Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure*

Tiffany Hyun-Jin Kim Lin-Shung Huang Adrian Perrig Collin Jackson Virgil Gligor
CyLab / Carnegie Mellon University
{hyunjin1,adrian,virgil}@ece.cmu.edu {linshung.huang,collin.jackson}@sv.cmu.edu

ABSTRACT

Recent trends in public-key infrastructure research explore the trade-off between decreased trust in Certificate Authorities (CAs), resilience against attacks, communication overhead (bandwidth and latency) for setting up an SSL/TLS connection, and availability with respect to verifiability of public key information. In this paper, we propose AKI as a new public-key validation infrastructure, to reduce the level of trust in CAs. AKI integrates an architecture for key revocation of all entities (e.g., CAs, domains) with an architecture for accountability of all infrastructure parties through checks-and-balances. AKI efficiently handles common certification operations, and gracefully handles catastrophic events such as domain key loss or compromise. We propose AKI to make progress towards a public-key validation infrastructure with key revocation that reduces trust in any single entity.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection—*Authentication*; C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General—*Security and protection*

Keywords

Public-Key Infrastructure, SSL, TLS, certificate validation, public log servers, accountability.

1. INTRODUCTION

Secure connection establishment on the Internet through SSL and TLS has been a tremendous success, as it is globally used for practically all secure web-based communication. Given that the security of the majority of network-based financial or commercial transactions relies on SSL/TLS, one would hope that its security is commensurate with its proliferation and importance.

Unfortunately, numerous attack vectors against SSL/TLS exist and recently several high-profile attacks have demonstrated its vulnerability in practice. The main weakness lies in the fact that current browsers trust hundreds of root Certificate Authority (CA) certificates, and a security breach of a single CA can compromise the security of sites protected by any one of the other CAs, powerfully illustrating the concept of weakest-link security. In fact, a maliciously issued certificate for a site can be used by an adversary to mount Man-in-the-Middle (MitM) attacks on connections to that site. To demonstrate the large extent of the SSL universe in which the weakest link could occur, EFF's SSL Observatory reports that Microsoft IE and Mozilla Firefox trust 1482 different CA public keys [13].

Since CAs are in the business of managing cryptographic keys, they are expected to have secure processes in place to protect their keys. Regrettably, recent events have highlighted the inability of several CAs to keep their keys and certificate issuance processes secure. We list a few high-profile cases that were recently discovered. In March 2011, in an attack on a **Comodo** reseller, fake certificates were issued for `mail.google.com`, `www.google.com`, `login.yahoo.com`, `login.skype.com`, `login.live.com`, and `addons.mozilla.org` [11, 27]. Comodo suggested that the attack originated from an Iranian IP address. In August 2011, news broke that **DigiNotar**, a Dutch CA, improperly issued a certificate for all Google domains to an external party [30]. It was claimed that as many as 250 false certificates for an unknown number of domains were released. It was reported that these certificates were used by the Iranian government to spy on Iranian citizens' communications with Google email during the month of August 2011. For the Stuxnet malware, two **Taiwanese CAs**' private keys were compromised, which the Stuxnet developers used to sign their malware [14]. Since not all CA vulnerabilities become public, we expect that an even larger number of CA-breaches may have occurred.

These examples demonstrate that CA breaches can result in real-world attacks. Besides CA-based attacks, SSL/TLS has other vulnerabilities, for example the fact that users click through browser warnings in case of self-signed certificates [12]. In Syria, such an attack was used to mount a MitM attack against Facebook [10], supposedly by the Syrian Telecom Ministry.

Addressing these problems is very challenging, as several seemingly conflicting requirements need to be satisfied. On one hand, adversarial events such as CA private key compromise or domain private key compromise need to be addressed. On the other hand, legitimate events such as switching to different CAs or key re-creation after private key loss need to be supported. For example, legitimate re-creation of a key pair and certificate after private-key loss may appear to be an impersonation attempt. Also, legitimately switching to a new CA to cease using a compromised CA that signs fraudulent certificates may also appear as a malicious event. Hence, we aim to create a certificate infrastructure that can prevent adversarial attacks yet gracefully handle legitimate key and certificate management events.

