Vantages: Using "Public Data" for Key Verification

Eric Osterweil
UCLA
eoster@cs.ucla.edu

Dan Massey
Colorado State University
massey@cs.colostate.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

A public key verification system for the global Internet has long been thought of as prerequisite for enhancing Internet security with cryptographic protections. However, after years of efforts by numerous groups, such a facility remains absent in the operational Internet. In this paper, we formally define a novel concept of *Public Data*, and through the design of a system called Vantages we describe how we can leverage the concept of Public Data to develop a public key verification system for the global Internet. More specifically, the Vantages system is designed to solve the DNSSEC key learning problem. As of the writing of this paper, DNSSEC is in the verge of wide deployment and is in desperate need of an operationally realistic key learning system that allows DNS resolvers to obtain and verify public keys known as `DNSKEY`s. We further demonstrate the improvement that Vantages provides over DNSSEC's native key verification by formally quantifying each of them and empirically measuring their effectiveness.

1. INTRODUCTION

Cryptography can provide effective security protections for critical Internet protocols, and most cryptographic system designs require an effective way to learn and verify public keys. Thus, a key verification system has long been thought of as a prerequisite for adding cryptographic protections in the Internet. Unfortunately, such a facility remains absent in today's operational Internet.

Most existing designs use conventional cryptographic system constructs such as a Public Key Infrastructure (PKI) as the basis for key verification, but the lack of success of any such system suggests that there is a fundamental misalignment between hierarchical systems that base their trust model on a universally accepted root and the nature of Internet systems that are composed of a large number of independent administrative organizations. Specifically, [22] details various key verification systems deployed in the Internet today, including SSL [28] and PGP [30], and outlines a few fundamental challenges in managing trusted keys in Internet-scale systems. All the evidence shows that we need to take a fundamentally different approach that embraces the properties of Internet systems as first-class design principles.

In this paper, we first describe the few fundamental properties of Internet-scale systems, and then use the original DNS Security Extensions (DNSSEC) [6, 8, 7] design as a concrete example to explain how these essential Internet properties invalidate designs that take a hierarchical PKI approach. We then introduce a novel concept called *Public Data* and describe how one can verify the validity of Public Data in the absence of prior established cryptographic trust. Based on the concept and properties of Public Data, we design a system called *Vantages* that can provide key verification for DNS resolvers that are using DNSSEC. At the time of this writing, DNSSEC is on the verge of being globally deployed, but faces a major obstacle of a broken hierarchical key verification system as originally designed. We use this difficulty as an opportunity to demonstrate the effectiveness of Vantages on an actual Internet-scale system.

We have built an implementation of Vantages and are running it in our labs as a core service. In order to evaluate its effectiveness, we use DNS resolver traces taken from several universities and emulate large-scale attacks against an emulated Vantages deployment. We then use empirical metrics defined for DNSSEC in a previous work [24] to quantify Vantages' deployment. Finally, we use these metrics to present a side-by-side comparison of Vantages to DNSSEC's original key verification design so as to quantify the improvement provided by Vantages.

The remainder of this paper is organized as follows. Because Vantages directly addresses DNSSEC's key verification, Section 2 provides an overview of DNSSEC's design and explains the mismatch between that design and the essential properties of the global Internet. Section 3 formally introduces the Public Data concept. In Section 4 we describe Vantages as a system and detail relevant points of its implementation. Next, in Section 5 we formally define metrics to assess a Vantages deployment. Based on that, in Section 6 we evaluate Vantages'

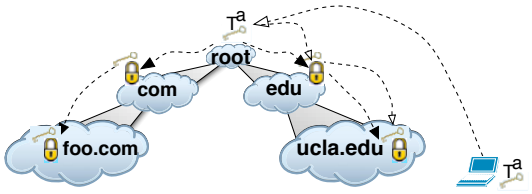


Figure 1: Using the root zone's key as a trust anchor (T^a) resolvers trace a "chain of trust" down the DNSSEC hierarchy to any zone.

performance in face of emulated attackers, and finally we conclude in Section 7.

2. BACKGROUND

2.1 DNSSEC

The Domain Name System (DNS) maps hostnames such as `www.ucla.edu` to IP addresses and provides a wide range of other mapping services ranging from email to geographic location. In this section we introduce a basic set of DNS terminology which is used throughout the text, including resource records (RRs), resource record sets (RRsets), and zones, followed by an overview of the DNS Security Extensions.

Security was not a primary objective when the DNS was designed in mid 80's and a number of well known vulnerabilities have been identified [10, 9]. DNSSEC provides a cryptographic solution to the problem, which seems pretty simple and intuitive. To prove that data in a DNS reply is authentic, each zone creates public/private key pairs and then uses the private portions to sign data. Its public keys are stored in a new type of RR called `DNSKEY`, and all the signatures are stored in another new type of RR called `RRSIG`. In response to a query, an authoritative server returns both the requested data and its associated `RRSIG` RRset. A resolver that has learned 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 `www.ucla.edu`, the resolver needs to construct a *chain of trust* that follows the DNS hierarchy from a trusted root zone key 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 then be used to authenticate the public key of `ucla.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`) RR that corresponds to a `DNSKEY` RR at the child zone, and creates an authentication link from the parent to child. It is the child zone's responsibility to request an update to the `DS` RR 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.

Moreover, the parent and child zones belong to *different* administrative authorities, each may decide independently is and when they turn on DNSSEC. This leads to the second and more problematic challenge. If the parent zone is not signed, there is no 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 K lower in the tree. The original DNSSEC design envisioned that its deployment would be rolled out in a top-down manner. Thus only the root zone's K would need to be configured in all resolvers' T^a sets and all secure delegations would follow the existing DNS hierarchy. However as of this writing, the root zone is not signed, and it is unclear when it will be. Moreover, many of the Top Level Domains remain unsigned. Without the root and top level domains deploying DNSSEC (as is the case today) there could be potentially millions of isolated trust anchors. In fact various approaches have been proposed for securely obtaining these trust anchors.

