Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing

Dan Wendlandt David G. Andersen Adrian Perrig
Carnegie Mellon University

Abstract

Widespread use of "Trust-on-first-use" (Tofu) host authentication, most commonly associated with protocols like SSH and SSL with self-signed certificates, demonstrates significant demand for a host authentication mechanism that is low-cost and easy to deploy. While Tofu applications are a clear improvement compared to completely insecure protocols, they can leave users vulnerable to even simple network attacks. Our system, PERSPECTIVES, thwarts such attacks using a network overlay that observes a server's public key via multiple network vantage points (detecting localized attacks) and keeps a record of the server's key over time (recognizing short-lived attacks). Clients that receive an unauthenticated key can contact this overlay and check the key against these records, detecting many common attacks. The PERSPECTIVES design explores a promising part of the host authentication design space: Tofu applications gain significant attack robustness while retaining the basic ease-of-use that makes Trust-on-first-use so popular. We present a full network overlay and client design, analyze the security provided by the system, and describe our experience building and deploying a publicly available implementation.

1 Introduction

Networked applications need secure communication channels capable of providing source authenticity, secrecy, and replay protection. Decades of research have shown that a client and server can establish a secure channel if they either share a secret value or know an authentic public value belonging to each other. In the absence of such protections, **Adversary-in-the-Middle (AitM)** attacks are possible. For example, the Diffie-Hellman key establishment protocol is vulnerable to an AitM attack unless the two parties have a mechanism to authenticate each other's public values. This problem remains an important challenge today, particularly as public wireless networks increasingly provide trivial opportunities to mount AitM attacks.

In this paper, we examine a novel approach to client authentication of a server's public key. Traditional approaches, such as a public-key infrastructure (PKI) [9, 5, 7], rely on trusted entities (e.g., Verisign) that grant credentials based on the validation of real-world identities. When done securely, such verification requires significant (often manual) effort. While a small number of hosts, primarily commercial websites, can afford to employ such a heavyweight authentication solution, clients have no simple and effective means to authenticate connectivity to the majority of other hosts on the Internet.

The approach used by protocols like SSH [28] and SSL/TLS [13, 8] with self-signed certificates is a simple alternative to heavyweight authentication, and serves as a starting point for our design. The *SSH model* for host authentication does not require the server to provide credentials from a PKI, but instead relies on the client's discretion to decide if an unauthenticated key is valid. Keys deemed valid by the client are cached locally and used to authenticate subsequent communication with the same server. The rapid adoption of SSH to replace completely insecure protocols like telnet underscores the importance of a simple and cheap alternative to PKI-based host authentication.

Within the general SSH model, clients can use different approaches to determine key validity. For example, while some users may verify all new server public keys in a secure manner (e.g., by memorizing a key fingerprint) a more common approach is for users to *assume the absence of an adversary on the initial connection and accept the initial key without verification.* We refer to the latter approach as **Trust-on-first-use (Tofu)** authentication. Tofu, sometimes also called "Leap-of-faith authentication", is one instance of the general SSH model.

While Tofu applications offer significant security benefits over completely insecure protocols, the approach has two primary weaknesses:

1. By accepting any key on the initial connection, the user creates a simple attack vector that can be exploited by any adversary either on the path between the user and the server or on a shared wireless LAN.
2. On subsequent connections, the user must still manually verify the correctness of any new key that conflicts with a cached key. A user who assumes that such keys are valid (e.g., legitimate server key changes) has no protection against AitM attacks.

These weaknesses in the Trust-on-first-use approach are particularly severe in the case of websites using self-signed SSL certificates, because clients tend to visit a large number of sites. As a result, clients have a large number of vulnerable initial connections, and the users often lack the means or expertise to manually verify keys or key conflicts.

Our system, PERSPECTIVES, improves on basic Tofu authentication by providing client applications with additional information regarding the validity of an unauthenticated public key. Applications can use this data to distinguish between attacks and legitimate keys, thereby reducing the vulnerability of clients when initially connecting to a server and helping to distinguish between a server key change and an attack when a key conflict exists. Because PERSPECTIVES generates this additional information using automated network probes, it is another example of the lightweight SSH model: server owners do not need to contact a certificate authority or even modify their software at all. Thus, *applications using* PERSPECTIVES *enjoy the same simple PKI-free deployment model that has made SSH so popular*.

PERSPECTIVES uses a semi-trusted overlay of Internet hosts that we refer to as *network notary* servers. Notary servers are located in diverse Internet locations and each node periodically probes a large set of server hosts in order to record the public key that the server is currently using for a service like SSH or HTTPS. When a client connects to a service and receives an unauthenticated key, the client also contacts several network notaries for additional information about the service's public keys. Each notary server supplies the client with a cryptographically signed (i.e., "notarized") statement listing all keys the notary has observed from the service in question. This list of observations from diverse network vantage points over a longer time span helps clients make strictly better security decisions: clients are often able to detect attacks during an initial connection or a key cache conflict, the two scenarios when the standard Tofu applications are most vulnerable.

While our network notary infrastructure adds some complexity to a Trust-on-first-use application, it exists independent of both clients and servers. Even if servers are unaware of it, updated clients benefit from the infrastructure. Furthermore, we design the notary infrastructure with redundancy in mind, meaning that notary servers do not need to be completely trusted.

While this paper focuses on authenticating unsigned keys (i.e., the SSH Model), PERSPECTIVES also helps protect users who access websites that use PKI-signed certificates. Because users frequently ignore browser security warnings [24, 11], an attacker can launch an AitM attack by injecting a bogus self-signed certificate in the place of a PKI-signed certificate. However, as discussed in Section 8 the user's browser can easily detect this attack by comparing the received certificate with those seen by the notary overlay.

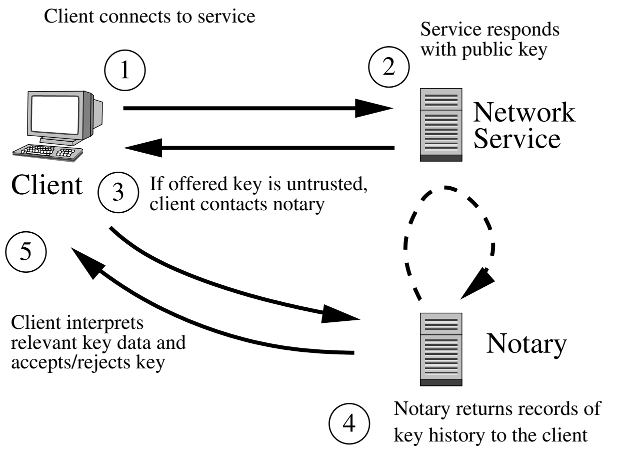
Client connects to service
1
Service responds with public key
2
Network Service
Client
3
If offered key is untrusted, client contacts notary
5
Client interprets relevant key data and accepts/rejects key
Notary
4
Notary returns records of key history to the client

Figure 1: Overview of a client using PERSPECTIVES. In practice, several notaries would be contacted in parallel before making a key trust decision.

This paper makes four primary contributions:

1. We present the design of a modular network notary system that can monitor multiple types of network services and tolerate internal failures and compromises (Sections 3 and 5).
2. We offer a client policy framework that aggregates multiple notary replies to determine if a key is trustworthy, potentially without any user interaction (Section 4).
3. We analyze our system's ability to resist a variety of network attacks within a realistic threat model, demonstrating that PERSPECTIVES can prevent the most common attacks against Tofu applications (Section 6).
4. We benchmark a robust notary server implementation, and describe our modified OpenSSH and Mozilla Firefox clients that use data from notary servers to make key trust decisions (Section 7).

