Deploying DNSSEC Without a Signed Root

Samuel Weiler

Information Networking Institute

Carnegie Mellon University

April 2004

Table of Contents

Abstract

DNS Security (DNSSEC) authenticates DNS data by building public-key signature chains along the DNS delegation chain from a root of trust, ideally the DNS root. Due to a myriad of technical and political concerns, it appears unlikely that many delegation-heavy zones, including the root and most generic top level domains (gTLDs), will sign their zones in the near future, which leaves DNS resolvers with no means to validate data from the children of those zones without maintaining a large number of preconfigured keys.

This paper presents two schemes for publishing secure delegation information outside of the DNS delegation chain. These will allow resolvers to validate data from zones whose parents either aren't signed or refuse to publish secure delegations to their children. Additionally, we introduce synthesis of negative DNS answers as a means for mitigating query load, and we show how DNSSEC security policy, once thought to be solely a resolver concern, impacts authoritative server load and protocol design.

1 Introduction

The DNS Security (DNSSEC) extensions aim to provide data authentication for the DNS as an aid to detecting and deterring assorted attacks, including the spoofing of DNS answers. DNSSEC has been in development for about ten years, and it appears that many of the technical problems that have plagued it have been worked out and it may be nearing deployability. Unfortunately, since it depends on a trust hierarchy that follows the DNS delegation hierarchy, DNSSEC's utility is largely dependent on having the root and top level domains (i.e. .com and .net) signed. Due to both technical and political constraints, it's unlikely that many of those zones will be signed in the immediate future.

Rather than tackle those technical and political hurdles, this paper presents two mechanisms, called Trust Authorities (TAs), that remove DNSSEC's dependency on having those high-value zones signed, by creating alternate trees of trust data. These mechanisms are designed to be sufficiently scalable, efficient, and reliable to be used as viable alternatives to a signed root.

1.1 DNS Background and Terminology

The domain name system (DNS) is a global distributed database primarily used for mapping hostnames to IP addresses and IP addresses back to names. It has a hierarchical namespace with a unique global root, and the namespace is broken up into zones. Within a zone, a domain name (i.e. www.example.net.) may have multiple DNS resource records (RRs) of multiple types (i.e. A, MX, SOA, NS). All records of the same name and type are known as a resource record set (RRset).

A parent zone can delegate a portion of its namespace to a child zone by inserting one or more NS RRs at some name in its zone. This name is known as a zone cut or delegation

point. The NS records give the domain names of the child zone's authoritative nameservers, which are responsible for answering queries for all names below the zone cut. (For example, the zone net. may delegate the example.net. namespace by publishing an NS record at the name example.net.) Both the parent and child zone contain records at the zone cut, also known as the apex of the zone: the parent contains only an NS RRset, and the child contains NS, SOA, and perhaps other RRsets.

A DNS client, known as a recursive resolver, caching resolver, or simply a resolver[1], looks up RRsets, identified by query name, query type, and query class (QNAME, QTYPE, QCLASS) by starting at the authoritative servers for the DNS root. In most cases, the root returns a referral to one of it's children that is responsible for the namespace identified by the QNAME (i.e. in response to a query for www.example.net., the root might return a referral to the authoritative servers for .net.). The resolver recursively queries the authoritative servers for each such delegation until it reaches the zone which is authoritative for the data sought. The authoritative servers for that zone return the data or an answer showing that no matching data exists.

Most end-user computers, rather than run their own recursive resolvers, include only stub resolvers. Stub resolvers depend on caching resolvers, which are commonly run by internet service providers (ISPs). These resolvers typically cache all data received for a length of time (TTL) indicated by the authoritative server. This widely deployed caching infrastructure has contributed greatly to the scalability and success of the DNS.

The DNS is very complex and many details of it are omitted from this discussion, including classes other than IN, wildcard processing, and many special rules for handling certain RRs (i.e. CNAME). While consideration of those details is vital when extending

1 Although it is possible to have a non-caching recursive resolver, it is common for all recursive resolvers to include a cache. The terms resolver, recursive resolver, full-service resolver, and caching resolver are used interchangeably in this paper.

the DNS protocol, they have not necessitated changes to the designs presented here. Anyone contemplating modifying these designs is advised to pay careful attention to the DNS specifications and to the lessons learned in the development of DNSSEC.

1.2 DNSSEC Background and Terminology

DNS does not provide any data authentication or other provisions to prevent the malicious modification or substitution of DNS data. Such attacks are very easy, and simple tools to perform them are widely available.

The DNS Security (DNSSEC) extensions were designed to address these weaknesses by using chains of public-key signatures to authenticate DNS data. Such signatures, combined with the data they sign, essentially form a certificate chain, similar to that used by SSL or PGP. These chains start with a trusted key for some zone, ideally the root, that is preconfigured into DNSSEC-aware resolvers or validators, and proceed along the DNS delegation hierarchy to authenticate a given piece of DNS data.

