Trustworthy TCB for DNS Servers

Arun Velagapalli and Mahalingam Ramkumar
(Corresponding author: Arun Velagapalli)

Department of Computer Science and Engineering, Mississippi State University
Box 9637, Mississippi State University, MS 39762, USA
(Email: av82@msstate.edu)

(Received June 7, 2010; revised and accepted Oct. 4, 2010)

Abstract

A simple atomic relay function is proposed as a minimal trusted computing base (TCB) for a domain name system (DNS) server. This TCB, composed of a fixed sequence of logical and cryptographic hash operations, can be amplified to ensure that a DNS server cannot violate rules. The paper also outlines elements of a TCB-DNS protocol that amplifies the simple TCB to secure the domain name system. The paper includes an extensive comparison of the proposed approach with DNSSEC, the current standard for securing DNS. The proposed approach is shown to overcome many issues associated with DNSSEC. Specifically, TCB-DNS demands substantially lower overhead for DNS servers and resolvers, eliminates the issue of zone enumeration, and is less susceptible to replay attacks.

Keywords: Domain name system, DNSSEC, trusted computing base

1 Introduction

The domain name system (DNS) simultaneously refers to i) a hierarchical naming scheme for Internet-based services, and ii) a distributed database of DNS records which can be queried by specifying *name* and *type* of record. DNS records pertaining to a DNS *zone* are created by the *authority* for the zone, and are intended for clients who utilize the services of the zone.

For example, a client who wishes to connect to a web-service named `www.yahoo.com` requires the IP address of the host running the web-server. This information is contained in an A-type (address-type) DNS record created by the authority of the zone `yahoo.com`. To obtain the required A-type record, the client invokes a DNS query `(www.yahoo.com, A)`. Similarly, a client desiring to send an email to an address `alice@xyz.com` invokes a DNS query `(xyz.com, MX)` for an MX-type (mail-exchange) record to obtain the name of the mail-server for the domain `xyz.com`. Once the name of the mail-server is determined (say, `mail.abc.net`), the client can then query the DNS for `mail.abc.net,A` to determine the IP address of the mail-server.

The important services offered by DNS are performed by DNS servers and resolvers. More specifically, DNS records are created by zone authorities, and are ultimately intended for clients. DNS servers and resolvers are the "middle-men" involved in storing and relaying the records to clients.

Ultimately, any solution to secure the DNS should not require the middle-men to be trusted. Some of the desired security assurances in the context of operation of the entire domain name system are as follows:

1) **A1 - Authentication and integrity:** DNS records cannot be modified in transit between their creator (zone authorities) and their ultimate destination (clients).
2) **A2 - Authenticated denial:** If a DNS server is queried for a nonexistent record (the queried name and type does not exist), the querier should receive a *believable response* that the queried record does not exist.

Currently, DNSSEC [1] is the standard security protocol for DNS. DNSSEC does not require the middle-men (DNS servers) to be trusted; only the zone authority is trusted to provide information regarding the zone. Unfortunately, due to substantial overhead imposed on DNS servers, resolvers, and clients, DNSSEC has seen low levels of adoption [7, 8, 15]. Furthermore, in its attempt to provide assurance **A2** DNSSEC enabled servers are forced to reveal names of unsolicited records to the querier, and is consequently susceptible to the *zone enumeration* or *DNS-walk* problem [27]. To provide authenticated denial without being susceptible to DNS-walk, we desire yet another assurance:

1) **A3 - Only explicitly solicited records should be revealed:** For providing assurance **A2**, DNS servers should not be required to reveal names of DNS records that are *not* explicitly queried by name and type.

Thus far, attempts to provide assurance **A3** in DNSSEC have not been successful. For example, the *hashed authenticated denial of existence* scheme (also referred to as NSEC3 [14]) does not reveal unsolicited names - it reveals only cryptographic hashes of unsolicited names. Unfortunately, simple dictionary attacks can reveal the names behind the hashes.

Some of the alternate approaches in the literature for securing DNS include symmetric key DNSSEC [22] and DNSCurve [10]. Such approaches only protect the *links* over which DNS records traverse (on the way from the zone authority to their ultimate destinations). The advantage of approaches that only secure links is that they demand substantially lower overhead compared to DNSSEC, and are thus more likely to be adopted. Unfortunately, unlike DNSSEC, link-security approaches require the middle-men (DNS servers and resolvers) to be trusted.

Trusting DNS servers and resolvers implies trusting all hardware and software that constitutes the server or resolver. Due to the large number variables involved in the realization of hardware and software, this is far from practical. In this paper we propose a novel approach for securing the domain name system. In the proposed TCB-DNS approach we identify a minimal trusted computing base (TCB) for a DNS server, and leverage this TCB to realize the desired assurances.

1.1 Trusted Computing Base

For any computing system with a desired set of security requirements $\mathcal{R}$ the trusted computing base (TCB) is "a small amount of software and hardware we rely on" (to realize the requirements $\mathcal{R}$) and "that we distinguish from a much larger amount that can misbehave without affecting security" [13]. In other words, *as long as the TCB is worthy of trust* the TCB can be amplified to realize the desired assurances $\mathcal{R}$ regarding the operation of the entire system.

In this paper we argue that the TCB for a DNS server is a simple *atomic relay* function. In practice, the atomic relay function can be realized as a fixed sequence of logical and cryptographic hash operations performed inside low-complexity trustworthy modules (TM) that can be realized at low-cost. Such TMs housed in DNS servers can be leveraged by the TCB-DNS protocol to realize assurances **A1** - **A3**. In doing so, TCB-DNS

1) will not require DNS servers (and operators with unfettered access to the DNS servers) to be trusted; only the TMs are trusted to provide assurances **A1** and **A2**;
2) is immune to DNS-walk; TCB-DNS provides assurance **A2** without the need to reveal names of unsolicited records, or even hashes of unsolicited names to the queries (thereby providing assurance **A3**);
3) will place very little additional demands (over plain-old DNS) on DNS servers, resolvers and hosts; and
4) will not mandate any changes to the structure of DNS queries and responses.

1.2 Organization

The rest of this paper is organized as follows. Section 2 provides an overview of "plain-old" DNS, security issues in DNS, and DNSSEC.

Section 3 outlines the atomic relay function, and the TCB-DNS protocol for securing DNS. An essential component for the atomic relay function is a mechanism for computing pairwise link-secrets, which is discussed in Section 3.3.

Section 4 provides a more in-depth description of the TCB-DNS protocol. Section 4.1 provides a formal listing of the atomic relay algorithm. Sections 4.2 and 4.3 describe how the atomic relay function is leveraged to realize the desired assurances **A1** - **A3** regarding the operation of the DNS. Section 4.2 enumerates the steps to be taken by zone authorities to prepare TCB-DNS zone files. Section 4.3 enumerates a typical sequence of events in relaying and verification of DNS RRs that are accompanied by some TCB-DNS specific values.

Section 5 addresses some practical considerations in the deployment of TCB-DNS, including the need for an infrastructure for verifying the integrity of DNS TMs. Section 6 discusses some of the similarities and differences between DNSSEC and TCB-DNS. The protocols are compared in terms of overhead, and susceptibility to replay attacks. Some modifications to DNSSEC have been proposed recently to reduce the overhead, with the intent of rendering DNSSEC more suitable for adoption. Some of the unintended side-effects of such modifications are discussed in Sections 6.4 and 6.5.

Conclusions are offered in Section 7. The notations and abbreviations used in this paper are summarized in Table 1.

2 Domain Name System

The domain name system is a tree-hierarchical naming system for services that can be accessed over the Internet. At the top of the inverted DNS tree (see Figure 1) is the *root*. Below the root are generic top level domains (gTLD) like `com`, `org`, `net`, `edu`, etc., and country-code top level domains (ccTLD) like `ca` (Canada), `in` (India), etc. A leaf named `b.cs.univ.edu` in the DNS tree is a server-host in a branch `cs.univ.edu`, which stems from a thicker branch `univ.edu`, which stems from an even thicker branch `.edu`, stemming from the root of the DNS tree.

A branch of the tree (including its sub-branches and leaves) under the administrative control of an authority, is a DNS zone.

The authority for a zone is responsible for i) assigning names for branches and leaves under the zone; ii) creating DNS resource records corresponding to such names,

Table 1: Typical states of SEIR model

DNS	Domain name system/server
ANS	Authoritative name server
PNS	Preferred name server
$h()$	A cryptographic hash function (like SHA-1)
$X \parallel Y$	Concatenation of values X and Y
$A, B, \ldots$	Upper case letters represent identities of entities
K_{XY}	A pair-wise secret privy only to entities X and Y
MAC	Message authentication code
$M_{XY,V}$	$M_{XY,V} = h(V \parallel K_{XY})$ – A MAC for a value V

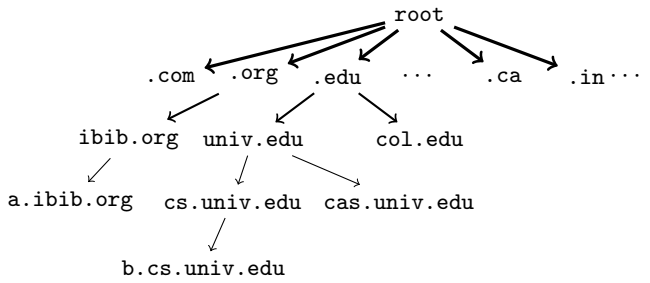

Figure 1: The domain name system (DNS) tree. A leaf of the tree is typically a DNS name of a server (for example, web-server, mail-server, DNS servers etc.). Some internal nodes correspond to starting points of DNS zones. Associated with each zone is i) an authority, and ii) a set of authoritative name servers (ANS) for the zone.

and/or iii) delegating an entity as the authority for a branch within the zone.

The authority for the root zone has delegated a gTLD zone like `.edu` to a `.edu`-gTLD authority, who has in turn delegated the zone `univ.edu` to another authority, who may have delegated a zone `cs.univ.edu` to yet another authority (say Z). All DNS records for the zone `cs.univ.edu` (or all DNS records with names ending with `cs.univ.edu`) are created by Z. The zone authority also specifies the names of *authoritative* name servers (ANS) for the zone. A "zone master file" which includes the set of all DNS RRs pertaining to the zone is then provided every ANS of the zone. ANSs of the root zone are also referred to as *root name-servers*.

A client desiring to access a HTTP service `www.cs.univ.edu` requires the IP address of the web-server with a domain name `www.cs.univ.edu`. This information is in an A-type record in the master file for a zone under which the name `www.cs.univ.edu` falls, and can be obtained by querying any ANS for the zone. To obtain this information, the querier only needs to know the IP address of a root name-server. While the root name server cannot directly provide the answer to the query "`www.cs.univ.edu, A`," it can provide the names and IP address of the ANSs for gTLD and ccTLD zones. In this case, the root server will respond with the names and IP addresses of all `.edu` ANSs.

The querier can now send the same query to any of the `.edu` ANS, which will respond with the name and the IP address of the ANSs for the zone `univ.edu`. When the same query "`www.cs.univ.edu, A`" is the directed to an ANS for the zone `univ.edu`, the response includes the names and IP addresses of ANSs for the zone `cs.univ.edu`.