2 Overview of PERSPECTIVES

We name our system PERSPECTIVES because it helps clients make sound security decisions by leveraging views from multiple network perspectives. PERSPECTIVES' task is to help clients determine whether they should accept an untrusted public key received while connecting to a particular *network service*. Example services include SSH access to an end-host or HTTPS access to a website that uses a self-signed SSL certificate.

PERSPECTIVES uses a set of publicly available servers, called *network notaries*, that monitor and record the history of public keys used by a network service. A notary cryptographically signs (i.e., notarizes) statements saying that at time t it observed service S using public key K. The basic operation of PERSPECTIVES is shown in Figure 1. When a client connects to a network service, it receives an *offered key* in reply. If the offered key does not match the client cache, the client must either accept the offered key, taking a security risk, or reject the key, losing the ability to communicate with the service. To obtain more information to make this decision, the client contacts a set of notary servers and requests all *observed key data* for that service. The client then uses application-specific *key-trust policies*, discussed in detail in Section 4, to interpret this data and accept or reject the key. These policies check for consistency between the offered key and the keys seen by each notary, often allowing clients to distinguish between a legitimate key and an attack.

2.1 Threat Model & Attack Resistance

Attackers mount AitM attacks by providing clients with a false public key in order to stealthily intercept or modify network communications. In our attack model, an adversary can compromise any path in the network as well as components of the notary overlay itself. Only the client and the private key of the network service must be trusted, a standard requirement for host authentication schemes.

While our model allows any network or notary component to be compromised, we borrow from Abraham Lincoln and assume that an attacker "can fool all of the [components] some of the time, and some of the [components] all of the time, but it cannot fool all of the [components] all of the time." That is, we assume that attacks are *either*: (1) localized to a particular network scope (e.g., a single network path or an insecure wireless LAN); or (2) of limited duration, since a larger attack is more easily detected and remedied.

With this attack model, PERSPECTIVES does not protect against all possible network attackers, meaning that services with stringent security requirements (e.g., large e-commerce websites) will likely still use traditional authentication mechanisms like a PKI. However, we believe that PERSPECTIVES' approach based on automated network probing represents an important (and relatively unexplored) portion of the overall host authentication design space that focuses on improving the attack resistance of services that currently either use Trust-on-first-use or are completely unauthenticated.

As a result, our primary goal in designing PERSPECTIVES is to *substantially increase the difficulty of an AitM attack without requiring the cost and complexity of a traditional PKI*. Throughout the paper we describe the added protections available to applications that use PERSPECTIVES in terms of "redundancy". *Spatial redundancy*, key observations gathered from multiple vantage points, means that even if an adversary has compromised the client-server path (and potentially other network elements as well), enough notaries will see the valid key so that clients can detect that an attack is underway. *Temporal redundancy*, provided by the key history data returned by each notary, can offer additional protection because even if an attacker compromises all paths to the server, clients can still detect that a recent key change occurred and regard the new key with suspicion. Finally, *data redundancy* (described in Section 5) helps clients detect compromised or malicious notaries that supply inconsistent information, thereby limiting the effectiveness of attacks on the notary infrastructure itself. Sections 4 and 6 explore in greater detail how the attack resistance achieved a PERSPECTIVES application depends on trade-offs made by the client's key-trust policies.

3 The Notary Architecture

We now explore how the notary system provides spatial and temporal redundancy to help clients evaluate an untrusted public key. We defer some implementation details until Section 7. *Notary servers* are hosts distributed across the Internet in a coordinated overlay. *Notary clients* are integrated into applications (e.g., an SSH client) and contact notary servers to learn more about the keys used by a particular service. Application-specific policies in the notary client interpret the resulting observed key data in order to accept or deny an unauthenticated key.

3.1 Notary Administration

We envision a network notary overlay to be a fixed set of ten or more servers in diverse network locations. The overlay may be run by a single entity, but we design it in a decentralized fashion to also support a cooperatively built overlay in which universities, ISPs, hosting providers, or other well-known organizations each contribute one or two nodes. Several existing projects have successfully used such a decentralized deployment model, including the Tor [10] anonymizing network, the large collection of publicly available traceroute and looking-glass servers [27], and the NTP Pool's large set of publicly available network time sources[19]. Cooperative Internet testbeds like PlanetLab [21] and RON [2] are another attractive deployment option. For simplicity we describe a single network notary, but in practice multiple notary systems controlled by different entities could operate in parallel.

Each notary overlay is organized by a *notary authority*, an entity that acts as the authoritative source on what machines are part of the notary overlay. The notary authority has a public/private key pair and publishes its public key ($K_{AUTHORITY}$) using an out-of-band mechanism (e.g., as with Tor, the key could be distributed with any software that accesses the notary system). To add a notary server X to the system, X's owner generates

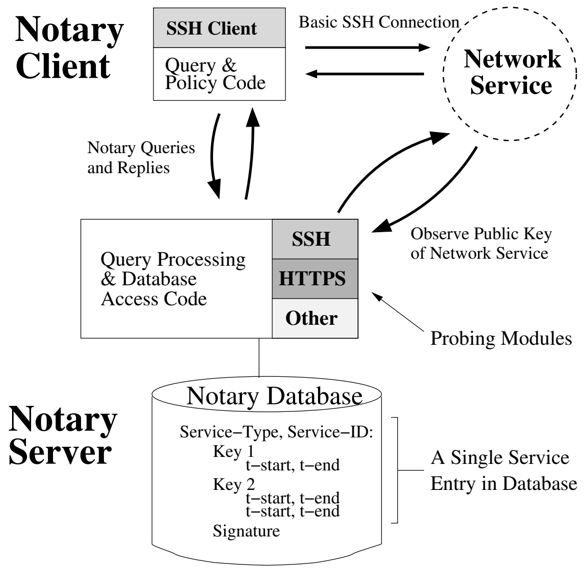

Figure 2: Schematic of a notary server and SSH client.

a public/private key pair and furnishes the public key K_X and its IP address A_X to the notary authority. Each day the notary authority publishes a list of notary server IP address (A_i) and public key (K_i) pairs to each of the notary servers in the systems, along with a signature $S = \{Date, (A_1, K_{A_1}), (A_2, K_{A_2}), \ldots, (A_n, K_{A_n})\}_{K^{-1}_{Authority}}$. A client can contact any notary server, download the list, and validate this signature using $K_{Authority}$ to receive a fresh list of notaries IP addresses and their public keys.

3.2 Notary Server Key Monitoring

Notary servers monitor the public key(s) used by a network service over time. In this section we describe how a notary collects, stores, and shares this observed key data. A notary server provides clients with an application-independent query interface, but uses application-aware *probing modules* to monitor different types of services (e.g., SSH or HTTPS). Probing modules observe keys by connecting to the service and mimicking the behavior of an ordinary client until it receives the server's public key, at which point it disconnects.