We design a new certificate validation infrastructure to address these challenges. Our proposal is called Accountable Key Infras-

*An earlier version of this paper was published as a technical report [18].

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

tructure (AKI), integrating an architecture for key revocation of all entities (e.g., CAs, domains) with an architecture for accountability of all infrastructure parties through checks-and-balances.

AKI efficiently handles common certification operations and gracefully handles catastrophic events such as domain key loss or compromise. We propose AKI to make progress towards a public-key validation infrastructure with key revocation that reduces trust in any single entity. To accomplish these goals, we leverage globally visible directories (i.e., public log servers) that enable public integrity validation for certificate information. Such public validation provides accountability for CA's actions, and thus creates deterrence against fraudulent CA activities. To reduce the number of trusted CAs, a domain can define which and how many CAs are required to update the certificate. To enable recovery from unanticipated events, certificates can be updated through another set of CAs; however, the certificates become active after a domain-specified hold time. In case of fraudulent updates, legitimate domains can react during the hold time to have the fraudulent certificate removed.

To evaluate the security, availability, and efficiency of our public-key validation infrastructure and to enable comparison between different systems, we propose a new set of metrics. Specifically, we propose Duration of Compromise (DoC), Duration of Unavailability (DoU), and several efficiency metrics. The DoC metrics provide insight into the security of a system, measuring the impact of a compromise or loss of various credentials, such as the private key of CAs or domains. The DoU metrics measure availability, again depending on various events such as key compromise or loss. Finally, the efficiency metrics measure the overhead of operating the public-key validation infrastructure and the overhead of secure session establishment. We present these metrics in detail in Section 9.

2. PROBLEM DEFINITION

The core problem we aim to address is the design of a new public-key validation infrastructure that reduces the amount of trust placed in any one infrastructure component (e.g., CA), yet reduces the system's attack surface and single points of failure. Additional goals are efficiency and incentives for deployment for all associated parties. In this section, we first describe a list of desired properties, followed by assumptions and the adversary model. Detailed metrics for evaluation are described in Section 9.

2.1 Desired Properties

- **Checks and balances:** The infrastructure should limit trust in any single party, and limited trust should be distributed over multiple parties to prevent a single point of failure. Furthermore, parties can monitor each other to detect misbehavior.
- **Brief compromise period:** Given the compromise of a private key of any trusted party, the time during which an attacker can successfully attack a legitimate client should be brief. This includes the compromise of CAs, public log servers, or domains.
- **Brief unavailability period:** After various events (benign or adversarial), the time during which a legitimate client (who is not under an attack) cannot verify a domain's certificate should be brief. This includes when a domain's certificate is newly registered or updated, and when the CAs', log servers', and domains' private keys are compromised.
- **Trust agility:** Users can decide which entities they trust to form their root of trust, and they can modify their trust decisions at any time [24]. Furthermore, such changes should be made without undue delay. We extend this notion of trust agility to domains, enabling domains to select their roots of trust and to modify their trust decisions at any time.
- **Privacy:** Clients should not reveal which entity/server they establish an SSL/TLS connection with.
- **Efficiency:** One of the most important efficiency requirements is to avoid increasing the latency of an SSL/TLS connection establishment — in particular, avoiding any additional round-trips to external servers. Moreover, no additional infrastructure servers should be needed.

2.2 Assumptions

In our approach of checks and balances, we assume a set of entities that do not collude: CAs, public log servers, and validators. We assume that these entities audit each others' accountable operations and disseminate detected misbehavior. In the context of the current Internet, the Electronic Frontier Foundation (EFF) may be one entity that would play the role of a validator entity, for example.

We assume that browsers store the authentic public keys of CAs, public log servers, and validators. We also assume that all parties are loosely time synchronized (up to a few minutes).

