Visualizing DNSSEC Configuration and Troubleshooting with DNSViz

ABSTRACT

The deployment of security to the Domain Name System (DNS) using the DNS Security Extensions (DNSSEC) is gaining ground in production DNS. While the advancement is significant, the protocol and administrative complexity added by DNSSEC has had an impact on DNS availability. Troubleshooting DNSSEC issues using traditional methods can be tedious and non-intuitive. In this paper we describe methodology for presenting administrators with a graphical representation of DNSSEC. This visualization aids administrators in understanding DNSSEC and identifying problems with configurations. We implement the visualization using a Web-based tool and include graphical examples of misconfigurations encountered in production DNSSEC deployments.

1. INTRODUCTION

The Domain Name System (DNS) is essential to proper Internet function. Most network applications use DNS for name resolution, and outages can render these applications effectively unusable. To maintain DNS availability, DNS must be both understood and well configured.

Problems affecting DNS availability are commonly the result of misconfiguration. The complexities of name resolution present a number of areas where misconfiguration may be introduced by DNS administrators. Resolution of a domain name is often dependent on proper resolution of other domain names. These dependencies exist to provide robustness and flexibility in configuration, but increase administrative overhead, particularly for troubleshooting problems that arise.

The addition of the DNS Security Extensions (DNSSEC) adds to complexity already inherent in DNS, extending the protocol to include provisions for a public-key infrastructure (PKI) suitable for cryptographic authentication of DNS answers. The authentication chain for a domain name follows a series of signatures and accompanying keys upwards through the namespace hierarchy to a trusted anchor. Problems at any point in that chain may result in a resolution failure.

To aid DNS administrators in understanding and troubleshooting a DNSSEC deployment, we introduce a graphical presentation of DNSSEC. With our approach, we visualize not only the DNSSEC authentication chain for a domain name, but integrate conventions to facilitate identification of misconfigurations affecting DNSSEC validation, which may span diverse DNS dependencies. The primary contributions presented in this paper are:

- A graphical representation of the DNSSEC authentication chain for a domain name.
- An visual convention to identify configurations causing validation errors.
- The implementation of a tool employing DNSSEC visualization techniques on production DNS data.

In this paper we provide a basic description of DNSSEC and discuss several specific behaviors that make its troubleshooting difficult. We describe our solution and describe the implementation of DNSViz, a Web-based tool for DNSSEC visualization.

The remainder of this paper is structured as follows. In Section 2 we briefly review the DNS and DNSSEC protocols, and Section 3 contains troubleshooting complexities related to DNSSEC. We describe our visual representation of DNSSEC in Section 4. In Section 5 we discuss our implementation and apply the model to production DNS data. Our conclusions are contained in Section 6.

2. DNSSEC BACKGROUND

The Domain Name System (DNS) [13, 14] is a distributed system for resolving Internet names to addresses. An autonomously managed piece of namespace is a DNS *zone*. For name resolution a *resolver* asks questions of *authoritative servers* to resolve the name in question. It begins its query process at the top of the hierarchically-organized DNS namespace, the *root zone*. The result from root authoritative servers is a downward referral to the servers authoritative for the appropriate top-level domain (TLD) (e.g., *com*) in *delegated* namespace. This referral process is iteratively repeated until the resolver contacts a server authoritative for the name in question, which responds with the appropriate answer.

DNS questions and answers consist of *resource records* (RRs), each of which has a name (e.g., *www.example.com*), a type (e.g., `A`), and record data specific to its type. RRs of the same name and type comprise a *resource record set* (RRset).

The DNS Security Extensions (DNSSEC) [1–3] add authentication to DNS using a public-key infrastructure (PKI). Public keys are embedded in the zone data for each zone using the `DNSKEY` RR type. Each `DNSKEY` RR specifies an algorithm (e.g., RSA-SHA1) that it uses for signing zone data. Zone data is signed on a per-RRset basis, and the signature is included in the zone data as an RR of type `RRSIG`, with the same name as the RRset it *covers*. `RRSIG` RRs also contain information to identify the `DNSKEY` that made the signature.

