Managing Trusted Keys in Internet-Scale Systems

Eric Osterweil
UCLA
eoster@cs.ucla.edu

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

Lixia Zhang
UCLA
lixia@cs.ucla.edu

1 Introduction

In order to meet security objectives, Internet applications are increasingly reliant on public key cryptography. For example, Web sites often use the Secure Sockets Layer (SSL) [14] or its successor TLS [7] to encrypt sensitive content, Pretty Good Privacy (PGP) [16] uses public key cryptography to digitally sign and verify email conversations, and the DNS Security Extensions (DNSSEC) [3, 5, 4] use public key cryptography to authenticate DNS lookups. We anticipate that this list of applications will continue to grow over time.

This paper considers a fundamental problem facing all applications that rely on public key cryptography: how to obtain, verify, and maintain the set of trusted public keys. Without some notion of where to find the right keys or which keys can be trusted, an adversary can create false keys, trick users into accepting the false keys, and impersonate legitimate online parties. For example an adversary could forge the public key for Amazon.com and steal customers credit card information. Furthermore, a valid key can also be revoked at a later time, thus the trusted key set needs continuous maintenance. Unfortunately, the problem of obtaining and managing trusted keys is made challenging by Internet's lack of a central authority: Generally speaking, there is no single point of authority to issue lists of trusted PGP keys, or SSL certificates, or DNSSEC keys for the entire Internet.

In this work we explore a new direction to address the challenge of distributing and maintaining trusted keys. First, we note many of today's more successful Internet cryptographic systems (such as the SSL and PGP) use various forms of *locally trusted* key lists, but managing this local list encounters the following 3 basic questions: 1) Where can one learn the needed keys? 2) How can one verify the trustworthiness of the keys, and 3) How does one maintain this trusted key lists to keep them updated? To find a systematic solution to these questions, we also observe that there exists a set of *existing* practices that people already use when making trust decisions based on *public* data. Examples include the way in which DNS resolvers learn the IP addresses of the DNS root name servers, the way Border Gateway Protocol (BGP) routers learn the global routing table, a way in which DNS resolvers learn of `DNSKEY`s, and even a way in which clients learn Secure Shell (ssh) keys. Conceptualizing these existing practices, we present a model called *Public-Space*, in which publicly available data can be used as self-verifying. By shifting the onus of verifying data's authenticity away from a central authority and onto the Public-Space, users can judge veracity for themselves within a structured framework.

2 Existing Key Management

Pretty Good Privacy (PGP): In PGP [16] 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. 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*.

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 [9] 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 verifications, encryptions, 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.

Secure Sockets Layer (SSL): SSL [14] is used by many protocols including HTTPS [11]. SSL uses X.509 [6] certificates for its cryptographic underpinnings. It allows

clients to verify *newly* discovered certificates by using preconfigured trusted (or *root*) certificates as authorities (CAs). The idea behind this system is that ultimate trust *can* be bestowed by root certificates (in contrast to PGP where there is no ultimate authority). However, SSL's design does not include any mechanism for securely distributing its root certificates to clients. SSL deployment gets around this problem by having vendors such as Microsoft or Firefox preconfigure root certificates into client applications. The presumption is that clients will trust the root certificates provided to them by software vendors.

Vendors that bundle SSL root certificates use informal conventions for deciding which certificates to include. For example, a web browser may come bundled with root certificates from organizations such as VeriSign [12]. Furthermore, when an SSL client finds a certificate whose signatory does not exist in the local list of roots, SSL often lets the user decide what to do (e.g. Web users have likely encountered a browser popping message asking whether to accept a certificate). The use of a local trusted key list allows individual users to add to and subtract from their local set of roots. For example, an organization can create its own root certificate and use it to sign locally generated certificates that are served by its website, mail server, etc. After users manually add the new root certificate, locally signed certificates can be accepted by browsers without paying the cost of obtaining a certificate signed by one of the well known roots. However, just as with PGP, SSL has no operational provisions for managing the keys in its trusted keys lists.