Each notary server uses a local database to store a *service entry* for each monitored service. A service entry contains all observed key data the notary has recorded while monitoring that service over time (see Figure 2). An entry is uniquely identified by the combination of a *service-type*, which identifies the protocol used to retrieve the key (e.g., HTTPS)[1] and a *service-id*, which provides the information necessary to contact the service (e.g., hostname and port). The observation history of each key is stored as one or more *key timespans*. A key timespan is a start and end timestamp pair (t_{start}, t_{end}) that indicates a period of time during which the notary observed only that key for the service. When the notary makes a new observation it updates the corresponding service entry in the following manner: if the observed key is the same key observed during the previous observation, the notary simply updates the t_{end} value of the most recent key timespan to the current time. Otherwise, if the key has changed since the last observation, the notary first updates the t_{end} value of the most recent to the current time minus one unit. The notary then creates a new key timespan with both t_{start} and t_{end} set to the current time and adds this timespan and the new key (if necessary) to the service entry. If a probe fails to receive a key from the service, the notary creates or updates a timespan with a "null key" containing no data.

[1] A single logical protocol may have multiple service-types (e.g., an SSH2 server can have both RSA and DSA keys).

Notary Protocol:

$C:$	$S =$ (service-id, service-type)
$C \rightarrow N:$	S
$N:$	$O, \{O\}_{K_N^{-1}} = find\_service\_entry(S)$
$N \rightarrow C:$	$O, \{O\}_{K_N^{-1}}$

Figure 3: The basic notary protocol between a single client (C) and the notary server (N).

The notary also stores a cryptographic signature for each service entry in the database. Using its private key, the notary server calculates a signature over all data in the service entry: the (service-id, service-type), as well as each key and its associated key timespans. This signature is updated following each modification of the service entry. Section 7) measures the overhead of this simple signature scheme and mentions potential optimizations.

3.3 Querying Notary Servers

A client application using PERSPECTIVES contacts the notary whenever it receives a key from a service that does not match an existing entry in its cache. This may occur because the client has never contacted the service or because the received key does not match the already cached key for the service.

When contacting an individual notary (Figure 3), the client specifies a (service-type, service-id) pair. The notary server finds the corresponding entry in its database and replies with observed key data consisting of keys and their associated timespan(s), along with a signature over that data using the server's private key.

The process by which a client queries for and receives notary data is described in Figure 4. Recall that a notary client learns about all notary servers and their public keys using a list distributed by the notary authority. The client's key-trust policy (Section 4) determines N, the number of

```
**Check-Unauthenticated-Key**($s$, $K_{offered}$)
  $O = \{\} //$all observations for service $s$
  All-Notaries= $load\_notary\_addresses()$
  Chosen-Notaries= $choose\_random(N$,All-Notaries$)$
  foreach $x$ in Chosen-Notaries in parallel
      $(O_x, \{O_x\}_{K_x^{-1}}) = contact\_notary(x, s)$
      $K_x = load\_notary\_key(x)$
      $if(verify\_signature(O_x, \{O_x\}_{K_x^{-1}}, K_x))$
          $O = O \cup O_x$
  $if($ ! $check\_trust\_policy(K_{offered}, O))$
      $abort\_connection()$
```

Figure 4: How a client checks an unauthenticated key K_s for service s. Details of $check\_trust\_policy(K_s, O)$ are discussed in Section 4.

notaries that the client should contact. The client then randomly chooses N entries from the list of notaries and queries these servers in parallel using simple UDP datagrams.[2] The querying process is complete once enough notaries have replied for the client policy to make a trust decision, or when the client determines that all remaining notaries are unreachable (clients are responsible for implementing a simple retransmission strategy). The client validates the signature for each response using that notary's public key, discarding any invalid responses.

4 Notary Client Key-Trust Policies

Once a client has received observed key data from notary servers, it must implement a *key-trust policy* to accept or reject an offered key based on this data. Code making the client policy decision will take the offered key and all valid observed key data as input, but may also take into consideration other inputs, such as previously cached keys for the service, user security preferences, or even active user input. Upon completion, the client application either accepts the key and continues running the protocol or rejects the key and disconnects.

4.1 The Security vs. Availability Trade-off

The SSH model (which includes both Trust-on-first-use and PERSPECTIVES) presents clients making a key-trust decision with a basic security vs. availability trade-off: faced with an untrusted key, the client can take a security risk and accept the key, or be safe and reject the key, at the cost of making the service (at least temporarily) unavailable.[3] For example, when faced with a new key the Trust-on-first-use approach makes two different security vs. availability trade-offs: First, in the case of the initial connection, Tofu completely favors availability at the cost of security by always accepting the offered key. Second, in the case of an offered key that conflicts with a cached key, a Tofu application must reject the new key, completely favoring security over availability (as noted earlier, automatically accepting a conflicting key would void all AitM resistance for a Tofu application).

Lacking useful information about the key's validity, Tofu applications are stuck making this inflexible security vs. availability trade-off. In contrast, because PERSPECTIVES provides additional data indicating whether a key is likely to be an attack, it allows application key-trust policies to make significantly more intelligent security vs. availability trade-offs.

In this section we explore several variations on how applications might implement client policies. We do not advocate that these are optimal policy choices (in fact, significantly more nuanced and complex policies exist), but rather offer them as evidence that even simple policies can support a wide range of security vs. availability trade-offs.

4.2 Quorum: A Key-Trust Primitive

Recall that PERSPECTIVES provides security by allowing clients to leverage spatial redundancy (key observations from multiple vantage points) and temporal redundancy (key observations over time). Therefore, the role of a client policy is to test the spatial and temporal consistency of the offered key against notary data. To provide a framework for reasoning about spatial and temporal consistency, we introduce threshold parameters that quantitatively represent these properties.

Definition: For a set of N notary servers, a service S, and a threshold Q ($0 \leq Q \leq N$) we say that a key K has *quorum* at time t iff at least Q of the N notaries report that K is the key for S at time t.

Intuitively, for values of Q that are large relative to N, a key that has quorum indicates consistency among the observations made by the entire notary overlay at a single point in time. We use another threshold parameter to extend the concept of quorum into the temporal realm.

Definition: For a set of N notary servers, a service S, and a quorum threshold of Q, a key K has a *quorum duration*

[2] Using the implementation parameters described in Section 7, a 1460 byte MTU-sized UDP datagram can hold a notary reply with 44 key/timespan pairs. If necessary, multiple UDP packets are used for large replies to avoid IP fragmentation.

[3] It may be possible for a client that sacrifices availability by rejecting a key to later verify that key out-of-band and reconnect. Because this is true for any authentication in the SSH Model (i.e., both Tofu and PERSPECTIVES), the distinction is not central to the paper and for the sake the brevity we refer to this scenario simply as a loss of availability.

of D at time t iff for all t' such that $t \leq t' \leq t+D$ the key K had quorum with threshold Q at time t'.

Quorum duration gives a measure of how long, without interruption, a notary overlay has consistently seen a particularly key.

Applications can make security vs. availability trade-offs by choosing to accept an untrusted key only if it exceeds a particular quorum duration threshold. Higher Q and D thresholds provide more security, but risk compromising availability by accidentally rejecting valid keys. For example, setting Q equal to the total number of notaries N provides the strongest protection against accepting a false key, but it also means that a single unavailable or compromised notary would cause the client to reject a valid key. Similarly, a larger quorum duration threshold D provides security even against a strong attacker that compromises many paths for a significant amount of time, but would also require clients to reject connections to services that are new or recently changed keys.

Next, we consider two examples of how the concepts of quorum and quorum duration might be used by a PERSPECTIVES client policy.

