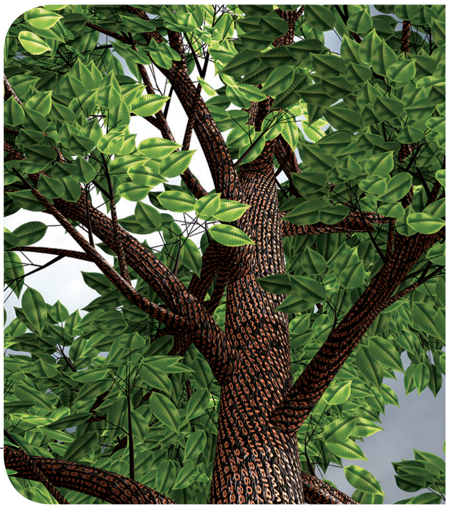
Interadministrative Challenges in Managing DNSKEYs

Although the visible deployment of Domain Name System Security Extensions is growing at a tremendous rate, evidence suggests that managing cryptographic keys is deceptively complex. Here, the authors outline the problem of managing DNSKEYs and present a survey comparison of existing proposed solutions.



Eric Osterweil and Lixia Zhang
University of California, Los Angeles

The Domain Name System (DNS)[1] is the Internet's de facto name resolution system. In fact, almost every transaction performed on the Internet is prefaced by a DNS lookup—for example, when a user types "www.bankofamerica.com" into his or her Web browser, it issues a DNS request to get Bank of America's IP addresses. However, in today's Internet, attackers can spoof DNS messages.[2] The DNS Security Extensions (DNSSEC) RFCs[3–5] specify how DNS domains (logical namespaces such as bankofamerica.com) can use cryptographic keys to digitally sign their content and gain the protection of origin authenticity, data integrity, and secure denial of existence.

DNSSEC specifies that each reply from authoritative DNS servers will have cryptographic signatures attached to it. DNS *resolvers* (clients) can obtain cryptographic keys for each domain and then formally verify that the key generated the signatures, the correct DNS server originated the data the signatures cover, and the data wasn't modified on the way to the resolver.

However, resolvers must ensure that the keys they have for a domain are authentic and not spoofed. Although DNSSEC's deployment has grown, the mechanisms by which resolvers can obtain and verify domains' cryptographic keys haven't evolved as needed. Specifically, it was envisioned that resolvers would begin with a trusted key for the DNS root domain (".") and recursively trace a secure delegation chain (*chain of trust*) from parent domains to their children until the resolvers reached the domain containing the queried name. For example, a resolver might want to get the `A` records (which contain IPv4 addresses) for the domain www.foo.com. This would require it to ask the root domain "." to refer it to the com domain, and then the com domain would refer it to the foo.com domain. At that point, the foo.com domain would be able to respond to the www.foo.com query. One essential problem facing DNSSEC deployment today is that neither the root nor many of the top-level domains (TLDs, such as com) have deployed DNSSEC. Consequently, DNS resolvers don't have an automated way to verify whether the keys they have for foo.com are valid or spoofed by an adversary (unless the keys are configured into the resolvers as *trust anchors* via some unspecified, out-of-band process).

In this article, we examine the space of various cryptographic key management issues involved in DNSSEC deployment and the approaches resolvers might use to identify the proper keys (trust anchors) for the DNS domains they visit. Further examination into these mechanisms leads to many more subtle issues that arise from how we currently manage the DNS.

Background

The DNS maps domain names such as www.ucla.edu to a wide range of data, including IP addresses, email services, and geographic locations.

All DNS data is stored in the same type of data structure, called a *resource record* (RR), each of which has an associated name, class, and type. For example, an IPv4 address for www.ucla.edu is stored in an RR with the name www.ucla.edu, class `IN` (Internet), and type `A` (IPv4 address). A host with several IPv4 addresses will have a set of several RRs, each with the same name, class, and type, but its own IPv4 address value. This *RRset* is the smallest unit that can be requested via query. For example, when a browser queries for ⟨`www.ucla.edu`, `IN`, `A`⟩, the reply will be the RRset for www.ucla.edu with all the IPv4 addresses for that name. All DNS actions—including cryptographic signatures, which we discuss later—apply to RRsets instead of individual RRs.