Even if root certificates *could* be revoked, the SSL protocol has no provisions for distributing new certificates. Rather, it relies on the initial maintainers (software vendors) to distribute new keys through software patches or new releases. This effectively means that a certificate's lifetime is loosely coupled to when it can be disseminated to clients. In spite of the vast number of deployed SSL clients, its means of managing its cryptographic keys can be characterized at best as being ad hoc, and at worst as problematic.

Though X.509 has provisions for Certificate Revocation Lists (CRLs), they are not implemented in SSL clients. One known approach to certificate revocation is the Server-based Certificate Validation Protocol (SCVP) [**?**]. The details of SCVP are beyond the scope of this writing. However, for completeness we observe one simple mismatch between this protocol and the Internet's distributed authority structure. SCVP follows a traditional PKI notion that its certificate policies should be managed centrally rather than implemented at the consumer endpoints. Perhaps the most critical ramification of this is that users must be able to learn of the proper authority for a certificate revocation chain to avoid being spoofed into using an attacker's server as an authority by mistake.

DNS Security Extensions: The DNS Security Extensions (DNSSEC) provide a cryptographic solution to the DNS spoofing problem. To prove that data in a DNS reply is authentic, each zone creates public/private key pairs and uses the private portions to sign data. Its public keys are stored in `DNSKEY` record, and all the signatures are stored in `RRSIG` records. In response to a query, an authoritative server returns both the requested data and its associated `RRSIG`(s). A resolver that has learned the `DNSKEY` of the requested zone can then verify the *origin authenticity* and *integrity* of the reply.

In order to authenticate the `DNSKEY` for a given zone, resolvers need to construct a *chain of trust* that starts from a locally configured trusted key which then recursively delegates to other keys following down the DNS hierarchy. For example, a resolver looking for the `DNSKEY` for `ucla.edu` would begin by querying the root zone (`.`), and then be referred to `.edu`, and then finally to `ucla.edu`. The design of DNSSEC envisioned that resolvers would only need to configure the public key of the DNS root zone as a trusted key, which can then be used to recursively authenticate zones in any branch of the DNS.

Generally speaking, parent and child zones in the DNS tree belong to *different* administrative authorities, each may decide independently whether and when they turn on DNSSEC. If a parent zone has not deployed DNSSEC, there is no chain of trust leading to its child zones' `DNSKEY`s. This orphaned key effectively becomes isolated in the DNS hierarchy. To verify the data in these isolated DNSSEC zones, one has to obtain the keys offline in a secure manner and store locally. Unfortunately, DNSSEC's design does not include provisions for securely obtaining `DNSKEY`s for local trusted key lists. Thus, DNSSEC also does not have provisions for maintaining its local trusted key lists as they evolve.

One notable approach that was not discussed above is the classical notion of a Public Key Infrastructure (PKI). The main reason for the omission is the absence of globally deployed PKIs in practice. We speculate that this absence is not accidental. In addition to the great dynamism of the Internet and the presence of constant misconfigurations and changes (which complicate but do not make PKI designs impossible in the Internet), the Internet is composed of a enormous number of independent and autonomous administrative domains. These domains cannot be compelled to agree upon any single agency to act as the root of all trust, as is required by traditional PKI designs. This simple freedom gates the deployment of any global PKI. Perhaps the closest examples of PKI-like designs in the Internet are the Secure Sockets Layer and The DNS Security Extensions.

3 The Public-Space

In many cases, if some data is made widely available, one can reasonably expect that inconsistencies, inaccuracies, or other problems in the data would result in publicized

objections and/or corrections. Examples of this behavior in daily life include the legal requirement for people that want to change their name to publicize this in a newspaper before it becomes official, so that people remain accountable for their deeds under their old names even after switching to new ones. The public record is an audit trail.

Perhaps more interestingly, the DNS top-level domain: `.se` has recently begun running newspaper adds that publicly announces its cryptographic key. This announcement serves to notify everyone of what value they should trust in the DNS resolvers. Clearly this concept and its usage is not limited to daily life, but also exists in cyberspace. We believe we are the first to formally recognize this use of information publicity and to suggest ways in which principles can be derived from existing approaches to solidify them into a coherent and formal approach: the *Public-Space* [10].