A resolver must be seeded with a `DNSKEY` as a *trust anchor* to validate `RRSIG`s. Rather than requiring a resolver to maintain a `DNSKEY` for every signed zone, DNSSEC scales by building an authentication chain upwards through the namespace hierarchy, so resolvers may anchor with the `DNSKEY` of a common ancestor zone. The link between zones is accomplished by the introduction of `DS` RRs in the parent zone. A `DS` RR maps to a `DNSKEY` in the child zone of the same name, and is secured by a cryptographic digest of the `DNSKEY` stored as record data in the `DS` RR. Delegation status is classified in one of the following ways:

- *Secure:* A `DS` RR in the parent zone corresponds to a `DNSKEY` in the child zone.
- *Insecure:* No `DS` RRs exist in the parent zone.
- *Bogus:* `DS` RRs exist in the parent zone, but none match any `DNSKEY` in the child zone.

A `DNSKEY` which either corresponds to a `DS` RR in the parent or is explicitly configured as a trust anchor in a validating resolver is considered a *secure entry point* (SEP) into the zone. A common setup is to have two `DNSKEY`s in a zone: one which signs the zone data—a *zone signing key* (ZSK)—and one which signs only the `DNSKEY` RRset—a *key signing key* (KSK).

Authenticated denial-of-existence of RRsets is accomplished using `NSEC`-type RRs. Each `NSEC` RR in a zone specifies all the RR types of RRsets existing in the zone having the same name as the `NSEC` RR. It also designates the name of the next RRset in DNSSEC *canonical* ordering [2]. Signed parent zones must use `NSEC` RRs in responses to prove the non-existence of `DS` RRs for insecure delegations. Otherwise resolvers cannot prove that the child zones of such delegations are in fact unsigned.

3. DNSSEC TROUBLESHOOTING

DNS administrators have historically had tools at their disposal for troubleshooting DNS-related problems at various levels of granularity. Two such well-known tools, `dig` (part of ISC BIND 9.7.0 [9]) and `drill` (part of ldns 1.4.0) [12], provide a command-line interface and a number of options to send, receive, and analyze DNS messages to and from recursive and authoritative servers, and are instrumental to troubleshooting DNS. Other Web-based tools have been designed to provide a graphical or textual summary of the DNS health of a domain or server [4, 8, 17]. However, the deployment of DNSSEC has introduced new challenges for administrators of resolvers and authoritative servers alike.

We consider a *stub resolver* that issues requests which are *recursively* handled by a validating resolver. Validation of an RRset by a resolver results in one of four outcomes [3] which determine the response returned to the stub resolver:

- *Secure*: The validating resolver establishes an unbroken chain of trust from the RRset to a trust anchor. In this case, the resolver returns the answer to the stub resolver, setting the *authenticated data* (`AD`) flag in the response.
- *Insecure*: The validating resolver can assure that no chain of trust should exist between the RRset and a trust anchor. In this case, the correctness of the answer cannot be determined, but the resolver has securely proven that there is no path wherein verification is possible. The resolver returns the answer but does not set the `AD` flag.
- *Bogus*: The validating resolver is unable to form a chain of trust between the RRset and a trust anchor and cannot securely determine that no such chain should exist. In this case the resolver issues a response with a status of `SERVFAIL`, a general name resolution failure, and does not include the answer.
- *Indeterminate*: The validating resolver cannot obtain sufficient information (i.e., the required DNSSEC RRs) to determine whether or not an RRset should be signed. However, the response from the resolver is the same as if it deemed the response bogus, i.e., with a `SERVFAIL` response.

The outcome determined by secure and insecure responses are clear to the stub resolvers. However, a bogus response is ambiguous and may be the result of any number of issues that cause validation failure, or it may be caused by a DNS misconfiguration unrelated to DNSSEC. While traditional DNS troubleshooting tools are quite powerful, tracking down the cause of a server failure in a DNSSEC deployment requires more effort. In the remainder of this section we discuss several challenges with regard to troubleshooting DNSSEC problems.