The DNS provides a tree-like hierarchical namespace; each node in the tree, except the leaf nodes, is called a domain. At the top of the tree, the root domain delegates authority to TLDs such as com, net, org, and edu. The com domain then delegates authority to create the google.com domain, edu delegates authority to create the ucla.edu domain, and so forth. The information repository that makes up the domain database is divided into sections called *zones*, each of which belongs to a single administrative authority and is served by multiple authoritative name servers to provide name resolution services for all names in the zone. By definition, a zone can contain one or more connected domains in the DNS name tree; in practice, many zones contain only one domain—this is the case for DNS TLDs as well as large domains in general. In the rest of this article, we use the terms domain and zone interchangeably when a zone contains a single domain.

The Three R's in DNS Name Management

Generally speaking, three types of entities manage the DNS namespace and are often referred to as the three R's of the DNS: the *registry*, the *registrant*, and the *registrar*. At a high level, the registry is an organization that serves the records for a zone. VeriSign, for instance, is the registry for the com zone. Any organization that would like to be assigned a new domain (like foo.com) is a registrant. Because com is foo.com's parent zone, a resolver trying to look up information about foo.com is first directed to com's DNS servers (which VeriSign runs). To make the com DNS servers redirect these queries to the foo.com registrant, the registrant must purchase the registration service from a registrar. Figure 1 illustrates the relationships between the three R's. It's possible for the same organization to play more than one role. In this figure, for example, VeriSign can be both a registrar and the registry, but the VeriSign registry must treat the VeriSign registrar in exactly the same way as it treats GoDaddy or any other registrars.

This three-party arrangement aims to remove the burden of directly interacting with potentially millions of registrants from large registries. Furthermore, to ensure a marketplace with competition, the Internet Corporation for Assigned Names and Numbers (ICANN) mandates that the roughly 20 large generic

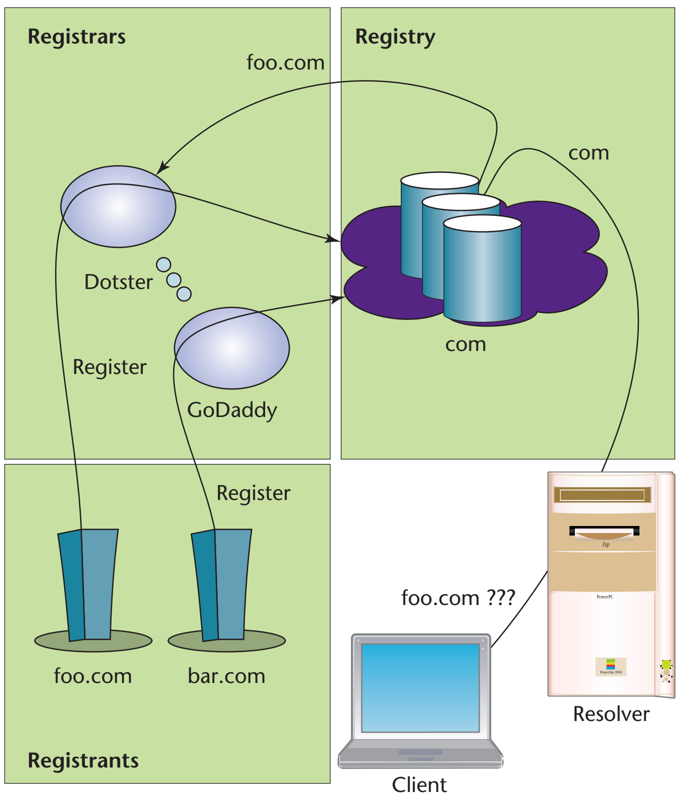


Figure 1. The three R's: registrants, registrars, and registries. The foo.com and bar.com companies want to create Domain Name System (DNS) zones and must use one of the many available registrars to do this. foo.com chooses GoDaddy, and bar.com chooses Dotster. The registrars then coordinate with the com registry, after which resolvers can ask com how to find foo.com.

TLDs (gTLDs, such as com, net, org, and edu) use ICANN-accredited registrars to sell domain names. Consequently, a registrant must purchase service from a registrar, which must in turn coordinate a delegation contract with a registry. More than 200 country code TLDs (ccTLDs), such as se, us, and de, often use registrars as well.