2.3 Adversary Model

We consider an adversary whose main goal is to impersonate a victim web site that is using HTTPS. To achieve this goal, the adversary may compromise some infrastructure servers that the victim trusts. An adversary may be able to *temporarily gain control of some infrastructure servers that the victim trusts*, but can gain *long-term control of infrastructure servers that the victim does not trust*. Gaining control means access to private signing keys.

3. BACKGROUND

Two main families of proposals to reduce trust in CAs exist that are related to our design: (1) certificate observatories and (2) timeline servers that provide public visibility into all certificate operations. We briefly provide an overview of these approaches.

3.1 Certificate Observatories

The first type to reduce the trust in CAs and prevent many of the attacks discussed in Section 1 is to create a public repository of SSL/TLS server certificates and enable browsers to compare the key they have received (presumably from the server) with the observation of the observatory which is received over an integrity-protected connection. This integrity-protected connection is set up by embedding a root public key of the observatory system into the client, creating a PKI just to authenticate the relatively small number of observatory nodes.

Perspectives. The Perspectives system [2, 33] has globally distributed notary servers that contact known SSL / TLS servers once a day to fetch the current server's certificate. These notary servers then *store the history of observed certificates and support queries into their database.* Perspectives offers a Firefox plugin, which contacts a random subset of notary servers after an HTTPS connection is opened, to compare the notary certificate observations with the received certificate. A configurable policy decides if the received certificate is presumed valid.

Convergence. Convergence [1] enhances Perspectives in several dimensions, most notably *providing privacy for certificate lookups by including a two-step onion routing approach*, where the first Convergence server redirects the query to a second server, and the second server responds (the first server knows the identity of the querier, but not the web site queried, and the second server only knows the query but not the querier).

SSL Observatory. EFF's SSL Observatory [13] also collects global certificate information. However, it does not frequently update the information nor support any online queries.

Discussion. The main advantage of these certificate observatories is that *server operators need not be aware of this approach.* Hence, no additional steps are necessary on their part. The approach even works to validate self-signed certificates. These systems are also effective to prevent numerous CA-based attacks, for example, all the attacks in Iran [11, 27] and Syria [10] would have been prevented, as the illegitimate certificates would have been detected as different from the legitimate server certificates.

The main disadvantage of these systems is that they require additional connections to query the observatories, resulting in higher latency for establishing an HTTPS connection. Another disadvantage is a period of unavailability for new and updated certificates. However, this can be remedied with a scheme where a new certificate is offered ahead of time to the observation servers, but this approach negates one of the main advantages of these systems in that they do not require assistance from the server operators.

3.2 Certificate Log Servers

Certificate Transparency [20–22] and Sovereign Keys [9] are two recent proposals that suggest a public log to record all certificate transactions.

Certificate Transparency (CT). In CT [20–22], the authors construct an append-only log by using an append-only Merkle hash tree structure, enabling *efficient validation of each certificate that was added to the log*. More specifically, domains register their certificates and obtain a non-repudiable statement (audit proof) that their certificate has been added to the append-only log. Domain owners provide the audit proof along with the certificate such that clients can validate whether the certificate was included in the log.

Sovereign Keys (SK). In the SK model, timeline servers act similar to timestamping servers [15]. In SK, the first registration of a certificate binds the key to the domain name, preventing any subsequent registrations for the same name. Only the legitimate owner of the certificate has the private key, enabling revocation and re-registration of the name with a new public key.

The browser can contact the timeline server to inspect if the received server certificate is indeed the correct certificate registered in the timeline server's log. To reduce the load on timeline servers, distributed mirror servers store a copy of the timeline server database to efficiently disseminate information.

Discussion. The public log proposed by CT and SK creates accountability for a CA's actions, an idea which we build on in this paper. The main advantage of the log is that it prevents the attacks mentioned in the introduction. For example, compelled certificates [28] are simply invalid in this framework, as the legitimate owner did not approve the newly generated key.

