network WG, April 2004.
- [15] U. of Oregon. Route Views Project. <http://www.routeviews.org>.
- [16] E. Osterweil, D. Massey, and L. Zhang. Observations from the DNSSEC Deployment. In *The 3rd workshop on Secure Network Protocols (NPsec)*, 2007.
- [17] E. Osterweil, V. Pappas, D. Massey, and L. Zhang. Zone state revocation for dnssec. In *LSAD '07: Proceedings of ACM Sigcomm Workshop on Large Scale Attack Defenses*, 2007.
- [18] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *ACM SIGCOMM*, 2004.
- [19] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. In *ICICS*, pages 495–506, 2007.
- [20] A. C. Weaver. Secure sockets layer. *Computer*, 39(4):88–90, 2006.
- [21] S. Weiler. DNSSEC lookaside validation (DLV). RFC 5074, SPARTA Inc., November 2007.