Verifying Keys through Publicity and Communities of Trust: Quantifying Off-Axis Corroboration [Supplemental Material]

Eric Osterweil[1], *Dan Massey*[2], *Danny McPherson*[1], *Lixia Zhang*[3]
Verisign Labs, {eosterweil, dmcpherson}@verisign.com[1]
Computer Science Department, Colorado State University, massey@cs.colostate.edu[2]
Computer Science Department, UCLA, lixia@cs.ucla.edu[3]

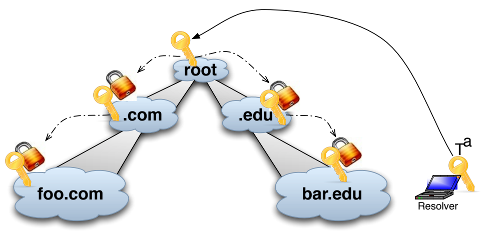


Fig. 1. The root zone's key is a trust anchor (T^a) of the "chain of trust"

I. Operational DNSSEC

The DNS Security Extensions' [5], [7], [6] (DNSSEC's) design goal is to prove that data in a DNS reply is authentic. In order to do this, each zone creates public/private key pairs and then uses the private portions to sign data. Its public keys are stored in DNS records called DNSKEYs, and all the signatures are stored in records called RRSIGs. In response to a query from a DNS resolver, an authoritative server returns both the requested data and its associated RRSIGs. A resolver that has learned and authenticated the DNSKEY of the requested zone can verify the *origin authenticity* and *integrity* of the reply data. To resist replay attacks, each signature carries a definitive expiration time.

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

There are two challenges in building the chain of trust. First, a parent zone must encode the authentication of each of its child zone's public keys in the DNS. To accomplish this, the parent zone creates and signs a Delegation Signer (DS) record that is a hash (SHA1, SHA256, etc.) of a DNSKEY record at the child zone, and signs it with *its own* key. This creates an authentication link from the parent to child. It is the child zone's responsibility to request an update to the DS record every time the child's DNSKEY changes. Although all the above procedures seem simple and straightforward, one must keep in mind that they are performed manually, and people inevitably make errors, especially when handling large zones that have hundreds or thousands of children zones.

The second, and more problematic, challenge comes when there is a break in the delegation chain. This could be because the parent zone is not signed, or simply because there has been an operational error (e.g. the .gov, .fr, and many other events). In either case there is no valid chain of trust leading to the child zone's DNSKEY. This orphaned key effectively becomes an isolated trust anchor for its subtree in the DNS hierarchy. To verify the data in these isolated DNSSEC zones, one has to obtain the keys for such isolated trust anchors offline in a secure manner. DNSSEC resolvers maintain a set of well-known "trust-anchor" keys (T^a) so that a chain of key sets + signatures (secure delegation chain) can be traced from some T^a to a DNSSEC key lower in the tree. A full description of these difficulties can be found in prior work [15]. Ultimately, without a fully deployed, 100% operational secure delegation hierarchy, DNSKEY messages can be spoofed as easily as ordinary DNS messages.

A. Operational Status

In our previous work [14], [1] we used the SecSpider distributed monitoring apparatus to measure and *quantify* various behaviors of DNSSEC's global deployment from a set of globally distributed vantage points. Our goal was to use our candidate quantification to understand how well DNSSEC's protections are functioning. We derived the following three measures, and then used them to construct system metrics that were normalized on a scale ranging from zero to one (inclusive):

- *Availability:* How well is data delivered to resolvers throughout the Internet?
- *Verifiability:* How well can resolvers cryptographically verify DNSSEC data?
- *Validity:* How often is the cryptographically verified data actually valid (as opposed to serving properly signed, but incorrect, data)?