Finally, any of the ANSs for the zone `cs.univ.edu` is queried to obtain the desired the A-type record. If the zone `cs.univ.edu` had not been sub-delegated, then the ANS for the zone `univ.edu` would have directly provided the response. Thus, knowing only the IP address of one root name server, any one can obtain any DNS record by specifying the name and type, and performing a series of queries.

2.1 DNS Records

Every DNS resource record (RR) is a five-tuple consisting of i) name, ii) class, iii) time-to-live (TTL), iv) type, and v) value; for example, `name=www.cs.univ.edu, IN, TTL=2345, type=A, value=159.43.7.82`. The class is always IN (for Internet RRs); the field TTL is specified in seconds, and indicates how long a RR can be cached. In the rest of this paper, to keep notations simple, we shall ignore the fields "class" and "TTL."

A-type RRs indicate an IP address in the value field. An NS-type record `name=cs.univ.edu, type=NS, value=ns1.dserv.net` indicates that a name-server with a domain name `ns1.dserv.net` is an ANS for the zone `cs.univ.edu`.

A set of records with the same name and type, but with different value fields, is collectively referred to as an RRSet. For example the NS-type RRSet for the name `cs.univ.edu` may include two NS records - one indicating the ANS `ns1.dserv.net`, and the other indicating another ANS named `ns1.cs.univ.edu`.

The NS type records are used for delegation. An RRSet of NS records for a delegated zone (say) `cs.univ.edu` can be found in the master file of the parent zone `univ.edu`. Similarly the NS RRSet for `univ.edu` can be found in the master file of the zone `.edu`, and so on. Along with NS records which specify ANSs, the A-type records for the

ANSs are also included in the master file as *glue* records[1].

The creator of RRs for a zone, viz., the zone authority, is always off-line. Once the master file for zone has been provided to the ANSs, and the names of ANSs conveyed to authority of the parent zone (and included as NS records in the master file of the parent), the zone authority simply expects the ANSs to faithfully accept and respond to DNS queries regarding the zone.

2.1.1 Query-Response Process

DNS queries and responses are typically payloads of UDP packets and have the same packet format. They include a header, and four sections: `QUESTION`, `ANSWER`, `AUTHORITY` and `ADDITIONAL`. In a query packet `QUESTION` section indicates the queried name and type (all other sections are empty). The response has an identical `QUESTION` section. The `ANSWER` section contains the desired RRSet. The `AUTHORITY` section includes NS records indicating the authoritative zone and the ANS for the zone. The `ADDITIONAL` section contains A-type glues for the NS records.

In practice, clients initiate queries using *stub-resolvers* running on their own host machine. Stub-resolvers do not directly query ANSs. Instead, they use preferred name-servers (PNS) as intermediaries. PNSs are also referred to as local DNS servers or local recursive resolvers or caching-only name-servers, and are typically operated by Internet service providers (ISP).

An application requiring the IP address of (say) `www.cs.univ.edu` queries a stub-resolver running on the same host. The stub-resolver redirects the query (`(www.cs.univ.edu,A)`) to a PNS. To do so, the host (or the stub-resolver) should know the IP address[2] of at least one PNS.

All PNS needs to be aware of the IP address of at least one root server. The PNS queries a root-server for `(www.cs.univ.edu,A)`, and receives NS records (with glued A-type records) for ANSs of `.edu`. The PNS then queries a `.edu` ANS to receive NS records of `univ.edu`, and so on. Finally, an ANS of the zone `cs.univ.edu` responds to the query with the desired A-type RRSet, which is relayed back to the stub-resolver.

PNSs may cache RRs for a duration specified by the TTL field in the RR, and may respond to queries from stub-resolvers using cached RRs. Similarly stub-resolvers may also cache RRs and respond to queries from applications running on the same host using the cached RRs.

[1]Note that zone `univ.edu` (or even `edu`) cannot be authoritative for the zone `dserv.net`. Thus, while `univ.edu` can provide an authoritative response regarding the *name* of the ANS for the child zone `cs.univ.edu`, it cannot provide an authoritative A-type record for the server `ns1.dserv.net`. To avoid possible circular dependency problems, the necessary *non authoritative* A-type records are included as glue records.

[2]Typically, IP addresses of PNSs are provided to a host by a DHCP server. In UNIX-like machines the IP addresses of PNS are stored in a file /etc/resolv.conf.

2.2 Securing DNS

The main goal of attacks on DNS is to simply divert traffic away from genuine services, or more often, to divert such traffic to impersonators phishing for personal information from unsuspecting clients. A common strategy for attackers is to impersonate ANSs to provide fake DNS responses to PNSs, thereby "poisoning the cache" of the PNS, and consequently the caches of many stub-resolvers which employ the poisoned PNS.

The header of a DNS query includes a 16-bit transaction ID t_{id}; the UDP packet carrying the query indicates a 16-bit source port p chosen by the querier. A DNS packet carrying the query will be accepted only if it is addressed to port p. The DNS response in the UDP packet will be accepted only if it indicates an expected transaction ID t_{id}.

To create a fake response that will be accepted by an PNS, an *out-of-path* (or external) attacker, who does not have plain-sight view of the query packet, will need to guess the values t_{id} and p. A typical strategy for an out-of-path attacker is to register a domain, run her own ANS for the domain, and query the targeted PNS for a name under her domain. When the query from the PNS is ultimately directed to the attacker's ANS, the attacker learns enough information to narrow down the two values t_{id} and p within small range.

Recently, Kaminsky [11] pointed out that DNS cache poisoning attacks can have even more severe consequences. Instead of attempting to poison RRs corresponding to a specific zone, the attacker can impersonate a root server and send fake glue records for "IP addresses of gTLD name servers." Thus, queries to every `.com` zone, for example, will then be directed to a computer under the control of the attacker, which could redirect such queries to other "ANSs" under her control.

2.3 Link-Security Approaches

While properly randomizing the two 16-bit values (t_{id} and p) is a good first step, they offer no defense against *in-path* attackers. In-path attackers who may be in the same LAN as the server or the resolver, or lie in-between the resolver and the server, have plain-sight access to the values t_{id} and p in the UDP DNS packets, and can thus easily fake responses. Securing links between DNS servers (for example, by using a secret shared between a resolver and the server queried by the resolver) can prevent such attacks.

Specifically, two entities A and B who share a secret K_{AB} can prevent even in-path attackers from impersonating them by i) encrypting the message sent over the link using the shared secret K_{AB}, or ii) appending a message authentication code (MAC) $h(V \parallel K_{AB})$ where $h()$ is cryptographic hash function, and V may be the message, or a cryptographic hash of a message $\mathcal{M}$ (or $V = h(\mathcal{M})$); as long as $h()$ is pre-image resistant, only an entity with access to the secret K_{AB} can compute a valid MAC for a message.

Strategies like SK-DNSSEC [22] and DNSCurve [10] adopt such an approach. In symmetric key DNSSEC [22] all PNSs have the ability to establish secure channel with the root servers. ANSs higher in the hierarchy act as trusted servers and facilitate establishment of secrets with ANSs lower in the hierarchy, using the Needham-Schroeder protocol [16] (which is the basis for the Kerberos [17] authentication protocol). When a PNS queries the root server for "`cs.coll.edu`, A", the root server's response includes a Kerberos-like ticket which permits the PNS to establish a secure channel with a `.edu` DNS server. The `.edu` DNS server then issues a ticket for securely communicating with an ANS for the zone `coll.edu`.

DNSCurve [10] employs a Diffie-Hellman scheme over a special elliptic curve $\mathcal{C}$ for setting up a private channel between DNSCurve enabled DNS servers. A DNSCurve enabled server A chooses a secret a. The secret between two DNSCurve enabled servers/resolvers A and B (where B's secret is b) is $K_{AB} = \mathcal{C}(b, \alpha) = \mathcal{C}(a, \beta)$, where $\alpha = \mathcal{C}(a, S)$, $\beta = \mathcal{C}(b, S)$, and S is a public parameter.

Link-security approaches assume that the DNS servers themselves are trustworthy. Note that while link-security approaches protect DNS RRs from out-of-path attackers (who do not have access to values t_{id} and p) *and* in-path attackers (those with access to t_{id} and p), there is nothing that prevents an entity controlling the DNS server from modifying an RR. In practice, such an attacker can be the operator of a DNS server, or some other entity who has somehow gained control of the DNS server. Such an attacker can receive RRs over protected links, illegally modify RRs, and relay fake RRs over "protected" links.

2.4 DNSSEC

Ideally, the "middle-men" should not be trusted: only the authority of a zone should be trusted for providing information regarding the zone. This is the approach taken by DNSSEC [1], where every RRSet in the zone master file is individually signed by the zone authority.

Every DNSSEC-enabled zone authority has an asymmetric key pair. The public portion of the key pair is certified by the authority of the parent zone. For example, the public key of the zone `cs.univ.edu` is signed by the authority of zone `univ.edu`. The public key of the zone `cs.univ.edu` can be obtained by querying for a DNSKEY-type RR for the name `cs.univ.edu`. To authenticate the public key in the DNSKEY RR, the parent zone `univ.edu` introduces two RRs in its zone file: a delegation signed (DS) RR which indicates a key-tag (a hash) for the public key of its child, and an RRSIG(DS) record which is the signature for the DS record.

For verifying the RRSIG(DS) record the public key of the parent zone `univ.edu` is required - which is the DNSKEY RR for the name `univ.edu`. To authenticate the public key of the parent, it is necessary to obtain the DS and RRSIG(DS) record from *its* parent zone - `.edu`, along with the DNSKEY RR for `.edu`. Finally, the public key of `.edu` can be verified by obtaining DS and RRSIG(DS) records from the root zone (by querying any root server). The public key of the root zone is assumed to be well publicized.

To summarize, corresponding to every RRSet for the zone `cs.univ.edu` is a RRSIG(RRSet) record which contains the digital signature for the RRSet. In response to a query for an RRSet, the corresponding RRSIG record is also included in the response. To verify the RRSIG, the required DNS RRs are

1) DNSKEY RR of `cs.univ.edu`
2) DS, RRSIG(DS) corresponding to DNSKEY RR of `cs.univ.edu`, and DNSKEY RR of the parent zone `univ.edu` (fetched from the parent zone `univ.edu`);
3) DS, RRSIG(DS) corresponding to DNSKEY RR of `univ.edu`, DNSKEY RR of `.edu`;
4) DS, RRSIG(DS) corresponding to DNSKEY RR of `.edu`, from the root zone.

2.5 Authenticated Denial

Consider a scenario where the zone authority for the domain `wesellstuff.com` outsources its DNS operations to `dnsnet.net`. It is indeed conceivable that a competitor `wealsosellstuff.com` could bribe some personnel in `dnsnet.net` (or any entity who has acquired control over the ANS) to remove the record for `wesellstuff.com` (thereby driving the competitor out of business).

To ensure that DNS servers and/or their operators need not be trusted, DNSSEC demands a pertinent response from an ANS for *every* query that falls under the zone. If the queried name exists, the ANS should provide a signed RRSet. If the queried name does *not* exist, the ANS is expected to provide *authenticated denial* by providing some information signed by the *zone authority*[3] which demonstrates that the queried record does not exist. If the ANS ignores the query, or provides a non pertinent response, the resolver will send the query again, or will query another ANS for the zone, till it receives a pertinent response.