DNSSEC Overview

When the DNS was designed in the mid 1980s, security wasn't a primary objective. Since then, researchers and security professionals have identified several vulnerabilities, including cache poisoning and message spoofing.[6,7] DNSSEC provides a cryptographic solution to these problems that has a simple and intuitive design.

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. Each of the zone's public keys are stored in an RR called a DNSKEY; each signature is stored in a different RR called an RRSIG. In response to a query, an authoritative server returns both the requested data and the associated RRSIGs. A

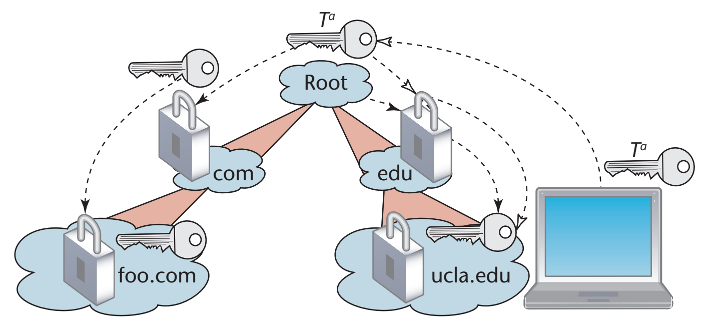


Figure 2. Secure delegation hierarchy. Secure zones vouch for their children's keys. Resolvers are preconfigured with the root zone's public key as a trust anchor (T^a) so that they can authenticate the edu public key, which then allows the resolver to authenticate the ucla.edu public key.

resolver that has learned the requested zone's correct DNSKEY can verify the reply data's integrity and the origin authenticity. To resist replay attacks, each signature carries a definitive expiration time.

To authenticate the DNSKEY set for a given domain—say, ucla.edu—the resolver must construct a chain of trust that follows the DNS hierarchy from a trusted root domain key down to the key for the domain in question, as Figure 2 shows. Ideally, resolvers would obtain the DNS root's public key in an offline and secure way.

Several challenges are inherent to building the chain of trust. First, a parent domain must encode the authentication of each child domain's public keys into its own zone. To accomplish this, the parent domain creates and signs a *delegation signer* (DS) RR that corresponds to a DNSKEY RR at the child. This signed DS RR creates an authentication link from the parent to the child. The child zone must request an update to the DS RR every time the DNSKEY changes. Although these procedures seem simple and straightforward, they are manual tasks, and humans inevitably make errors, especially when handling large zones with hundreds or thousands of child domains.

Second, when the parent and child zones belong to different administrative authorities, namespace management is often coordinated through the DNS's three-R structure, as we described earlier. Having the registrar between a registry (the parent) and the registrant (the child) introduces additional complications in coordinating the change of the child's key, as we explain later. Moreover, the parent and child domains might decide independently if and when they turn on DNSSEC, which increases operational challenges. If the parent zone isn't signed, no chain of trust leads to the child zone's DNSKEY, so this orphaned key effectively becomes an isolated trust anchor for its subtree in the DNS hierarchy. In these cases, DNSSEC doesn't specify how resolvers can learn DNSKEYs securely. Thus, DNSSEC resolvers must maintain a set of these *trust-anchor keys* (T^a) so that they can trace a chain of key sets + signatures (*secure delegation chain*) from some T^a to a DNSSEC key K needed to verify the signature in a DNS query reply. The original DNSSEC design envisioned a top-down deployment in which resolvers would need to have only the root zone's key configured in their trust-anchor sets, and all secure delegations would follow the existing DNS hierarchy. However, the root and most TLDs haven't yet deployed DNSSEC, so a potentially very large number of isolated trust anchors exist, as our measurement results show.

DNSSEC Deployment Status

Figure 3 shows data taken from our project, SecSpider (http://secspider.cs.ucla.edu), the first monitoring project to track DNSSEC's global rollout. The data set covers October 2005 through January 2009 and includes the histories of more than 11,000 secure zones. SecSpider's corpus is learned via user submissions, crawling from DNS monitors, crawling from a search engine, and NSEC (Next Secure) walking (NSEC RR types let you iterate through a DNSSEC zone because they chain names together; a more detailed description is out of this article's scope). Although this set of signed zones includes many well-established DNS zones, such as the se ccTLD, it also includes other signed zones that are clearly deployed only for testing.