As a result of this problem, the DNSSEC community has begun investigaing how to augment the DNSSEC secure delegation hierarchy with systems call Trust Anchor Repositories (TARs). These are systems whose designs vary quite widely, but whose goals are to help resolvers learn the public keys (or trust-anchors) for zones without needing the secure delegation hierarchy. Thus, the DNSSEC community has identified the key learning problem in DNSSEC's design as a major problem and it remains an area of intense investigation at the time of this writing. The various approaches proposed have been discussed on the DNSSEC Deployment Initiative [1] mailing lists, and a description of open issues and further comparison and taxonomy of approaches can be found on our website [5].

2.2 Internet-Scale Requirements

In this work we employ the term "Internet-scale" that was defined in previous work [24]. We to mean any system that meets the following four requirements: First, the system must scale to large number of elements. Second, the system must be deployed across multiple independent administrative domains. In other words, different operational entities operate their own part of the system and interact with each other to create the global system. Third, the system must allow local policies. In other words, each administrative domain is able to decide for themself with whom they will interact. Finally, the system must survive and continue to operate in the presence of constant misconfigurations, errors, and attacks.

Thus we consider that even very large systems that are deployed under a single administrative authority are not Internet-scale systems (as defined above) since they do not need to address the complexity and in particular conflict of interests caused by different administrative domains, local polices, and errors. Similarly, systems that have distributed administration but lack large scale or local policies are not Internet-scale by our definition. We only call a system Internet-scale if it has all of the four conditions above.

Examples of prominent Internet-scale systems include the Border Gateway Protocol (BGP) [26] and the Domain Name System (DNS). These systems have sustained the Internet as core protocols for decades and have the above properties. BGP has a large number of links and routers, is deployed across many distinct Autonomous systems, each system sets its own local polices, and the system as whole functions despite the fact that every day there is some router somewhere that suffers from a misconfigurations, error, or attack. Similarly, DNS has vast numbers of resolvers, caches, and servers, is deployed across independent DNS zones, each zone controls polices ranging from dynamic update rules to access rules, and again there is always some cache, resolver, or server that is suffering a misconfiguration, error, or attack.

In contrast, DNSSEC follows a more traditional model of hierarchical key verification. This is similar to a PKI in that all users must begin by trusting a pre-specified root. Next, any break in the delegation chain (due to misconfigurations or disynchrony) breaks the native design's key verification.

3. LEVERAGING PUBLIC DATA

As we argued earlier, cryptographic protections must have some means to validate public keys, yet the traditional PKI designs are infeasible in Internet-scale systems. To deploy cryptographic protections such as DNSSEC and other systems in Internet, we need a way to obtain valid public keys in the absence of a classical PKI that starts with a central authority. Toward this end, we have developed a new approach that uses the concept of *Public Data* as its foundation. The result allows us to extract valid data in the presence of adversaries and in the absence of pre-existing cryptographic systems.

To define this new concept of Public Data and demonstrate its power and usefulness, we define three properties that are required for data to be considered *Public Data* and discuss this in Section 3.1. In Section 3.2, we reason why meeting these 3 properties can give robust validity in the absence of cryptographic verification. Section 3.3 discusses how this might be achieved and leads to our system implementation, called Vantages.

3.1 Public Data

We begin the discussion by considering some examples of possible Public Data. In one example, news posted on the New York Times website can be considered as Public Data because it is accessible by anyone with an Internet connection, and it is indeed accessed by large number of people daily, i.e. it is widely disseminated. But what if a piece of news was posted on the NYT website by a hacker and removed after only a few minutes? Does this still constitute Public Data? Does the persistence of the data make a difference in whether it is considered to be Public Data?

Our first objective is to offer a simple but also rigorous definition of Public Data. Using this definition, we can apply a clear unambiguous rule to declare data as public or not. In addition, a rigorous definition helps us later to evaluate the degree of trustworthiness of Public Data in the validation of critical information.

Definition of Public Data: a piece of data is considered to be Public Data if and only if

- The data is accessible to any and every user who wants to access it;
- The data is persistent; and
- The data approaches a consistent value the number of distinct paths that can used to learn the data approaches infinity.

The first requirement is intended to capture the idea that Public Data is available to the public. Our definition only requires that users in general can access the data. Note this requirement allows us to exclude special cases such as local restrictions and filters that hinder the accessibility by some particular users. It also allows us to exclude transient unavailability due to network failures that can create conditions where some set of users cannot access the data. The fact that some groups of users may be blocked from the data does not change the public property of data itself.

The second requirement captures the notion that once data becomes publicly accessible, it stays public. This

does not preclude posting updates and modification to the data, but the old value remains accessible. The persistency of Public Data also has important implications for nonrepudiation and we will make use of this fact later in both the definition of our system and the Vantages design.

Finally, the third requirement is that as the number of independent paths grows large, a consistent value of the data will emerge. By definition, the value of Public Data converges toward a consistent value if one could observe the data from an infinite number of independent paths.

These three properties define what it means to be *Public Data*. Clearly, not all data will have these properties. However, we will show that data which has these properties can be validated *without* assuming the existence of cryptographic PKI.

3.2 Validating Public Data

The intuition behind validating Public Data can be illustrated with an example. Currently, the "`.se`" ccTLD chooses to post their DNSSEC public key in one of the country's leading newspapers. The paper is accessible to a generic user. The paper persists in archives and for all practical purposes, the data persists. Now suppose the same public key was also posted in not just one newspaper, but in the top 10,000, or even top 10,000,000, different newspapers. One cannot guarantee that no one will publish a newspaper with an invalid copy, but note also that the true owner of the key can also see the value published in the paper, and be able to raise a flag in the case the value is incorrect. As the number of distinct channels reporting the *same* consistent public key approaches a very large number, one's confidence in the validity of the public key also approaches 100%.

We make three observations on the definition of Public Data. First, the three requirements are defined in terms of *system properties*, i.e. the accessibility, persistency, and independent access channels are the properties of the system that hosts the data. We are trying to derive data validity based on these system properties, instead of cryptographic verifications.

Second, as system properties, although the above definition is theoretically precise, it has obvious practical limitations. In particular, persistence over time needs some notion of practical bound in any real system. Continuing the newspaper example, it is unlikely, nor it is useful, to be able to access the copy of the newspaper thousands of years from now. Similarly, there are obviously not an infinite number of papers that could publish the key.

The third observation is derived from the first two: given Public Data is defined by system properties, and system properties have varying degree of conformance and practical limitations, the validity of Public Data may not necessarily be stated as either black or white, or absolutely valid or invalid as a cryptographic system does.