3.1 Internet-Space

Many of the Internet's core technologies already use elements of the Public-Space, and we note these, and some emerging systems that capitalize on some of its concepts.

Currently, DNS resolvers can learn the mapping of any domain name (such as `www.ucla.edu`) to data (such as its IP address). This process is done by starting for the DNS root servers and then recursively learning and querying until the target name is found. However, the DNS root servers must be known ahead of time. The mechanism by which this is done is that a file (often called `root.hints`) is disseminated to all DNS resolvers by: software vendors, open source packages, operating system distributions, etc. The contents of this file are generally considered trustworthy because they are expected to be globally consistent. Rarely does it change and when it does it is widely publicized. Any questions about whether one has the proper contents are immediately answerable by polling websites, friends, etc. Essentially, the contents are too well-known to fake.

Two well known results [2, 13] and [8] use historical routing data to help defend BGP routes against hijacks. These approaches revolve around the notion that if false data gets injected into BGP, it will be corrected within a "short" period of time. The result is that valid data becomes far more prevalent than invalid data when looking at BGP history.

Regional Internet Registries (RIRs) announce newly allocated Autonomous System Numbers (ASNs) over email lists so that broad knowledge of which agencies own which numbers can be used to authenticate what operators may see. Thus, whenever anyone wonders (perhaps in the face of conflicting reports) who owns which ASNs, operators can refer back to the public record of the announcement. For example, IANA (one of the RIRs) announces its allocations over its email list, signs the message with its PGP key, and includes a URL where the registry can be found: `http://www.michnet.net/mail.archives/nanog/msg16203.html`

Another example of self-verifying public data is they way in which DNSSEC resolvers can learn and maintain a list of verified `DNSKEY`s. Currently, SecSpider [1] tracks the global rollout of DNSSEC and polls each zone for its `DNSKEY` records from a set of distributed DNS monitoring sites (or pollers). The results from the pollers are compared against each other so that global consistency can be verified. This means an adversary must spoof all pollers at the precise time that they query a zone in order to subvert SecSpider. All globally consistent keys are then entered into a *trust-anchors* file that resolver operators can download and configure into their resolvers whenever it changes. The public view of DNSSEC zones acts as self-verification of DNSSEC data.

As mentioned above, some organizations use PGP keys to certify the authenticity and integrity of their announcements. However, just as mentioned in Section 2, learning PGP keys is not always straightforward. Many of these organizations such as APNIC, RouteViews, RIPE, and SecSpider (just to name a few) post their PGP keys on their webpages. This act serves as public notice of what their keys are. The persistence of these keys on these websites can be attested to by search engines and casual queries, but provides evidence of which are the proper keys for an organization.

Finally, a new project called ssh-Perspectives [15] uses pollers similar to its predecessor SecSpider, but designed to pull ssh keys. However, rather than running as a global census, ssh-Perspectives polls ssh servers on-demand and returns to the querier the instantaneous values seen. This public view of ssh keys also acts as self-verification of ssh data.

3.2 Using the Public-Space

Beyond recognizing the trend of applications increasingly using public data for verification, we also propose that the Public-Space concept must be formalized. We propose that to formally participate in the Public-Space, systems must incorporate a few important elements with their "public data:" i) a notion of time; the presence of data must include when it was seen how long it has been visible. This requirement allows the presence of data to be rebutted. ii) a notion of space; we define public data as being widely visible from many spatially separate locations. This allows public-data to be visible to any party that may want to rebut its validity. iii) a notion of non-repudiation; we require that public data must offer some form of non-repudiation. However, we note that the traditional definition of non-repudiation evokes notions of cryptography and the provable veracity of data. We soften this requirement slightly. The Public-Space allows one to prove that data existed in public. The veracity of data is always left to the end-user to determine.