One example is bogussig.bogussig.test.jelte.nlnetlabs.nl. In this case, the zone's actual name indicates that it's used for testing, and other zones in this same delegation (under test.jelte.nlnetlabs.nl) account for a substantial portion of the signed zones. To focus on how DNSSEC deployment is proceeding in "production" zones, our analysis prunes zones that appear to be operating in a test capacity. We broadly classify any zone as production if it's signed and

- is a TLD,
- belongs to the delegation hierarchy under the arpa TLD,
- has an active Web server at www.⟨zone name⟩, or
- has an active mail server pointed to by a mail exchanger (MX) record.

This pruning process reduced the set of zones we considered from roughly 11,000 to roughly 2,200 signed production zones. Although our test might have missed some legitimate production zones and included some test zones, it serves as an automated way to identify reasonable candidates for measurement.

Figure 3 shows production zone growth over recent years; this list is also posted on the SecSpider project Web site and announced on DNSSEC deployment mailing lists. Zone administrators can use the site's Web interface to change a zone's status from testing to production or vice versa. Note that due to the announcement of a new cache poisoning attack variant in summer 2008,[2] the adoption rate has continued to increase dramatically. Furthermore, of all the security islands known, roughly 98 percent are isolated without secure delegations from their parents. In some cases, such as ripe-ncc.com, the parent zone (com) isn't signed. In others, the DS record exists but doesn't match the child's DNSKEY, and in still other cases, the parent zone has DNSSEC turned on but simply doesn't have a DS for its child zone. Although not having secure delegations from parent zones might seem a trivial problem, we show in the next section that it's deceptively complex.

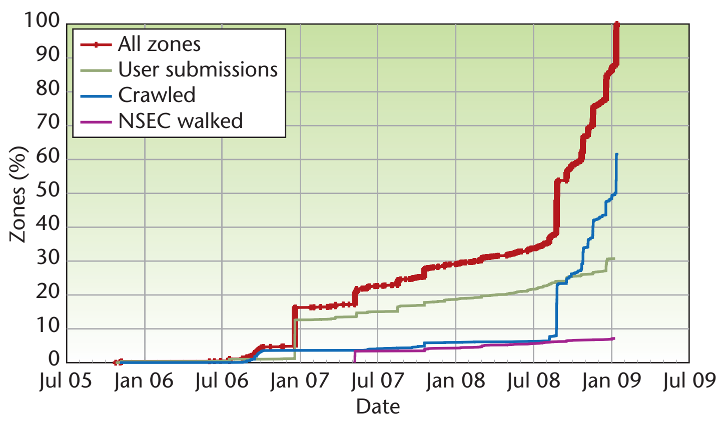


Figure 3. Cumulative distribution function (CDF) showing a dramatic increase in Domain Name System Security Extensions (DNSSEC) deployment. This increase followed the announcement of the Kaminsky cache poisoning attack. The *y*-axis shows the percentage of zones SecSpider is tracking today (where 100 percent is roughly 2,200 zones), and the *x*-axis is the date. The three subcurves plot how SecSpider learned of the zones and sum to the main curve.

How Zones Manage Their Public Keys

The operational issues that surround DNSSEC's key management are a combination of the challenges faced from both the authoritative zone's (publishing) and the resolver's (consuming) perspectives. Each perspective brings separate complications when considering DNSKEY lifetimes, rollovers, and especially verification.

When a Zone's Parent Is Signed

When a zone administrator decides to deploy DNSSEC (or sign his or her zone), having a parent zone that has already signed provides a way to verify the zone's DNSKEYs. The administrator will want to get the signed parent to serve a DS record that matches the zone's keys. Having this delegation in place lets resolvers transit any trust they have in the parent zone to the child.