3.1 Computation Required

One of the biggest difficulties with troubleshooting DNSSEC is that some values are not immediately available in a response, as they typically have been for standard DNS. For example, `RRSIG`- and `DS`-type RRs both reference `DNSKEY` RRs using a *key tag*, a 16-bit checksum-like value computed from the `DNSKEY` record data [2]. The key tag itself is not included in the `DNSKEY` record data and must be computed by resolvers to help match `RRSIG` and `DS` RRs to `DNSKEY` RRs for verifying signatures and establishing a SEP from a parent zone, respectively. Since computation is required to generate the key tag for a `DNSKEY` RR, it is not immediately apparent to DNS administrator without additional help from a tool. This functionality is provided by `drill` but not by `dig`.

Visually verifying cryptographic digests and signatures is also infeasible. The digests and signatures contained in DS and RRSIG RRs, respectively, must be verified by tools capable of cryptographic functions. However, one component that an administrator may evaluate sans computational help is the expiration time of an RRSIG RR, which is included as a field in the RRSIG record data.

3.2 DNS Key Roles

Among the record data for the DNSKEY RR is the flags field. The only flag required to be set by the specification to be set is the *zone key* flag, which designates the DNSKEY as one pertaining to DNS. Two other flags defined are the *secure entry point* (SEP) flag and the *revoke* flag. The SEP flag was introduced as a way for DNS administrators to distinguish between a ZSK and a KSK. However, the specification dictates that a validating resolver should ignore the SEP flag when performing validation [11]. Likewise, resolvers are not limited to using DNSKEYs with the SEP flag set as trust anchors, and there is no restriction that DNSKEY RRs corresponding to DS RRs have the SEP flag set. Thus the status of the SEP flag for a DNSKEY is independent of its signing role.

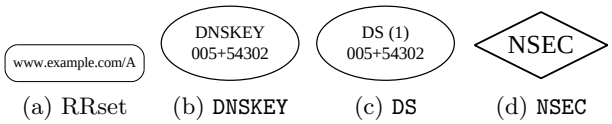
The revoke flag was introduced as a way to designate a DNSKEY as revoked [18]. The revoke bit itself is a necessary attribute of a revoked DNSKEY; however, the DNSKEY must also be self-signed, that is, it must be used to sign the DNSKEY RRset. Without this additional characteristic, the DNSKEY is not considered revoked.

Since DNSKEY attributes (aside from the zone key flag) do not necessarily indicate a DNSKEY's actual role in DNSSEC, administrators cannot rely on flags alone when troubleshooting. DNSKEYs must be distinguished by their role in the chain of trust. If the key tags for the DNSKEYs are known, then an administrator may with some effort determine their role, whether they are ZSKs, KSKs, or simply published in the zone with no signing role. The latter might be the case when a zone is being prepared to *roll* its ZSK or KSK [10]. They may also determine which DNSKEYs, if any, provide a SEP into the zone by examining DS RRs in the parent zone. However, compiling a chain of trust from these observations can be tedious and error-prone.

3.3 DNSSEC Dependencies

Troubleshooting DNSSEC becomes more difficult when dependencies are considered. Name and server dependencies exist as part of standard DNS and are discussed in detail in other work [6, 7, 16]. In this paper we consider parent, alias, and trust anchor dependencies, which have relevance to visualization, as presented in our paper.

Parent dependencies. Because name resolution follows referrals from the top of the namespace downwards, the resolution of a domain name is always dependent on its hierarchical parent. Thus resolution of *sub.example.com* is dependent on *example.com*, which is in turn dependent on *com*, etc. DNSSEC increases this parent dependency since the authentication chain extends upward. For example, an expired signature in the *com* zone can invalidate names in *example.com* and *sub.example.com*. A view of the entire hierarchy of a domain name is essential for troubleshooting DNS- and especially DNSSEC-related issues.



(a) RRset (b) DNSKEY (c) DS (d) NSEC

Figure 1: Node types.

Alias dependencies. Aliasing in DNS is accomplished using the CNAME RR. If in response to a query a resolver learns that a domain name is an alias for another name, it must follow the resolution process for the domain name listed as the *target* or *canonical name* of the CNAME RR. A domain name alias is dependent on its target name, as well the ancestry of the target name. Additionally, if the target itself is an alias, then a new dependency is created. RFC 1912 discourages the practices of alias chaining [5]. However, this configuration exists in practice, particularly in content distribution networks (CDNs), where companies offload their DNS to a third party that replicates server content on globally distributed servers for load balancing and failover. One example is *www.intel.com*, which is the first in a chain of four different aliases, spanning a total of four distinct zones.