3.3 Systems to Validate Public Data

There may by a number of ways by which Public Data can be validated to various degree. In this work we present one candidate approach to verify Public Data. However, first we need to introduce the notion of an observer of the Public Data. The observer has some location (or more precisely vantage point) within the network and we refer to an observer as *vantage* v_k. The objective is to learn the value of some Public Data item. We assume the data item is available from any source s_i from the set of all sources $s_i \in S$ and the actual value of the data item is d_g.

Next, we consider an adversary *Eve* whose attack model is to trick v_k into believing d_e is the proper value instead of d_g. In order to do this, *Eve* must insert d_e along the network path between the vantage v_k and its chosen data source s_i. If one simply considers a path to the data, the chance of success is entirely dependent on the chance that *Eve* is on the path. Note *Eve* may control some node or link on the path to s_i or may control s_i itself. To exploit the notion of Public Data, the user attempting to learn the data should obtain observations from not just one vantage, but several vantages that they trust (friends). We can see that the set of vantages and aggregate observations of the Public Data items is critical. Thus, we define each user's set of vantages as their Community of Trust. As the number of $v_k \in V$ with non-intersecting paths $\rightarrow \infty$, the number of paths that *Eve* must compromise in order to cause damange without being noticed $\rightarrow \infty$.

The currently deployed Vantages system, discussed in the next Section, takes a very aggressive view of whether the numerous different observations are converging toward a consistent view. In this aggressive version, if any vantage is able to report the correct value d_g it will be seen as conflicting with the d_e value seen by other vantage(s). This formalization follows the logical method of distributed consistency checking that was originally tested and deployed in a well known and widely used Monitoring Tool [4]. One useful side effect of the public nature is that since data is verified through (essentially) taking measurements, the approach lends itself to ease of deployment and quantifiable results, as we will see more of in Section 5.

4. THE VANTAGES SYSTEM

The Vantages system uses the verification properties of Public Data to verify `DNSKEY`s for DNS resolver operators. In addition, Vantages' design embraces the requirements of Internet-scale systems so as to ensure its

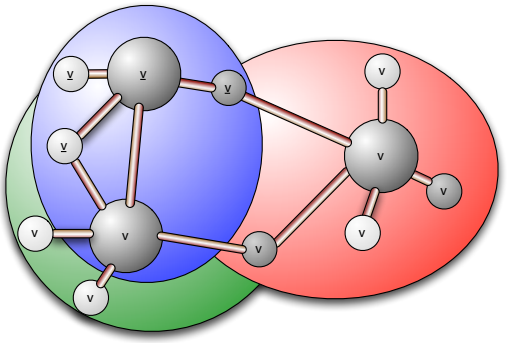


Figure 2: **This figure highlights the overlapping COTs of three Vantages (v) that share some friends with each other.**

operational relevance.

The Community of Trust (COT) concept in the definition of Public Data specifies that a set of vantages that can act as *friends* and cooperate to verify public data. However, in a real system, not all parties will want to communicate with or trust each other. In fact, it is likely that there may be overlapping networks of people interested in trusting each other. A system implementation has to address the reality that trust is not transitive and a user may not trust the friends of their trusted friends (Figure 2). Furthermore, the number of friends describes how many paths can be used to reach data, and in a real system this will clearly not approach infinity.

In addition, DNS resolvers do not necessarily query the same set of DNS zones, and therefore do not all look at the same values. It is unreasonable for resolvers that belong to different authorities prompt each other to perform additional DNS queries. Further, when different resolvers *do* query the same zones, they will not necessarily do so at the same time. Thus, a system must consider issues of data-overlap and the rate at which data may have changed.

Another relevant issue that arises for DNSSEC is that some data owners serve their `DNSKEY`s over multiple protocols. For example, there is a growing practice for zones to serve their keys on web pages in addition to serving them from DNS name servers. This allows data to be checked for consistency across different types of protocols as well as just network vantages.

Finally, it is important for vantages to be sure that data they have received from *each other* was not spoofed. When a vantage considers an observation from another vantage in the COT, it must be certain that the data has not been spoofed. Moreover, to meet the Public Data requirement of nonrepudiation Vantages must ensure that once a Vantage has observed data it must that any other vantage can always prove that observed data existed (even if it is no longer served).

4.1 Design

Vantages meets its design goals with the following basic building blocks: i) each user of the Vantages system runs a daemon that is a single *Vantage* point, ii) COTs are implemented as a set of peer-to-peer Vantages that form overlapping communities of independent *friends*, iii) it uses an expressive verification taxonomy for `DNSKEY`s, iv) it has the ability to take data from a variety of sources and protocols, and v) it has the transparency to show how unprocessed data is converted into usable `DNSKEY`s,

Multiple Vantage Points: After processing data into `DNSKEY`s, Vantage daemons check the *consistency* of the keys with their list of trusted *friends* (remote Vantage daemons). Having multiple views (or *vantages*) of the same `DNSKEY` is the primary mechanism that Vantages uses to do its verification. The result is that having a trusted set of friends at different points in the network fulfills the Public Data requirement of having multiple paths. The actual mechanism to spoof DNS resolvers has been extensively discussed in the literature [10, 9, 23], and is beyond the scope of this paper. The enlistment of community members as pollers is a notable contrast to previous distributed key verification systems [4, 29] in which users must rely on a 3^{rd} party infrastructure. The importance of this contrast is that, unlike other approaches, Vantages' protections do not have the prerequisite of a "trusted" 3^{rd} party. Thus, Vantages' trust model is dictated by its operator's decision of whom she will configure as *friends*.

Vantages is both flexible and easily configurable and its design goal is to let operators use real-world trust when deciding who to configure as *friends*. This is a notable contrast to self-organizing peer-to-peer networks like [14, 27, 20, 21]. The success of these networks comes (in large part) because of their ability to scale and self-organize. However, because of their anonymous nature, the peers in these networks can be arbitrarily malicious to each other. This can result in peers sending erroneous data (called pollution [15]) or even attacking each other [19, 18]. This has not abated their success because users rarely trust these open networks to provide them with important transactions like banking or e-commerce. By contrast, DNS is an integral part of essentially all Internet activity, and allowing anonymous peers to influence security decisions should be a decision left to each operational group, rather than mandated by a system design. Thus, Vantages makes the decision of whom its daemons can become friends a user-specific attribute.