For example, in response to a query for name `abc.xyz.fgh` the querier expects a signed RRSet by the authority for the zone under which the name `abc.xyz.fgh` falls, or alternately, expects

1) a signed response from the authority of the root zone that no record for a name `.fgh` exists; or
2) a signed response from the authority of the zone `.fgh` that no record for the name `xyz.fgh` exists; or
3) a signed response from the authority of the zone `xyz.fgh` that no record for the name `abc.xyz.fgh` exists.

[3]Only the zone authority is trusted to provide information regarding the zone - even information indicating that a record does *not* exist.

As the zone authority is off-line, a response denying every possible (as yet unknown) query, regarding the almost infinitely many possible names and types that can fall under the zone, should somehow be signed by the zone authority and included in the master file provided to ANSs. This is accomplished cleverly through NSEC records [27]. A signed NSEC record `abc.example.com, NSEC, cat.example.com` indicating two *enclosers* is interpreted as an authenticated denial of all enclosed names: viz., names that fall between `abc.example.com` and `cat.example.com` in the dictionary order. For example, if queried for a record named `cab.example.com`, this NSEC RR signed by the authority of the zone `example.com` (the signature included in a RRSIG(NSEC) RR) is proof that no such record exists.

2.5.1 DNS-Walk

Even while DNS RRs are not meant to be private they should only be provided when explicitly queried by name and type. NSEC permits one to query random names and learn about unsolicited names of enclosers that *do* exist in the zone master file. For example, a querier may send a query for a random name like `axx.example.com` and get to know the two enclosers `abc.example.com, cat.example.com` that actually exist. The attacker can then query for a random name like `cate.example.com` and obtain its enclosers, say `cat.example.com, data.example.com`, and so on.

The ability to easily enumerate all services under a zone is obviously a useful starting point for any attacker. An attacker wishing to obtain all DNS records for a zone can easily "walk-through" all records in the zone master by simply making a sequence of random queries. Such supercilious queries also have the ill-effect of further burdening the DNS infrastructure.

It is this unintended side effect of providing assurance **A2** that created the need for assurance **A3**. NSEC3 [14], a recent modification to NSEC, employs hashes of names as enclosers instead of using the names themselves as enclosers. For example, an NSEC3 RR indicating a hash encloser (v_l, v_h) indicates that no record with a name y exists, *if* $v_l < h(y) < v_h$. The enclosers v_l and v_h are hashes of real names corresponding to records that do exist in the master file.

Unfortunately, simple dictionary attacks can be used to reveal the names behind hashes like v_l and v_h. Thus, NSEC3 fails in its attempt to provide assurance **A3**.

3 Overview of TCB-DNS

DNSSEC has seen poor levels of adoption as upgrading a "plain-old" DNS server to support DNSSEC will often necessitate a hardware upgrade due to an order of magnitude increase in the size of DNSSEC records (compared to plain DNS records), and substantial increase in the size of DNS responses [7, 8, 15]. In many cases DNSSEC may require more expensive TCP instead of UDP as the transport protocol for carrying large DNS responses. DNS resolvers and clients will also need to endure substantial computational burden due to the need to verify multiple digital signatures. Furthermore, the feasibility of zone enumeration also encourages attackers to perform supercilious queries, thus exacerbating the issue of high DNSSEC overhead.

3.1 Extending Link-Security Approaches

Cryptographic mechanisms for individually securing each link traversed by DNS records, viz., the links i) between the off-line zone authority and ANSs of the zone (for securely conveying master files); ii) between PNSs and ANSs; and iii) between clients and their PNSs, demand substantially lower overhead compared to the hierarchical PKI-like approach employed by DNSSEC. Unfortunately, link-security approaches implicitly assume that the middle-men are trustworthy: while RRs are protected in transit, there is no protection for RRs while they reside in the DNS servers.

Specifically, symmetric key DNSSEC [22] and DNSCurve [10] simply *assume* that

1) the keys employed by DNS servers (which are used to compute the link secrets) are well protected from untrustworthy entities (else, any entity with access to the secrets of a DNS server can impersonate the DNS server to send fake RRs); and that

2) the intermediary DNS servers i) will not modify RRs, and ii) will not deny RRs that do exist.

The proposed TCB-DNS, where every DNS server houses a low complexity trustworthy module (TM), is also a link-security approach. However, TCB-DNS does not make such unjustifiable assumptions regarding the trustworthiness of DNS servers. Instead, TCB-DNS assumes that

1) secrets protected by the TM (which are used to compute link-secrets) cannot be exposed; and

2) the trivial functionality of the TM cannot be modified.

To warrant trust, an obvious requirement is that the TM functionality is as simple as possible. To motivation for TCB-DNS stems from the intuition that TMs that perform trivial functions are sufficient to provide the desired assurances **A1**, **A2** and **A3**.

3.2 Principle of Operation

In TCB-DNS every DNS server is equipped with a low-complexity TM. From the perspective of DNS servers, the TMs are black boxes that accept a formatted stream of bits as input, and output a message authentication code (MAC). Such MACs accompany plain DNS responses sent by DNS servers. The operations performed inside the TM

(to map the input bits to a MAC) are a fixed sequence of logical and cryptographic hash operations. This simple TM functionality is the TCB of a DNS server, which is leveraged to realize all three assurances **A1** - **A3** with negligible overhead.

3.2.1 Atomic Relay

In the path of a RRSet originating from zone authority Z to the client (a stub-resolver C), are an ANS for the zone Z and the PNS used by the host C. An *atomic relay*, as the name suggests, relays a value from one entity to another, in one atomic step. A TM A in the ANS performs an atomic relay of a value V from the zone authority Z to a PNS TM P, thus eliminating the need to trust the ANS in which the TM A is housed. Similarly the TM P in the PNS performs an atomic relay of the value V from the ANS TM A to a stub-resolver C, eliminating the need to trust the PNS.

From the perspective of the TM A, it receives some input bits which specify the identity of the source Z, a value V to be relayed, a message authentication code (MAC) $M_{V,ZA}$, and the identity of the entity P to which the value V needs to be relayed. The TM A uses its secrets to compute pair-wise secrets K_{ZA} and K_{AP} (the precise mechanism for doing this is explained later in this paper). Using the pair-wise secret K_{ZA}, TM A verifies the MAC $M_{V,ZA} = h(V, K_{ZA})$ appended by Z. Following this, the TM A computes a MAC $M_{V,AP} = h(V, K_{AP})$ using the secret K_{AP}.

The values relayed by TMs are hashes of RRSets. The hashes of RRSets are computed by zone authorities and individually authenticated to each ANS TM using MACs. ANS TMs can atomically relay the hashes to any PNS TM which can then atomically relay the hash to any stub-resolver. The TMs thus provide a parallel channel for securely conveying hashes of RRSets by leveraging link-secrets (which are computed using secrets protected by the TM).

Note that in both DNSSEC and TCB-DNS end-to-end integrity of an RRSet is realized by securely conveying a pre-image resistant hash of the RRSet. In DNSSEC this is achieved by signing the hash. In TCB-DNS the integrity of the hash is assured to the extent we can trust the TMs involved in relaying the hashes.

DNSSEC provides assurance **A1** by signing hashes of regular DNS RRs, and provides assurance **A2** by signing hashes of NSEC/NSEC3 RRs. Obviously, by atomically relaying the hashes of regular RRSets and NSEC/NSEC3 records, TCB-DNS can also provide both assurances provided by DNSSEC. However, a simple addition to the capability of the TMs can provide TCB-DNS with yet another useful feature - the ability to provide assurance **A3**, and thereby eliminate the possibility of DNS-walk.

3.2.2 "Intelligent" Atomic Relay

If we merely relay hashes of NSEC/NSEC3 records, then TCB-DNS will also be susceptible to DNS-walk. Fortunately, to realize assurance **A3**, the only additional intelligent feature required of TMs is the ability to recognize that "a value V falls inside an enclosure (V_l, V_h)."

The atomic relay function performed by a TM with identity X, takes the form

$$M_{XD} = \mathcal{AR}_X(S, D, V, V_l, V_h, M_{SX}).$$

In executing this TCB function the TM X accepts some fixed-length inputs like i) S and D: the identities of a source and destination; ii) cryptographic hashes V, V_l, and V_h; and iii) a MAC M_{SX} provided by the source S. The TM outputs a MAC for the value V under two conditions:

1) the MAC M_{SX} is consistent with V; or
2) the MAC M_{SX} is consistent with values $V_l \parallel V_h$, *and* V is enclosed by (V_l, V_h).

In the latter case, the TM interprets a pair of values (V_l, V_h) authenticated by the zone authority as proof that no value enclosed by (V_l, V_h) exists in the master file. If V is enclosed, the TM outputs a MAC for V to inform D that an "enclosure for V was found."

In DNSSEC that a value V is enclosed by (V_l, V_h) is checked by the querier. The need to reveal the enclosures to the querier is the reason that assurance **A3** cannot be provided. In TCB-DNS the enclosure is checked by the ANS TM (**not** provided to the querier). To the extent that the TM can be trusted, the querier trusts that an enclosure exists for the value V (and consequently, is convinced that an RR with the name corresponding to V does not exist).

More specifically, in TCB-DNS,

1) if the queried name and type exists the response includes the desired RRSet in the `ANSWER` section; a MAC for a value V (where V is hash of the RRSet) is also included in the response.
2) To deny a name and type $n_i \parallel t_i$ the `ANSWER` section indicates the name and type $n_i \parallel t_i$; the MAC for the value $V = h(n_i \parallel t_i)$ is included in the response to imply that the indicated name and type does not exist.

Typically, to provide authenticated denial for a queried name-and-type, a plurality name-and-types will have to be explicitly denied (as will be explained later in Section 6.1).

3.3 Computing Link Secrets

For performing the atomic relay, a TM needs to compute two pairwise secrets - one shared with the sender (the previous hop), which is used to verify the hash V (or the encloser (V_l, V_h) for authenticated denial), and one shared with the destination (the next hop). Specifically,

1) ANS TMs require the ability to establish a pairwise secret with i) the zone authority for receiving hashes of RRSets, and ii) all PNS TMs for securely conveying hashes of RRSets.

2) PNS TMs require the ability to establish a pairwise secret with i) all ANS TMs for receiving hashes, and with ii) all clients who employ the PNS for conveying the hashes.

For reducing TM complexity it is essential to identify a low complexity mechanism for computing pairwise secrets. While many efficient strategies exist, the modified Leighton-Micali scheme (MLS) proposed in [19] is particularly well suited for establishing pairwise secrets between TMs in DNS servers (between a large number of ANS TMs and a large number of PNS TMs). In this paper we also extend MLS to facilitate link-secrets between i) zone authorities and ANS TMs and ii) between PNS TMs and clients.

3.3.1 MLS

Let K_X represent a secret privy only to an entity with identity X and a key distribution center (KDC). Similarly, let K_Y represent a secret known only to an entity Y and the KDC. In MLS, the secret shared between two entities X and Y is $K_{XY} = h(K_X, Y)$ *or* $K_{YX} = h(K_Y, X)$.