Trust anchor dependencies. A validating resolver must maintain the trust anchors with which it has been configured. The DNSKEYs corresponding to the anchors may periodically change (e.g., as corresponding KSKs are replaced during routine rollovers [10]), and the resolver must maintain the current version of the trust anchor, or validation of the domain names at and below the zone to which the trust anchor belongs will fail. RFC 5011 specifies a protocol by which resolvers can automate this process in-band using already established trust.

4. VISUALIZING DNS AUTHENTICATION

In this section we discuss our visual model for DNSSEC analysis. We present a DNSSEC authentication chain for a RRset, r, as a directed graph, $G_r = (V, A)$. Node and edge placement are discussed in the remainder of the section.

4.1 Graph Nodes

Each node $u \in V$ is classified as one of several types:

- *RRset node*: A DNS RRset (e.g., *example.com*/A).
- *DNSKEY node*: A DNSKEY RR.
- *DS node*: A DS RR.
- *NSEC node*: A set of one or more NSEC or NSEC3 RRs returned by an authoritative server, which collectively prove non-existence of a DS RRset for a delegated child zone.

Different shapes are used to visually distinguish different types of nodes. All nodes are labeled with their RR type,

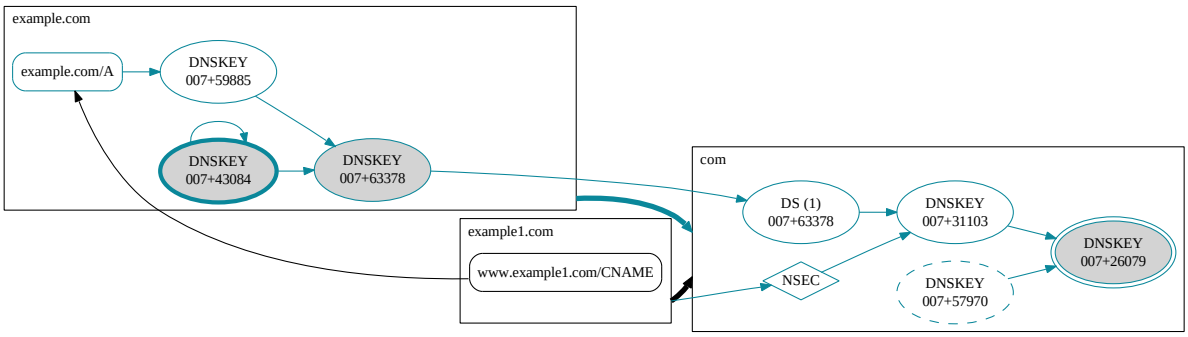


Figure 2: A graphical representation of the DNSSEC authentication chain for *www.example1.com*.

(a) SEP flag (b) Revoke flag (c) Published only (d) Trust anchor (e) Non-existent

Figure 3: `DNSKEY` node attributes.

and `DNSKEY` and `DS` nodes are additionally labeled with identifying information, such as DNSSEC algorithm, key tag, digest method (for DS RRs) to distinguish one from another. Different node types are shown Figure 1. Figure 1(a) represents the RRset with name *www.example.com* and RR type `A`, Figures 1(b) and 1(c) correspond to a `DNSKEY` and `DS` RR, respectively, both with DNSSEC algorithm 5 (RSA-SHA1) and key tag 54302. The `DS` RR uses digest method 1 (SHA1). Figure 1(d) is an arbitrary `NSEC`, whose meaning depends on its placement in the graph.

Each node is associated with the zone in which its corresponding data is maintained. These nodes are visually clustered by their corresponding zones. This allows an administrator to easily identify the area of administrative control from which a particular problem originates. Each zone cluster is labeled with the origin of the zone. An example of the different node types and zone clustering is shown in Figure 2.