However, this is not to imply that identifying friends is challenging. Many DNS zone operators already do this when following operational best-practices and set-

ting up secondary name servers in remote networks. Additionally, many operators meet each other at operational conferences such as NANOG [2] and RIPE [3]. The operational community is rife with opportunities for operators to aid each other in such a lightweight operation and we anticipate that sites like [4] will act as open Vantages friends for operators who want to use them.

Verification: When querying a friend, Vantages does not expect that the remote daemon will necessarily have previously queried for the `DNSKEY`s in question. In fact, the opposite is often the case, or if the friend does have the data it may be too old to rely on. In these cases, Vantages can still proceed. Stale data can often be recognized by using the existing inception and expiration dates on `DNSKEY`s' signatures.

The definition of Public Data uses the notion that as the number of paths approaches infinity, the consistency reaches its absolute maxima, and data can then be verified. As a real system of friends, Vantages will not approach this number of paths. Therefore the system uses available evidence and a classification scheme called CPUC to classify `DNSKEY`s into one of four exclusive states: The first state is *Confirmed*, and it indicates that no conflicting values for the `DNSKEY` were seen over any protocol or from any friend, and at least a threshold number (n) of friends have also seen the same value for the key. The second state is *Provisional*, and it means that a key was seen by other friends, but the number of friends is less than the configured threshold of n. The third state is *Unknown*, and these are keys which Vantages can provide no protection for because they have not been seen by any friends. Finally, the last state is *Conflict*, and this state is an alert that either a conflicting set of keys were seen over different protocols or from any friend. The main benefit of the CPUC scheme is that it is an evidence-based verification scheme that lets end-users specify their own local policy of which states qualify a key as "verified."

Diverse Protocols: Vantages collects each of its `DNSKEY`s via any combination of DNS queries, HTTP or HTTPS requests. This list of protocols was chosen based on the current DNSSEC operational practices. It has become an operational convention, when constructing a trust-anchors file, for operators to augment their DNS lookups with `DNSKEY`s that are served from zones' web pages (where available). Vantages takes this a step further and automates this process. Thus, each Vantage daemon may either collect its `DNSKEY`s from just a single source, or from multiple sources. In order to allow operators to specify data sources in a general way, data sources are specified via standard URLs [12, 17]. In addition to just Public Data such as DNS and HTTP, users can advertise keys that they *own* via local attestations (specifying a local non-public data source). This

Figure 3: Vantages' dashboard admin page

allows zone owners to specify which `DNSKEY`s they are authorities for so that their friends (who already trust the owners as Vantage-friends) can bootstrap each others' `DNSKEY`s.

Data Provenance: Vantages implements and shares the data provenance of its `DNSKEY`s with friends so that the origin source that friends have learned keys from is transparent to users. This level of transparency allows operators to accept or reject each others' Vantage data based on its original source, and/or scraping. For example, a Vantage operator may decide not to trust anyone's manual attestations or may blacklist certain TARs. This configuration simply uses the provenance of all remote key looks to prune keys from those data sources.

4.2 System Implementation

Vantages is a fully implemented system that is deployed and running in several research labs, and is being evaluated at other operation centers. It is implemented as a community of individual daemons (`vantaged`) that each operational group runs on one of their local servers. These daemons accept data sources that operators want to poll (such as web pages), and allow operators to configure a list of *friends* that they would like their daemon to confer with when verifying keys. The main configuration interface to Vantages is through a web-based configuration dashboard that is served from an embedded web server, which runs on a configurable port, and can optionally be password protected (Figure 3).

The `vantaged` daemon is written in C++, and is open source (available for download from [5]). The data it gathers as well as its meta-data are stored in a local SQLite database. When setting up a Vantage daemon, the operator must specify a PGP [30] key to use as a signing key by the daemon.

Operators can add data sources to be polled by submitting URLs [12, 17] through the administrative web dashboard. In addition, Vantages can be configured to use `libpcap` to automatically learn and poll DNS sources of zones that the host machine sees DNS traffic to. This option is useful when Vantages is deployed on a machine that hosts a recursive resolver.

Configuring `vantaged` with friends involves bootstrapping the local daemon with the public PGP key from a friend's daemon, the HTTP URL that their daemon listens on, and optionally an HTTP password. This

allows `vantaged` to automatically connect to, authenticate with (if a password is specified), and begin verifying data with that friend. The data is returned with PGP signatures, and the friend's PGP key can be used to verify its origin authenticity.

The default behavior for the daemon is to poll all of the configured data sources at fixed intervals (though polling is randomized to avoid pattern detection). The polling behavior has been optimized for speed so that many thousands of DNS zones can be polled in minutes. This allows the daemon to (if desired) maintain a very fresh view of the sources it tracks.

After each polling cycle, the daemon contacts all of its configured friends with the list of zone names it has just queried. Friends then respond with the `DNSKEY` values they have seen, and when the last time they saw them. With the responses from friends, `vantaged` classifies each of keys with its CPUC taxonomy.

4.3 Meeting Requirements

One of the main design goals of Vantages is that it meets the requirements of Internet-scale systems. Here we demonstrate how it: is able to support large-scale deployments, supports multiple independent administrative domains, allows users to manage their own policies, and finally is resilient against errors, misconfigurations, and attacks.

Large-scale: Vantages scales from the perspectives of both data owners and clients. Administrators add data by placing it on a public server such as a web server or a DNS name server. Clients use their own Communities of Trust to verify data without the need for a central authority.

Multiple Independent Administrative Domains: Independent administrative domains are essentially first-class elements of Vantages. Administrators can use their own local policy of what data they would like to publish, and how to so. In addition, clients can cull public from any Public Data sources.

Local Trust Policies: Trust is entirely determined by clients. They use their own view of Public Data and those views of other vantages that they choose to trust (not a mandated authority). In addition, because Public Data is defined on the notion that clients determine the trustworthiness of data for themselves, the *provenance*[1] of where data came from lets its history further contribute to clients' trust assessments.