4.3 Power User Example

We first consider a client application policy targeted for a "power user" who understands the risk of an AitM attack and is familiar with the SSH authentication model. Here, the policy code uses quorum and quorum duration values to provide concise messages that help the user make a better security vs. availability trade-off when deciding to accept or reject the key.

Similar to common use of a Tofu client, we apply a different policy depending on whether the key conflicts with an existing cache entry:

Case 1: No Server Key Cached
In this case, the user is not necessarily suspicious of the new key, but she will use observed key data to confirm that the key is consistent across many notaries and that the duration of the key history is commensurate with her expectation of key age. The policy module may supply a message like:

```
This key has been seen consistently for the
past Z days.
```

If the key fails to achieve quorum or does not have sufficient quorum duration based on policy parameters, the user might see one of the following warnings:

```
SUSPECTED ATTACK: The offered key is not
consistent.  Only X of Y notaries currently
see it.
```

```
WARNING: Server key has only been seen
consistently for the past Z days.
```

Case 2: Offered Key Differs from Cached Key
In this case, the user must distinguish between a legitimate server key change and a falsely injected key. As in Case 1, the user is of course interested in the prevalence and duration of the offered key. A key conflict in which the new offered key has established quorum duration may help the user recognize that the new key is likely the result of a legitimate server key change:

```
Offered key conflicts with cached key, but has
been consistently seen for Z days.
```

Additionally, because the policy has access to the cached key that is currently trusted for the server, the policy might also highlight portions of the key history that cast suspicion on the new key. For example, a warning might indicate that some notaries are still currently observing the cached key, suggesting that the actual server key has not been changed:

```
LIKELY ATTACK: Offered key conflicts with
cached key and cached key is still observed
by X of Y notaries!
```

Interpreting such statements require little additional work on the part of the user but provides her with vastly more information than warnings in existing Tofu applications like SSH. On the rare occasion that such a summary is insufficient, power users may view all observed key data, similar to how web browsers allow users to examine SSL certificates.

4.4 Automated Key-Trust Policy Example

It is critical that non-expert users also benefit from using PERSPECTIVES. For these users, we examine a client policy example that accepts or rejects keys without requiring the user to make a key-trust decision. We expect that automated policies will be particularly beneficial in the case of self-signed HTTPS, because average web users often make bad security decisions [24, 11] and because automation could eliminate the intrusive but frequent security dialogs that may make website owners hesitant to use self-signed certificates at all.[4]

Automated policies use quorum and quorum duration thresholds as the sole factor determining key-trust. For example, a simple high availability automated policy might

[4]Automated policies also let PERSPECTIVES apply to protocols that operate without human interaction, such as SMTP mail relay.

```
Simple-Policy($K_{offered}$, $O$, security-level):
  if(security-level == HIGH-SECURITY)
    $Q$ = 90% of notaries
    $D$ = 7 days
  else
    $Q$ = 75% of notaries
    $D$ = 2 days
  t = current_time()
  if(key_quorum_duration($K_{offered}$,O,$t$,$Q$) $\geq$ $D$)
    ACCEPT
  else
    REJECT
```

Figure 5: This policy maps a high-level security preference to quorum and quorum duration values that automate the decision to accept/reject a key.

accept a key as long as at least three-quarters of the notary servers currently see that same key (i.e., $Q = 0.75N$ and $D > 0$). In more complex policies, the value of these threshold parameters could be set according to high-level security setting chosen by the user, similar to security settings in modern web browsers. Figure 5 demonstrates how user-level goals such as "high security" or "medium security" might be translated into threshold values to support automated key-trust decisions.

Similar to the power user scenario described above, automated policies can also distinguish between the initial connection to a server and the more suspicious scenario of a key conflict, for example, by using higher Q and D thresholds in the case of a key conflict.

5 Detecting Malicious Notaries

The final aspect of the notary design that we discuss is *data redundancy*, a notary cross-validation mechanism that limits the power of a compromised or otherwise malicious notary server.

To implement data redundancy each notary server acts as a *shadow server* for several other notaries. As described below, a shadow server stores an immutable record of each observation made by another notary. Whenever a client receives a query reply from a notary, the client also checks with one or more of that notary's shadow servers to make sure that the notary reply is consistent with the history stored by the shadow server.

5.1 Benefits of Data Redundancy

An adversary in control of a notary server and its corresponding private key and can provide clients with false observed key data. Similar to forward-secure signatures [3], data redundancy prevents a notary from changing data already recorded in its observation history. As a result, an attacker that compromises a notary cannot, for example, create a new malicious key and falsely claim that this key has been stably seen over a long period of time. This cross-validation guarantees that PERSPECTIVES' assumption supporting temporal redundancy (i.e., that an adversary can make a malicious key appear "stable" only by sustaining a network attack for a commensurate amount of time) holds true even with respect to notary compromises. Additionally, data redundancy guarantees that a notary, even after compromise, cannot give conflicting answers to two clients querying about the same service. This property (which cannot be achieved using forward-secure signatures) can let hosts monitor the notary overlay to help detect network attacks and compromised notaries (See Section 8).

5.2 Cross-Validation Protocol

Each entry in the list published by the notary authority also lists MAX-REDUNDANCY other servers that each act as a shadow server for the notary specified in the entry. A notary server is responsible for keeping all of its shadow servers up-to-date. When a notary server contacts a service S, it updates its local database and then sends the new service entry (including signature) to each of its shadows.[5]

Shadow servers update their shadow copies in a way that prevents malicious notaries from eliminating previously shadowed data. To do so, the shadow server requires that each key timespan in the old shadow copy either exists in the new shadow copy or is contained within a larger timespan in the new copy (i.e., both t_{start} values are identical, but the new t_{end} is more recent than the old one). Additionally, no timespans in the new copy can overlap. If the old and new data are consistent, the old data is discarded. If an inconsistency exists, the shadow server stores both sets of observed key data and signatures.

Client policy chooses R ($R \leq$ MAX-REDUNDANCY), the number of shadow servers that must corroborate a notary's observation for it to be deemed valid. For each notary n the client contacts, it randomly selects R_q ($R_q \geq R$) of n's shadow servers to query. When contacting a shadow server (Figure 6), the client specifies the IP address of n as well as the service-id from the original query to n. The shadow server replies with a service entry (observed key data and signature) created by n. If fewer than R of n's shadows provide valid responses signed by K_n, the client disregards all data from n. After verifying the signatures, any inconsistencies among the original and shadowed data will cause the client to reject data from n. Additionally,

[5]Client consistency checks are specifically designed so that a malicious notary cannot deceive clients by failing to keep its shadows up to date.

Cross-Validation Protocol:

$C \rightarrow SH:$	n, s
$SH:$	$DB_n = get\_replica\_databse(n)$
$SH:$	$O_n, \{O_n\}_{K_n^{-1}} = find\_service\_entry(DB_n, s)$
$SH \rightarrow C:$	$O_n, \{O_n\}_{K_n^{-1}}$

Figure 6: A client (C) contacting shadow server (SH) for a shadow copy of notary *n*'s observed key data for service *s*.

because of non-repudiation, clients can provide any inconsistent data and signatures to the notary authority as evidence of a notary compromise.

6 PERSPECTIVES Security Analysis

To demonstrate the benefits of network notaries, in this section we enumerate realistic attack scenarios and compare the security provided by PERSPECTIVES to that offered by basic Trust-on-first-use. We analyze both an adversary's ability to launch an adversary-in-the-middle (AitM) attack, as well as its ability to deny availability (DoS) by preventing a client from reaching the legitimate service.