Unfortunately, CT and SK have some shortcomings. CT and SK do not specify any revocation mechanism for audit proofs, which indicates that audit proofs are valid indefinitely. Instead, CT relies on other approaches such as Certificate Revocation Lists, Online Certificate Status Protocol, Revocation Transparency [19], etc. Thus, these approaches currently do not handle events such as private key loss or compromise of a domain. For SK, the browser needs to query a mirror server to download the server's certificate history, increasing latency, decreasing availability, and sacrificing privacy. In Section 9, we discuss these systems in more detail.

4. AKI: Accountable Key Infrastructure

Before we describe the details of AKI, we first provide a high-level overview. The main entities in our system are as follows.

- A **Domain (server)** represents a named entity with which clients desire to establish secure connections.

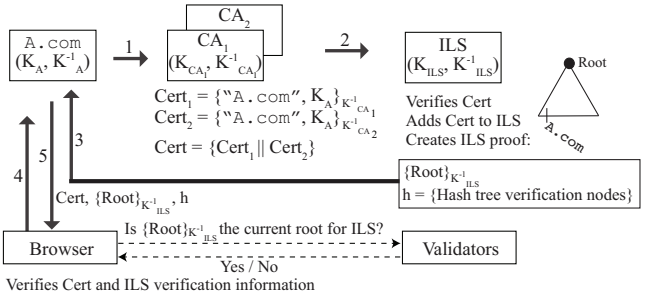

Figure 1: AKI certificate registration process. This figure depicts `A.com` registering a certificate (signed by two CAs) to a single ILS, but the domain can acquire multiple certificates from multiple CAs and register to multiple ILSs. Dashed arrows represent occasional communications.

- A **Client (browser)** is an entity establishing TLS connections with domains (servers).
- A **Certification Agency (CA)** is similar to a current certificate authority that certifies domains' public keys, but "agency" instead of "authority" indicates that CAs are not absolute authorities any more in AKI.
- In addition to current systems, **Integrity Log Server (ILS) Operators (ILSO)** are Internet services that operate ILSs that log domains' certificates and make them publicly available. Each ILS maintains an **Integrity Tree**, which is a hash tree of all the registered certificates in lexicographic order. All ILS operations (i.e., log data entries such as registration, updates, and revocation) are digitally signed. We envision that large ISPs or widely available Internet-based corporations such as Amazon or Google will operate ILSs.
- **Validators** are entities that monitor ILS operations, by downloading the entire ILS data structure and performing consistency checks. Consistency checks include validation that certificate updates follow the certificate policy or that certificates do not suddenly disappear from the log. Non-governmental Internet governance organizations, such as the EFF, could serve as validators. In the case of ILSO misbehavior, validators disseminate incriminating information without requiring their own authentication information.

Figure 1 depicts an overview of our AKI architecture. Alice owns domain `A.com` and wants to obtain an AKI-protected certificate, as she wants to protect herself against compromise of the CAs that signed her certificate and other rogue CAs, and protect her clients against compelled certificates. To define the security properties that she intends to achieve for her domain, Alice defines CAs and ILSOs that she trusts, the minimum number of CA signatures that she recommends her clients for validation, and rules for certificate revocation, replacement, and updates. Alice includes these parameters with her public key and contacts more than the minimum number of trusted CAs (according to her security policy) to sign her certificate. She then registers the certificate with one or multiple ILSs. Each ILS adds `A.com` to its database, by placing it in the Integrity Tree. The ILS then re-computes hash values over all stored certificates for updated verification information.

Alice now supplements her certificate with the verification information that she downloads from every ILS, and sends it to browsers that connect to her web site via HTTPS. For certificate validation, the browser uses the trusted root-CA certificates as in current practice, and uses the pre-installed ILS public key(s) on her browser to validate ILS information.

Before trusting the ILS information received from Alice, the client browser occasionally checks with validators to confirm that the ILSs' current root hash values are valid. To reduce latency, we also describe later how the the web server can staple validator information during the HTTPS connection setup.

4.1 AKI Details

We discuss the details of AKI based on the following stages: certificate creation, CA signature acquisition, ILS registration, browser-based validation, ILS tree update, certificate update, and certificate revocation and recovery.