If the pairwise secret is $K_{XY} = h(K_X, Y)$, then entity X computes the pairwise secret by directly hashing its secret K_X; entity Y employs a public (non-secret) value $P_{YX} = h(K_X, Y) \oplus h(K_Y, X)$ to compute $K_{XY} = h(K_{XY})$ as

$$\begin{aligned} K_{XY} &= h(K_Y, X) \oplus P_{YX} \\ &= h(K_Y, X) \oplus h(K_X, Y) \oplus h(K_Y, X) \\ &= h(K_X, Y). \end{aligned}$$

On the other hand, if the pairwise secret is $K_{YX} = h(K_Y, X)$, entity Y computes the pairwise secret directly, and entity X employs the public value $P_{YX} = h(K_X, Y) \oplus h(K_Y, X)$.

MLS is an identity-based scheme, where the identity assigned to an entity can be chosen to reflect the credentials of the entity. For example, the identity of a zone authority can simply be the hash of the name of the zone. In MLS some bits of the identities (say of two entities X and Y) are also used to determine which of the two entities should employ the pair-wise public value.

3.3.2 Key Distribution for TCB-DNS

In TCB-DNS the KDC can be entity under the control of a regulatory authority (for example, ICANN). The *core* TCB-DNS entities are TMs associated with DNS servers. The *fringe* TCB-DNS entities include zone authorities (who need to securely convey RRs to ANSs) and clients (stub-resolvers) who query DNSs. Pair-wise secrets for TCB-DNS can be i) between two core entities (between two TMs), or ii) between a core entity and a fringe entity.

For the former case, a sequence number included in the TM identity specifies which of the two core entities should use the public value to compute the pairwise secret. For the latter (pairwise secret between a core entity and a fringe entity) the fringe entity employs the public value - the core entity does not.

The identity X of a TM (associated with an ANS or a PNS) is of the form $X = X_t \parallel q_x$ where X_t is a succinct code describing the nature of X and duration of validity; the value q_x is a unique number assigned sequentially to every DNS server TM. To establish a secret between TMs $X = X_t \parallel q_x$ and $Y = Y_t \parallel q_y$ where (say) $q_x < q_y$, Y is required to use the value P_{XY} to compute the pairwise secret K_{XY}; X can compute K_{XY} directly using its secret K_X.

The TCB-DNS identity Z of a zone authority is of the form $Z = Z_t \parallel Z_{name}$ where Z_{name} is a one-way function of the domain name of the zone. To enable Z to compute a pairwise secret K_{ZA} with an ANS TM A the zone authority is issued

1) a secret $K_{AZ} = h(K_A, Z)$ by the KDC, or

2) a secret K_Z, along with a public value P_{ZA}; or

3) a TM with identity Z (with secret K_Z stored inside the TM), along with a public value P_{ZA}.

In the TCB-DNS identity of a client $C = C_t \parallel C'$, C' can be a unique random value. If P is the identity of a TM in a PNS used by the client C the client C is issued i) a secret $K_{PC} = h(K_P, C)$ or ii) a secret K_C and a public value P_{CP}.

Thus, once keys have been distributed to TCB-DNS entities, computing any link-secret will require the TM to only perform a single hash computation (or a hash computation and an XOR operation). Periodically, the KDC disseminates signed revocation lists indicating identities of entities revoked.

Note that unlike the "basic" key distribution scheme for a static network of size N (where each node is issued $N-1$ secrets) MLS can cater for a dynamic network - as new core entities (DNS server TMs) can be added at any time. In the basic scheme, to add a new node every old node should be provided a new secret, which is impractical. In MLS the new node is provided with one public value corresponding to every "old" node - old nodes do not need a public value to establish a secret with newly added nodes.

3.3.3 Multiple KDCs

At the top of the hierarchy of DNSSEC is a single root CA - which is the authority for the root zone. Though the root zone is expected to sign only public keys for gTLD and ccTLD zones, the all powerful root zone authority has the ability to misrepresent public keys for *any* zone. While ideally we would desire that this power be distributed amongst multiple independent entities, such an approach can further increase the overhead for DNSSEC.

However, MLS can be easily extended to support multiple KDCs with minimal overhead. If we use m (for example, $m = 4$) independent KDCs, an entity X receives m secrets (one from each KDC), $K_{X_i}, 1 \leq i \leq m$. Two entities X and Y can compute m independent pairwise secrets of the form $K^i_{XY}, 1 \leq i \leq m$ (one in each of the m parallel systems). All these secrets are simply XORed together to compute K_{XY} as $K_{XY} = K^1_{XY} \oplus K^2_{XY} \cdots \oplus K^m_{XY}$.

The secret K^i_{XY} can be computed only by X, Y, and the i^{th}KDC. The secret $K_{XY} = K^1_{XY} \oplus K^2_{XY} \cdots \oplus K^m_{XY}$ can be computed only by TMs X and Y (and together, by all m KDCs). While there are m public values associated with each secret, the m values can be XORed together and stored as one value; for example, $P_{XY} = P^1_{XY} \oplus \cdots \oplus P^m_{XY}$, where $P^i_{XY} = h(K_{X_i} \parallel Y) \oplus h(K_{Y_i} \parallel X)$.

Thus, a TM with identity X stores m secrets $K_{X_i}, 1 \leq i \leq m$ inside its protected boundary. To enable the TM to compute K_{XY}, the entity (DNS server) using the TM X provides two inputs: (Y, P_{XY}). The TM performs m hash operations and m XOR operations to compute

$$\begin{aligned} K_{XY} &= h(K_{X_1}, Y) \oplus \cdots \oplus h(K_{X_m}, Y) \oplus P_{XY} \\ &= K^1_{XY} \oplus \cdots \oplus K^m_{XY}. \end{aligned}$$

In the rest of this paper we shall use the notation $K_{XY} = F(Y, P_{XY})$ to represent the process of computing the pairwise secret K_{XY} by entity (or TM) X.

3.3.4 Renewal

For renewal of secrets of a TM $X = X_t \parallel q_x$ the regulatory authority simply issues a new TM with TCB-DNS identity $X' = X'_t \parallel q'_x$, with secrets $K_{X'_1} \cdots K_{X'_m}$. If at the time of renewal, the last issued sequence number was q, the new TM is issued a sequence number $q'_x = q + 1$. The owner of the TM is issued q public values (where each public value is the XOR of m public values). If the secrets of an ANS TM A is renewed, only the zone authorities using the ANS need to be issued new public values for A. If the TM P of a PNS is renewed, only the clients who use the PNS are issued with new public values corresponding to P.

More specifically, a node with sequence number q is the q^{th}node to join the network, and is issued one secret and $q-1$ public values (or m secrets and $q-1$ public values if we use multiple KDCs). For renewal we simply add a new node. The public values are the same size as the pair-wise keys (say 160-bits). A DNS server with a TM sequence number 10 million will need access to at most 200 MB of storage for public values (which can easily be stored in the hard-disk of the DNS server). There is no practical limit on the number of fringe entities (zone authorities and clients). Each fringe entity requires access only to a small number of public values (as they need to establish a pairwise secret only with a small number of core entities - zone authorities with TMs of all ANSs for the zone, and clients with all its PNSs).

4 The TCB-DNS Protocol

In this section we begin with a detailed specification of the atomic relay algorithm. We then we outline the operation of TCB-DNS by outlining the steps for creating TCB-DNS master files (in Section 4.2) and illustrating the sequence of events in typical a query-response process (in Section 4.3).

4.1 The Atomic Relay Algorithm

A TM with identity X stores a secret K_X inside its protected boundary - which is known only to TM X and the KDC. To relay a value from S to D the TM requires to compute secrets K_{XS} and K_{XD}. For this purpose the TM needs two additional inputs - public values P_{XS} and P_{XD}. Thus, the atomic relay function of a TM X takes the form

$$M_{XD} = \mathcal{AR}_X((S, P_{XS}), (D, P_{XD}), V, V_l, V_h, M_{SX}).$$

In a scenario where X does *not* require to use a public value to compute K_{XS}, the input $P_{XS} = 0$ is provided to the TM (as XOR-ing by 0 leaves a value unchanged). It is the responsibility of the (untrusted) DNS server to store and provide appropriate public-values to its TM; if a DNS server provides incorrect public values to its TM the MAC will be rejected by the next-hop[4] which verifies the MAC.

The TM X accepts a formatted stream of bits $\mathbf{b}_i = (S \parallel P_{XS}) \parallel (D \parallel P_{XD}) \parallel V \parallel V_l \parallel V_h \parallel M_{SX}$ as input from the DNS server which houses the TM; the TM performs a simple sequence of logical and cryptographic hash operations, and outputs a MAC M_{XD} or a fixed constant `ERROR.`. An algorithmic description of the sequence of operations is depicted in Figure 2.

$\mathcal{AR}_X(S, P_{XS}, D, P_{XD}, V, V_l, V_h, M_{SX})$ {
$K_{XD} = F(P_{XD}, D)$;
`IF` $(S == X)$;
  `RETURN` $h(V \parallel K_{XD})$;
$K_{XS} = F(P_{XS}, S)$;
`IF` $(V_l == 0)$
  $V_i = V$;
`ELSE IF` $(((V_l < V) \wedge (V < V_h)) \vee ((V > V_l) \wedge (V_l > V_h)))$
  $V_i = h(V_l \parallel V_h)$;
`ELSE RETURN ERROR`;
`IF` $(M_{SX} != h(V_i \parallel K_{SX}))$;
  `RETURN ERROR`;
`RETURN` $M_{XD} = h(h(S \parallel V) \parallel K_{XD})$; }

Figure 2: The Atomic Relay Algorithm $\mathcal{AR}_X()$. K_{XS} and K_{XD} are pair-wise secrets that X shares with TCB-DNS entities S and D respectively.

[4] If the next-hop is a PNS, when an invalid response is received, the PNS will send the query again or query another ANS. Similarly if the next-hop is a stub-resolver C, then C will resend the query or query another PNS.

As shown in the algorithm in Figure 2, the TM computes the pairwise secret K_{XD} for authenticating TM output to destination D. If $S = X$ (source is indicated as the TM itself), the TM construes this as a request to output a MAC $h(V \parallel K_{XD})$ verifiable by D. This feature, as we shall see soon, permits zone authorities to use DNS TMs for protecting zone secrets. In general (for $S \neq X$) the TM proceeds to compute the pairwise secret K_{XS} required for validating the inputs (V, V_l and V_h authenticated by source S using a MAC M_{SX}:

1) if V_l is zero the TM verifies that the MAC M_{SX} is consistent with V and K_{XS};

2) if the value V_l is non-zero, the TM verifies that i) the input MAC M_{SX} is consistent with the two values $(V_l \parallel V_h)$, *and* ii) that V is *enclosed* by (V_l, V_h). A value V is enclosed by (V_l, V_h) if $V_l < V < V_h$. If $V_l > V_h$ then V is enclosed by the "wrapped around" pair if $V > V_l > V_h$ or $V < V_h < V_l$.

On successful verification the TM outputs a MAC for the value $(S \parallel V)$ computed using the pairwise secret K_{XD} between X and D.