Our analysis considers three possible system components which an adversary may control. Enumerating each combination of these possible assets lets us analyze all scenarios relevant to the attack resistance of PERSPECTIVES:

$\mathbf{L_{client}}$:	A compromise of the client's local link places the adversary "on path" for all client-to-service and client-to-notary communication.
$\mathbf{L_{server}}$:	A compromise of the server's local link lets an attacker inject arbitrary keys when either clients or notaries contact the server.
$\mathbf{k \cdot N_M}$:	A compromise of *k* distinct notary servers.

Note that because an adversary controlling *k* notary-to-service paths is a weaker version of the $\mathbf{k \cdot N_M}$ attacker, we do not examine it separately. $\mathbf{L_{server}}$ covers the case when an attacker controls most or all of the notary-to-service paths.

As described in Section 4, the actual security and availability that a client will receive depends on its choice for the following policy parameters:

N:	Number of notary servers contacted by the client.
Q:	Quorum threshold of client.
R:	Number of shadow servers (per notary) the client requires for data redundancy.

We do not directly model the length of at attack or a client's quorum duration. Instead, we use the concept of "temporal safety", which means that clients will be safe as long as their quorum duration threshold is larger than the actual duration of the attack.[6] Also, since control over the client or server access link ($\mathbf{L_{client}}$ or $\mathbf{L_{server}}$) allows the attacker to mount a trivial DoS attack in either scenario, we do not explicitly mention this attack below. Table 1 summarizes the results.

$\mathbf{L_{client}}$ **Compromise:** When the client's access link is compromised, Tofu provides no defense against an AitM attack, while data from network notaries allows a client to easily detect and avoid the same attack. This protection provided by PERSPECTIVES is important, because client link attacks are easily performed, for example, on a shared wireless network.

It is important to recognize that the attacker gains no AitM advantage by using $\mathbf{L_{client}}$ to disrupt client-to-notary communication. Clients verify the signatures on all observed key data, meaning that notary responses cannot be spoofed by an attacker. Furthermore, adversaries do not benefit by making notaries or shadow servers unreachable, since blocking such communication will prevent any key (including a malicious one) from being reported by enough notaries and shadows to achieve quorum. Blocking client access to notaries could result in a legitimate key being rejected (i.e., a DoS attack), but $\mathbf{L_{client}}$ already allows the attacker a trivial vector with which to deny availability.

$\mathbf{L_{server}}$ **Compromise:** A server link compromise also renders a basic Tofu client vulnerable to AitM attacks. For PERSPECTIVES, the compromise of all paths to the server will prevent spatial diversity alone from detecting an attack. However, historical key data provides a client with temporal safety against network attacks.

$\mathbf{k \cdot N_M}$ **Compromise:** The compromise of notaries alone will not enable an adversary to inject a false key to the client and launch an AitM attack. However, this adversary can attack availability by trying to cause a client to incorrectly reject a valid key. Disabling k > N - Q notary servers, either by compromising them or by making them unreachable, prevents the client from establishing a quorum even if the remaining servers all agree on the valid public key. Because this attack is possible even if the adversary is not on the client-to-service path, it represents an availability vulnerability that does not exist with the Tofu approach. However, this attack is limited to scenarios when clients receive a new key for a service; it does not apply to repeated connections between a client and a server using a cached key.

$\mathbf{L_{client} + L_{server}}$ **Compromise:** The analysis of this

[6]Additionally, to simplify our analysis, we assume a sufficiently large number of notary servers such that no overlap exists among the Q notary servers and the $Q \cdot R$ shadow servers used by a client. A notary overlay that is too small to satisfy this assumption would reduce the number of compromised notaries needed to undermine data redundancy.

	Tofu		PERSPECTIVES	
Compromise	**DoS**	**AitM**	**DoS**	**AitM**
$\mathbf{L_{client}}$	✘	✘	✘	**safe**
$\mathbf{L_{server}}$	✘	✘	✘	**temporal safe**
$\mathbf{k \cdot N_M}$	**safe**	**safe**	k > (N - Q) : ✘ k ≤ (N - Q) : **safe**	**safe**
$\mathbf{L_{server} + L_{client}}$	✘	✘	✘	**temporal safe**
$\mathbf{L_{client} + k \cdot N_M}$	✘	✘	✘	$k \geq (Q + Q \cdot R)$: ✘ $k \geq Q$: **temporal safe** $k < Q$: **safe**
$\mathbf{L_{server} + k \cdot N_M}$	✘	✘	✘	$k \geq (Q + Q \cdot R)$: ✘ $k < (Q + Q \cdot R)$: **temporal safe**
$\mathbf{L_{server} + L_{client} + k \cdot N_M}$	✘	✘	✘	$k \geq (Q + Q \cdot R)$: ✘ $k < (Q + Q \cdot R)$: **temporal safe**

Table 1: Summary of attack resistance provided by PERSPECTIVES in comparison to the standard Tofu approach. The left column contains abbreviated attack descriptions as defined in the text. Columns show resistance to availability (DoS) and AitM attacks. ✘ indicates no resistance, "safe" indicates that attacks are detected, and "temporal safe" indicates temporal safety, as defined below.

scenario is identical to $\mathbf{L_{server}}$, since, as discussed above, using $\mathbf{L_{client}}$ to restrict client access to notary servers provides no attack benefits.

$\mathbf{L_{client} + k \cdot N_M}$ **Compromise:** Control over the client link and some notary nodes lets the attacker use notaries to "promote" an invalid key using false observations. An attacker cannot perform an AitM attack unless it compromises a full quorum Q of notaries, since the client rejects keys that do not achieve quorum. If an attacker compromises Q notary servers, the situation is identical to the $\mathbf{L_{server}}$ scenario described above: the client is still protected for a time period determined by its quorum-duration. However, if the adversary compromises an additional $Q \cdot R$ notaries beyond the basic quorum, it can overcome the data redundancy of the system. Without data redundancy the attacker can forge the observation history, eliminating the protections of temporal safety. [7]

$\mathbf{L_{server} + k \cdot N_M}$ **Compromise**: This attack is stronger than the previous $\mathbf{L_{client} + k \cdot N_M}$ scenario, since control over the destination service's link means that even legitimate notaries will observe the attacker's key. As a result, even if fewer than Q notaries are compromised, the client relies entirely on temporal safety.

$\mathbf{L_{server} + L_{client} + k \cdot N_M}$ **Compromise:** This scenario is identical to the previous attack. As described in the $\mathbf{L_{server} + L_{client}}$ case, client link access grants no additional power if an adversary already has server link access.

[7] We note that this is a worst-case analysis. If the client selects notaries randomly and the total number of available notaries is larger than $(N + N \cdot R)$, the attacker cannot easily predict which notaries or shadows it must compromise in order to mislead the client.

This attack analysis demonstrates that PERSPECTIVES significantly improves resistance to AitM attacks compared to Trust-on-first-use. Additionally, PERSPECTIVES is robust to limited compromises of the notary infrastructure itself.