One of the most poignant findings from [14] was that DNSSEC's secure delegation hierarchy (i.e. the key verification design) has left a great many zones without a chain of trust, and thus, wholly unverifiable. The *verifiability* metric concisely quantifies a growing number of incidents in which operational errors at large TLDs (`.gov`, `.arpa`, `.uk`, `.fr`, `.de`, etc.) have caused disruptions in the delegation chains and left descendant zones unverifiable [9], [16], [12], [10].

B. Challenges

The operational lessons learned thus far lead to the following critical challenges:

Lookup and Verification Are Not Equivalent: DNSSEC overlays a key verification hierarchy on top of the existing DNS name hierarchy. However, these two operations have very different goals and requirements. The hierarchical DNS name space is primarily used to enforce name uniqueness; DNS name resolution also utilizes the loose coordination between parent and child zones to locate resource records (name servers, IP addresses, mail servers, etc.). On the other hand, a single root key requires global trust, which runs counter to the administrative autonomy in the Internet's operational model. Key verification involves trust relations between parties which may not follow or match the DNS hierarchical structure.

Robustness: DNSSEC is vulnerable to all sorts of data delivery failures and imperfect operations. For example, resolvers may fail to learn keys due to PMTU issues [3], [4] and zones (including TLDs) have become unverifiable due to human errors ranging from failing to update signatures to disagreements over policies on cryptographic algorithms and parent/child key exchange procedures. The concern is not that a particular error has occurred; the key insight is that operations will never be perfect. We argue that in the absence of full provably correct operations (which will never occur), DNSSEC must still be able to verify keys.

Unfractured Deployment: In DNSSEC, a child zone cannot have its keys verified until its parent has turned on DNSSEC. In fact, this has been a major issue in rolling out DNSSEC over the last several years. Furthermore, the lack of strong robustness discussed above means we not only need full deployment, we need full 100% operational correctness. We argue that an operational system should tolerate gaps in deployment, regardless of whether those gaps are caused by failure to deploy or failure to deploy correctly.

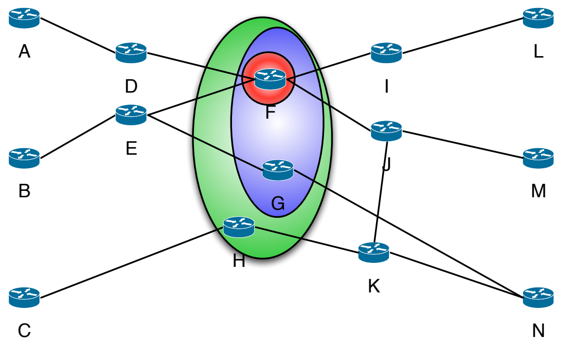


Fig. 2. Here we can see that how adding either name servers or CoT members can increase the size of a min-cut set. if Alice is at vantage A and $V_Z = \{L, M\}$, then her min-cut set is order 1 (if vantage F is cut then she is disconnected). However, if she adds (say) vantage B to V_{CoT}, then her cut set grows to the oval $\{F, G\}$. Here we note that even if V_Z grows to $V_Z = \{L, M, N\}$ the min-cut set remains the same. However, if Alice adds C to her V_{CoT} the cut set grows to the circle: $\{F, G, H\}$.

C. Verification Processes

Algorithm 1: $verif(O, p)$: Returns a datum with higher than p proportion of agreement, or the empty set.

```
begin
    /* Init variables to 0/Null */
    max_count = 0; $d^{verif} = counts[] = d^{max} = \emptyset$
    foreach $o_i \in O$ do
        /* Data item in observation */
        $d_i$ =extract_data($o_i$)
        counts[$d_i$]++

        if counts[$d_i$] $>$ max_count then
            max_count = counts[$d_i$]
            $d^{max} = d_i$
    /* If max exceeds threshold */
    if max_count $> (p \times |O|)$ then
        $d^{verif} = d^{max}$
end
```