While attributes of `DNSKEY` RRs don't necessarily correspond to their DNSSEC authentication role, they may be represented visually by modifying appearance of the corresponding nodes, so they can be easily identified by administrators to ease troubleshooting. Figure 3 demonstrates these attributes. We distinguish a `DNSKEY` with the SEP flag set by filling the background of its node, as opposed to a clear background when the flag is not set. If the revoke flag of a `DNSKEY` is set, we enlarge the border surrounding the corresponding node; otherwise, the border is of normal size. If the border of the `DNSKEY` is dashed, then the `DNSKEY` is published only—i.e., it is not being used to sign any RRsets. If the user designates a `DNSKEY` node as a trust anchor, then the corresponding node has a double border.

Nodes of any type that are required but missing are filled with red. An example is the signature of an RRset that was made by a non-existent `DNSKEY`.

4.2 Graph Edges

An edge has one of several different meanings in G_r, and the significance of each edge $(u, v) \in A$ depends on the destination node, v. The color of the edge conveys the validity of the relationship between the two nodes. For example, an edge representing a signature is colored differently depending on whether it is valid or not. Edge types are listed below, by destination node, and examples reference Figure 2.

- `DNSKEY`: An `RRSIG` created by `DNSKEY` v covering an RRset corresponding to u, which is either a `DS` node, `DNSKEY` node, `NSEC` node, or RRset node. Valid signatures are colored blue, and invalid signatures are colored red. Example: the edge from *example.com*/`A` to `DNSKEY` 007+59885 represents a valid signature.

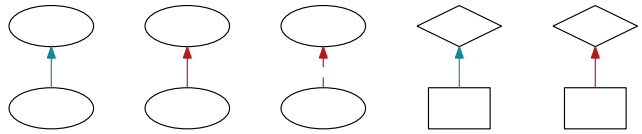
- `DS`: A cryptographic digest of `DNSKEY` u corresponding to the `DS` RR data of v. Valid digests are colored blue, and invalid digests are colored red. Example: the edge from `DNSKEY` 007+63378 to `DS` (1) 007+63378 represents a valid digest.

- `NSEC`: Node u is the zone cluster corresponding to the child zone of the insecure delegation, and v is the set of `NSEC` node proving non-existence of `DS` RRs for the child zone in its parent. If the set of `NSEC` or `NSEC3` RRs in the comprising the `NSEC` node is sufficient to prove non-existence, then the edge is colored blue; otherwise, it is colored red. Example: the edge from the *example.com* cluster to the `NSEC` node in *com* represents the `NSEC` RRs included in *com* to establish an insecure delegation.

- RRset: An alias relationship, wherein RRset u aliases RRset v (i.e., with the `CNAME` RR). Alias edges are always black, as there is no valid/invalid status. Example: the edge between *www.example1.com*/`CNAME` and *example.com*/`A` represents an alias relationship.

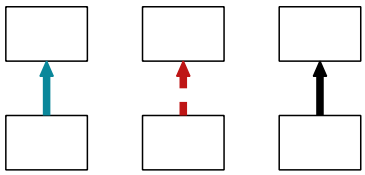
These rules are summarized in Figure 4.

Edges between zone clusters represent delegations from one zone from another (i.e., from parent to child zones). They are distinguished from edges between DNSSEC-related nodes by their thickness. The color of delegation edges also has significance, depending on whether the delegation is secure



(a) Valid RRSIG or digest (b) Bogus RRSIG or digest (c) Missing RRSIG (d) Sufficient NSEC covering (e) Insufficient NSEC covering

Figure 4: Edge types.



(a) Secure (b) Broken (c) Insecure

Figure 5: Delegation status.

(blue), insecure (black), or broken (red and dashed). These are illustrated in Figure 5. In Figure 2 the only statuses exemplified are secure (*com* to *example.com*) and insecure (*com* to *example1.com*).