7 Experience with Notary Server and Client Implementations

To demonstrate the viability of a notary system and to gain experience with its deployment, we have implemented and are running a publicly available network notary on the RON testbed [23]. Additionally, we have created two different PERSPECTIVES clients: a modified version of the OpenSSH [20] client for SSH and an extension to the Mozilla Firefox [18] browser for use with HTTPS certificates. We have made both server and client source code publicly available at: `http://www.cs.cmu.edu/~dwendlan/perspectives/`. Our performance measurements indicate that a single notary server can monitor several million hosts a day while simultaneously handling client queries.

7.1 Notary Server Implementation

Our notary server code is written in C and uses the Berkeley DB library for storing observed key data and signatures. The notary probes services running either SSL or SSH using probing modules based on code from OpenSSL and OpenSSH. The only substantial difference between our implementation and the design described in Section 3 is

Operation	Operations / Sec	
	S_{Slow}	S_{Fast}
Monitor service key	26	195
Monitor service key (no sig.)	112	270
Handle query (in memory)	21,700	25,600
Handle query (disk-bound)	114	-

Table 2: Summary of notary server benchmarks for machines S_{Slow} and S_{Fast}.

that we do not implement data redundancy, in part because our notary deployment is run by a single trusted entity.

We benchmarked our notary server with respect to service monitoring and query response operations to demonstrate that network notaries are practical on commonplace hardware. In our implementation, notaries use 1369 bit RSA keys for service entry signatures[8] and store public keys as 128 bit MD5 fingerprints (while MD5 collision resistance has been compromised, the security of public key fingerprints depends instead on second pre-image collision resistance, which is still considered secure for MD5).

We run our each benchmark on two different machines. One is a modern server (S_{Fast}) with two dual-core 2 GHz AMD Opteron CPUs and 8 GB of RAM. The other server (S_{Slow}) is a three year old machine with a single-core P4 2.4GHz CPU and 512 MB of RAM, intended to demonstrate that a modest notary infrastructure could still be run on older (perhaps donated) hardware. A summary of the results is provided in Table 2.

Monitoring Load: To benchmark notary monitoring, we identified a set of *.com* domains running HTTPS using a web-crawl and had our notary server monitor these sites. (Our SSH scans exhibited nearly identical rates and are therefore omitted). When monitoring a service, the notary must perform the protocol negotiation necessary to retrieve the service's key, load the existing service entry from the database, update the entry, recompute the entry's signature, and store the entry again. Our code uses OpenSSL for RSA signatures and takes the simple (albeit heavyweight) approach of forking a process running a slightly modified OpenSSH or OpenSSL client for each probe. S_{Fast} and S_{Slow} performs 195 and 26 such operations per second respectively. While these rates may seem small, even at only 50% utilization they correspond to 8.4 million and 1.1 million probes per day. Table 2 also shows monitoring rates when signatures are not computed, indicating that optimizing the cryptographic processing by batching service records or creating a hash-tree over multiple service entries would significantly improve performance. Additionally, if future notary overlays require high-performance key monitoring, our experience suggests that software designed specifically for parallelized key scanning would provide a dramatic performance boost over the our existing approach.[9]

Query Handling Load: The primary consideration for query processing performance is whether the notary's databases fit in memory (recall that a notary may have multiple databases due to shadowing). With the cryptographic parameters described above, a single service entry with several keys and timespans will consume approximately 250 bytes. Thus, a database with one million entries (250 MB) can easily fit even in the small memory of S_{Slow}. Table 2 shows benchmarks for both servers responding to randomized client queries (measured over the loop-back interface). If the database is in memory, the response rate is above 20,000 requests/sec for both servers (because our server code is single-threaded, there is little difference between S_{Fast} and S_{Slow} for this test). To identify likely response rates when the database must be accessed from disk, we tested S_{Slow} with a database that was four times the size of its main memory. In this case, the speed of a disk-seek limits the response rate to 114 queries/second. However, even this query-processing rate (which translates to just under 10 million per day) may be viable when clients contact notaries only on rare cache misses.[10]

7.2 Notary Client Implementations

To demonstrate the general nature of the PERSPECTIVES approach, we created two client implementations that access our notary overlay. Both clients share common library code that reads notary configurations, creates and parses protocol messages and implements basic quorum duration policies. As a result, we expect that porting other clients to work with PERSPECTIVES will not be difficult.

OpenSSH Client: We modified the popular OpenSSH [20] client software to contact notary servers when no cached key exists or when the offered key differs from the cached key. The modifications consisted of a few library calls within the SSH code that checks the key cache, as well as functionality to interact with the user when making a key-trust decision (our client roughly follows the "power user" example described in Section 4). We have used the modified client within our university for over half of a year and have found the added latency of

[8] Notary signatures do not need long-term security, as signatures are recomputed frequently and public keys are easily updated. 1369 bit RSA is deemed secure through 2010[15].

[9] OpenSSH's *key-scan* utility provides such functionality and served as our initial SSH scanner, but it exhibited bugs that caused us to discontinue its use.

[10] This disk-bound query-handling process does not significantly contend with the CPU-bound service monitoring, meaning that their respective rates are unlikely to diminish significantly when both are run simultaneously.

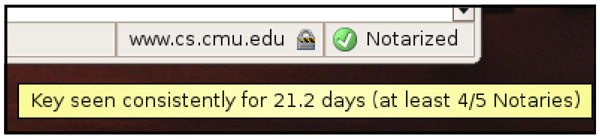

Figure 7: This screen capture shows the bottom right-hand corner of a Firefox browser window, which contains the HTTPS notary extension's visual indicator. In this case, the website satisfied quorum duration, so the browser automatically accepted the certificate and suppressed the usual certificate pop-up usually displayed to the user. The image also shows the tool-tip that summarizes the notary data for the user.

contacting notary servers to be negligible.

Mozilla Firefox HTTPS Client: We implemented our notary client for HTTPS within the Mozilla Firefox extension framework, meaning that users can easily install it without having to download a new browser. The HTTPS notary extension (Figure 7) is written in a combination of C++ and javascript. Whenever a certificate validation fails, the notary client extracts the public key from the certificate and contacts the notary overlay using the host and port specified in the URL bar.[11]

Our implementation allows users to set a preference for "High Security" or "Medium Security", which correspond to different quorum duration thresholds. Keys that achieve quorum duration are accepted automatically, while other keys result in disconnection with an error message. The "High Availability" security setting gives power users the ability to connect even when a key does not achieve quorum duration. These users are provided with the certificate and a summary of notary data in order to make their own key-trust decision.

Because use of our OpenSSH and Firefox clients is currently limited to the Linux and FreeBSD/Mac OS X platforms, we also provide a web page that allows users to query notary servers even without a modified client.

7.3 Generating a List of Services to Monitor

Section 3 assumes a priori knowledge of what services the notary server should periodically monitor. We now consider two possibilities for how notaries might build this list.

The first approach, which we use in our implementation, is for notaries to add a service to its database the first time a client queries for it. While this approach does not initially provide clients with temporal redundancy, when bootstrapping the system a notaries might probe "on-demand" to provide even the first client with spatial redundancy that is capable of detecting $\mathbf{L_{client}}$ attacks. A client could seed the notary with services (e.g., from the `known_hosts` file in SSH) and a server could register for monitoring simply by querying a notary and specifying its own address as the service-id.