Vantages Administrative Page

Lookup Data Trigger Poll Submit New Data Friends Monitored URLs

Fig. 3. Vantages' web administrative dashboard

II. Minimum Cut-Set

In order for Eve to fully subvert Alice's Community of Trust (V_{CoT}), she must be able to intercept all messages between V_{CoT} and the servers Alice is trying to reach (V_Z). From this observation, we generalize that Eve needs to be in the position to be able to partition V_{CoT} from V_Z, and she would like to minimize her acquisition costs in doing so. The lower bound on the *number* of nodes she needs to compromise is on the order of the minimum cut set: $|V_e| = O(MinCut(v_j, V_Z))$. The intuition here is that Eve's vertices must be able to disconnect (or partition) all messages from V_Z to anyone in Alice's V_{CoT}. Figure 2 illustrates this general idea, and that We note that the cut set may not grow every time a source or destination is added, but sometimes it can. We therefore define $V_{cut} = MinCut(V_{CoT}, V_Z)$, and use $|V_{cut}|$ as a lower bound on the number of nodes needed for Eve's attack to succeed.

III. System Implementation

Vantages is an open source package that is written in C++, it uses a SQLite backend, and has been publicly available since early in 2009 [2]. CoTs are peer-to-peer networks where each witness is an individual `vantaged` daemon. These daemons are designed to be installed alongside recursive resolvers (on the same hosts) so that it can use `libpcap` to automatically learn which zones to monitor. The daemon uses an embedded webserver so that configuration can be done through a web *dashboard* (Figure 3), and so that peer-to-peer communication within CoTs can use HTTP too. Also, when setting up an instance, operator must specify a PGP [19] key that is used to sign all data seen by the daemon.[1]

Getting Data: Vantages continuously polls the zones in its corpus to verify their `DNSKEY`s. The set of data sources for each zone is comprised of its list of name servers, and any URLs (such as dnssec:org.?type=48 or http://secspider.cs.ucla.edu/--zone.html) that have been manually added through the dashboard. An observation is stored from each source, the `DNSKEY`s are extracted from each observation, and then all values are check to see what fraction are consistent. This process creates a dependency (or provenance) tree leading from each key seen to all of the observations that it was seen in. Thus, given a `DNSKEY`, an operator can trace the derivation history, know where it was observed, by whom, and verify its original observations' origins by their attached PGP signatures.[2]

[1]PGP was chosen because it is widely used and accepted by many operators.

The Community of Trust: Configuring `vantaged` with witnesses involves bootstrapping the local daemon with the public PGP key from a remote witness' daemon (to verify communications), the HTTP URL that their daemon listens on, and optionally an HTTP password.

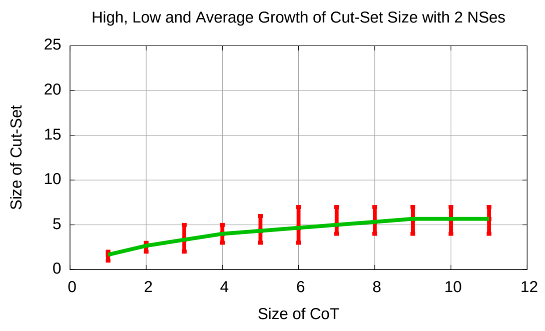
Whenever the daemon polls a zone, and successfully processes its keys, it then queries the other witnesses in its CoT. If the remote witness has processed data values for the query, then they are returned with the remote witnesses PGP signature covering the data and the timestamp at which it was processed. With the responses from witnesses, `vantaged` verifies each of the keys by classifying them with its CPBUC verification scheme.

Resolver Integration: The simple way that Vantages interfaces with DNS resolvers is intended to be one of its strengths. Rather than requiring any modification of resolver code, `vantaged` is designed to be deployed alongside a typical resolver (BIND or unbound). The results of the daemon's verification are stored in a `trusted-keys` file, whose format is purposely the same as the format accepted by these resolvers. Thus, an operator simply needs to configure these resolvers to refer to the same file that `vantaged` is configured to create, and it will begin verifying zones.