DNSSEC introduces four new RR types to the DNS: the DNSKEY, RRSIG, DS, and NSEC RRs. We'll first discuss the DNSKEY, RRSIG, and DS records, which are used to build DNSSEC chains of trust, then we'll discuss the NSEC RR. DNSKEY RRs appear at a zone apex and store public keys used to authenticate that zone's data. Each RRset (all RRs of the same name and type) in that zone is signed by the private key(s) corresponding to one or more of those DNSKEYs. The digital signatures are stored in RRSIG RRs, which also include a time interval during with the RRSIG is considered to be valid. RRSIG RRs, together with the corresponding DNSKEYs, authenticate a particular RRset. In this paper, we'll use the notation RRSIG(A) to denote an RRSIG that authenticates an A RRset.[2] DS (Delegation Signer) RRs may appear at a delegation point

2 A records store IPv4 addresses

in the parent zone and, along with an RRSIG(DS), authenticate a particular DNSKEY in the child zone. The child zone may use that particular DNSKEY, known as a secure entry point (SEP) key or key signing key (KSK) to sign it's DNSKEYset. Any of the DNSKEYs in the DNSKEYset may then be used to sign other data in the zone.

Any given DNS zone may choose to be signed or not. If a zone's parent includes a DS RR for the zone (along with the usual NS RRset), it is said to be secure (or a secure delegation). If there is no DS in the parent, it is unsecure (or an unsecure delegation). In essence, the presence of a DS RR in the parent tells a validator to expect the child to be signed. Similarly, a preconfigured key tells a validator to expect some zone (again, ideally the root) to be signed. If such an expectation is established for a given zone, whether by the validator having a preconfigured key for it or by having a signed DS at the zone's parent, the validator may give an error if any data in that zone fails to validate.

The NSEC (Next SECure) RR is used to prove which data is and is not in a zone. One NSEC appears at each name in the zone, and it lists the RR types present at that name as well as the next name in a canonical ordering of the zone. A NSEC RR (along with the RRSIG(NSEC)) proves that there are no other records in the zone between the record which is the name of the NSEC and the 'next name' to which it points. The NSEC at the last name in that canonical ordering points back to the apex of the zone, forming a complete ring. Authoritative servers generally send NSEC RRs as part of proofs that data does not exist (i.e. when an NXDOMAIN response would be appropriate). In particular, a signed NSEC is sent by each parent sending a delegation to an unsecured zone to prove that is has no DS RR for that child.

As currently defined, DNSSEC allows a resolver to classify data into one of three categories: secure, (verifiably) unsecure, or bad/bogus. Absent a preconfigured trusted key, all data is necessarily unsecure. Any data for which the validator can't determine a security status, whether because the appropriate RRs weren't accessible or because a

cryptographic signature validation failed, is classified as bad.

The DNSSEC protocol specifications are currently scattered across a number of RFCs dating back to 1999, many of which are nearly unreadable. An effort is underway in the IETF's DNS Extensions (DNSEXT) working group to rewrite and clarify those specifications, consolidating all of the changes into a single set of three documents. [DNSSECbis-1][DNSSECbis-2][DNSSECbis-3] Although those three documents, commonly known as DNSSECbis, are only available as works-in-progress, their technical content is relatively stable and they are nearing publication as RFCs. Because they contain some important clarifications, this paper is based on the understanding of DNSSEC contained in them.

1.3 Unsolved Problems: Scaling

DNSSEC causes significant growth in both zone size and DNS query response size. Specifically, DNSSEC adds an NSEC RR for each name in a zone and an RRSIG RR for each RRset at that name, including the NSEC RR. Each RRSIG is likely to be much larger than the RRs it covers. An RRSIG record has an 18 octet preamble, a domain name (indicating the zone that generated the signature), and a digital signature. Assuming that a 1024 bit RSA key is used, an RRSIG is likely to be over 150 bytes long. For zones consisting primarily of A or PTR records, this adds three records per name in the zone, increasing the zone size by an order of magnitude.

At a delegation, DNSSEC adds at least two RRs to the parent zone: an NSEC RR and and RRSIG(NSEC). Compared to an unsigned delegation, which likely contained two NS records and nothing else, DNSSEC at least doubles the number of records at a delegation and increases the number of bytes at the delegation by an order of magnitude (assuming signatures made by 1024 bit RSA keys). Secure delegations add at least two more

records (DS and RRSIG(DS)), causing another factor of two growth.

For both of the above cases, the described growth occurs in both the zone size and the response size. For a small site running a few zones incidental to the operation of a small network (a small university, for example), DNS probably consumes only a small portion of the site's available bandwidth and the zone size growth doesn't impose significant new network or hardware provisioning requirements. For zones with a large number of records or referrals, the zone size growth can be particularly problematic -- at least one popular open-source authoritative nameserver requires that the entire zone be loaded into RAM, and the growth caused by DNSSEC can force the use of 64-bit hardware for serving those zones. Even if the zone is small, a zone with a high query load may also need to increase its network provisioning to deal with the extra load caused by DNSSEC. To take a real example, researchers at NLnet Labs saw the .nl zone grow by a factor of five when they signed it with a 1024-bit RSA key, even though it contained almost no secure delegations (DS records), their NSD authoritative server software needed 4-6 times the memory to serve a signed zone[Rozendaal].

As a result of this growth in both hardware and bandwidth requirements, large infrastructure zones including the generic top-level domains (gTLDs) such as .com, .net, and .org face a large hurdle for adopting DNSSEC. Representatives of VeriSign, the operator of the .com and .net zones, have said that the costs of implementing DNSSEC are so prohibitive that VeriSign is unlikely to find a business case sufficient to justify signing .com and .net.