These requirements offer convenient facilities when de-

signing Internet-scale trusted key learning and maintenance systems. However, the Internet's size, heterogeneous administrative authorities, and rampant misconfigurations and dynamism (i.e. constant and often frequent changes) make managing public keys quite complex. Never the less, the Public-Space is an ideal concept on which to build a key management system for the Internet. The propensity of successful cryptographic Internet systems to manage their trusted key list locally, and the preference of users to specify their own local view of trustworthiness support designs that emphasize a user's ability to understand how her keys are managed and to customize the list of keys that she does and doesn't trust.

The Public-Space fully embraces this approach. Keys that exist in the Public-Space can have their veracity derived from their presence. This has the further benefit that keys can be periodically regathered by clients. When key owners need to change their keys, they simply do so in the Public-Space and all clients can pull the updates at their own rate. Furthermore, when key owners need to revoke their keys, they can announce this in the Public-Space so that all clients can use this as a public record. Moreover, whenever a client wants to use their local preference to override something in the Public-Space, the key list is manged locally and can be amended in any way the operator wants.

It is important to note that any system attempting to act as a Public-Space must create a notion of global visibility. Systems like ad-hoc peer to peer systems in which clients query untrusted peers for data may create attack vectors in which adversaries can fool clients with large bot-nets, or by pre-publishing keys for other users before those users have the opportunity to refute them. In contrast, in a Public-Space anyone can attest to data or refute it. Thus, we feel that clients may face issues of spam in the Public-Space, when adversaries publish erroneous data, they will still be able to observe genuine data owner's submissions. Thus, the challenge will evolve into discerning signals from noise.

4 Challenges and Open Issues

This paper argues that the formalization of the Public-Space presents a new and promising direction to address the public key learning, authentication, and maintenance challenges in the Internet. At the same time, it also brings a number of open issues.

As we discussed earlier in the paper, none of the applications using public key cryptography actually has a secure and scalable means to revoke and reissue keys to a potentially large number of users. We believe the Public-Space offers the channel for this key management task, and we have developed one specific form of solution for DNSSEC key lookup. However providing a general solution for all other application remains an open issue.

The second issue is a problem introduced by the exact nature of the Public-Space: putting information in public can potentially lead to exposure of privacy. For example, if party A shows the DNSSEC keys it has obtained, it implicitly reveals the DNS domains it has visited. We believe solutions exist that limit the exposure during information sharing, while still allowing Communities of Trust to reinforce each other.

The third challenge concerns potential attacks against the Public-Space itself. Miscreants may try to break down the systems that are built on the Public-Space by disrupting its operations through DDoS attacks, impersonating trusted parties, or simply compromising user machines. In developing effective protections, we see two factors in our favor. First is the Public-Space itself; anything altered can be observed. Second is the scale, different from hierarchical PKIs where a single failure at the root can lead to cascading compromises, here we use *Communities of Trust* instead of a single party so that an adversary must fool a whole community, rather than a single party, before he could cause a damage.

References

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

[2] Protecting the bgp routes to top level dns servers. In *NANOG 25*, 2002. `http://www.nanog.org/meetings/nanog25/presentations/massey.ppt`.

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

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

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

[6] D. Chadwick. The X.509 Privilege Management Infrastructure. In *Security and Privacy in Advanced Networking Technologies*. 2004.

[7] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[8] J. Karlin, S. Forrest, and J. Rexford. Pretty good bgp: Improving bgp by cautiously adopting routes. In *ICNP '06: 2006 IEEE International Conference on Network Protocols*, 2006.

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

[10] E. Osterweil, D. Massey, B. Tsendjav, B. Zhang, and L. Zhang. Security through publicity. In *First USENIX Workshop on Hot Topics in Security*, 2006.

[11] E. Rescorla. Http over tls. RFC 2818, RTFM Inc., May 2000.

[12] VeriSign. Licensing verisign certificates securing multiple web server and domain configurations, 2005. `http://www.verisign.com/static/001496.pdf`.

[13] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Protecting bgp routes to top level dns servers. In *ICDCS '03: International Conference on Distributed Computing Systems*, 2003.

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

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

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