For ease of following the discussion in the rest of this section, note that

$$\begin{aligned} M_{ZA,V} &= \mathcal{AR}_Z(Z, 0, A, P_{ZA}, V, 0, 0, 0) \\ &= h(V \parallel K_{ZA}), \end{aligned}$$

is a MAC for a value V computed by a TM Z (for verification by a TM A). We shall see soon that zone authorities can employ TMs in this fashion to authenticate hashes of RRSets for verification by ANS TMs. Also note that

$$\begin{aligned} M_{AP,V_Z} &= \mathcal{AR}_A(Z, 0, P, P_{AP}, V, 0, 0, M_{ZA,V}) \\ &= h(h(Z \parallel V) \parallel K_{AP}), \end{aligned}$$

is a MAC computed by TM A which can be verified by an entity P. The MAC represents A's claim that "a value V was received from Z." If the MAC is verifiable, to the extent P trusts A, P can accept the claim that the value V was provided by Z. Finally,

$$\begin{aligned} M_{AP,V'_Z} &= \mathcal{AR}_A(Z, 0, P, P_{AP}, V', V_l, V_h, M_{ZA,V'}) \\ &= h(h(Z \parallel V) \parallel K_{AP}), \end{aligned}$$

is also a MAC verifiable by an entity P; on successful verification, P concludes that "a value V was received from Z." P does not need to differentiate between the two cases. In the former case, V was explicitly conveyed to A by Z through a MAC $M_{V,A}$. In the latter case, V is any value, not explicitly conveyed by Z, but happens to fall within an enclosure (V_l, V_h) (and the enclosure is authenticated by Z using MAC $M_{V',A}$).

4.2 Preparation of TCB-DNS Master File

Consider a zone `example.com`, which employs ANSs with TMs A and B for the zone. The sequence of steps performed by the zone authority to prepare a master file are as follows. Let the TCB-DNS identity of the zone be Z where $Z = Z_t \parallel Z_{name}$, where Z_{name} is the hash of the name of the zone (`example.com`). Recall that Z_t includes a succinct representation of the time of expiry of the secrets assigned to Z.

Step 1. Prepare a regular plain DNS master file. Some of the required additions to plain DNS RRs are as follows:

1) Each RR will indicate an absolute value of time as the time of expiry. This value can be a 32-bit value like UNIX time, and can be different for each RR. In general the time of expiry of any RR should not be later than Z_t.

2) NS-type RRs (which indicate the name of an ANS) includes two additional values
   a. the TCB-DNS identity of the ANS-TM, and
   b. the value Z_t (note that from the name of the zone in the NS RR, one can compute the value Z_{name}; along with the value Z_t the TCB-DNS identity of the zone can be computed as $Z = Z_t \parallel Z_{name}$).

In general, a RRSet **R** has multiple RRs with the same name and type, and each RR indicates its own a time of expiry.

Step 2. Let r be the total number of RRSets. For an RRset **R** with name n_j and type t_j compute i) the hash of the RRSet $v_j = h(RRSet)$; and ii) $u'_j = h(n_j \parallel t_j \parallel \tau)$, where τ is the time at which the authentication for all enclosures expire. Repeat for all r RRSets.

Step 3. Sort the hashes $u'_1 \cdots u'_r$ in an ascending order; Let the sorted hashes be $u_1 \cdots u_r$. Now, compute values $d_1 \cdots d_r$ as

$$d_j = \begin{cases} h(u_j \parallel u_{j+1}) & j < r \\ h(u_r \parallel u_1) & j = r \end{cases}$$

Note that for the last "wrapped around" enclosure (u_r, u_1) the first value u_r is greater than the second (u_1).

Step 4. For each of the $2r + 1$ values in $\{v_1 \cdots v_r, d_1 \cdots d_r, \tau\}$ compute MACs $M_{ZA,i} = h(v_i \parallel K_{ZA}), 1 \leq i \leq r$, $M'_{ZA,j} = h(d_j \parallel K_{ZA}), 1 \leq j \leq r$, and $M_{ZA,\tau} = h(\tau \parallel K_{ZA})$. If the zone authority Z employs a TM Z then a MAC like $M_{ZA,i} = h(v_i \parallel K_{ZA})$ is computed by using the atomic relay function of the TM as

$$\begin{aligned} M_{ZA,V} &= \mathcal{AR}_Z(Z, 0, A, P_{ZA}, v_i, 0, 0, 0) \\ &= h(v_i \parallel K_{ZA}). \end{aligned}$$

Prepare a supplementary master file with

1) the values $(\tau, M_{ZA,\tau})$;

2) r rows of the form $(i, M_{ZA,i})$, $1 \leq i \leq r$, and
3) r rows of the form $((u_j, u_{j+1}), M'_{ZA,j})$, $1 \leq j \leq r$.

Step 5. Provide the supplementary master file to ANS with TM A along with the regular master file. The zone authority repeats Step 4 for ANS B to create a supplementary master file with values $(\tau, M_{ZB,\tau})$; r rows $(i, M_{ZB,i})$, and r rows $((u_j, u_{j+1}), M'_{ZB,j})$.

4.3 Verification of RRSets

We shall consider a scenario where an ANS with TM A is queried for an RRSet `cad.example.com,A` by a PNS with TM P. Let us further assume that the query was initiated by a stub-resolver C.

4.3.1 Events at ANS with TM A

Let the identities A and P of the TMs be $A = A_t \parallel q_a$ and $P = P_t \parallel q_p$. If $q_p < q_a$ (sequence number of P is less than that of A) the ANS fetches P_{AP} from storage (else $P_{AP} = 0$). If the queried name and type (n_i, t_i) exists, or if a suitable delegation exists, the ANS

1) extracts the RRSet from the plain DNS master file, and computes the hash of the RRSet, v_i;
2) extracts the MAC $M_{ZA,i}$ for v_i from the supplementary master file;
3) requests TM A to compute

$$\begin{aligned} M_{AP,i_1} &= \mathcal{AR}_A((Z,0),(P,P_{AP}),v_i,0,0,M_{ZA,i}) \\ &= h(h(Z \parallel v_i), K_{AP}). \end{aligned}$$

In the response sent to the PNS, the ANS includes the RRSet in the `ANSWER` section along with the value M_{AP,i_1}. If the response is a delegation, the NS RRSet can be included in the `AUTHORITY` section along with the value M_{AP,i_1}. The TM A does not know, or care, if the response is an answer or a delegation. To deny a name-and-type (n_i, t_i),

1) ANS extracts the values $(\tau, M_{\tau,ZA})$ from the supplementary master file for zone Z.
2) ANS computes $v_i = h(n_i \parallel t_i \parallel \tau)$;
3) ANS finds encloser for v_i (say (u_j, u_{j+1})), and corresponding MAC $M_{ZA,j}$ from the supplementary master file;
4) ANS requests TM A to compute M_{AP,τ_1} and M_{AP,i_1} as

$$\begin{aligned} M_{AP,\tau_1} &= \mathcal{AR}_A((Z,0),(P,P_{AP}),\tau,0,0,M_{ZA,\tau}) \\ &= h(h(Z \parallel \tau), K_{AP}) \\ M_{AP,i_1} &= \mathcal{AR}_A((Z,0),(P,P_{AP}),v_i,u_j,u_{j+1},M_{ZA,j}) \\ &= h(h(Z \parallel v_i), K_{AP}). \end{aligned}$$

For reasons that will be explained later in Section 6.1, typically the ANS will need to deny multiple name-and-type values in a response. Let us assume that q name-and-type values need to be denied. For each such name-and-type (n_l, t_l) the ANS computes $v_l = h(n_l \parallel t_l \parallel \tau)$, finds an encloser for v_l and the corresponding MAC in the supplementary master file, and requests the TM to compute MACs of the form M_{AP,l_1} (each of the q requests are made independently - each request results in the use of the atomic relay function $\mathcal{AR}_A()$ by the TM A).

In the response sent to the PNS the ANS includes (in the `ANSWER` section)

1) values τ and M_{AP,τ_1},
2) q denied name-and-type values $n_i \parallel t_i$,
3) q MACs of the form M_{AP,i_1}.

4.3.2 Events at the PNS with TM P

Before the PNS had sent a query to the ANS for a name and type belonging to zone Z, the PNS would have queried an ANS for the parent zone of Z - say $W = W_t \parallel W_{name}$, and obtained an NS-type RRSet for the name Z_{name} (where $Z = Z_t \parallel Z_{name}$).

Let us further assume that the NS-type RRSet was authenticated by a TM $G = G_t \parallel q_g$ (housed in an ANS for the zone W). In other words, the PNS would have received a value M_{GP,j_1} to authenticate the NS-type RRSet, where

$$M_{GP,j_1} = h(h(W \parallel v_j), K_{GP}),$$

and, v_j is the hash of the NS-type RRSet.

In TCB-DNS, the PNS is expected to verify the NS RRSet before sending a query to the delegated server. In this case, where the PNS had chosen to approach the ANS A for the zone Z (based on the information included in the NS-type RRSet authenticated by G) the PNS computes $v_{j_1} = h(W \parallel v_j)$, and requests its TM P to compute

$$x = \mathcal{AR}_P((G,P_{GP}),(A,P_{PA}),v_{j_1},0,0,M_{GP,j_1}).$$

As long as $x \neq ERROR$, the PNS considers the NS records to be valid.

Similarly, prior to querying G (ANS for W, the parent of Z) the TM would have received a response from an ANS for the parent of W (unless W is the root zone which has no parent - in our case W is the gTLD zone `.com`). Such a response from an ANS of W's parent zone would have also been verified as above before a query was sent to G. Thus, after the response from the parent zone W was verified, the PNS P had sent a request to A for a name and type under zone Z.

TCB-DNS does not require queries to be authenticated. Queries merely indicate the TCB-DNS identity of the querier.

Now, after the response is received from A, the PNS P has all the necessary information to send the answer to the stub-resolver C which initiated the query.

Typically, the PNS will need to include an RRSet in the ANSWER section (along with a MAC computed by its TM P). For responses containing authenticated denial for q name-and-types the response will include $q+1$ values authenticated individually using $q+1$ MACs. For both cases, an NS-type RRSet will be included in the AUTHORITY section indicating ANS for the zone, along with a MAC computed by the TM P.

More specifically, the hash of the RRSet in the ANSWER section is relayed atomically from A to C, by P. Similarly, for authenticated denial, the q hashes corresponding to multiple non-existing names, and the value τ, are relayed atomically from A to C by P. The hash of the NS-type RRSet is however relayed atomically by P from G to C.

For example, to relay the RRSet with hash v_i received from A the PNS first hashes the RRSet to obtain v_i and requests its TM to compute

$$\begin{aligned} M_{PC,i_2} &= \mathcal{AR}_P((A,P_{PA}),(C,0),v_{i_1},0,0,M_{AP,i_1}) \\ &= h(h(A \parallel v_{i_1}) \parallel K_{PC}) \\ &= h(h(A \parallel h(Z \parallel v_i)) \parallel K_{PC}). \end{aligned}$$

If the hash of RRSet v_i computed by the PNS is not the same as the one authenticated by the zone authority Z, the MAC M_{AP,i_1} will be found inconsistent by the TM P, which will return $ERROR$.

Similarly, to relay the NS-type RRSet received from G along with a value M_{GP,j_1}, the PNS hashes the RRSet to obtain v_j, and uses its TM P to compute[5]

$$\begin{aligned} M_{PC,j_2} &= \mathcal{AR}_P((G,P_{PG}),(C,0),v_{j_1},0,0,M_{GP,j_1}) \\ &= h(h(G \parallel v_{j_1}) \parallel K_{PC}) \\ &= h(h(G \parallel h(W \parallel v_j)) \parallel K_{PC}). \end{aligned}$$

The response from the PNS to C thus includes

1) NS-type RRSet (with hash v_j) for Z along with the values W, G and M_{PC,j_2} in the AUTHORITY section;

2) The queried RRSet with hash v_i, along with a MAC M_{PC,i_2}, and the identity A of the ANS;

3) Authenticated denial of q name-and-type values (by including $q+1$ values and $q+1$ MACs), and the identity A of the ANS.