VeriSign proposed a DNSSEC extension to mitigate some of the effects of this zone growth. Called opt-in, this extension allowed zones to mark spans between NSEC records as unsigned and allowed unsigned delegations to appear in those spans. This permitted various attacks to names in those spans including allowing delegations within a span to be spoofed away and allowing bogus delegations to be inserted. The benefit of

opt-in was that only secured delegations (ones with DS records) required NSEC and RRSIG(NSEC) records -- while the remaining delegations were allowed to have NSEC and RRSIG records, those records were not required. Opt-in would have allowed delegation-heavy zones such as .com to grow approximately linearly as secured delegations were added. It would also have allowed for registry business models that charged for having a signed delegation, whether or not the delegation was secure (had a DS record). The IETF's DNSEXT working group failed to reach consensus to add opt-in to the DNSSEC specifications, and it appears unlikely that opt-in will be revived.

Scaling presents a relatively minor problem for the root zone. With only ~300 entires in the zone, zone file growth is not a problem. Response size growth would be a problem except for the efforts that the root operators have made in recent years to build capacity, largely in response to distributed denial of service attacks.

1.4 Importance of a Signed Root and Signed TLDs

It's possible for zone administrators to sign their zones without involvement from their parents, but clients will not be able to validate those signatures without a secure entry point into the zone. If the parent isn't providing that in the form of a signed DS record, a client's only alternative is to preconfigure a key for the zone. While that's certainly possible, it's impractical for resolver operators to preconfigure keys for a large number of signed zones. As discussed above, keeping even a single preconfigured key up-to-date is expected to present a challenge to most resolver operators.

Since much of the value in DNSSEC is in allowing parties that have limited or no trust association with each other to communicate securely, it's desirable for resolvers to have access to as wide a range of secure entry points as possible. The obvious way to achieve that is by signing high value zones, such as the root and TLDs. As discussed above, for a

variety of economic and political reasons, it is unlikely that many of those high value zones will be signed in the immediate future.

2 Problem Statement

This paper presents two ways of publishing and retrieving information about secure delegations that don't require the cooperation of the secured zones' ancestors. These schemes make it possible for resolvers to validate data from a large number of signed zones without requiring that each resolver maintain a large list of preconfigured trust associations and without requiring high-value zones (such as the root and TLDs) to be signed.

The primary design constraint for both schemes is deployability -- they need to be both technically and politically feasible. On the technical side, that means that the code, particularly resolver code, must not be excessively complicated; that the mechanism not impose excessive load on the network; and that the startup cost to run a trust authority be as low as possible. Furthermore, the schemes must not cause harm to non-participants. On the political side, the schemes should not be vendor-specific nor create any artificial monopolies, lest they be shunned by potential users as having too little utility or disproportionately benefiting some party.

This paper assumes that zones wishing their own data to be secured are capable of signing their own zones, but that an ancestor zone is either unwilling to be signed or unwilling to include a DS record for the relevant child zone.

3 The Designs

Since the primary motivation behind this work is the need to work around particular high-value zones (such as the root and TLDs) not being signed by their respective zone operators or registries, it is valuable to specify a scheme that is only usable to make trust statements about a single zone. A single-zone scheme is also much simpler than a scheme that will work for an entire hierarchy. A scheme that will work for multiple zones is discussed in section 3.2.

3.1 Single Zone, Single Label Trust Authorities

A Trust Authority (TA) is a portion of the DNS namespace that contains trust statements about zones that are not its own children. Specifically, it contains the keying information needed to authenticate answers from those zones. For simplicity, TA domain must be used exclusively as a TA; it must not contain other data.

This section presents a design for a Trust Authority (TA) that can make statements about children of a single zone, called the 'target zone'. This particular design is limited to making trust statements about children of a target zone which only have one additional label. For example, a TA for .net may include statements about example.net but may not include statements for child.example.net. Such a TA would be most usefully used when the target zone has few or no multi-label delegations. The root, most TLDs, and several zones that act as TLDs (i.e. co.uk, ac.uk) fit that description.

Throughout this section, we will use as an example the ta.com zone, which is a TA for the target zone .net. ta.com is assumed to be signed with DNSSEC, and a DNSSEC-aware resolver is assumed to be configured to use ta.com as a TA for .net. The resolver is also assumed to have a secure entry point into ta.com, which may be a preconfigured key for ta.com, a preconfigured key for .com along with a secure delegation (DS record) into ta.com or a preconfigured key for the root along with a secure delegation into .com

and from .com to ta.com.

A TA consists of a DNS zone containing one name for each of the target's child zones that has registered security information with the TA. For a given zone, the corresponding name in the TA zone is formed by replacing the target zone name with the TA zone name. Using the example, information concerning the zone example.net can be found at example.ta.com.

At each such name, there is a TA resource record. The TA resource record uses the same format as the Delegation Signer (DS) record, shown in Figure 1.

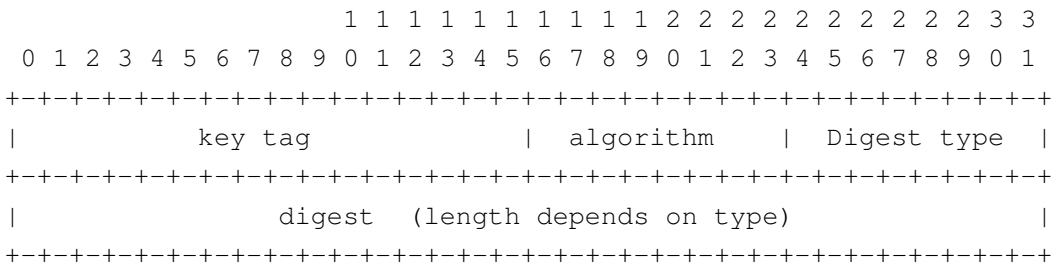

Figure 1

The fields in the TA record contain exactly the same data as the DS record and use the same IANA-assigned values in the algorithm and digest type fields as the DS record. (Those IANA registries are known as the "DNS Security Algorithm Numbers" and "DS RR Type Algorithm Numbers" registries.)