Certificate creation. AKI certificates contain several extensions over standard X.509 certificates and feature the following additional fields:

- `Trusted CAs (CA_LIST)`: This field contains a list of trusted CAs for creating a new certificate.
- `Trusted ILSs (ILS_LIST)`: This field contains a list of trusted ILSs where the certificate is registered.
- `ILS validation proof timeout (ILS_TIMEOUT)`: This field indicates how long an ILS proof is acceptable to the browser after the proof creation time. The tradeoff is among efficiency, availability, and robustness to compromise. A long timeout requires fewer queries to the ILS for an updated proof, but increases the amount of time until a certificate can be revoked. This parameter typically varies from one hour to one day.
- `Minimum number of CAs to generate an AKI certificate (CA_MIN)`: This field indicates the number of CA signatures required to initially register a certificate to ILS and to update a certificate. This parameter is typically set to 1 or 2.
- `Threshold number of CAs for certificate re-establishment (CA_TH)`: This field indicates the minimum number of CA signatures needed to re-establish a certificate in case of a lost private key. In other words, this parameter indicates the number of different CA signatures that can activate the new key for the domain that lost its key. An attacker can register a certificate to an ILS for a domain who is unaware of AKI, and select `CA_TH` to be high such that the domain can never revoke the adversary's certificate. To prevent such an attack, we set `CA_TH` = `CA_MIN` + 1.
- `Cool-off period for an unlinked certificate (COP_UNLINKED)`: This field indicates the minimum cool-off period for a new certificate that is not linked to the old certificate (i.e., the new public key is not signed by the previous private key of the domain). In AKI, registering a new certificate does not automatically validate it:. Instead, AKI enforces a "cool-off" period until a new certificate becomes valid which will replace the previous certificate. This enables protection against an adversary who quickly registers a new key following a CA compromise, as the legitimate owner can revoke the new certificate during the cool-off period. An attacker can register a new key for a domain that is unaware of AKI and set `COP_UNLINKED` to be high to prevent the domain owner from re-acquiring the ILS entry. To prevent such an attack, an upper bound exists for `COP_UNLINKED` (e.g., 7 days).
- `Cool-off period for a certificate from an untrusted CA (COP_UNTRUSTED)`: This field indicates the minimum cool-off period for a new certificate that is signed by a CA that is not in `CA_LIST`. In case of a lost or compromised private key, an attacker can acquire a certificate signed by some CA (that the domain owner does not trust), and this parameter provides time to enable the legitimate owner to revoke the bogus certificate. We recommend that `COP_UNTRUSTED` is defined to be longer than `COP_UNLINKED`. For similar reasons as for `COP_UNLINKED`, `COP_UNTRUSTED` has an upper bound (e.g., 10 days).

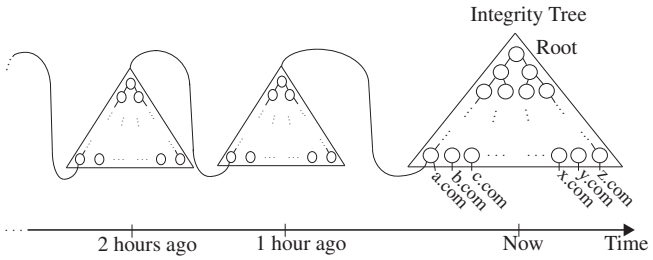

Figure 2: AKI Integrity Tree structure.

CA signature acquisition. After creating a certificate with AKI extensions, the domain contacts the `CA_MIN` number of CAs from `CA_LIST` to acquire CA signatures. In AKI, the combination of all the CAs' and ILS signatures validates the domain's AKI certificate. Hence, care must be taken to include all the CAs' serial numbers and timestamps, and the X.509 standard will need to be amended to enable such multi-signatures (discussed in Appendix A).

ILS registration. The domain then contacts one or more of the trusted ILSs (from `ILS_LIST`) to register the AKI certificate.