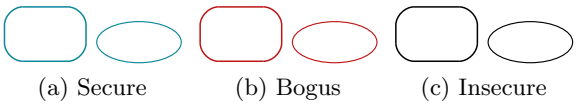
Edge placement is determined by the existence of signatures and the role of DNSKEYs in a zone (e.g., ZSK, KSK, revoked). For example, in Figure 2 DNSKEY 007+63378 is a KSK, not because its SEP bit is set (although it is set), but because it draws edges from other DNSKEY nodes. Alternatively, the only in-edge for DNSKEY 007+59885 is from zone data, indicating that it is a ZSK, independent of the status of its SEP bit. DNSKEY 007+43084 is revoked because it both has the revoke bit set and is self-signed.

Relationships required but missing are represented by a red, dashed edge. An example is the lack of a signature for an RRset in a signed zone.

4.3 Trust Propagation

The graph displayed in Figure 2 is conveniently lacking problems, such as invalid signatures. However, such issues may be found in real-world deployments. Regardless of problematic edges or nodes in the DNSSEC authentication graph, the bottom line for proper validation is an unbroken chain from a trust anchor to the RRset being queried. For example, if no trust anchor is designated, or if problems occur below a properly configured insecure delegation, the problems do not directly impact validation. We reserve the border color of nodes in the graph for designating the security level of the node: secure (blue), bogus (red), insecure (black). These statuses are shown in Figure 6.

Nodes are colored by beginning at the trust anchor, and recursively following and coloring predecessor nodes, depending on edge characteristics. Nodes which have no path to the trust anchor, but should (i.e., they are not separated from the trust anchor zone by an insecure delegation) are



(a) Secure (b) Bogus (c) Insecure

Figure 6: Node status.

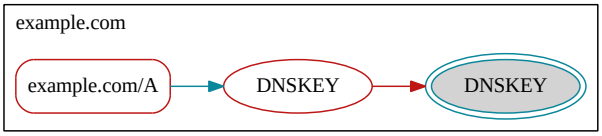


Figure 7: Bogus or expired signature.

bogus. In Figure 2 all nodes are deemed secure except *www.example1.com*/A, which is part of an insecure delegation and therefore is insecure.

5. IMPLEMENTATION

We implemented the DNSSEC visualization as a Web-based tool we call DNSViz. The images are created from a relational database containing DNSSEC data compiled by an analysis tool written in-house. Domain names used to seed the database came from user-submitted data (via the Web interface), in addition to a number of signed hostnames extracted from URLs indexed as part of the Open Directory Project at DMOZ [15].

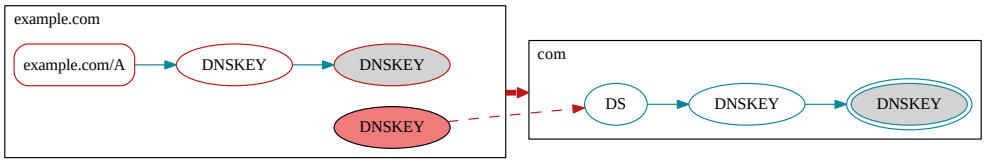
We present the graphical output representing some of the actual problems we encountered as produced by our DNSViz tool, shown in Figures 7 – 10. Although in a real-world example DNSKEY details such as algorithm and key tag values would be included in the graph, we have abstracted these from our examples to emphasize the visual effect. In each case we use *example.com*/A as the name being resolved, and the trust anchor varies.

Figure 7 demonstrates the effect of invalid signatures, which include bogus and expired RRSIGs. Note that although the RRSIG covering *example.com*/A made by *example.com*'s ZSK is valid, the chain of trust is broken by the invalid RRSIG made by the KSK, which results in a bogus result.

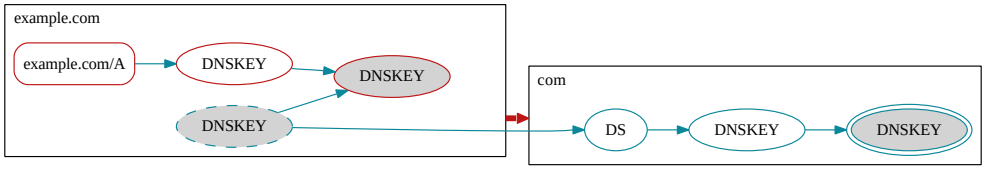
Figure 8 demonstrates two examples in which a chain of trust is broken because of a missing or "misplaced" DNSKEY in the child zone. In Figure 8(a) the is missing entirely; in Figure 8(b) it exists, but only in a published state, so there is no path to the *example.com* zone data. In either case the result is a bogus response.