When a TA wants to include a record for a particular child of the target zone, it merely inserts a TA RR into its zone, using the naming convention described above. Just as with the DS records, the digest field of the TA record contains a hash of a DNSKEY used to sign the apex DNSKEYset in the target's child's zone.

3.1.1 Resolver Algorithm

In summary, a resolver first gets the answer it desires using normal DNS lookups, then validates the answer using a TA.

To validate a record, a resolver checks its preconfigured list of TA's for any that are applicable to the name being queried (the QNAME). It starts by looking for the most specific match then strips off leading labels until a match is found. Having found such a TA, it asks that TA for the TA RR corresponding to the name being sought. If there is a corresponding TA RR, the resolver validates it then uses it as though it were a DS RR to validate the remainder of the answer.

Using the example above, to look up the www.example.net A RRset, a resolver first completes its normal lookup process for the A RRset. Then it searches its list of preconfigured TA's, starting by looking for a www.example.net TA and proceeding to a .net TA. Having found a TA for .net, it then asks that TA for the example.ta.com TA RR. If there is a TA RR, and it validates, the resolver treats that TA RR as a DS RR for example.net and proceeds with validating the www.example.net A RRset (or the negative response indicating that no such RRset exists). Specifically, the resolver uses the TA RR to validate example.net's DNSKEY RRset then uses the DNSKEY RRs to validate the www.example.net A RRset.

If there is no TA RR for example.ta.com, then no trust information is available, and the resolver proceeds just as though .net were signed and contained no example.net DS record, treating the result of the A record lookup as an insecure response. A special case occurs when the .net nameservers say that there is no example.net delegation, but there exists an example.ta.com TA RR. In that case, the presence of a TA RR asserts that the example.net zone should exist, and the resolver should treat the non-existence answer as

bad data. Similarly, a TA lookup occurs even if the original query returns NXDOMAIN (the usual response showing that a name doesn't exist) -- if a TA record exists at the QNAME, real signed data should have existed there, too.

It's quite possible that the zone containing the sought-after data will be more than one delegation below the TA's target zone. In such cases, the TA is treated as a secure entry point into the target's child, then normal DNSSEC chaining is used below that point. For example, if child.example.net is a secure delegation from example.net, meaning that example.net contains a DS record for child.example.net, the example.net TA RR and the example.net DNSKEYset can be used to validate the child.example.net DS record, etc.

3.2 Hierarchical Trust Authorities

Extending the above design to handle the entire DNS hierarchy or some subtree of it requires only minor changes to the data stored. Using a hierarchical TA, though, makes the resolver logic noticeably more complex.

A hierarchical TA tree, rather than being a flat zone with single-label names, may have depth and the target, rather than being a single zone, becomes an entire hierarchy. Using the example in the previous section, where ta.com was a TA targeting .net, ta.com could contain TA records named example.ta.com as well as child.example.ta.com. It could also have a TA record at the name ta.com that refers to the .net zone. A TA targeting the root could have TA records for any DNS name, including one for the root itself. Within this tree, there can be normal secure delegations (using DS records, just as in the main tree).

3.2.1 Trust Paths

Providing an alternate hierarchy of trust data allows resolvers to implement multiple

security policies. These are primarily of interest when validation would give different results depending on which tree(s) is used. In this discussion, the term "trust path" refers to a chain of signed DS, DNSKEY, and TA records from a trusted key (as would be preconfigured into a resolver) to the answer being validated. Absent the mechanisms introduced in this paper, DNSSEC trust paths contain only signed DS and DNSKEY records.

A resolver can use a trust path that starts in a TA tree and then flows into the main delegation tree. It's not possible, as currently specified, to follow a path from the main delegation tree back into a TA tree. Thus, the number of possible trust paths to any given RR, assuming only one TA tree, is limited to the number of labels in the name of the RR plus two. For example, a resolver attempting to validate an A record at www.example.net can follow five paths:
1) using only the main delegation hierarchy (DS and DNSKEY records), starting at a preconfigured key,
2) starting with a TA record for www.example.net, looking for DNSKEYs at www.example.net, and using those DNSKEYs to validate the A record.
3) starting with a TA record for example.net,
4) starting with a TA record for .net, or
5) starting with a TA record for the root.

It's possible for each of these trust paths to give different validation results.

3.2.2 Resolver Algorithms

Which trust paths a resolver uses can have a noticeable impact on the utility of TAs and on the DNS query load experienced by the TA servers. In this section, we introduce five possible resolver policies (again, assuming only a single TA), then discuss the resolver

algorithm required to implement each and the impact of those algorithms on the TA servers and the DNS as a whole.

Each of these algorithms is complicated by the fact that DNS does not require a delegation at every label cut. For example, the zone foo.bar.example.net could be a child of example.net; there need not be a bar.example.net zone nor even any record with that name. While we can specify that TA records may only exist at zone cuts, depending on the policy chosen, a resolver may not be able to rely on the main delegation tree to tell us where zone cuts are.[3] Instead, the resolver must search the TA tree looking for TA records at all possible zone cuts.[4] The policies discussed are:

1. Accept any success
2. Failure trumps
3. Closest encloser trumps
4. Parent masks
5. Accept first answer

Using Policy 1, "accept any success", a resolver looks for any trust path that validates. If a path results in a validation failure or in a proof of insecurity, the resolver keeps trying other paths until it gets a success or exhausts all possible paths. If no path results in a successful validation, and at least one fails validation, a failure is returned.