The ILS data structure to store AKI certificates is based on a binary hash tree, called *Integrity Tree*, as depicted in Figure 2. All AKI certificates are placed at the leaf nodes of the binary hash tree, sorted in lexicographic order. AKI uses a sorted hash tree as opposed to a linear list as in previous work [9, 22] for the following reasons:

1) The hash tree efficiently represents the current state of all distinct names, and its height only depends on the number of different entries but not on time (i.e., it does not grow taller with revocations/re-establishments, thus removing a source of DoS).
2) The height is logarithmic in the number of entries. Hence, a validation of any leaf node can be efficiently represented based on an authenticated root node and a logarithmic number of nodes to re-compute the root node from the leaf node.
3) The sorting enables quick verification of the *absence* of an entry, whereas in a linear list, the entire list needs to be searched.

As depicted in Figure 2, the Integrity Tree enables independent validators to check the integrity of the entire data structure. The hash chaining of the trees enables temporal re-construction of all operations, similar to a timestamping service or the timeline server data structure [9, 22].

When adding a new AKI certificate, the ILS first verifies whether an entry already exists in its Integrity Tree for the same domain name. (In Section 6 we discuss how an ILS checks other ILSs to ensure name uniqueness.) If the name is indeed new, the ILS schedules the AKI certificate to be added to the Integrity Tree. The ILS creates a *registration confirmation* of the successful addition through a digital signature with its private key, and returns it to the domain. The ILS registration confirmation contains a proof of absence that the name is not present in the Integrity Tree, preventing a malicious ILS from minting forged registration confirmations if the name is already registered. The domain also obtains a signed statement from validators that they have seen the registration confirmation, and the domain can use this ILS registration confirmation and validator statement to start using the new certificate for a limited time, until the ILS generates the next Integrity Tree which will include the new certificate as a leaf node. Figure 3 depicts the certificate registration process.

The `ILS update period (ILS_UP)` is the interval between two tree updates; at every `ILS_UP`, an ILS finalizes and commits the

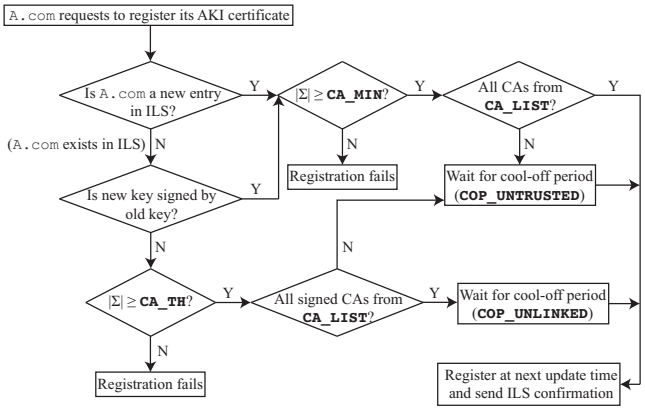

Figure 3: Flowchart of ILS certificate registration, update, revocation and recovery. Upon receiving a certificate registration request from A.com, the ILS follows this flowchart and plans to add the certificates. Σ stands for the CA signatures that A.com obtained.

next Integrity Tree. At this point, the domain contacts the ILS to request the signed root node ($\{Root\}_{K_{ILS}^{-1}}$) and the hash tree verification nodes (h) that are needed to validate its certificate as depicted in Figure 2, where ILS_UP is set to one hour. In practice, ILS_UP is set to one or two hours, to enable quick certificate revocation. ILS verification information is combined with the AKI certificate to enable client browsers to validate the ILS information without the need to contact an ILS during the TLS connection setup.

Browser-based validation. The browser receives the validator and ILS information together with the server/domain's AKI certificate during the second phase of the TLS protocol.[1] The CA signatures are validated using the browser's CA root certificates, and the ILS and validator information is validated using the ILS and validator public keys stored in the browser. The ILS_TIMEOUT field in the AKI certificate is validated to ensure that the ILS information is sufficiently recent depending on the domain's preferences specified in the certificate. In Section 6, we discuss incremental deployment issues, considering the actions an AKI-enabled browser should perform when presented with a legacy TLS certificate.