The second approach is to proactively discover services. For example, HTTPS websites can be found using web crawling or search engines and SMTP servers can be identified using MX queries for any domain. For less public services, such as SSH, TCP-layer scanning on standard protocol ports could discover a large number of services. We built one such scanning engine to evaluate this approach on SSH servers within our campus network and a few large public IP blocks. We found that while scanning can quickly identify tens of thousands of SSH servers, it has three key limitations. First, if reverse DNS is unavailable for an IP address, we cannot reference a service entry in terms of the DNS name. Second, such scanning misses services running on non-standard ports. Finally, scanning may be misinterpreted as an attack.

Because active discovery cannot identify all services, one promising approach is to seed notaries with data from active discovery, and then identify additional services as clients query for them.

7.4 Notary Overlay Parameters

We now outline the primary considerations for when choosing a notary overlay's global parameters.

Number of Notaries: Because AitM attacks are most likely at the network edge (i.e., $\mathbf{L_{client}}$ or $\mathbf{L_{server}}$), increasing spatial redundancy is likely to have diminishing returns: in the case of $\mathbf{L_{client}}$, even a few valid notaries will detect an attack, while for $\mathbf{L_{server}}$, all notaries will see the same false key. We therefore suggest that clients query from 4 to 10 notaries, depending on their desired robustness to notary compromises or failures.

Notary Monitoring Frequency: A notary would like to minimize the time between when a server changes its public key and when the notary server observes this change. However, the scalability of probing depends directly on the number of hosts being monitored. Our performance evaluation indicates that with modern hardware, a notary monitoring several million different servers could probe each one several times a day. A slightly modified notary design could allow clients to request "on-demand" probes for a service. However, the requirement that the notary cryptographically sign the results of each probe would substantially reduce the peak query response rate of the server.

Degree of Data Redundancy: The degree of data redundancy required for a notary overlay depends greatly on

[11]Interestingly, because the notary extension treats the certificate simply as a public key value, it also handles other certificate validation errors, including expired certificates, and certificates containing hostnames that do not match the hostname specified by the user.

who administers individual notary nodes. If nodes are run by one or a few trusted entities that take great care to secure the machines, little or no data redundancy may be needed. However, even when data redundancy is needed, MAX-REDUNDANCY can likely be small (2 to 4), because clients can detect inconsistencies if any one of the contacted shadow servers are not compromised.

8 Discussion

Notaries and DNS Attacks

So far, we have focused on adversaries that compromise IP-level paths or notary servers. Adversaries could also manipulate DNS to falsely map a service's hostname to an IP address that places the adversary "on path". When notaries and clients use DNS names to identify services, a compromised local or remote DNS server present the same threat as the corresponding $\mathbf{L_{client}}$ or $\mathbf{L_{server}}$ attack. Thus, the analysis in Section 6 applies to DNS attacks as well.

Additionally, if an attacker controls only DNS, notaries can help clients detect and even circumvent such attacks. To do so, notaries would also record the IP address used when monitoring a service. If a client sees that both the service key and IP have changed from a previously trusted or stable key, it can connect directly to the IP addresses associated with the old key to test if that key is still visible on the network. If the client receives the prior key from any of the past addresses, it can (depending on local policy) disregard the new key as a DNS attack and instead connect directly to the address using the prior key.[12]

Client Privacy Considerations

While benign notary servers would not record a client's IP address, a malicious notary could link client addresses and destination services, impinging on client privacy. Because clients access the notary system only rarely (when the offered key is not cached) and furthermore do so only when a legitimate security threat exists, we believe that most clients will consider this potential privacy risk acceptable.

However, clients that desire additional privacy could contact the notary system through a proxy. One promising design for such a proxy is to have the notary overlay run a DNS nameserver for a special notary domain (e.g., *notary.com*). A notary client looking for observed key data for *server.domain.com* from *num-notary* different servers would perform a DNS look-up for *num-notary.server.domain.com.notary.com*.[13] The nameserver for *notary.com* would then randomly query *num-notary* servers and return the base64-encoded results as a DNS TXT record. Because of the recursive nature of DNS look-ups, the notary nameserver would learn only general information about the client (e.g., that they use CMU's DNS servers), and the rest of the notaries would learn no client-specific information. . Clients are unlikely to have privacy concerns about notary queries via local DNS resolvers, since they already expose such connection information to the resolver with standard DNS look-ups.

As we discuss in Section 10, these privacy concerns would be eliminated completely if servers were modified to be notary-aware.

[12] Adding IP addresses to the observed key data returned by a notary also helps clients handle cases when DNS-based load-balancing maps a single hostname to different machines that each have their own key.

[13] A similar DNS trick is used by the popular Coral Cache content distribution network [12]. Using a TTL of 0 and appending random data to the beginning of the DNS name can prevent DNS caching from providing stale data.

Detecting Authentication Downgrade Attacks

Our primary motivation for designing PERSPECTIVES was to help authenticate services that do not have certificates signed by a global PKI. However, hosts can benefit from using PERSPECTIVES even when accessing websites with PKI-signed certificate by gaining protection against *authentication downgrade attacks*. In an authentication downgrade attack, an AitM adversary between the client and the server injects a self-signed certificate in place of the PKI-signed certificate sent by the legitimate server. Because the attacker can spoof legitimate names in the domain name and issuer fields, the significant number of users who routinely ignore many browser security warnings [24, 11] would fall victim to such an attack. In fact, a malicious exit-node in the Tor anonymizing network was recently observed running such an attack [26]. However, if a notary overlay also monitors services with PKI-signed certificates, clients could detect this attack by comparing the received certificate against the notary replies. This same approach would also help prevent the even more damaging attack in which an attacker tricks one of the many certificate authorities trusted by a browser into signing an invalid certificate.[14]

9 Related Work

Significant work exists on the problem of authenticating remote Internet hosts. Standard solutions include X.509 certificates [5] within a global PKI, or Kerberos [17], which assumes that each participant has a shared-secret with a single trusted third party. Such solutions are extremely useful, but the popularity of the SSH model demonstrates the need for lightweight alternatives.

Ali and Smith [1] propose improving SSH key authentication using a "portable key cache", which the user stores along with an authenticating MAC on a personal webserver. With a modified SSH client, the user can access this cache

[14] To accommodate legitimate key turnover, the site owner can sign the new public key using the older key that is already recognized as valid by the notary overlay.

from any machine, use a passphrase to verify the integrity of its contents, and then compared the offered key to entries in the cache. This design helps users who would otherwise see the same new or changed key warning several times when connecting to the same server(s) from multiple client machines. However, unlike PERSPECTIVES, it provides no help in determining a key's validity when a user either accesses a service for the first time or when an offered key does not match the key in the user's portable cache.

Advocating a more significant departure from the standard SSH authentication model, RFC 4255 [25] proposes storing SSH host key fingerprints in DNS. Unfortunately, this proposal relies on the deployment of the secure DNS PKI to authenticate the fingerprint data itself, and secure DNS has shown little traction to date. Additionally, the proposal requires that a domain's DNS administrator fulfill the responsibilities of a certificate authority: verifying that a real-world entity who contacts him with a key to sign is the rightful owner of a particular host. In contrast, PERSPECTIVES requires no heavyweight verification and instead creates authentication data automatically using probes.

ConfiDNS [22] suggest performing DNS look-ups from diverse network vantage points. However, the primary focus of the ConfiDNS work was dealing with the fact that DNS replies (unlike public keys used in PERSPECTIVES) frequently have legitimate inconsistencies due to considerations like DNS load-balancing. Additionally, because the system was designed to avoid pollution in cooperative DNS systems, their design only protects against a malicious or failed *local* DNS server, not an "on path" adversary (e.g. $\mathbf{L}_{\mathbf{client}}$) launching an AitM attack.