Misconfigurations: Vantages deals with myriad of misconfigurations, errors, and attacks that Internet-scale systems face by cooperation, consistency, and compartmentalization. Clients join together into COTs and cooperate with each other to form distributed vantage points. Thus, the distributed observation points of Vantages are formed by those that use its data. As a result, an attack that subverts one vantage point's view will be detected when its data's consistency is compared to another vantage. Based on this, an adversary must subvert an entire COT to pull off an successful attack. Furthermore, in the event that an adversary is able to subvert an entire COT, the damage is compartmentalized to only that COT. Other COTs are not affected.

[1]Provenance is a term often used in fields such as art to describe the custody chain of an item. It is, increasingly, gaining mindshare among computer scientists as a way to describe the derivation history of data.

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

Table 1: DNSSEC verification vs validity matrix.

5. DEPLOYMENT QUANTIFICATION

In this Section we propose to use system metrics to compare Vantages' performance against its forerunner, DNSSEC's native key verification. Previous work [24] has demonstrated that DNSSEC's classical cryptographic deployment can be quantified, and by creating metrics to measure the same qualities of a Vantages deployment, we will directly compare these systems to each other.

5.1 Quantifying the DNSSEC Deployment

In [24], the authors identify three general measures to quantify the DNSSEC deployment: *availability* is defined as the network-level operation of DNSSEC, *verifiability* characterizes the cryptographic design and captures how much operational effort is needed in order to enable the deployment to offer its cryptographic protections, and *validity* describes how often the deployment verifies data that is correct (versus verifying data that is no longer valid).

The authors use these general measures to define DNSSEC-specific metrics (on a 0 to 1 scale) and then use those to quantify the actual DNSSEC deployment. The details of, and the motivation for, these equations is beyond the scope of this writing, but we provide a brief summary of them here.

Availability: The availability metric was designed to reflect how difficult it is for DNS resolvers to actually retrieve the `DNSKEY`s from DNS in order to use them.

Verifiabilty: The verifiability metric followed DNSSEC's hierarchical delegation design and formalizes how much additional configuration is needed by resolvers beyond a single key for the DNS root zone.

Validity: Finally, the validity metric that was described captured how frequently DNSSEC avoided giving false negatives and false positives (Table 1).

Based on the values of these metrics, the authors suggest that the DNSSEC deployment can be longitudi-

	Confirm	Provisional	Unknown	Conflict
Valid	Ideal Behavior	User Policy	User Policy	Bad/Safe
Invalid	Bad	User Policy	User Policy	Good

Table 2: Vantages verification vs validity matrix.

nally quantified, and they go on to propose that such a technique should be useful to other types of cryptographic systems.

5.2 Quantifying a Vantages Deployment

We now use these same measures to define metrics for Vantages. Each of the metrics will also produce values on a 0 to 1 scale.

Availability: Vantage daemons run on the same machines as resolvers and generate local trust-anchor files. Therefore, the availability of `DNSKEY`s to resolvers is always 1. Any time data needs to be added or removed, the daemon does it locally, and the resolver never needs to conduct any network transactions.

Verifiability: Vantages' `DNSKEY` verification combines any knowledge of ground truth (such as the keys for one's own DNSSEC zone), with a the ground truth from a user's COT (friends' keys), and Public Data verification. Thus, there are two types of data that must be manually configured in order for Vantages to be able to verify all other secure zones Z^s (where $|Z^s|$ denotes the number of secure DNS zones): local keys K^l (where $|K^l|$ denotes the number of local keys) and the list of friends in the COT: V (where $|V|$ denotes the number of friends in a COT). We, therefore, define the verification metric for Vantages as Equation 1.

$$V^f = 1 - \frac{|V| + |K^l|}{|Z^s|} \quad (1)$$

Validity: The approach that Vantages takes in verifying data is based on building up evidence. A Vantage derives local confidence in a key's veracity from the number of friends and number of different data sources that report to see the same value. Based on this, Vantages uses a user's local policy to control if an observed key is classified as: Confirmed, Provisional, Unknown, or in Conflict (CPUC). Based on these states users may decide to adopt a very liberal policy and accept Unknown and Provisional keys (those for which there is little or Public Data), or a more conservative approach in which only Confirmed keys are used. Table 2 depicts the possible states of data, based on the ground truth of being either valid or invalid.

6. EVALUATION

In this Section we both evaluate Vantages against trace-driven emulation of attackers and quantify it against the current native key verification system used by DNSSEC.

6.1 Threat Model

We define our threat model against Vantages as cases in which a Vantages COT is operating properly, none of its members is purposely serving invalid data, and one or more external adversaries is attempting to insert invalid `DNSKEY`s for one or more zones. This model does not focus on insider attacks, in which one of the members of a Vantages community becomes malicious and serves invalid data to its friends. We leave this to future work.

Next we sub-classify our model into two categories: uncoordinated attacks, an coordinated attacks. Uncoordinated attacks are those in which an arbitrary number of adversaries individually attack a number of resolvers trying to spoof them into accepting invalid data. Examples of this type of attack from the wild include the rampant cache poisoning attack seen in the summer of 2008, dubbed the Kaminsky Attack [13]. In these cases, numerous uncoordinated individuals were able to exercise a newly discovered weakness in DNS caching behavior.

In contrast to this, we define coordinated attacks as those in which a powerful adversary is able to fool a large number of resolvers into using false keys. For example, a powerful enough adversary might serve a specific set of bogus `DNSKEY`s to all of the resolvers in North America.

In either case, the attack vector may vary without changing the nature of the attack. For example, an uncoordinated attack may use cache poisoning (such as the Kaminsky attack), or it may be done by an *on-path adversary* using a Man in the Middle (MitM) attack. For the purposes of this analysis, the attack vector is immaterial because we presume that the attacks succeed against individual resolvers. We focus our evaluation on the overall effect on a Vantage COT.

6.2 Setup

Current measurements of DNSSEC's deployment size [4] suggest that it is still relatively small compared to the hundreds of millions of normal DNS zones. However, it is clearly important to evaluate Vantages' design at the target scale of DNS' global deployment. Thus, actual traces of DNSSEC queries are insufficient, and we instead turn to trace-driven emulation.