Using policy 2, "failure trumps", a resolver explores all possible trust paths until it finds

3 If a resolver relies on the main delegation tree to report where zone cuts are, it leaves itself vulnerable to attack if any of its ancestors are insecure -- an attacker need only forge an answer from an insecure ancestor, spoofing away all of the delegations down to the QNAME. Since the resolver sees no delegations, it doesn't check the TA tree as described in section X to see if such delegations should have existed.

4 The algorithm for this is similar to the algorithm described in RFC3658 section 2.2.1.2 for finding a zone's parent. Before DNSSEC, it was never necessary to find a zone's parent (or any other zone cut), and the protocol has no provision to do so.

one that fails. If any path results in a validation failure, the entire result is treated as a failure.

Using policy 3, "closest encloser trumps", a resolver looks for the longest TA record which could be an ancestor of the requested name, and attempts validation from that point downward. With this policy, a TA record for www.example.net would take precedence over TA records for example.net and its parents. This is the same policy that is described in section 3.1 (Single-Zone, Single-Label TAs) for choosing which TA to use.

Using policy 4, "parent masks", a parent's TA record takes precedence over any for it's children. If there is a TA record for .net, a TA record for example.net will be ignored.

Using policy 5, "accept any answer", a resolver can start with any of the possible trust paths. The first validation result is taken as the result of the validation.

With all of the above policies, it is assumed that if all paths show the answer to be insecure (no relevant TA records and no path in the main tree), the answer will be flagged as insecure.

A resolver is not completely free to choose among these policies: some violate design constraints on the problem and others have a major impacts on the deployability of TAs, mostly due to the query load they impose on the TA zones' servers.

Policy 5, accept any answer, can be quickly discarded because it allows for a resolver to get a different validation result depending on what queries it has seen before. In general, to minimize query load, a resolver would choose those trust paths that can be tested based on cached data. Since contents of a resolver's cache depends almost entirely on the history of queries it has received, the choice of path taken, hence the validation result,

would depend on the history of queries that resolver has made.

Policy 4, parent masks, can also be discarded. It is an extremely efficient policy, particularly when many TLDs or the root are signed, but it needlessly limits the utility of TAs. It prevents a TA from having both an entry for the root and entries for second-level names, such as example.net.

Policy 3, closest encloser trumps, is an attractive policy. First, it helps zones work around the problem of an uncooperative or unresponsive parent that refuses to update a DS record. A resolver using this policy, though, must generate at least one query to the TA tree for each new query name. It starts by searching the TA tree for a TA record of the exact name being sought. If it finds one, it uses it for validation. If it doesn't find one, it removes one label from the QNAME and tries again, all the way up to the target zone of the TA. This algorithm is relatively simple and easy to understand. The main drawback to it is that the TA servers become uber-resolvers -- every new query handled by each resolver that uses that TA winds up hitting the TA servers.

Policy 1 and 2 have similar algorithms and similar performance impacts. Each makes excellent use of caching, particularly when the root and TLDs are signed. To minimize query load, resolvers using either policy would first test those trust paths that can be most easily tested using only cached data. Only then would they begin querying the TA tree for new data. Absent further optimizations such as the 'closer encloser answers' discussed in section X, it would be most efficient for these resolvers to then look for a TA RR for the TA's target zone and then proceed to add labels until they reached the QNAME. At the first validation success (in the case of policy 1) or failure (in the case of policy 2), an answer could be returned.

Policy 2, failure trumps, makes the DNS unnecessarily brittle. With it, any misconfiguration at any ancestor zone can make a zone unreachable. For that reason, this

policy is best avoided. It is also more inefficient when data is properly signed, since it exhaustively looked for bad data.

Policy 1, accept any success, is a very attractive policy. It is the most liberal of the policies discussed, providing the most opportunities to work around an uncooperative or malicious parent zone. It can also be very efficient when data is properly signed. Policy 1 can be much less efficient.

In summary, policies 1 and 3 are noticeably more appealing than 2, 4, and 5 because both provide a trivial means for working around an uncooperative or malicious ancestor zone. Because policy 3 will impose so much more load on TA server than policy 1, policy 1 is recommended.

3.3 Design Commonalities

3.3.1 Scaling

Both of the designs presented here require, as part of the resolution algorithm, one or more queries to the TA zone(s) for each client DNS lookup.

For both designs, query load on the TA servers scales linearly in the number of resolvers using the TA. For the single-zone design, the load is a proportion of the load seen by the target zone's authoritative servers. For the hierarchical design, the load is a proportion of the load seen by the authoritative servers for all zones in the hierarchy targeted by the TA.

It's reasonable to expect that some TA's will be sparsely populated compared to their target zones. In particular, it's likely that any TA for a global top-level domain such as

.com will have vastly fewer entries than its target zone. To facilitate deployment of a TA, it would be preferable if the query load on the TA servers scaled linearly as the number of TA entries increased and tapered off to near zero when there are few entries in the TA zone. Aggressive negative caching, presented below, accomplishes this.