Single administrative domain. In some cases, a newly signed zone belongs to the same administrative organization as its parent zone—for example, cs.ucla.edu is the parent zone of secspider.cs.ucla.edu. Although different operators might run each zone, they work within the same organization (UCLA's computer science department). Thus, the process of getting a DS record for secspider.cs.ucla.edu into the parent zone can be as simple as sending an email or making an in-person visit. Furthermore, changes and key rollovers can easily be governed by the processes outlined in the DNSSEC operational practices RFC.[8] (In this work, we illustrate the complexities of managing keys and refrain from discussing the further implications of unplanned key rollovers.) For example, when a child zone needs to change its DNSKEYs, it can coordinate a DS record rollover locally so the parent and child zones remain synchronized. Failure to maintain synchronization could cause resolvers to believe that the observed keys for a child zone are false because the parent might be serving a stale DS record. The essential observation here is that the DNSSEC key management process can be streamlined between child and parent zones if they're within the same administrative organization.

Multiple administrative domains. DNSSEC's operational needs become complicated when signed zones are run by different administrative organizations than their parent zones. This is the case whenever a company wants to buy a zone whose name is directly below a TLD. In such cases, DNSSEC key management generally requires coordination among the three R's. As we described earlier, the three R's conduct DNS namespace management jointly. Although the management process between these entities is well defined for DNS operations, the added complications of communicating new types of data, more rigid failure modes, and DNSSEC's general cryptographic complexity bring up many new coordination issues.

The typical non-DNSSEC communications among the three R's are done to change name server (NS) record delegation information. Specifically, registrants that want to change their NS records might often receive a Web interface (from their registrar) that lets them log in and update their information. After this, the registrar will use some provisioning protocol (such as the Extensible Provisioning Protocol [EPP]) to communicate this information to the registry. Such communications are relatively infrequent on a per-zone basis. Thus, overall communication between the three R's is usually light for any single child zone. However, in DNSSEC, zones should change their keys at regular intervals (RFC 4641 suggests, for example, that some

DNSKEYs be rolled over every few months). Because the registrant's zone owns DNSKEY records, and the registry has DS records that correspond, this periodic change of keys introduces new operational hurdles.

The DNSSEC community has begun investigating ways to design trust-anchor repositories (TARs) that resolvers can use as off-hierarchy systems to verify DNSKEYs for zones.

The actual communications load among the three R's might not increase significantly, but the operational procedure to coordinate between the parent and child zones' administrative domains and across all three R's to perform key rollovers in synchrony is an open issue. Furthermore, in some cases, operators have noted that the full communication channels between registrants that run their own zones and their registries (which go through the registrar) aren't necessarily secure enough to coordinate DNSSEC data, but this still leaves the registrant with no other recourse.

Note that many registrars provide hosting services to their registrants. Thus, while hosting a zone, a registrar might be able to simplify some inherent operational difficulties because the registrant wouldn't be an online participant. However, this could complicate matters when registrants wish to change registrars. Consider a simple case in which a registrant (foo.com) wants to change registrars (perhaps to one offering a lower yearly fee), but the current registrar manages the registrant's zone and actually owns the public and private keys for that zone. With DNS, an operator need only transfer the zone's data to the new registrar so it can begin serving the zone. However, with DNSSEC, it's unclear whether the current registrar is expected to pass the keys over to the new registrar so that the zone doesn't need to re-sign, how to do this securely, or whether the new registrar should create new keys (which might invalidate the chain of trust). In the latter case, how and when should the parent zone (the registry) be notified? Furthermore, after notifying the parent, how should the three parties coordinate the rollover process? Recall that each party is an independent company and likely has its own operational schedules and agendas—they might not all easily reach agreement on timing issues. Moreover, what business model motivates everyone to coordinate in this process? One subtlety in this case is that no one can mandate what each party must do, and they need not cooperate. One proposed solution is to simply let the new registrar add new DNSKEYs to the registrant's existing set. This solution raises yet another concern about having too many keys for a zone and its inability to serve them to resolvers due to the *path maximum transmission unit* (PMTU) limitations. The PMTU issue is discussed in detail elsewhere.[9]

Consider another example: suppose that a registrant uses a third party to sign its zone. In this case, the registrant would operate its own DNS zone but contract out to a third party all the cryptographic operations that support DNSSEC (www.signmyzone.com). The third party takes the registrant's DNS zone and returns a signed version; it also handles key management rollovers and other related duties. The main distinction between the third-party signers and the registrars that host services for registrants is that a given third-party signer need not have any specific contractual qualities that ICANN mandates for registrars. This is a subtle point that highlights the lack of formal procedures in place today.