Notary client policies bear some similarity to client behavior in the "web-of-trust" model used by Pretty-Good-Privacy (PGP) [4], a decentralized PKI for email. However, because PGP uses human contact to bind entities to keys, its primary challenge is estimating the strength of key trust chains that include multiple links, each representing a pair of real-world acquaintances. PERSPECTIVES policies do not have trust chains (each notary probes a service directly), but do have other complexities not seen in PGP, including the temporal nature of key histories.

Other recent work uses multiple network paths to improve security properties other than authentication. For example SPREAD [16] improves confidentiality and Information Slicing [14] provides anonymity.

10 Future Work

We believe that the network notary concept introduced by PERSPECTIVES opens several promising avenues for additional exploration in the area of host authentication.

Notary-Aware Services

As presented, PERSPECTIVES only requires client modifications. However, If notaries become common, servers might be modified to also communicate with notaries. This would provide three primary benefits:

1. **Immediate Probing of New Keys:** A server could alert notaries when a new service comes online or has changed its key, allowing notaries to immediately begin building a observation history for the new key.
2. **Reduced Need to Query Notaries:** The server could act as a caching proxy by querying notaries on behalf of clients. This would eliminate privacy issues related to clients querying notaries directly and also allow clients to receive cached observed key data even if notaries were temporarily unavailable.
3. **Attack Detection:** With access to notary observations about itself, a server could alert its administrator if the notary data includes any illegitimate keys. Such a false key is a likely indication that either a network element near the server is malicious or that a notary is compromised, thereby aiding attack detection.

Improving Security Indicators with PERSPECTIVES

Many users ignore client application security indicators, including the warnings about accepting unauthenticated keys [24]. Based on recent usable security research [11], we believe that PERSPECTIVES can improve the usability of security indicators in two ways. First, it can reduce "banner blindness" by eliminating unneeded security warnings when an unsigned key is validated by notaries. Second, notary data that suggests a reasonable probability of an actual attack can justify stronger and more intrusive user warnings, which have been shown to be effective.

Applying PERSPECTIVES to Additional Protocols

While SSH and HTTPS have served as running examples, PERSPECTIVES also opens the door for more widespread use of SSH-style authentication with other protocols because, unlike Tofu, it can authenticate keys even on the first connection to a service.

SMTP: Many SMTP servers already have self-signed SSL certificates so that local clients who manually install these certificates can authenticate their outgoing mail server. However, because no PKI exists for one server to trust another server's certificate, emails are often transmitted "in-the-clear", leaving them vulnerable to snooping by whomever controls the intermediate networks. PERSPECTIVES could support mutual authentication of inter-SMTP server communication, allowing, for example, a server to refuse to transmit a message the user has deemed "sensitive" to an unauthenticated server.[15]

[15]DomainKeys [6] could also benefit from PERSPECTIVES, as keys are currently acquired using unauthenticated DNS look-ups.

Incremental DNSSEC: The deployment of secure DNS (DNSSEC) is hampered by the fact that a sub-domain (e.g., example.com) cannot protect its hosts until its parent domain (e.g., .com) publishes its own public key and signs the sub-domain's public key. Unfortunately, to date, major top-level domains have shown little enthusiasm for deploying DNSSEC. With PERSPECTIVES, the authoritative nameserver for any sub-domain could publish an un-signed key used to sign its zone, with resolving name servers using notaries to validate and then cache the key.

11 Conclusion

As evidenced by the proliferation of SSH and self-signed SSL certificates, the simplicity and ease of management of the SSH authentication model makes it exceedingly popular. Unfortunately, the commonly used Trust-on-first-use approach is vulnerable to even simple AitM attacks. To enhance security without requiring a PKI, we designed PERSPECTIVES to supplement Tofu with spatial and temporal redundancy. Our implementation demonstrates that the notary concept is practical, and after using our PERSPECTIVES clients for several months, we have found them invaluable at several occasions: when logging in to a new server while connecting through a public wireless network, or when connecting to a known server after a server key change. From one author's personal experience, AitM attacks on Tofu protocols like SSH *do* happen in public spaces, and we consequently anticipate that PERSPECTIVES will serve as a practical light-weight measure to enhance security.

References

[1] Y. Ali and S. Smith. Flexible and scalable public key security for SSH. In *EuroPKI*, pages 43–56, 2004.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Experience with an Evolving Overlay Network Testbed. *ACM Computer Communications Review*, 33(3):13–19, July 2003.

[3] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. *Lecture Notes in Computer Science*, 1666:431–448, 1999.

[4] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*. Internet Engineering Task Force, Nov. 2007. RFC 4880.

[5] S. Chokhani and W. Ford. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*. Internet Engineering Task Force, 1999. RFC 2527.

[6] M. Delany. *Domain-based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys)*, Aug. 2004.

[7] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet Draft, Nov. 1998. Expires May 12, 1999.

[8] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet Request for Comment RFC 2246, Internet Engineering Task Force, Jan. 1999. Proposed Standard.

[9] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, pages 107–125, 1992.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, Aug. 2004.

[11] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: An empirical study of the effectiveness of web browser phishing warnings, 2008.

[12] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. In *Proceedings of the 4th USENIX Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[13] A. O. Freier, P. Kariton, and P. C. Kocher. The SSL protocol: Version 3.0. Internet draft, Netscape Communications, 1996.

[14] S. Katti, J. Cohen, and D. Katabi. Information slicing: Anonymity using unreliable overlays. In *Proceedings of the 4th USENIX Symposium on Network Systems Design and Implementation (NSDI)*, Apr. 2007.

[15] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. 14 (4):255–293, 2001.

[16] W. Lou, W. Liu, and Y. Fang. Spread: Enhancing data confidentiality in mobile ad hoc networks. In *Proc. INFOCOM*, 2004.

[17] S. Miller, B. Neuman, J. Schiller, and J. Saltzer. Kerberos authentication and authorization system. Technical report, MIT, Oct. 1988. Project Athena Technical Plan.

[18] Mozilla Firefox. Firefox web browser. `http://www.mozilla.com/firefox/`.

[19] NTP-Pool. pool.ntp.org : the Internet cluster of NTP servers. `http://www.pool.ntp.org`.

[20] OpenSSH. OpenSSH. http://www.openssh.com.

[21] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. In *Proc. 1st ACM Workshop on Hot Topics in Networks (Hotnets-I)*, Oct. 2002.

[22] L. Poole and V. S. Pai. ConfiDNS: Leveraging scale and history to improve DNS security. In *Proceedings of Third Workshop on Real, Large Distributed Systems (WORLDS 2006)*, November 2006.

[23] RON Testbed. The RON/IRIS testbed. http://www.datapository.net/tb/.

[24] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[25] J. Schlyter and W. Griffin. *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints*. Internet Engineering Task Force, Jan. 2006. RFC 4255.

[26] Tor Exit Node Hijacks. TOR exit-node doing MITM attacks. `http://www.teamfurry.com/wordpress/2007/11/20/tor-exit-node-doing-mitm-attacks`.

[27] Traceroute.org. Traceroute.org. `http://www.traceroute.org`.

[28] Y. Ylonen and C. Lonvick. *The Secure Shell SSH Protocol Architecture*. Internet Engineering Task Force, Jan. 2006. RFC 4251.