3.3.2 Aggressive Negative Caching

One of the long-standing assumptions about DNS cache design has been that resolvers should cache all answers, including negative responses, indexed by (QNAME, QCLASS, QTYPE). A cache is not expected to synthesize an answer to any query; it is expected to answer only those questions when it has been asked before.

DNSSEC slightly changes those caching rules by adding an additional bit to the index tuple -- the CD, or checking disabled, bit. This is a bit that can be set by a resolver (or stub resolver) instructing other resolvers to not perform DNSSEC validation. It is supposed to be set only by a resolver that is itself capable of DNSSEC validation and is intended to allow such a resolver to receive data that might otherwise be blocked by a misconfigured resolver/validator.

DNSSEC's architecture also allows for a more drastic change to the caching architecture for negative answers. NSEC records, since they specify a range in which no names are to be found, give a caching resolver information about more names that just the one queried for. It is possible for a resolver to take an NSEC from a previous query and synthesize an answer to a different query. For example, a resolver that, in response to a query for UL., had received an NSEC for the name UK. showing a next name of US., could infer that UM. does not exist and could synthesize an answer to that effect.

Such synthesis has previously been discouraged because it needlessly extended the scope

of DNSSEC -- there was reluctance to use DNSSEC to add new features to DNS (such as query load reduction) that weren't specifically related to data authentication. There was also some fear that aggressive negative caching would make the DNS more brittle by introducing new failure modes. The dangers of negative caching chronicled in RFC2308 section 9 may have driven this concern.

With a sparsely populated TA, aggressive negative caching is very helpful in reducing query load on the TA servers. Aggressive negative caching caps the number of queries a resolver will issue to a TA hierarchy to twice the number of entries in that TA hierarchy -- one query for each entry, and one query for each NSEC. In a very sparsely populated TA hierarchy (i.e. one that contains 1,000 entries for children of .net), even a very active resolver will probably issue no more than ~1,000 queries to the TA servers during every TTL.

3.3.2.1 Aggressive Negative Caching: NSEC Validation

Aggressive negative caching increases the need for resolvers do some basic validation of incoming NSEC records before caching them. In particular, the 'next name' field in the NSEC record must be within the zone that generated (and signed) the NSEC. Otherwise, a malicious zone operator could generate an NSEC that reaches out of its zone -- into its ancestor zones, even up into the root zone -- and use that NSEC to spoof away any name that sorts after the name of the NSEC. We call these overreaching NSECs. More insidiously, an attacker could use an overreaching NSEC in combination with a signed wildcard record to substitute a signed positive answer in place of the real data. This checking is not a new requirement -- these attacks are a risk even without aggressive negative caching.

However, aggressive negative caching makes the checking more important. Before

aggressive negative caching, NSECs were cached only as metadata associated with a particular query. An overreaching NSEC that resulted from a broken zone signing tool or some misconfiguration would only be used by a cache for those queries that it had specifically asked before. Only an overreaching NSEC actively served by an attacker could cause misbehavior. With aggressive negative caching, an overreaching NSEC can cause more broader problems even in the absence of an active attacker. This threat, which can be easily mitigated by checking the bounds on the NSEC, can be compared with the cache poisoning attacks which have long plagued DNS.

It is suggested that all validators using a trust authority implement aggressive negative caching.

3.3.3 The TA Resource Record

Both of these designs use a new DNS RR type, the TA RR. This choice was driven by two factors: the unsuitability of existing type codes, particularly DS, and the availability of new type codes.

Reuse of the DS RR might appear to be very sensible, given that the information needed in the TA record is exactly the same as that in the DS record. The DS RR, though, has some very special semantics that make reusing it risky. First, resolvers only expect to see it at a zone cut. It's not known how resolvers will behave when they see a DS that is not at a zone cut. Furthermore, the DS RR exists only at the parent side of a zone cut, and it's the only record with that distinction. That characteristic has caused numerous compatibility problems with legacy resolvers [RFC3755] [RFC3658]. Rather than risk causing more problems by putting DS RRs at non-delegations, it seems wiser to use a RR with more normal semantics. Furthermore, avoiding DS leaves us free to use it in the normal way in a full-hierarchy TA tree to make secure delegations. Delegating away

parts of a TA zone could help make the service more scalable.

Use of another existing DNS RR type with "normal" semantics, such as the TXT RR or the appropriately-named SINK RR, would avoid the need to allocate a new RR, but it could lead to confusion when other applications use those records to store data. The problems with reusing one RR type for multiple purposes is well-documented in RFC3445, which prohibited the use of KEY resources records (the predecessor of DNSKEY) to store keys for anything other than DNSSEC.

Using a new RR type requires that a type code be assigned by the Internet Assigned Numbers Authority (IANA). While there are type codes that can be assigned without IETF consensus, DNSSEC is presently limited to securing type codes under 128, and assignment of the codes in that range requires IETF consensus [RFC2929]. Experience has show that achieving IETF consensus for the allocation of new type codes takes many months, even in the best of cases [IPSECKEY] [RFC3755], making it unlikely that the process could be completed within the timeframe of this work. An effort is underway in the IETF's DNSEXT working group that will allow DNSSEC to secure typecodes greater than 127. For the purposes of this work, it is reasonable to assume that DNSSEC will soon be able to secure type codes greater than 127 and that IANA will assign such typecodes without great delay.