Other complications are matters of scale. Suppose a large registrar manages the zone data for many thousands of registrants. Large zones might experience changes in their data at frequent intervals, and every change requires that data to be re-signed. The computation power needed simply to generate the cryptographic keys for new zones and rollovers is already significant. The additional resources required to sign zones at the frequency that they change can be nontrivial. To address this, an increasing number of software tools allow operators to sign zones "incrementally," meaning that when a change to a zone occurs, an operator must sign only the relevant RRsets. We note this primarily because some of the most popular DNS name server software packages either don't support this facility or have only recently begun to, and would require existing installations to upgrade. Thus, without incremental signing, users must pay the computational costs of re-signing their whole zone whenever a change occurs. Moreover, DNSSEC requires several additional record types for each DNS RRset (all RRsets need RRSIGs and must have associated NSEC or NSEC3 records for secure denial of existence), which increases the storage size by roughly a factor of three.

Finally, a fundamental difference exists between DNS and DNSSEC operations: in DNS, when there is a misconfiguration (such as an NS record that points to the wrong server), the redundancy in the servers often ensures that name resolutions can still proceed successfully. In DNSSEC, if, for example, a stale DS record points to the wrong DNSKEY for a zone, resolvers might consider all data from that zone to be invalid and discard it, leading to denial of service.

These and other problems aren't insurmountable from a technical perspective, but a true appreciation for their complexity comes from realizing that the barriers sometimes stem from managing operational difficulties at a large scale.

When a Zone's Parent Isn't Signed

Another complicated case is one in which a zone *C* decides to sign but can't get a secure delegation from its parent (that is, get a DS record signed by its parent zone). This can happen if the parent zone isn't signed, or if no clearly defined, generally usable, and secure mechanism exists through which *C* can pass its DS record to its parent zone. A resulting question is then, why would a zone choose to deploy DNSSEC if it didn't have a secure delegation from its parent? The answer can vary, but one major reason is that DNSSEC can offer isolated (or singleton) zones both origin authenticity and data integrity, assuming resolvers can find means to verify the DNSKEYs for a singleton zone.

SecSpider's observations show that more than 98 percent of DNSSEC-enabled zones are singletons (zones without a valid secure delegation). This fact illustrates that for early DNSSEC adopters, the secure delegation hierarchy isn't there to help their key verifications. Consequently, the DNSSEC community has begun investigating ways to design trust-anchor repositories (TARs) that resolvers can use as off-hierarchy systems to verify DNSKEYs for zones.

Trust-anchor repositories. TARs are systems designed to let resolvers determine the validity of keys for DNSSEC-enabled zones. Existing TARs vary widely in how they're implemented and maintained, and in the ways they present trust anchors to resolvers.

One of a TAR's first considerations is how to obtain valid keys for trust anchors. This is directly related to resolver operators' most important concerns: How much should they trust the keys a TAR validates? Currently, two main schools of thought exist:

- TAR operators should vet keys manually and enter a given zone's DNSKEYs into a TAR only after some real-world assurances are made (where "real-world" might not be clearly defined); or
- keys should be obtained from large-scale monitoring of all visible DNSSEC zones by polling them from multiple independent vantage points.

Each approach has its pros and cons. For example, the TARs with manual inputs offer traditional cryptographic assurances that keys found in them should belong to the zones they report. However, such a manual verification and maintenance process faces scaling challenges. In the presence of the hundreds of millions of DNS zones that exist today, key rollovers could require even more TAR maintenance efforts than large registrars face today. Conversely, the polling TARs use concepts from distributed system design (off-axis polling, independent paths, and so on) to significantly raise the bar and make it impractical for an adversary to spoof keys. Clearly, this approach doesn't offer the provable verifiability that the manual TARs do, but it does offer systems-level assurances, and its automated operation can scale as DNSSEC continues to grow.

Another issue facing TARs relates to how resolvers get data from them. Again, two major schools of thought exist:

- DNS queries should spawn an additional inline query to a TAR (or a look-aside) while the resolver waits, or
- resolvers should obtain static entries from TARs so that all trust anchors are known locally a priori.