In order to emulate Vantages' behavior we need to model four separate degrees of freedom: multiple independent Vantage daemons installed at separate locations (alongside recursive resolvers), independent sets of zones looked up by (and the corresponding overlap in query habits of) separate resolvers, independent rates at which those zones are queried and re-queried, and

attack patterns. However, simulating all of these parameters raises the chances of introducing unintentional experimental bias. To avoid this, we used DNS resolver traces (taken from actual resolvers at several sites) to reissue DNS `A` record queries (the DNS lookup for an IPv4 address) from multiple distinct locations. This allowed us to accurately emulate multiple Vantages, independent sets of zones, actual query rates, and we were even able to use the effects of Content Distribution Networks (CDNs) to emulate attack traffic.

To issue queries from multiple geographically distinct locations, we assigned each trace file to a separate PlanetLab [25] node. For example, if a set of `A` records were logged in one trace file at a specific set of times, our emulation apparatus would re-issue those same queries from a specific PlanetLab node at the same frequency so that we could record the answers seen. This allowed us to emulate which zones a Vantage daemon would likely query, and at what rate. Next, we observed when different resolvers saw different `A` records (and the associated `NS` records that are generally returned with `A` responses), and used the distinct values to represent cases where different Vantages saw different `DNSKEY` values. This type of behavior is called *geo-balancing* and is often used by CDNs to enhance performance of Internet services, even over very small regions. This fine-grained geo-balancing for `A` records serves as an excellent behavior to represent uncoordinated attacks, because it is precisely the way we would expect an adversary to behave in relation to `DNSKEY`s[2].

In addition to geo-balancing `A` records, some CDNs geo-balance name server (or `NS`) record sets as well. When these `NS` sets vary, they tend to be consistent across large regions. For example, querying "`mail.yimg.com`" reveals that Akamai [16] acts as their CDN, and in many locations around the world their `NS` set is "`n0g.akamai.net`," ... "`n8g.akamai.net`." However, in China the `NS` set served differs: "`gtm1.glb.cn2.yahoo.com`" .... We use these cases to represent coordinated attackers who have the ability to spoof entire regions of the Internet, but not the entire World.

6.3 Trace Results

The data used in our evaluation of Vantages came from four separate computer science departments at major Universities. DNS traces were captured at each department's set of recursive resolvers for approximately one month. From these traces we extracted the query time and the domain name being queried.

In order to evaluate Vantages' behavior with larger communities, each of these traces was broken into smaller non-overlapping five day traces. This technique (described in [11]) provided twenty separate sets of DNS queries and timing between the queries. Next, we selected twenty distinct PlanetLab nodes and assigned each node a trace file. Four nodes were in China, nine in Europe, three in Japan, one in Korea, one in the Middle East, two in South America, and five in North America. Starting at a common time, each PlanetLab node replayed the five days worth of queries from its trace, preserving the query order and timing found in the original trace. Each node represents one Vantage daemon in the following analysis.

[2]This would be similar to the large-scale uncoordinated attacks like the Kaminsky cache poisoning attacks [13].

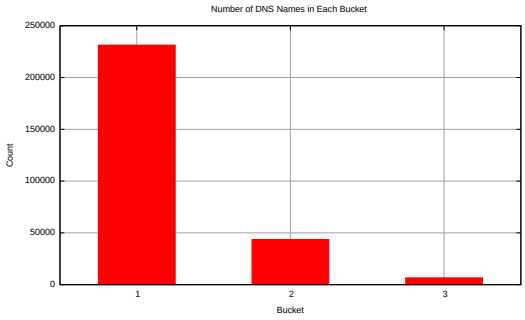
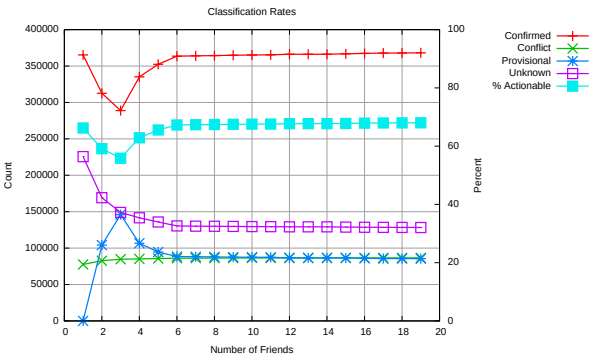


Figure 4: Domain name popularity (bucket) vs. number of names

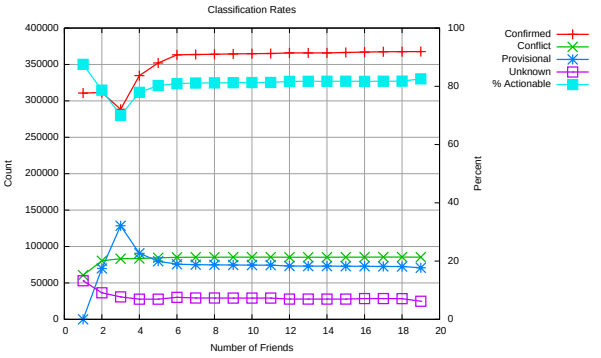
Since Vantages relies on its Community of Trust members to verify each other's query results, it requires some degree of overlap in the zones each Vantage queries. Popular zones will be queried by many Vantage daemons. Unpopular zones may have too few diverse queries to make a strong claim about keys validity. Figure 4 divides the domain names found in our query sets into three buckets. The first bucket has 231,240 domains that were rarely queried. The middle bucket (number two) has 43,620 domains that were queried with moderate frequency. The final bucket has 6,576 domains that were the most frequently queried. We introduce this division in order to demonstrate that Vantages performs better on more popular domains.

Our results focus on the effectiveness as viewed from a single Vantage's daemon (a driving-trace). Due to space restrictions we limit our presentation of the results to one arbitrary driving-trace. This trace represents an "average" example (neither the best nor the worst performer).