3.4 Comparison of the Designs

Each of the two designs presented in this section have merits. The single-zone, single-label design is much simpler, particularly in the resolver. It requires, though, running a separate TA for each target zone. If one wants TAs covering the root, .com, .org, and .net, four TAs must operate, and each resolver needs four preconfigured TA keys. The hierarchy design allows for a single TA to handle the entire DNS namespace with only a

single preconfigured key. Furthermore, the single-label design imposes an artificial constraint on TAs: since DNS delegations can comprise multiple labels, this design needlessly prohibits securing some subset of DNS delegations.

There is another reason, though, to prefer the single zone design. Imagine that neither the root nor .net is signed, and a hierarchical TA starting from the root has an established and thriving business selling entries in its hierarchy to children of .net. Its zone would contain no TA records for ta.com (the root) nor net.ta.com, but many thousands of records for ???.net.ta.com. If .net were to subsequently sign its zone, it might want this TA to insert a TA record at com.ta.com. If the TA did this, however, the children of .net would have little need to maintain their own entries in the TA hierarchy (since they could be securely reached through the net.ta.com entry into .net followed by the regular DS record in the delegation from .net), and the TA would presumably lose quite a bit of business. Consequently, the TA would probably be unwilling to insert a TA record for net.ta.com or it would charge an obscene sum for doing so. Given this analysis, a resolver vendor that wanted to encourage vigorous competition among TAs at multiple levels of the DNS hierarchy might not want to implement the hierarchical scheme. Instead, that resolver vendor might want to find some way to make it relatively easy to preconfigure multiple TAs.

4 Alternate Designs and Related Work

4.1 Out-of-band

It would be relatively easy to design a mechanism for storing and retrieving trust statements outside the DNS. An HTTP-based lookup engine, much like a PGP key server, could serve the purpose, as could a custom-built directory service.

One obvious objection to such a scheme is that finding the directory service could depend on the DNS. That objection is more one of availability than a security concern -- assuming the data were still signed with a well-known key, data authenticity wouldn't be an issue.

Another objection to going out-of-band is the number of port-filtering firewalls in the world. There are many firewalls that allow clients behind them to make arbitrary DNS queries, but do not allow connections to an arbitrary port, such as the directory service might use.

The most compelling reason to keep the security data in-band is to take advantage of the deployed DNS caching infrastructure while avoiding unpleasant interactions with other proxies and caches. Without a caching system, the servers providing this data must be prepared to handle one or more queries each time a client does a DNS lookup. The present DNS caching architecture is quite effective and should cause TA servers to see dramatically fewer queries than the aggregate number of DNS queries issued by the DNS resolvers using that TA. This will lower the provisioning needs of TAs and make operation of them much less expensive. Layering the directory service on top of HTTP, while it's likely to avoid port-filtering firewalls, subjects the data to "transparent" HTTP caches, which could deliver stale data to resolvers while giving the resolver no way to ask for fresh data.

4.2 SIG/TA at Child

An alternative to setting up a Trust Authority that maintains secure delegations for children is to store the credentials issued by the TA in the child zone. This would relieve the query load constraints that result from running large nameservers (or any large

directory service). In many ways, this is comparable to the SSL security model, in which public key certificates are stored and distributed to clients by the servers being authenticated, not by the certificate authority.

It would seem relatively simple to store the TA credential, signed by the Trust Authority, in the zone it refers to. A similar scheme was used by DNSSEC before the advent of Delegation Signer: RFC2535 describes how a parent can generate a SIG over each child's KEY RR. The child then stores that SIG in its own zone, a scheme known as SIG@child. Experimental operation with SIG@child led to the conclusion that it was unscalable. Unlike SSL, in which certificates typically have year or multi-year validity intervals, the DNSSEC designers assumed that parents would want to change their keys on a regular basis, perhaps monthly or even daily. At the least, it was assumed that the validity period for a SIG would be set relatively short, in part because no revocation mechanism is available. Given that assumption of short SIG validity or rapid KEY rollover, it follows that the parent must keep in frequent contact with all of its children. That imposes a number of timing interdependencies between parent and children which were seen to be unmanageable for zones with many delegations. For the same reasons that SIG@child was rejected, storing TAs in the zones they refer to is problematic.

Furthermore, DNSSEC's security model differs from that of SSL, in that SSL provides no mechanism to help a user determine whether a service should be using SSL, where DNSSEC insists that the security status of zones be definitively proved. With SIG@child, for each child zone that wasn't signed, the parent zone had to generate and store 'NULL KEYs' and SIGs in its own zone to prove that the child was insecure. Because we want entities other than the target zone owner to be able to run TAs, the mechanisms in this paper don't require that a TA know all of the delegations that are present in its target zone -- instead, we prove the insecurity of zones by the absence of a TA record, substantiated by the TA's NSEC chain. Whether we use the NSEC chain mechanism or the per-child enumeration mechanism from SIG@child, these DNSSEC

design constraints still require the operation of a large database service, avoidance of which is the main draw of storing certificates in the zones to which they refer.

4.3 Alternate Tree

One alternative to publishing trust data to supplement the data in the main delegation tree is to operate an entirely separate signed tree. Such a scheme would present multiple difficulties, both technical and political. On the political side, there is great opposition to any sort of "alternate" namespace, largely grounded in the opinion that the DNS should be consistent -- it should never be possible to get different answers to the same query. On the technical side, to sign a zone, the entire zone must be available. This generally requires the participation or at least the consent of each such zone operator, which imposes limits on the scheme. For example, an alternate tree starting at the root could not contain any secure delegations to the children of .net without access to the entire .net zone.