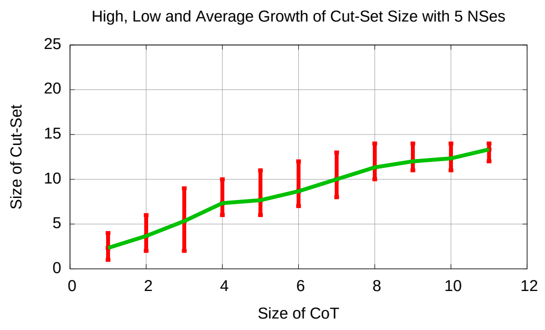
One example configuration an operator might use would be to enable `vantaged`'s `libpcap` mode and only place keys in the output file if they have reached the Confirmed state of CPBUC. In this case, the operator has chosen to be liberal in discovering zones, but conservative in trusting the observed keys. Alternately, an operator might choose to *manually* select web pages that contain well know trust anchors and require the results to be Confirmed. Thus, the daemon will be conservative in both the zones it trusts, and the level of verification needed. If the operator wants to be very liberal, he may choose to enable `libpcap` and trust any key in the Confirmed, Provisional, or Byzantine states.

Vantages' trusted-keys file gives the operator the final arbitration to manually control what is exported to a resolver. Specifically, even after the daemon has applied all of its policies, a fair amount of user-specific policy work can still be done to it before it is presented to a resolver. For example, an operator can use `awk`, `perl`,

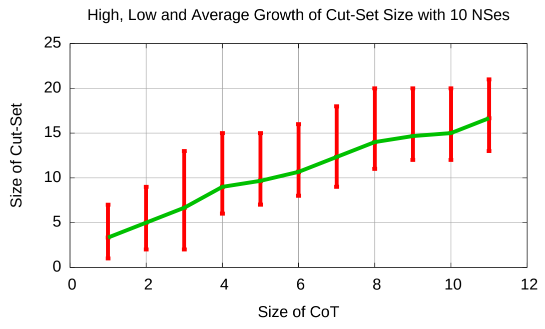
[2]Extracting `DNSKEY`s from non-standard sources, such as web pages, includes running either pre-existing perl code or operator-specified snippets. However, due to space restrictions, the details can be read on the project web page [2].



(a) $|V_Z|$ chosen to represent small zones (V_Z = 2)



(b) Here we have the V_Z size chosen to represent medium sized zones (V_Z = 5)



(c) $|V_Z|$ chosen to represent larger TLD/root zone sizes (V_Z = 10).

Fig. 4. These are our lower and upper bounds of the of $|V_{cut}|$ when randomly choosing V_{CoT} and V_Z.

`grep`, `python`, `split`, etc. to remove keys from certain zones, add local keys that the operator wants to manually configure, split the file into smaller files, or any number of other actions.

IV. Simulated Minimum-Cut Sets

Cut-Set: Using our topology, we mapped each combination of V_{CoT} and V_Z to the min-cut set V_{cut} between them. Figure 4 shows a general trend of increase in min-cut set size as V_{CoT} and V_Z grow. Intuitively, the min-cut set should grow as the CoT increases. Adding more witnesses to the CoT adds both nodes and paths. However, the addition of a new witness does not necessarily add value if its paths to the authoritative servers go through the previous cut set. The figures show how likely it is that when Alice adds a witness to V_{CoT}, her V_{cut} will grow.