An inline lookup lets a TAR be a separate service operated by focused operational groups separated from resolvers and can provide up-to-date information. On the other hand, this approach also adds obvious latency to DNS queries because resolvers can't send responses until after receiving a separate response from the TAR and any further validation occurs (using multiple look-aside TARs would clearly compound this). Furthermore, a resolver operator must trust the TAR operators to provide reliable and faithful service with limited outages and without compromised servers. The second approach has the advantage that all trust-anchor information is preconfigured and that no appreciable latency occurs when using their trust anchors. However, one drawback is that the list might grow large enough to become infeasible to manage locally.

TARs also face issues related to how they stay abreast of changes to the DNSKEYs of the zones they represent. It's just as important for TARs to keep updated as it is for parent zones to maintain their children's DS records. Thus, TAR designers must consider keys' lifetimes when designing a TAR. Can a TAR keep up with the frequency at which zones change their keys? Also, during a zone's key rollover (when it changes from one key to another), a few short transition phases exist in which the zone uses different combinations of the old and new keys—how will a TAR keep up during these phases? Keeping up with the key rollovers is particularly challenging if the trust-anchor keys are configured into all resolvers rather than queried inline. Additionally, many believe another question is much larger in scope: How will TARs let a zone transition to using a DS record once its parent signs and begins offering secure delegations? These questions remain largely open challenges at the time of this writing. The operational and research communities have various opinions about how to answer these questions and how TARs should evolve in the future.

Example TARs. Today, several examples exist of systems that act as TARs for resolvers.

The Internet Software Consortium (ISC) runs a DNSSEC key repository known as a *DNSSEC Look-aside Validation* (DLV) service,[10] which defines a new type of DNS RR called a DLV. Resolvers use this DLV repository as a TAR in the following way: when a resolver gets a DNSKEY from a zone ⟨*Z*⟩, it issues a query to ⟨*Z*⟩.dlv.isc.org for a DLV record. This record is similar to the DS record in that it's a cryptographic fingerprint of the associated DNSKEY. If the record matches the DNSKEY, the DLV repository has verified the key's authenticity. Thus, the resolver should be satisfied, and query responses that this key signs can be returned; otherwise, the resolver returns a `SERVFAIL` error to the client because the DNSSEC key validation failed. ISC's DLV repository is a manually maintained, inline TAR that uses its own DNSKEY to sign its DLV records. So, all resolvers must obtain ISC's key offline in a secure way and use it as a trust anchor to verify DLV records.

The *Interim Trust-Anchor Repository* (ITAR) is supported by the Internet Assigned Numbers Authority (IANA), which coordinates the DNS root zone. This TAR is manually maintained but contains trust anchors only for the DNS TLDs. The ITAR is specifically chartered to be decommissioned once the DNS root is signed, because a signed root would replace the ITAR's role and would do so through DNSSEC's secure delegation design. The ITAR expects all resolvers to statically configure its entries. Thus, the ITAR is a manually maintained, statically configured TAR.

The *SecSpider* TAR is an inline TAR that uses the same DLV mechanism as ISC's. Lookups to SecSpider's TAR take the form ⟨*Z*⟩.secspider.cs.ucla.edu for DLV records. Rather than using manual verification to collect and validate DNSSEC keys, SecSpider uses distributed polling from vantage points distributed worldwide; keys are inserted into the TAR only if they're globally consistent. Every time SecSpider runs, it randomizes the order in which it polls the zones in its corpus. SecSpider queries each zone from all of its pollers (lightweight DNS query repeaters) simultaneously. Its pollers are deployed at multiple vantage points around the world, and the communication channel between SecSpider and its pollers is cryptographically secured. SecSpider takes these three steps (unpredictable schedule, spatially diverse vantages, and secure communication channels) to severely handicap a would-be adversary who attempts to spoof it and falsely insert a DNSKEY for a zone. Because the scope of the SecSpider TAR is as large as the observable DNSSEC deployment, it helps verify the keys for all the zones it can find via its automated crawls of the DNS hierarchy. Thus, SecSpider is an automated, inline TAR.