4.4 Overlay Tree

A variation on the above scheme is to have the alternate tree contain only a portion of the data in the "real" tree. Resolvers would first query the alternate (signed) tree. Only if they didn't find an answer there would they turn to the unsigned tree. A new signed zone could be inserted into the signed tree even without the cooperation of the zone's ancestors.

This scheme is very similar to what has been proposed in this paper, but it does allow for inconsistencies between the two trees. In particular, delegations in the alternate tree could (and likely would) point to different authoritative servers that the delegation at the same name in the "real" tree. Zones in the alternate tree could also have different data

than zones in the main tree. For example, there might be an A record for virus.microsoft.com in the alternate tree but no such entry in the "real" tree, or the www.cnn.com A records in each tree could be different.

4.5 FMESHD

USC's Information Sciences Institute and Network Associates Laboratories had a DARPA-funded research project that proposed to join "islands of trust" by providing some way for resolvers that trusted some portion of the hierarchy to gain knowledge about a portion of the hierarchy. For example, it aimed to have resolvers that only had a preconfigured trusted key for af.mil be able to trust data from .com. Called "FMESHD: Flexible Mesh of Trust Applied to DNSSEC", the project would, in its extreme form, have provided a way for any zone to make trust statements about any other zone. The process and policy for handling those statements in the resolver would necessarily be complicated, particularly when chains of transitive trust were being followed. This problem is reasonably tractable when the entire set of connections is known in advance, such as when finding paths in the PGP web of trust. It becomes more complicated and much slower when each of those paths must be independently discovered with a DNS lookup.

The FMESHD project was waylaid by more pressing concerns about DNSSEC's operational viability, and the above goals were never achieved.

In many ways, the designs presented here are more limited variants of FMESHD -- they eliminate the infinite chaining of trust and permit only certain special zones to make trust statements about other zones that aren't their children. This noticeably simplifies the resolution process.

5 Conclusion

Each of the Trust Authority designs presented in section 3 are either more scalable or more in keeping with the DNS and DNSSEC design philosophies than any of the credible competing designs. Because of it's greater flexibility, we recommend the hierarchical Trust Authority design from section 3.2. Resolvers using such TAs need to be very clear about what security policy they are using. Primarily because it allows the greatest chance for successful validation, a resolver policy of "accept any success" is suggested. To allow TAs to scale, though, that policy requires that resolvers implement aggressive negative caching, as described in section 3.3.2.

5.1 Future Work

While this paper has primarily talked about how to use a single TA (or a set of single-zone TAs) to supplement the trust data in the DNS's main delegation tree, it's conceivable that multiple entities will operate TAs. Future DNSSEC validators may want to make provision for multiple TAs. The algorithms described in 3.2.2 should be easily extensible to handle multiple TAs. Further, more complex algorithms will be made possible: a resolver may wish to accept a validation only when using data from two out of three TAs lead to validation. More likely, a validator may want to refrain from treating DNS answers and bad or bogus unless more than one TA indicates that the zone that data came from should be signed.

Section 3.2.2 discusses an algorithm for finding a name's "closest encloser" -- the TA record most closely matching that name. That algorithm assumes that an authoritative server will only return an answer when it has an exact match and that the only answers returned that have a different name will be referrals to another zone. Some additional performance improvement could come from allowing authoritative servers to send the

closest enclosing TA record in response to the first TA query (i.e. sending a example.ta.com. record in response to a query for grand.child.example.ta.com.). We haven't recommended that behavior for two reasons: first, beacuse the logic described in section 3.2.2 would still be necessary as a fallback. Second, because it's not known how resolvers, particularly legacy resolvers, will respond to seeing this form of response. This sort of perturbation to the DNS protocol has triggered several bugs in widely deployed resolvers, both new and old. Additional testing will be necessary before such behavior can be trusted not to cause unpleasant side effects. It should be noted, though, that an authoritative server would need to return different data if the resolver's policy is "access any success" than if the policy is "closest encloser trumps". In the latter case, only the closest enclosing TA record need be sent (along with a proof than no others exist). In the former case, ALL TA records need to be sent (along with all available proofs than no others exist). Again, the resolver would still need to have logic for fetching any missing TA records. Because of this difference, it is strongly recommended at all resolvers follow the same security policy.

6 References

[DNSSECbis-1] Arends, R., Austein, R., Larson, M., Massey, D. and S. Rose, "DNS Security Introduction and Requirements", work in progress, February 2004.

[DNSSECbis-2] Arends, R., Austein, R., Larson, M., Massey, D. and S. Rose, "Resource Records for DNS Security Extensions", work in progress, February 2004.

[DNSSECbis-3] Arends, R., Austein, R., Larson, M., Massey, D. and S. Rose, "Protocol Modifications for the DNS Security Extensions", work in progress, February 2004.

[IPSECKEY] Richardson, M, "A Method for Storing IPsec Keying Material in DNS" work in progress, February 2004.

[RFC2929] Eastlake, D., E. Brunner-Williams, and B. Manning, "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 2929, September 2000.

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[RFC3445] Massey, D. and S. Rose, "Limiting the Scope of the KEY Resource Record (RR)", RFC 3445, December 2002.

[RFC3658] Gudmundsson, O., "Delegation Signer (DS) Resource Record (RR)", RFC 3658, December 2003.

[RFC3755] Weiler, S., "Legacy Resolver Compatibility for Delegation Signer", RFC3755, April 2004.

[Rozendaal] Rozendaal, E. "Implementing DNSSEC in NSD", Presentation at RIPE47, Amsterdam, January 2004.