In Figure 9 the insecure delegation from *com* to *example.com* is properly handled by a sufficient set of NSEC RRs, yet the signature on one or more of these NSEC RRs is missing. The result is that the resolver cannot prove that the delegation is insecure, so names in subdomain space are bogus.

The DNSKEY with revoke bit set in Figure 10 is lacking a self-signature, which is required to make it truly revoked [18]. Note that this does not directly cause a failure, but if validating resolvers or parent zones use RFC 5011 for the rollover



(a) Non-existent



(b) Misplaced

Figure 8: Missing `DNSKEY` corresponding to `DS`.

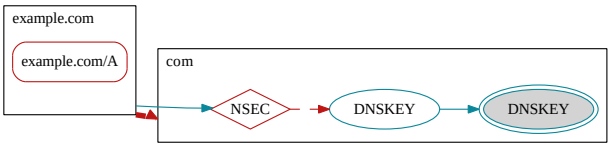


Figure 9: Missing `RRSIG` covering necessary `NSEC` RRs.

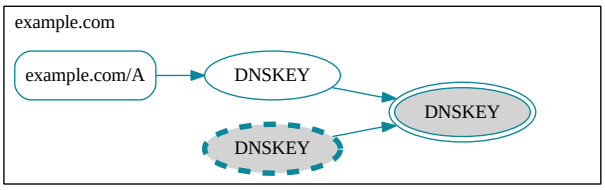


Figure 10: Improper revocation of `DNSKEY`.

of a trust anchor or KSK, then this could result in a situation such as that demonstrated in Figure 8(b) because the `DNSKEY` will not be recognized as truly revoked.

6. CONCLUSION

We have presented in this paper an intuitive convention for graphically representing the authentication chain for a DNS domain name and have described our method of implementation. This visualization helps DNS administrators gain a better understanding of their DNSSEC configurations, especially when troubleshooting validation errors. We anticipate that the functionality provided by DNSViz will facilitate DNSSEC deployment, as it has been used by actual DNSSEC administrators to understand their configurations and to identify, diagnose and fix real DNSSEC problems.

7. REFERENCES

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4033: DNS security introduction and requirements, 2005.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4034: Resource records for the DNS security extensions, 2005.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. RFC 4035: Protocol modifications for the DNS security extensions, 2005.

[4] B. Krishna. http://www.bala-krishna.com/web-tools/domain-dns-record-check-tool/.

[5] D. Barr. RFC 1912 - common DNS operational and configuration errors, 1996.

[6] C. Deccio, C.-C. Chen, J. Sedayao, K. Kant, and P. Mohapatra. Quality of name resolution in the domain name system. In *ICNP '09, Proceedings*, 2009.

[7] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra. Measuring availability in the domain name system. In *INFOCOM 2010. The 29th Conference on Computer Communications*, 2010.

[8] DNSCheck by .se. http://dnscheck.iis.se/.

[9] ISC BIND. http://www.isc.org/products/BIND/.

[10] O. Kolkman and R. Gieben. RFC 4641: DNSSEC operational practices, 2006.

[11] O. Kolkman, J. Schlyter, and E. Lewis. RFC 3757: Domain name system KEY (DNSKEY) resource record (RR) secure entry point (SEP) flag, 2004.

[12] ldns by NLnet Labs. http://www.nlnetlabs.nl/projects/ldns/.

[13] P. Mockapetris. RFC 1034: Domain names - concepts and facilities, 1987.

[14] P. Mockapetris. RFC 1035: domain names - implementation and specification, 1987.

[15] Open Directory Project.

[16] V. Ramasubramanian and E. G. Sirer. Perils of transitive trust in the domain name system. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 379–384, Berkeley, CA, USA, 2005. USENIX Association, USENIX Association.

[17] squish.net dns checker. http://www.squish.net/dnscheck/.

[18] M. StJohns. RFC 5011: Automated updates of DNS security (DNSSEC) trust anchors, 2007.