V. Targeted Adversary

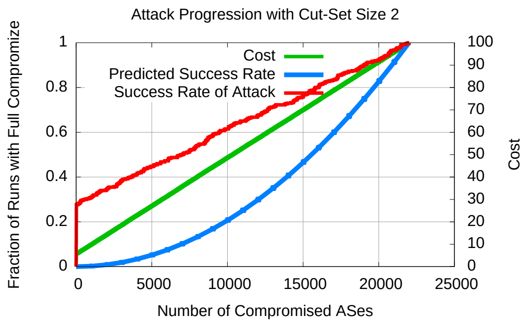
We next consider the *Targeted* adversary. Recall that she has taken the time to learn Alice's AS path, and compromises ASes in it, before resorting to a general attack against the rest of Alice's CoT. When Alice has a very small min-cut set (presumably from a small CoT), Eve's success rate surpasses our model's predictions. However, from Figure 5 we can see that as Alice's CoT grows, Eve's chances begin to closely resemble those of a general adversary. Additionally, we can see that even in the case where $|V_{cut}| = 14$, Eve's probability of success only begins to reach 10% after she has spent approximately 80% of the total cost to compromise the entire Internet. Thus, Alice's road to salivation, as with a general adversary, is through augmenting her CoT.

VI. Vantages Deployment Analysis

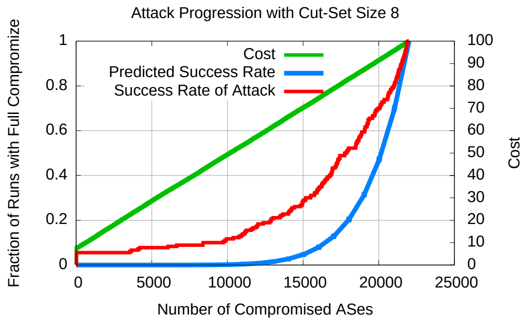
previous work quantified the operational status of DNSSEC's global deployment. However, the deployment metrics we derive for an operational deployment of Vantages only make sense if they include an operator's CoT, and the local policy for how much evidence is needed to make local trust decisions. As a result, each metric is formulated as a function whose inputs include this configuration information.

In this Section, we describe one candidate way to quantify the robustness of our theoretical model, Section I-A describes one method for quantifying the robustness of DNSSEC's deployment, and here we propose a way to map the same measures used for DNSSEC into system metrics that quantify the robustness of a live Vantages deployment.

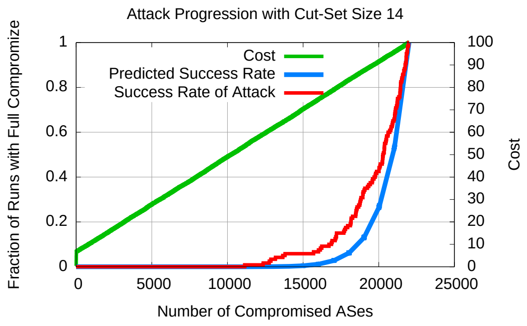
Vantages does not change *how* `DNSKEY`s are delivered and thus, its deployment affects the verifiability and validity of DNSSEC, but not *availability metric*. The intuition behind Vantages' *verifiability metric* follows from the observation that the minimum-cut set size is the lower bound on the cost an attacker incurs. The larger this set, the greater Alice's chances of overcoming an adversary. We use both Alice's specific V_{CoT}, and a parameter p (the number of independent paths between V_Z and V_{CoT}) to quantify her ability to verify DNSSEC data to be able to classify data as *Confirmed*. We define



(a) This figure shows that for a small $V_{cut} = 2$ the Targeted adversary has a better than predicted chance to spoof Alice.



(b) For $V_{cut} = 8$ the adversary is already significantly less likely to be able to spoof Alice, though not quite as unlikely as predicted in the general case.



(c) For $V_{cut} = 14$, Alice's protection approaches Equation **??**.

Fig. 5. Targeted adversaries with various V_{cut} sizes.

$Z^p_{V_{CoT}}$ as the set of secure zones who offer at least p independent paths from V_{CoT} to a zone's set of name servers (V_Z) and we define Z^s to be the set of all DNSSEC enabled zones. With these, we formulate the verifiability of a Vantages deployment as:

$$V^f(V_{CoT}, p) = \frac{|Z^p_{V_{CoT}}|}{|Z^s|} \qquad (1)$$

a) Validity: For the validity metric we follow the same derivation process as described in [14]. In that work, DNSSEC's operation practices were measured rather than a specific attack, simulated threats, etc. We, therefore, base the validity metric for Vantages on deployment data as well. In this previous work, we used Table I to characterize *ideal behavior* and *intended defense* states. These became the two components of our two-tuple validity metric for DNSSEC, and we used incidences of false positives and false negatives to calculate these two quantities. The metric took the form of delegation integrity (ideal behavior) and freshness (intended defense):

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

TABLE I
VERIFICATION VS VALIDITY MATRIX.