If the parent zone W does not support TCB-DNS (the ANS for W is not equipped with a TM) then the NS RRSet is relayed without any TCB-DNS authentication.

4.3.3 At the Stub-Resolver C

The stub resolver performs the following steps:

Step 1. Extracts name of zone from the AUTHORITY section; hashes name to compute Z_{name} and hence $Z = Z_t \parallel Z_{name}$;

Step 2. If the NS RRSet in the AUTHORITY section has TCB-DNS authentication

1) Client C computes the hash v_j of the NS-type RRSet in the AUTHORITY section and verifies that $M_{PC,j_2} = h(h(G \parallel h(W \parallel v_j)) \parallel K_{PC})$.
2) Parses W as $W = W_t \parallel W_{name}$ and verifies that W_{name} is a legitimate parent of Z_{name}.

Step 3. C verifies that Z_{name} is a legitimate parent of the queried name.

Step 4. C verifies that name of the zone is a parent of the queried name[6] in the ANSWER section.

Step 5. If the ANSWER is the desired response, hash the RRSet to compute v_i; compute $v_{i_1} = h(Z \parallel v_i)$, $v_{i_2} = h(A \parallel v_{i_1})$, and using key K_{CP} verify that $M_{PC,i_2} = h(v_{i_2} \parallel K_{PC})$.

Step 6. If the ANSWER is an authenticated denial indicating q values of the form $n_i \parallel t_i$, for each of the q values compute $v_i = h(n_i \parallel t_i \parallel \tau)$, $v_{i_1} = h(Z \parallel v_i)$, $v_{i_2} = h(A \parallel v_{i_1})$, and verify that $M_{PC,i_2} = h(v_{i_2} \parallel K_{PC})$.

RRs which have expired (as indicated by time-of-expiry field added to each RR in an RRSet) will be ignored. If the ANSWER section indicate authenticated denial and the value τ smaller than the current time, the response is ignored. If any of the TMs A and P and G involved in relaying the RRSets has been revoked by the KDC, the RRSet is ignored.

4.4 Proof of Correctness

Consider a scenario where the verifier C determines that the set of values $\{Z, A, v_i, M_{PC,i_2}\}$ satisfy

$$M_{PC,i_2} = h(h(A \parallel h(Z \parallel v_i)) \parallel K_{PC}).$$

In concluding that the RRSet (with hash v_i) in the ANSWER section was indeed created by the zone authority Z (as indicated in the AUTHORITY section), TCB-DNS assumes

1) The integrity of TMs A and P: more specifically, i) secrets assigned to TMs are not privy to other entities, and ii) the atomic relay function cannot be modified;

2) The keys of the zone authority Z (possibly protected by a TM Z) are not privy to anyone else; and

3) The hash function $h()$ is pre-image resistant.

With these assumptions, it is easy to see that:

[5] The value W is obtained from the NS type RRSet for the parent zone W, which was obtained by querying W's parent - the root.

[6] Just as there is nothing that stops an authority of `example.com` from signing an RRSet for `www.yahoo.com` in DNSSEC, in TCB-DNS a zone authority can authenticate any value. However, resolvers will not accept RRSet as valid as $Z_{name} = h($`example.com`$)$ is not a parent of `www.yahoo.com`.

1) As the hash function $h()$ is pre-image resistant, the value M_{PC,i_2} was computed by an entity with access to the secret K_{PC} (thus the verifier can conclude that the value M_{PC,i_2} was computed by TM P).

2) The TM P can compute M_{PC,i_2} *only* if it was provided values $v_{i_1} = h(Z \parallel v_i)$ and M_{AP,i_1}, satisfying $M_{AP,i_1} = h(v_{i_1} \parallel K_{AP})$.

3) Only TM A could have computed the value M_{AP,i_1} provided to P.

4) TM A can compute M_{AP,i_1} only if it was provided values $\{v_i, M_{ZA,v_i}\}$ satisfying $M_{ZA,v_i} = h(v_i \parallel K_{ZA})$.

5) As only Z has access to secret K_{ZA}, the value v_i was created by Z.

Note that to conclude that "value v_i was indeed created by Z," it is *not* necessary that the parent zone W supports TCB-DNS. However, it is indeed desirable that all zones adopt TCB-DNS. If the parent zone also supports TCB-DNS, then the client can also verify the integrity of the NS RRSet for zone Z, and thereby confirm that A is indeed a TM associated with an ANS for the delegated zone Z.

5 Practical Considerations

TCB-DNS can be implemented with minimal modifications to current DNS servers. The specific modifications required to support TCB-DNS are as follows:

1) Every RRSet will indicate an absolute time of expiry (say, 32-bit UNIX time) specified by the zone authority; this value is unrelated to the TTL value[7] specified in each RR.

2) Each NS record will indicate the TCB-DNS identity of the ANS TM (this is similar to the requirement in DNSCurve where the elliptic-curve public key of the ANS is indicated in the NS record).

3) DNS queries will indicate an additional field - the TCB-DNS identity of the querier.

If an NS record for a zone W provided by a parent zone does not indicate the identity of a TM, the implication is that the indicated ANS for the zone W does not support TCB-DNS. It is also possible for a zone to employ as its ANSs, some TCB-DNS aware servers and some plain DNS servers. The NS records corresponding to TCB-DNS compliant ANSs will indicate the TCB-DNS identity of the ANS. NS records corresponding to plain DNS servers will not. Thus, a PNS which supports TCB-DNS may prefer to query a TCB-DNS compliant ANS for the zone W. Similarly, a plain DNS PNS may choose to direct its query to a plain DNS ANS for zone Z.

[7]The TTL value specifies how long an RR can be cached by resolvers.

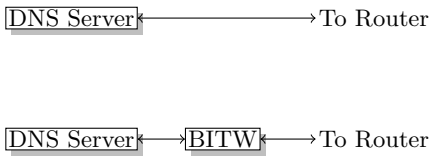

Figure 3: Top: Original Configuration. Bottom: Bump-in-the-Wire (BITW) Implementation.

If a TCB-DNS server receives a query which does not indicate the TCB-DNS identity of the querier, the querier is assumed to be unaware of TCB-DNS. In this case a plain DNS record is sent as a response. If a TCB-DNS unaware server is queried by a TCB-DNS compliant resolver the DNS server will simply ignore the additional field.

TCB-DNS can easily support bump-in-the-wire implementations (see Figure 3). Converting a plain DNS server to TCB-DNS server can be as simple as adding an additional BITW unit equipped with a DNS-TM. Only the BITW unit will need to have access to the TCB-DNS supplemental master file. The BITW unit will

1) Verify TCB-DNS authentication appended to incoming DNS packets, strip authentication, and relay plain DNS packets to the DNS server;

2) Append TCB-DNS authentication to outgoing DNS packets.

5.1 Ideal TMs

Deployment of TCB-DNS requires an infrastructure in place for some regulatory authority (for example, ICANN or IANA) to oversee the production and verification of trustworthy DNS TMs. Mandating rigid and simple functionality can go a long way in reducing the cost for deploying such an infrastructure, reducing the cost of the TMs, and rendering them more worthy of trust.

Just as guaranteed TCB functionality can leveraged to provide some assurances regarding the security of the system, for realizing guaranteed TCB functionality two fundamental assurances provided by trustworthy computing modules are leveraged - *read-proofing* and *write-proofing* (see Figure 4).

Read-proofing of secrets protected by the TMs is necessary to ensure that the modules cannot be impersonated. Write-proofing of software executed by the TMs is necessary to ensure that the TCB functions (usually dictated by the software) cannot be modified. The two requirements are however not independent. With the ability to modify software at will, an attacker can force the TM to reveal its secrets (for example, by inserting a set of commands to write the secret bytes out to the serial port). On the other hand, secrets that are protected can be used to authenticate software. Without the knowledge of the secret the attacker cannot modify the software (more specifically, such modifications will be detected due to failure

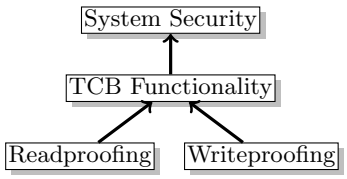

Figure 4: Leveraging Read-proofing and Write-proofing to Realize System Security.

of authentication).

In practice read-proofing is easier to realize [20], and is often a stepping stone to the more elusive goal of write-proofing [9]. Attacks aimed at modifying software to reveal secrets can be prevented by ensuring that software does not have access to at least some of the secrets that are protected. Some secrets may be *generated, stored and used* by dedicated hardware [21]. However, authenticating software with the secrets provides a boot-strapping problem [18]. After all, some software should be loaded which includes instructions to load a secret and carry out the steps required to perform authentication.

Realizing TMs that truly *deserve* trust calls for some simple common-sense restrictions to be imposed on the TMs. If the entire functionality of the TM is trivial enough to be *hard-wired* (and thus eliminate the need for mutable code), we can side-step issues associated with guaranteeing the functionality of the TM. If the TM functionality is simple and immutable, it is less expensive to *verify* such functionality as the testing infrastructure can be easily automated. If the TMs do not draw significant electrical power to perform its tasks (and consequently disseminate very little heat) we can then afford to physically shield the TMs extremely well from external intrusions aimed at exposing secrets.

It is for these reasons, that in identifying good TCB functionality for TCB-DNS we enforce these restrictions. The low-complexity, low-power, hard-wired TMs employed by TCB-DNS will merely require i) protected registers for storing a few secrets, ii) a hash function (for example SHA-1), and iii) hard-wired logic which drives a fixed sequence of logical and hash computations.

5.2 Leveraging TPMs

While creating a dedicated infrastructure for DNS-TMs is the preferred approach to realize highly trustworthy TMs, a practical alternative to lower the cost of the infrastructural requirements is to take advantage of an existing infrastructure for *trusted platform modules* (TPM).

The trusted computing group (TCG) approach to realize a trusted platform includes a thorough specification of trusted platform modules (TPM) [25], and recommendations on how such modules can be leveraged to provide some assurances regarding the integrity of the software stack running on a platform (a general purpose computer) equipped with a TPM chip. Several manufacturers of TPM chips exist today. Many desktop/laptop computers already posses a TPM chip, or have the capability (a slot in the motherboard) to accept a TPM chip.

The TPM exposes several interfaces which can be used by the platform to submit values for secure storage inside the TPM, and conveying such values to other parties, attested using secrets protected inside the TPM. Specifically, about 120 TPM interfaces have been specified in the current version of the TPM [24]. These interfaces are intended for a wide variety of purposes like i) taking ownership of the TPM; ii) generation/regeneration of keys; iii) submitting "measurements" of loaded software (in the form of hashes of loaded software) and extending such hashes into platform configuration registers (PCR) inside the TPM; iv) attestation of PCR values by the TPM for reporting the state of the platform; v) binding secrets to specific platform states (PCR values), etc.