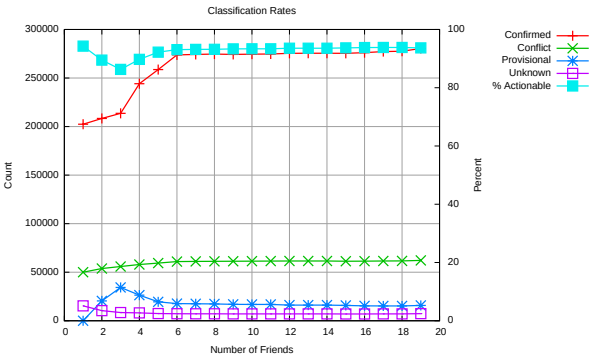
We noted in Section 4.2 that Vantages' CPUC taxonomy classifies keys as confirmed after n friends see consistent values (with no conflicts). In this evaluation we set $n = 3$, and in a case where a COT has fewer than three Vantage daemons, a record is confirmed if all the other daemons saw the same value. In addition we adopt the conservative policy that if a record is either confirmed or in conflict we say that Vantages has declared the record to be *actionable*. When names are provisional or unknown, Vantages does not have enough



(a) Performance with all keys from all buckets.



(b) Performance with keys from buckets two and three.



(c) Performance with keys from bucket three only.

Figure 5: Impact of COT size in uncoordinated attacks.

information to be authoritative about them.

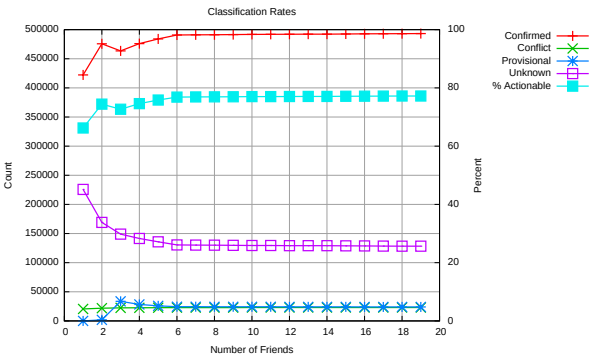
Figure 5 illustrates Vantages' performance under *uncoordinated attacks* as a community grows from one friend to the full list of nineteen. This figure illustrates that the number of other Vantage daemons in a community is initially a critical factor. In the transition from one to three, we can see that the number of confirmed keys and the actionable percentage drops before rising again. This pattern indicates that with an insufficient number of Vantage daemons in the Community of Trust, a Vantage succumbs to attacks and confirms keys that are actually invalid. In fact, this verifies the notion that without enough paths, Vantages cannot use the Public Data concept to properly offer protection. However, the emulation indicates that as the community grows in size and diversity, Public Data becomes effective. After reaching a size of six, Vantages converges and the attacks have all be detected. Even for unpopular zones, Vantages is able to classify roughly 75% and almost 80% of the zones as actionable (Figures 5(a) and 6(a) respectively). Furthermore, as we narrow our focus on the most popular keys (bucket three), the percent of keys Vantages classifies as actionable approaches the high 90s. This is in stark contrast to DNSSEC, which is only able to verify any keys that have a delegation hierarchy leading to them. Previous work [24] has reported that only about 3% of DNSSEC zones have delegation chains leading to them.

It is noteworthy that having six friends is not a special number, and that different attacks and different query patterns may need more or less overlap. Nonetheless, this set of traces show that a Community of Trust with size as small as six can be sufficient.

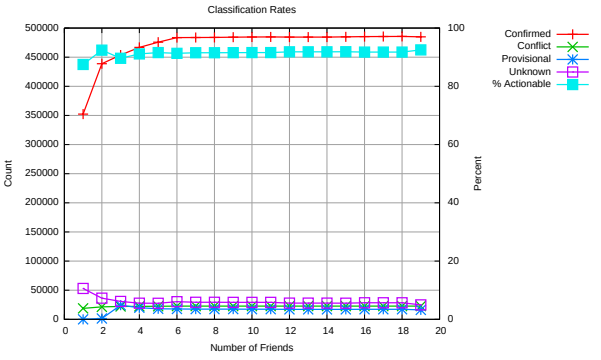
Figure 6 depicts *coordinated attacks* on Vantages and its resulting performance. As under the uncoordinated attacks using the same traces, Vantages converges after six Vantage daemons have been added to the Community of Trust. Because of the way coordinated attacks were modeled, there are far fewer of them than the uncoordinated attacks above, and the figures show that a much greater number of keys are confirmed because of this. However, one can see that Vantages' actionable-rates exceed those from its uncoordinated performance. The reason for this is that while the first few Vantage daemons can be in the same geographic region, adding just one node who is beyond the coordinated attacker's control completely overcomes the attack. Thus, a coordinated attack can be viewed as being more brittle (in some ways) than the uncoordinated maelstrom.

Once again, we can see that Vantages performs better when protecting keys from bucket three. We take from this a broader lesson: those keys that a community uses more often are more vital to protect. Thus, the simple fact that they are in use more creates a greater assurance that Vantages can verify them. By contrast, when a user visits a new zone for the first time and no one has previously queried it, there are no hard assurances that the key returned is valid. However, there is an equally small chance that an adversary will know when to spoof that key, and even if the key gets spoofed, Vantages will only classify it as unknown.

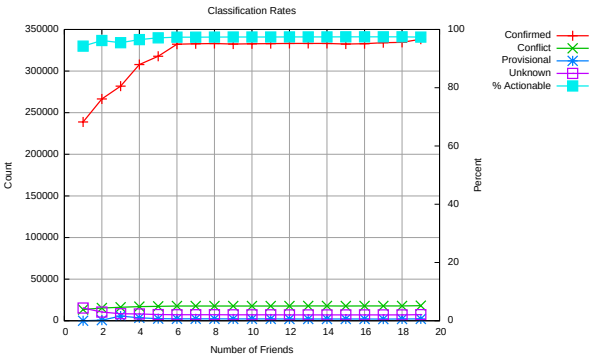
One could speculate that provisional keys should be included as well or more complex policies can be introduced for accepting these keys. Based on Vantages' design, however, this is a local policy decision, and operators can choose for themselves. But given the solid



(a) Performance with all keys from all buckets.



(b) Performance with keys from buckets two and three.



(c) Performance with keys from bucket three only.

Figure 6: **Impact of COT size in coordinated attacks.**

performance of the simple policy, we leave that as future work and this analysis considers them as unactionable.

6.4 Quantification

In the preceding evaluation, Vantages converged after its COT grew to six friends. Due to space limitations, we are unable to present the following analysis over the entire spectrum of friend configurations, and limit our quantification to a COT of size six and for zones from bucket three.