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

The V_{fresh} component (derived from the false positive rate) was calculated from observations of rapid re-signing, and did not reflect a facet of `DNSKEY` verification. Thus, the Vantages deployment does not have an analog for this component of the validity metric.

Rather, we focus our analysis of the Vantages validity metric on the V_{deleg} component (derived from false negatives). To characterize this, we measure rate of incidence where the `DNSKEY`s for a zone have changed, but a Vantage daemon has not learned the new value and is still reflecting the old values. We define the set of stale `DNSKEY`s in all of the Vantage daemons in a V_{CoT} as K^{stale}, the total number of keys in V_{CoT} as K, and we define the Vantages validity metric V^d as:

$$V^d = 1 - \frac{|K^{stale}|}{|K|} \qquad (2)$$

VII. Vantages Deployment Evaluation

Since SecSpider is able to act as a peering point for Vantage daemons, we opted to augment our CoT's topological diversity with SecSpider's polling base. These poller locations are UCLA, NLnet Labs, Colorado State University, Tsinghua University, a cable ISP in Los Angeles, Toshiba Corporation in Japan, SWITCH in Switzerland, Telx, and NIC.br. Thus, we evaluated our operational metrics using data from each `vantaged`'s database and using the topological diversity of SecSpider's polling locations.

Verifiability: We found that the the average min-cut set size was 14.33, and if resolver operators use *only* the SecSpider polling base and use a p threshold of 6,

Poller ID	Location	AS #
0	UCLA	52
1	NLnet Labs	12859
2	Colorado State Univ.	12145
3	Tsinghua Univ.	4538
4	Cable Modem in LA	7757
5	Toshiba Corp.	2500
6	SWITCH	559
7	Telx	19080
8	NIC.br	22548

TABLE II
AS #S AND LOCATIONS OF SECSPIDER POLLERS.

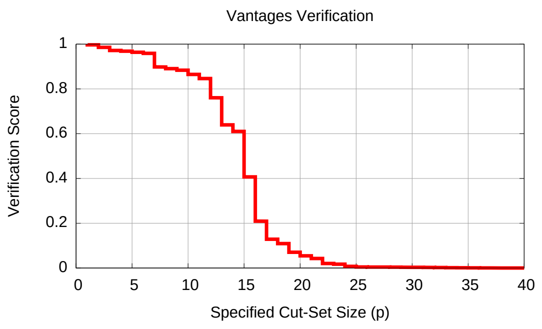


Fig. 6. A plot of Vantages' verification metric from SecSpider's V_{CoT} and varying p.

they can verify 96.4% of all DNSSEC zones.[3] In this case, Vantages' verifiability score (in September 2010) was $V^f = 0.964$, compared to DNSSEC's score of just $V^f = 0.437$, from the same time. If they use a threshold of 10, then roughly 86.5% of all zones are verifiable. It is important to note that setting p to a certain value does not limit Vantages from gaining more verifiability from zones with larger min-cut sets. It simply lets it consider zones that it might otherwise flag as *Provisional*. Thus, this value is not a hard cutoff, but a way to refine Vantages' classification.

Validity: To evaluate Vantages' validity we examine the data provenance stored by the daemons in our CoT since 2009. During this time, we observed cases in which Vantage daemons got conflicting values when comparing keys from their databases and from querying the network. We verified that these were not DNS attacks, and were simply mismatches do to polling cycles. That is, some zones had either rolled their keys over to new values, or revoked old keys and not all daemons noticed the changes at the same time. Thus, for Equation 2, we designated K^{stale} as any key who appeared to have conflicting values in the absence of DNS attacks, and chose K to be all keys whose state was determined to be Confirmed, Provisional, Byzantine, or in Conflict. We did not consider keys in the Unknown state because the Vantages system did not have enough data to act on.