The TCG model relies on three *roots* of trust: the root of trust for measurement (RTM); root of trust for storage (RTS); and root of trust for reporting (RTR). RTS and RTR are housed inside the TPM chip. The RTM is however constituted by components *outside* the TPM. More specifically, trusting the RTR and RTS amounts to trusting the integrity of the TPM chip. However, trusting the RTM implies trusting numerous components of a general purpose computer like the BIOS, CPU, RAM, CPU-RAM bridge, and possibly even some peripherals which have direct access to the RAM. In addition, to an infrastructure for verifying code is trusted to verify software and disseminate measurements (hashes) of authentic verified software.

Almost every attack on the integrity of a TCG trusted platform [4, 23] is a result of the fact that the RTM is constituted by components outside the TPM, which obviously do not merit the same extent of trust as components inside the TPM chip. Furthermore, the problem of verifying functionality of software is becoming increasingly intangible due to the ever increasing size of software and frequency of updates. For these reasons, some researchers have focused on strategies for securing applications running on untrustworthy platforms by relying *only* on the TPM [5]. The recent TPM 1.2 version added some additional TPM interfaces which expands the scope of applications that can be secured by relying only on the TPM as the TCB. Some desirable additions to TPM interfaces have been suggested in [6].

The atomic relay interface can be simply be an additional interface offered by the next version of TPMs. More specifically, TPMs will require two additional interfaces to serve as the TCB for a DNS server: i) an interface for securely receiving MLS keys (which can then be stored in some TPM PCRs reserved for this purpose), and ii) the atomic relay interface.

6 TCB-DNS vs. DNSSEC

The main reasons for the poor adoption of DNSSEC are i) the significant increase in the size of zone files (over

that of plain-old DNS) due to the addition of RRSIG, DS, DNSKEY, and NSEC/NSEC3 records; ii) increased bandwidth overhead for DNSSEC responses; and iii) the susceptibility of DNSSEC to DNS-walk. Due to the substantial overhead, it is especially expensive for large zones (for example, `.com`) to adopt DNSSEC.

DNSSEC and TCB-DNSSEC have many significant similarities:

1) Both do not require DNS servers and their operators to be trusted. Both protocols achieve this requirement by their ability to securely convey a pre-image resistant one-way function of RRSets created by zone authorities to end-points (clients), without the need to trust the intermediary servers.

2) In both protocols lifetimes are imposed on the validity of zone keys. Both specify validity periods for the authentication appended for RRs (which can at most be till the expiry of the keys used for validation).

3) Both use a strategy for ordering names (or a one-way function of names) to provide authenticated denial of enclosed names (or hashes of names).

4) Both protocols do *not* possess a mechanism for revoking authentication. Consequently both protocols are susceptible to replay attacks - under some conditions. If the authentication appended for an RRSet is indicated as valid till some time t, and if for some reason, there arises a requirement to modify the RRSet before time t, then an attacker may be able to replay the old RRSet (with a signature valid till time t) until time t.

The primary differences between DNSSEC and TCB-DNS include

1) The cryptographic mechanisms employed - DNSSEC relies on digital signatures, while TCB-DNS relies on TMs to atomically relay MACs.

2) Unlike TCB-DNS, DNSSEC does not provide assurance **A3**. Some of specific differences in the mechanism for authenticated denial, and the rationale for the choices made in TCB-DNS are outlined in Section 6.1.

3) DNSSEC demands substantially higher overhead compared to TCB-DNS; Section 6.2 provides a comparison of the storage bandwidth overheads of DNSSEC and TCB-DNS.

4) DNSSEC is more susceptible to replay attacks compared to TCB-DNS; Section 6.3 outlines the reasons for this phenomenon.

Since the discovery of the Kaminsky attack [12] the need to secure DNS has attracted renewed attention. Some modifications have been proposed to DNSSEC to address the main reasons for its poor adoption. However, while such efforts reduce some of the overhead for DNSSEC (and thereby reduce the resistance to adoption of DNSSEC), they are at the expense of watering-down some of the originally intended assurances of DNSSEC.

In Section 6.4 and we discuss such a mechanism, TSIG [26], which can reduce overhead for clients, but has the unfortunate side-effect of requiring to trust the PNSs. In Section 6.5 we discuss another modification (NSEC3 opt-out) [3] which is intended to facilitate easier adoption of DNSSEC by large zones like `.com`. This feature has an unfortunate side effect of interfering with the ability to provide authenticated denial. More recently, some attacks that exploit the NSEC3 opt-out feature have also been demonstrated [2].

6.1 Authenticated Denial

Consider a scenario where the ANS for the zone `example.com` is queried for a non-existent record "`a.b.example.com, A`." A negative response indicates that i) the queried name does not exist; *and* ii) no wild-card name like `*.b.example.com` exists; iii) no delegation exists for a zone `b.example.com`; and iv) no alias (type CNAME) record exists for the name `a.b.example.com`.

In NSEC the enclosures are textual strings indicating names. A single NSEC record [`example.com` [A,MX,NS], NSEC, `cat.example.com`] is adequate for the resolver to verify that all three conditions are true (all three names that have to be denied fall within the single NSEC enclosure). In NSEC3 the enclosures are hashes of names. Each NSEC3 enclosure can only be used to deny a *specific* name (which hashes to a value inside the encloser). Thus, proof of enclosure of three different name-hashes have to be provided to the resolver.

If a record of a type different from NS does exist for `b.example.com` or if a record with name `a.b.example.com` does exist (but not the solicited type A), then the NSEC3 record has to indicate the list of types that *do* exist[8].

Though intended as an improvement over NSEC, in some ways NSEC3 is actually inferior to NSEC. In response to a query for a non existent record, NSEC revealed two unsolicited names; NSEC3 typically reveals six hashes corresponding to six unsolicited name-hashes (which are subject to brute-force attacks). Furthermore, three RRSIG(NSEC3) signatures have to be verified (instead of one RRSIG(NSEC)).

The mechanism in TCB-DNS for authenticated denial is closer to NSEC3 than NSEC. The difference is that in TCB-DNS the name and type are hashed together (in NSEC3 only the name is hashed). As with DNSSEC-NSEC3, multiple name-and-type hashes will have to be denied by the ANS by using different enclosures. At first sight, it may seem that an NSEC-like approach may be preferable for TCB-DNS. After all, if only the TM is privy to the enclosures - viz., textual strings (names) in NSEC and hashes (of names) in NSEC3, there is no need for

[8]Thus, there are two ways in which NSEC3 fails to realize assurance **A3**: i) by being susceptible to simple dictionary attacks; and ii) by disclosing unsolicited types for a name.

hashing names. However, checking NSEC enclosures will require TMs to compare variable length text-strings, possibly of different formats (for example, ASCII, Unicode), which can substantially increase the complexity of TMs. With the NSEC3-like approach only fixed-length hashes need to be compared. Thus,

1) In DNSSEC with NSEC3 the purpose of hashing is to "hide" names (albeit ineffectively);

2) In TCB-DNS the purpose of hashing the names is *not* to hide the names - it is to lower the TM complexity.

In TCB-DNS, the reason for hashing name-and-type together is to ensure that (unlike NSEC3) names do not have to be disclosed if queried for a non-existent type. In TCB-DNS the number of encloser pairs equals the number of unique name-and-type values (which is the same as the number of RRSets). In NSEC3 the number of hashes correspond to the number of unique names.

The disadvantage of TCB-DNS is a small increase the number of encloser pairs. However, this is not a problem in practice. The increase in the number of encloser pairs would only be an issue for zones which have a very large number of names, *and* many types corresponding to each name. However, such large zones (like gTLD `.com`) which have large number of names, have only a single type (type NS) corresponding to most names!

Another difference between NSEC3 and TCB-DNS is the mechanism used for hashing names. In TCB-DNS the hash is computed as a function of name, type and a value τ. The value τ is the time of expiry of the authentication. In DNSSEC-NSEC3 the time of expiry is indicated in the RRSIG record; the name-hash is computed after including a salt to the name. Furthermore repeated hashing is employed to derive the name-hash. The reason for using the salt is to prevent *precomputed* dictionary attacks. The purpose of repeated hashing is to increase the computational complexity for dictionary attacks. As dictionary attacks are not possible in TCB-DNS (as the enclosers are never sent), TCB-DNS does not need to deliberately increase the computational overhead for generating name-hashes.

6.2 Overhead

Table 2 provides a quick comparison of TCB-DNS and DNSSEC.

The large size of DNSSEC records is due to the fact that public keys and signatures are large (1000 to 2000 bits). This increases the cache memory requirements for name servers. The longer RR sizes, and that multiple RRSIGs, DS and DNSKEY records need to be fetched and verified, results in substantial bandwidth overhead for responses.

For a typical query, the DNSSEC specific bits that accompany the response (over and above the plain DNS records) can easily be of the order of 2000 bytes. As described in Section 2.4, the additional DNSSEC specific records required to verify a RRSet include (typically) one RRSIG(RR), 3 DNSKEY records, 3 DS records, and 3 RRSIG(DS) records. Additionally, verification of an RRset requires the computational overhead for verification of 4 signatures.

For TCB-DNS the additional TCB-DNS specific bits that accompany the response (to a typical query) include i) the identity of the ANS and ii) one MAC in the `ANSWER` section, and two identities (TCB-DNS identity of the parent zone, and identity of an ANS TM of the parent zone) and one MAC in the `AUTHORITY` section. If the identities of TMs are 10 bytes long and identities of zones are 20 bytes long, and all MACs are 20 bytes long, the additional bandwidth overhead is of the order of 70 bytes. If we consider the additional values inserted in NS records the overhead may be close to 100 bytes (compared to 2000 bytes for DNSSEC).

For DNSSEC the increase in cache memory size for any RRSet is due to the addition of one RRSIG for every RRSet (about 200 bytes for every RRSet if 1600-bit RSA modulus is used). Additionally (for authenticated denial), corresponding to every unique name in the master file, one NSEC/NSEC3 record along with an RRSIG(NSEC/NSEC3) record are required: amounting to an overhead of roughly 300 bytes for every unique name in the master file.

In TCB-DNS, corresponding to every RRSet two additional values are required for regular responses - an index (of the RRSet within the master file) and a MAC. For authenticated denial, corresponding to every unique name and type (the total number of which is the same as the number of RRSets) three values are required: two hashes (enclosers), and a MAC for the enclosure. Assuming 20 byte hashes and MACs, the overhead is about 60 bytes for every unique name and type.

6.3 Replay Attacks

The fact that anybody can obtain verify a digital signature is a powerful feature of digital signatures. This power can also be abused more easily when no mechanism exists for revocation. A signed packet with prematurely invalidated contents can be more easily abused, compared to a packet authenticated using a MAC.

A DNS RRSet signed by the zone authority can be sent by anyone, from any place, to any place. However, in TCB-DNS, a MAC appended by a zone authority is intended only for the TM of a specific ANS. This implies that only the entity with control of the specific ANS (who has access to the TM) can replay such packets. This substantially reduces the *scope* of possible replay attacks.

For both protocols, reducing the scope of replay attacks requires choice of short life-times for signatures (MACs for TCB-DNS). Unfortunately shorter life-times imply more frequent re-computation of authentication. Due to the substantially lower computational overhead required for TCB-DNS we can can actually afford to recompute MACs more frequently.