Finally, *Vantages* (www.vantage-points.org) represents yet another type of TAR. It's designed to be run by resolver operators, as a daemon that polls specific data sources the operator specifies. Thus, it can poll the keys of all DNS zones that the resolver queries, or it can poll the Web pages where zone owners publish their keys. It can also use manual trust assertions from users that the operator personally believes are trustworthy. Vantages then compares the results of the local polling against the key information obtained from a list of the operator's manually configured and trusted friends who are also running Vantage daemons, providing off-axis corroboration. The Vantage daemon checks the results for consistency and then configures them into the local resolver. So, we classify Vantages as a hybrid automated/manually maintained, statically configured TAR.

Turning DNSSEC Off

One final case is one in which a zone administrator elects to discontinue using DNSSEC. Here, it's important for validating resolvers to be able to distinguish between a zone that was formerly serving a DNSSEC zone but has now reverted to the basic DNS service, and a DNSSEC zone that's under a downgrade attack. In this attack, an adversary attempts to serve non-DNSSEC data to a resolver rather than trying to spoof the cryptographic key used in the verification. If an adversary can convince the resolver that a zone doesn't use DNSSEC, then he or she can serve unsigned spoofed DNS data.

How to address this issue is largely an open challenge. One notable example of an operational approach used today is the one followed by the se ccTLD. When a zone under se chooses to discontinue DNSSEC, se simply stops serving the DS record for that zone. For a zone *C* under se, if resolvers don't find a DS RR for *C*, they don't expect to see DNSKEYs or signatures when querying it. However, with TARs' increasing numbers and usage, resolvers might continue to find DNSKEYs for zone *C*, until all the TAR operators remove their entries.

The active community surrounding DNSSEC has clearly identified the challenges listed in this work and discusses solutions to open issues daily on mailing lists and in community meetings. Much debate exists on how best to move forward. However, despite the remaining open issues, for many, DNSSEC is usable in its current form, and its deployment has entered a rapid growth phase, as evidenced by SecSpider's monitoring results. ☐

References

1. P. Mockapetris and K.J. Dunlap, "Development of the Domain Name System," *Proc. SIGCOMM Conf.* (SIGCOMM 88), ACM Press, 1988, pp. 123–133.

2. CERT, Cert vulnerability note vu#800113, 2008; www.kb.cert.org/vuls/id/800113.
3. R. Arends et al., *DNS Security Introduction and Requirement*, IETF RFC 4033, Mar. 2005; www.ietf.org/rfc/rfc4033.txt.
4. R. Arends et al., *Resource Records for the DNS Security Extensions*, IETF RFC 4034, Mar. 2005; www.ietf.org/rfc/rfc4034.txt.
5. R. Arends et al., *Protocol Modifications for the DNS Security Extensions*, IETF RFC 4035, Mar. 2005; www.ietf.org/rfc/rfc4035.txt.
6. S.M. Bellovin, "Using the Domain Name System for System Break-ins," *Proc. 5th Usenix Unix Security Symp.*, Usenix Assoc., 1995, pp. 199–208.
7. D. Atkins and D. Austein, *Threat Analysis of the Domain Name System (DNS)*, IETF RFC 3833, Aug. 2004; www.ietf.org/rfc/rfc3833.txt.
8. O. Kolkman and R. Gieben, *DNSSEC Operational Practices*, IETF RFC 4641, Sept. 2006; www.ietf.org/rfc/rfc4641.txt.
9. E. Osterweil et al., "Quantifying the Operational Status of the DNSSEC Deployment," *Proc. 8th ACM SIGCOMM Conf. Internet Measurement* (IMC 08), ACM Press, 2008, pp. 231–242.
10. S. Weiler, *DNSSEC Lookaside Validation (DLV)*, IETF RFC 5074, Sparta, Nov. 2007; www.ietf.org/rfc/rfc5074.txt.

Eric Osterweil *is a PhD candidate at the University of California, Los Angeles. His research focuses on large-scale network measurement systems, network security, and distributed data verification. His thesis work focuses on a concept called the Public-Space. Contact him at eoster@cs.ucla.edu.*

Lixia Zhang *is on the faculty of the University of California, Los Angeles computer science department. Her research interests include network architecture, system security, and protocol designs. Zhang has a PhD in computer science from the Massachusetts Institute of Technology. She cochairs the Routing Research Group under IRTF and is a fellow of the ACM and the IEEE. Contact her at lixia@cs.ucla.edu.*