[3] p is implemented as a configuration variable for refining Vantages' classification.

Vantage 1	Vantage 2	Vantage 3	Vantage 4[6]
0.954	0.993	0.938	0.989

TABLE III
VALIDITY SCORES FOR EACH VANTAGE DAEMON IN OUR COT.

We applied Equation 2 to each of the daemons in our CoT, and the results are in Table III. What we can see from this is that Vantages was able to properly verify its data anywhere from 96.4% of the time to 99.5% of the time in our specific deployment. The variability between daemons comes from the degree of zone overlap each daemon has with the other. Each daemon learns which zones to poll by observing the local DNS queries (i.e. the daemons had not always seen keys that their peers request information about).

VIII. Related Work

SecSpider: SecSpider [11] has been a distributed measurement apparatus since late 2005, but has also functioned as a distributed key learning and verification system since 2007. Users of the SecSpider system can download the trust-anchors file (containing all of the keys that SecSpider has verified using its nine distributed pollers) and perform local lookups.

The SecSpider system verifies keys that are seen as consistent across all of a zone's name servers and from all of the SecSpider pollers. Users can verify any DNSSEC-enabled zone, but they must rely on SecSpider as a trusted 3^{rd} party infrastructure.

Perspectives: Perspectives [18] is a distributed key verification system that is similar in design to SecSpider, but instead of `DNSKEY`s, it is designed to verify SSH keys on demand. The design of the system begins by having clients learn the PGP keys for a set of Perspectives' *notary* servers, which are positioned at four points in the Internet. Clients learn the PGP keys for these servers, and are then able to examine the responses to their queries.

While Perspectives does not require clients to use their notary network, any client that wants to set their own notary network up must do all of the work on their own. Thus, while the approach does not *require* a 3^{rd} party infrastructure to operate, clients are not given much help in setting up their own, and there is no clear alignment of operational costs and benefits in deploying a notary network. In addition, Perspectives does not have a formal model for arbitrating conflicting or historical data.

Lastly, the Perspectives system does not have formal model for arbitrating cases in which conflicting data is retrieved. The notary servers maintain a certain amount of history for the keys they have seen, and this data is available to clients. However, the framework of how to use this is unspecified.

Public-Space Key Infrastructure (PSKI): The PSKI was proposed in [13] along with a novel concept called the *public-space*, which suggested the need for complete information of digital entities' actions to be made publicly available to every user. It described the need for a structured framework to maintain a large number of entities, their actions, relationships, and histories. Its concept was that posting such information in public would not endorse the information's correctness, but it would provide users with a quantifiable set of information that would enable them to detect faults and make informed security decisions. Combined with traditional cryptographic techniques, the public-space system was intended to support the intrinsic heterogeneity of user security requirements in Internet-scale infrastructures and applications.

Pretty Good Privacy (PGP): PGP [19] embraces the notion that all crypto and verification should exist without any global authorities. In concert with a concept called the Web of Trust [8], users are able to create attestations for each others' identities. By themselves, the combination of these approaches has generally been regarded as offering insufficient protection for any operationally critical systems.

In PGP [19] there is no formally identified authority for verifying keys or data. PGP users generate their own public and private key pairs and then immediately use them to sign, encrypt, and verify data. These keys contain identification information, such as the name of the person who generated them, their email address, and sometimes even their picture so that when someone obtains a key they can associate it with the person who owns it. Without a global authority, there is no de facto way to distinguish between keys that are spoofed and valid keys. using other peoples' keys in Some form of real-world trust should exist before accepting a PGP key from someone else. In other words, *Alice* should use some real-world mechanism to verify that key K belongs to *Bob*. is does not mean that *Bob* is necessarily a trustworthy individual. Rather, it simply means that *Alice* trusts that key K belongs to *Bob*.