Table 2: Comparison of TCB-DNS and DNSSEC

	Overhead in Bytes			
	Bandwidth	Cache per RRSet	Cache per Name	Assurances
DNSSEC	2000	200	300	**A1**, **A2**
TCB-DNS	100	24 + 60		**A1**, **A2**, **A3**

6.4 DNSSEC with TSIG

As originally intended, DNSSEC provides the end-points with the ability to verify the integrity of RRs. For most clients this is a substantial computational burden. This is especially true for an ever increasing number of battery operated mobile devices. Furthermore, to receive the large number of DNSSEC specific records from the PNS the clients may have to employ more expensive TCP instead of UDP as the transport layer. Note that for PNSs this is less of an issue as it receives the multiple records required for verification as multiple packets from different ANSs.

It is for this reason that in most standard installations of DNSSEC the verification of RRs is performed only by the PNS. Stub-resolvers are expected to establish a secure channel with PNSs using some light-weight mechanism like TSIG [26], and obtain verified RRSets over the secure channel. TSIG is a protocol which leverages shared symmetric keys (established by other means - outside the scope of TSIG) for establishing secure channels. In DNS, TSIG is used by zone authorities to securely send master files to ANSs. This same strategy can also be used for establishing a secure channel between clients and PNSs.

The implication of using such an approach to lower overhead for clients is that *DNSSEC can no longer claim that end-points do not have to trust the middle-men.* With this approach, clients are required to trust the PNSs (and consequently their operators). More specifically,

1) If a DNSSEC enabled PNS X is compromised, or if the TSIG secret of X is privy to an attacker, then X (or the attacker) can disseminate fake RRSets for *any* zone; such RRSets will be blindly accepted by *all* stub-resolvers which query PNS X.

2) Similarly, if a DNSCurve enabled PNS X is compromised, or if the DNSCurve secret of X is privy to an attacker, then X (or the attacker) can disseminate fake RRSets to all stub-resolvers that query PNS X.

3) On the other hand, in the case of TCB-DNS, if a PNS X (with DNS-TM P) is compromised, the attacker cannot disseminate fake RRSets. It is only if the secrets of the TM P become privy to the attacker (*and* if the TM P has not been revoked) can the attacker disseminate fake RRSets to the stub-resolvers that query PNS X.

6.5 NSEC3 Opt-out

For consummate realization of DNSSEC assurances even top level domains should adopt DNSSEC. While authenticated denial is an especially important feature for gTLDs, the overhead for this purpose can be substantial for large zones, and especially for zones where new names are frequently added. More specifically, zones with frequent addition/deletion of names become more susceptible to replay attacks.

Consider a scenario where an RRSet corresponding to a new name (or name-hash) x needs to be added. Before the name is added, a signed encloser (x_l, x_h) will exist for x. However, after inserting x the encloser (x_l, x_h) needs to be revoked. Two new enclosers should be added instead - (x_l, x) and (x, x_h). Similarly, consider a scenario when an existing name y needs to be removed, and two signed enclosers (y_l, y) and (y, y_h) currently exist. In this case, both enclosers (y_l, y) and (y, y_h) need to be revoked and replaced with a new encloser (y_l, y_h).

Due to the fact that it is not possible to foresee which of the currently valid records will need to be revoked due to the addition of an (as yet unknown) name in the future, it is necessary to choose small enough life-times for all NSEC/NSEC3 enclosers. Obviously, for gTLDs like `.com` with several tens of millions of names, this is far from practical.

In TCB-DNS, due to the low overhead for computing MACs even gTLDs can afford to recompute enclosers more frequently. However, frequent re-authentication of NSEC3 records in DNSSEC is expensive for two reasons. The obvious reason is that the computational overhead for digital signatures is high. The other reason is that NSEC3 *deliberately* increases the complexity of hashing names to render dictionary attacks more time-consuming. Due to the substantial overhead involved in re-generation of signed NSEC3 records, DNSSEC is forced to employ larger life-times for NSEC3 signatures and consequently become more susceptible to replay attacks.

Recently, NSEC3 with an opt-out specification [3] has been proposed to make it more practical for gTLDs to adopt DNSSEC. Using opt-out NSEC3 can reduce the instances leading to revocation of RRSIG(NSEC3) RRs, thereby permitting longer lifetimes for NSEC3 RRSIGs. An NSEC3 record indicating an encloser (x_l, x_h) with an unset opt-out bit is proof that no enclosing records exist. However, if the opt-out bit is set, the implication is that zero or more unsigned delegations *may* exist - thereby diluting assurance **A2**. Furthermore, some serious security exploits resulting from using the opt-out approach have been identified recently [2].

7 Conclusions

Adoption of DNSSEC has been marred by the substantial overhead required, and the issue of DNS-walk. The large increase in the zone file size is especially severe for gTLD DNS servers and DNS servers employed by specialized DNS service providers who run DNS services for a large number of zones. This is further exacerbated by supercilious queries from "DNS-walkers." While the need to address DNS-walk is especially crucial for large DNS operators, ironically, using NSEC/NSEC3 feature of DNSSEC for this purpose is risky for such DNS servers due to the possibility of DNS-walk. Recent attempts to reduce resistance to adoption of DNSSEC have unfortunately come at the expense of security.

The primary insight for TCB-DNS approach stems from the fact that cryptographic techniques for independently securing each link in a query response process (like the approaches in symmetric key DNSSEC and DNSCurve) demand substantially lower overhead. As the light-weight link-security approaches require the intermediaries (DNS servers) to be trusted, a natural question then is "what is the minimal TCB for a DNS server?" As long as this TCB is trustworthy we will not be required to trust other components of the intermediary servers.

A TCB which simply relays hashes can provide assurances **A1** and **A2**. By adding some intelligence to the simple relay function (to verify that "one input lies between two other inputs") we can realize assurance **A3**, and thus eliminate the problem of DNS-walk. Due to the negligible overhead (a few tens of bytes of bandwidth overhead, and computation overhead amounting to a few hashes) even gTLDs can easily switch to TCB-DNS.

To summarize, the main advantages of TCB-DNS over DNSSEC are

1) TCB-DNS demands substantially lower overhead;

2) TCB-DNS eliminates the issue of DNS-walk;

3) TCB-DNS is less susceptible to replay attacks;

4) Due to the low overhead for verification, TCB-DNS will not require clients to trust PNSs (as in DNSSEC with TSIG);

5) Due to the low overhead for re-authentication of enclosers, TCB-DNS does not need to employ potentially dangerous practices like "NSEC3 opt-out."

Deployment of TCB-DNS ideally requires a dedicated infrastructure in place for some regulatory authority to oversee the production and verification of trustworthy DNS TMs. Alternately, two additional interfaces (one for for accepting MLS keys, and the second for performing the atomic relay) can be added to the next version of the trusted computing group (TCG) specification for trusted platform modules (TPM).

References

[1] R. Arends, R. Austein, M. Larson, and D. Massey, S. Rose, *DNS Security: Introduction and Requirements*, RFC 4033, Mar. 2005.

[2] J. Bau and J. C. Mitchell, "A security evaluation of DNSSEC with NSEC3," *The Seventeeenth Annual Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, 2010.

[3] CommunityDNS, *DNSSEC:Way Forward for TLD Registries*, White paper, Sep. 2009. (http://www.communitydns.net/DNSSEC.pdf)

[4] S. Bratus, N. Dunha, E. Sparks, and S. W. Smith, "TOCTOU, traps, and trusted computing," *TRUST 2008*, LNCS 4968, pp. 14-22, Springer-Verlag, 2008.

[5] M. V. Dijk, L. F. G. Sarmenta, J. Rhodes, and S. Devadas, "Virtual monotonic counters and count-limited objects using a TPM without a trusted OS," *Proceedings of The First ACM Workshop on Scalable Trusted Computing*, pp. 27-31, 2006.

[6] M. V. Dijk, L. F. G. Sarmenta, J. Rhodes, and S. Devadas, *Securing Shared Untrusted Storage by using TPM 1.2 Without Requiring a Trusted OS*, MIT CSG Memo 498. May 2007.

[7] DNS Survey: August 2006. (http://dns.measurement-factory.com/surveys/200608.html)

[8] C. Fetzer and T. Jim, *Incentives and Disincentives for DNSSEC Deployment*, June 2004. (http://research.att.com/~trevor/papers/dnssec-incentives.pdf)

[9] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin, "Tamper Proof Security: Theoretical Foundations for Security Against Hardware Tampering," *Theory of Cryptography Conference*, Cambridge, MA, Feb. 2004.

[10] (http://dnscurve.org/)

[11] D. Kaminsky, "Catching up with Kaminsky, " *Network Security*, vol. 2008, no. 9, pp. 4-7, Sep. 2008.

[12] D. Kaminsky, *DNS 2008 and the New (old) Nature of Critical Infrastructure*, BlackHat DC, Feb. 2009.

[13] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems*, vol. 10, pp. 4, pp. 265-310, 1992.

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

[15] T. Moreau, *A Short Note about DNSSEC Impact on Root Server Answer Sizes*, Document Number C003924, Aug. 2006. (http://www.connotech.com/dnssec_root_impact.pdf)

[16] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993-999, Dec. 1978.

[17] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications*, vol. 32, no. 9, pp 33-38. Sep. 1994.

[18] P. C. V. Oorschot, A. Somayaji, and G. Wurster, "Hardware-assisted circumvention of self-hashing software tamper resistance," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 82-92, Apr. 2005.

[19] M. Ramkumar, "On the scalability of a "non scalable" key distribution scheme," *IEEE WoWMoM SPAWN*, pp. 1-6, 2008.

[20] M. Ramkumar, "Trustworthy computing under resource constraints with the DOWN policy," *IEEE Transactions on Secure and Dependable Computing*, vol. 5, no. 1, pp. 49-61, Jan-Mar 2008.

[21] S. W. Smith and S. Weingart, "Building a high-performance programmable secure coprocessor," *Computer Networks*, vol. 31, pp. 831-860, 1999.

[22] A. D. Sorbo, *Network Security - Sk-DNSSEC: An Alternative to the Public Key Scheme*, PhD Thesis, Department of Computer Science, University of Salerno, Baronissi, Italy.

[23] E. Sparks, *A Security Assessment of Trusted Platform Modules*, Technical Report TR2007-597, Dartmouth College, Computer Science, Hanover, NH, June 2007.

[24] TPM Main, *Part 3: Commands*, specification version 1.2, October 2006.

[25] Trusted Computing Group. (http://www.trustedcomputinggroup.org)

[26] *TSIG: Secret Key Transaction Authentication for DNS*, RFC 2845, May 2000.

[27] S. Weiler and J. Ihren, *Minimally Covering NSEC Records and DNSSEC On-line Signing*, RFC 4470, Apr. 2006.

Arun Velagapalli is a PhD student in the Department of Computer Science and Engineering, Mississippi State University. He received his MS in Computer Science and Engineering (208) from Mississippi State University, MS, and B.Tech degree in Computer Science (2006) from GITAM, Andhra University, India. His research interests include Trustworthy Computing, Network Security, Cryptography, and SCADA Security.

Mahalingam Ramkumar is an Associate Professor in the Department of Computer Science and Engineering, Mississippi State University. He received his Ph.D. in Electrical Engineering (2000) from New Jersey Institute of Technology, Newark, NJ, MS in Electrical Communication Engineering (1997) from Indian Institute of Science, Bangalore, India and Bachelor Degree in Engineering (1987) from University of Madras, India. His research interests include Trustworthy Computing, Network Security, Cryptography, and Mobile Ad hoc Networks.