In Section 5 we described three measures (*availability*, *verifiability*, and *validity*) that can be used to comprehensively quantify the deployments of both DNSSEC's native key verification system and Vantages'. Each defined specific metrics to quantify three general measures. In the remainder of this section we present the quantified values of the DNSSEC deployment that were reported in [24], and contrast them to Vantages'.

	Confirm	Provisional	Unknown	Conflict
Valid	0.947	0.0006	0.0001	0.000
Invalid	0.0001	0.000	0.000	0.053

Table 3: **CPUC during coordinated attacks.**

	Confirm	Provisional	Unknown	Conflict
Valid	0.876	0.0031	0.000	0.000
Invalid	0.0001	0.000	0.000	0.12

Table 4: **CPUC during uncoordinated attacks.**

We can see by inspection that Vantages' deployment serves keys to resolvers without the need for network communication. Previous work showed that the availability of DNSSEC's deployment was calculated to be 0.8 on a 0 to 1 scale, and by comparison Vantages' availability is 1.0.

The verifiability metric for Vantages is defined in Equation 1. We note that since both traces involve the same number of zones and the same configurations, the expressions will be the same for both uncoordinated and coordinated attacks. Previous work calculated DNSSEC's verifiability as being $V^f = 0.241$ on a 0 to 1 scale. Calculating the the verifiability metric for Vantages with six friends and one local key to add (assuming the operator runs their own DNSSEC zone): $V^f = 1 - \frac{6+1}{359,573} = 0.99998$.

Finally, we consider the validity metric. Previous work used evidence of misconfigurations to indicate validity. However, in this work we calculate validity based on emulated attacks. Thus, these values do not lend themselves well to direct comparison. Rather, we point out that in DNSSEC's native design, data is narrowly classified as either valid or invalid. In contrast, Vantages uses the CPUC taxonomy to allow clients the flexibility to create their own policies.

In order to evaluate the validity of Vantages, we require a notion of ground-truth (to compare the system's CPUC values against). We approximate that by considering zones whose asymptotic state reaches either "Confirmed" or "Conflict" when evaluated with the full set of nineteen friends as being in a ground-truth state. We then compare the states of those zones when using six friends to represent how well Vantages assessed each zone's validity. What we find from the coordinated and uncoordinated attacks is shown in Tables 3 and 4 (respectively).

7. CONCLUSION

A large number of Internet protocols today utilize public key cryptography protections, yet a global scale public key verification system is still missing, and a fundamental challenge in deploying such a key verification system is the lack of global trust. This paper takes on a brand new approach to solve the problem: developing a key verification system through the use of *Public Data*.

In this work we presented the first formal definition of the concept *Public Data*. We derived this notion from observed common practices in the Internet and formalized the concept by giving it a rigorous definition. Based on the concept of Public Data we crafted the design of a key verification system called Vantages and used it to address the shortcomings of the native hierarchical key verification design in DNSSEC. Vantages can serve as an alternate or a complementary key verification system for the global DNSSEC rollout.

In order to demonstrate Vantages' robustness against attackers, we used trace-driven emulation of two forms of large-scale attacks and, further, we have quantified Vantages' deployment in a way that has allowed us to present a side-by-side comparison of it with DNSSEC's actual deployment.

It is our belief that the concept of Public Data provides a foundation on which brand new solutions to Internet security can be developed that do not rely on centralized authority or prior trust relations.

8. REFERENCES

[1] Dnssec deployment initiative. `http://dnssec-deployment.org/`.

[2] North American Network Operators Group (NANOG). `http://www.nanog.org/`.

[3] RIPE Community. `http://ripe.net/ripe/index.html`.

[4] SecSpider. `http://secspider.cs.ucla.edu/`.

[5] Vantages. `http://secspider.cs.ucla.edu/vantages/`.

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

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

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

[9] D. Atkins and D. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004.

[10] S. M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the Fifth Usenix Unix Security Symposium*, pages 199–208, 1995.

[11] D. Berengut. Statistics for experimenters: Design, innovation, and discovery, (2nd ed.). george e. p. box, j. stuart hunter, and william g. hunter. *The American Statistician*, 60:341–342, November 2006.

[12] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738, IETF, December 1994.

[13] CERT. Cert vulnerability note vu#800113, 2008.

[14] I. Clarke, S. Miller, T. Hong, O. Sandberg, and B. Wiley. Protecting freedom of information online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[15] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The pollution attack in p2p live video streaming: measurement results and defenses. In *P2P-TV '07: Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, pages 323–328, New York, NY, USA, 2007. ACM.

[16] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE INTERNET COMPUTING*, pages 50–58, 2002.

[17] S. Josefsson. Domain Name System Uniform Resource Identifiers. RFC 4501, ISOC, May 2006.

[18] jtcfrost.sourceforge.net. News: Freenet nodes under attack, 2008. `http://jtcfrost.sourceforge.net/news.html`.

[19] A. Kalafut, A. Acharya, and M. Gupta. A study of malware in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 327–332, New York, NY, USA, 2006. ACM.

[20] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network. *Internet Applications, IEEE Workshop on*, 0:112, 2003.

[21] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploring the robustness of BitTorrent peer-to-peer content distribution systems: Research Articles. *Concurrency and Computation: Practice & Experience*, 20(2):179–189, 2008.

[22] E. Osterweil, D. Massey, and L. Zhang. Managing trusted keys in internet-scale systems. In *The First Workshop on Trust and Security in the Future Internet (FIST'09)*, 2009.

[23] E. Osterweil, V. Pappas, D. Massey, and L. Zhang. Zone state revocation for dnssec. In *LSAD '07: Proceedings of ACM Sigcomm Workshop on Large Scale Attack Defenses*, 2007.

[24] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the operational status of the dnssec deployment. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008.

[25] L. Peterson, V. Pai, N. Spring, and A. Bavier. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. Technical Report PDN–05–028, PlanetLab Consortium, June 2005.

[26] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). RFC 1771, IETF, March 1995.

[27] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella Network. *IEEE INTERNET COMPUTING*, pages 50–57, 2002.

[28] A. C. Weaver. Secure sockets layer. *Computer*, 39(4):88–90, 2006.

[29] D. Wendlandt, D. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2008.

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