One person's trust in a key does not necessarily influence anyone else's. There is a notable addition to PGP called the Web of Trust [8] in which users can cross-certify each others' keys so that someone can use other people's attestations as evidence that a PGP key belongs the its purported identity. Whether a user wants to believe these attestations, and whether they are rigorous enough evidence to judge trustworthiness is beyond the scope of this paper.

All trust in PGP is local in scope and PGP operates based on a *local* trusted key list (a keyring). A user must first decide for herself if a key is trustworthy (via key signing parties, the Web of Trust, etc.) and then enter that key in their keyring. All PGP verification, encryption, and signatures are based on this local file. When an existing key expires, the new version needs to be fetched and verified manually (as described above). When keys are revoked, there is no clear way to ensure that users can even learn of the revocation. PGP makes no provisions for the maintenance of its trusted key lists.

SDSI: One other key learning approach that is relevant is SDSI [17]. SDSI is not an operational Internet key learning system (as those above are), but it is a framework in which the veracity of keys can be determined through local attestations. That is, when a user decides to give trust to an entity, that entity is then able to vouch for other keys, which allows for a grassroots style of key verification.

References

[1] SecSpider. http://secspider.cs.ucla.edu/.

[2] Vantages. http://www.vantage-points.org/.

[3] Availability problems in the dnssec deployment. In *RIPE 58*, 2009. http://www.ripe.net/ripe/meetings/ripe-58/content/presentations/dnssec-deployment-problems.pdf.

[4] Network path problems in dnssec's deployment. In *IETF 75 - dnsext*, 2009. http://www.ietf.org/proceedings/75/slides/dnsext-3.pdf.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirement. RFC 4033, March 2005.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034, March 2005.

[8] Rohit Khare and Adam Rifkin. Weaving a web of trust. *World Wide Web J.*, 2(3):77–112, 1997.

[9] Matt Larson. [dnssec-deployment] rrsig for arpa expired. *DNSSEC-Deployment mail list*, June 2010. http://dnssec-deployment.org/pipermail/dnssec-deployment/2010-June/004009.html.

[10] Vincent Levigneron. Key Deletion Issues and other DNSSEC stories. *ICANN 41*, June 2011.

[11] E. Osterweil, D. Massey, and L. Zhang. Deploying and Monitoring DNS Security (DNSSEC). In *ACSAC '09*, 2009.

[12] Eric Osterweil. *Measurable Security: a New Substrate for DNSSEC: Chapter 3.1.1 (Trials of the .gov TLD)*. PhD dissertation, UCLA, Department of Computer Science, 2010. http://irl.cs.ucla.edu/~eoster/doc/osterweil_thesis.pdf.

[13] Eric Osterweil, Dan Massey, Batsukh Tsendjav, Beichuan Zhang, and Lixia Zhang. Security through publicity. In *First USENIX Workshop on Hot Topics in Security*, 2006.

[14] Eric Osterweil, Michael Ryan, Dan Massey, and Lixia Zhang. Quantifying the operational status of the dnssec deployment. In *IMC '08*, 2008.

[15] Eric Osterweil and Lixia Zhang. Interadministrative challenges in managing dnskeys. *IEEE Security and Privacy*, 7(5):44–51, 2009.

[16] Gregory Patrick. Re: [dns-operations] Expired DNSSEC signatures in .gov. *dns-operations*, Spetember 2012. http://www.mail-archive.com/dns-operations@lists.dns-oarc.net/msg00661.html.

[17] R.L. Rivest and B. Lampson. SDSI–A simple distributed security infrastructure. Citeseer, 1996.

[18] Dan Wendlandt, David Andersen, and Adrian Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX Annual Technical Conference*, 2008.

[19] Philip R. Zimmermann. *The official PGP user's guide*. MIT Press, Cambridge, MA, USA, 1995.