ILS tree update. Periodically at a well-specified time, the ILS updates its Integrity Tree by purging AKI certificates that have been revoked or expired without renewal. The ILS also activates certificates that have passed their cool-off periods.

We envision update intervals of an hour up to a day. Hourly updates enable more fine-grained certificate revocation but increase overhead, as servers/domains need to frequently query the new signed root value to ensure that their certificates remain unchanged.

Certificate update. Before an AKI certificate expires, the domain creates a new private key, and requests the trusted CAs in CA_LIST to sign the new key. The domain also signs the new key with its previous private key. After gathering CA_MIN number of CA signatures, the domain combines all signatures and other relevant information into an AKI certificate. The domain then sends the AKI certificate with an update request to the ILS, which will readily accept the new AKI certificate since it is signed with the domain's old key and the update request is signed by both new and old keys. (Requiring a signature with the new key confirms possession of the new private key.) There is no cool-off period in this case, and the new AKI certificate is added when the ILS creates the next Integrity Tree. Hence, the new key can be readily used.

Certificate revocation and recovery. In case a key needs to be prematurely removed, a certificate revocation message needs to be sent to the ILS. Either the private key corresponding to the certificate's public key is used to sign the revocation message, or a special revocation key can be used, for which the public key is included in the certificate. The point of using a different revocation key could speed up recovery for the case where the main private key is compromised, as a shorter cool-off period can be used if the new public key would be signed by the revocation key.

The cool-off periods (COP_UNLINKED, COP_UNTRUSTED) in the AKI certificate specify the amount of time that needs to elapse before the new certificate becomes active. In case of private key compromise (and potentially private revocation key compromise), the COP_UNLINKED and COP_UNTRUSTED values enable the legitimate owner to react and revoke a fraudulent certificate that was potentially registered by the adversary.[2]

Since some domains may not have the best key secrecy and availability practices in place, we need to consider the case of catastrophic key compromise and loss when only the adversary is in possession of all secrets. In that case, we need recovery mechanisms where the legitimate owner can re-gain control of its domain. By contacting CA_TH number of CAs and obtaining signatures on a fresh key, the legitimate owner can eventually re-gain control. However, the adversary will be able to use the key until a valid revocation message arrives.

4.2 Checks and Balances among Parties

In this section, we describe how AKI achieves checks and balances among CAs, ILSs, validators, domain owners, and clients to reduce trust and prevent misbehavior by any party. Figure 4 illustrates what each party monitors and how each party reports.

An important aspect of AKI is that all actions are digitally signed, such that any misbehavior can be demonstrated based on the entities' signatures. Consequently, an accusation for misbehavior can be checked without trusting the accuser, ruling out slander attacks. Equivocation is an example for a misbehavior, where one party provides different answers to the same query – for example an ILS server who would create two different Integrity Trees in a given time period and uses either tree to respond to different entities. Since only a single Integrity Tree can exist per ILS per time period, the demonstration of the two ILS-signed Integrity Trees demonstrates ILS misbehavior.

Validation by CAs. Once the domain owner acquires signatures from trusted CAs for the certificate, the CAs monitor the ILS for any malicious changes in the domain's ILS entry. If the ILS makes a potentially invalid update (e.g., updated certificate without any of the trusted CAs' signatures), the CAs immediately inform the domain owner.

Validation by validators. Validators maintain a list of revoked ILSs that are detected for misbehavior. Validators disseminate the revoked ILSs, especially to the domain owners who are registered to those revoked ILSs, in which case the domain owners attempt to register with other valid ILSs. Thanks to the fact that all ILS operations are signed, the validator can easily demonstrate misbehavior

[1] To avoid additional latency and privacy leakage by the browser, the domain directly contacts relevant ILSs and validators, and passes the ILS and validator information to the browser. The browser would specify as part of the TLS handshake which CA/ILS/validator information it already has and which information it still seeks to receive. Note that the domain only needs to contact the ILS once every ILS_TIMEOUT time period and when the ILS switches to a new Integrity Tree (i.e., every ILS_UP). Then the domain can give that ILS information to the client browser. The validator needs to be contacted every ILS_UP time period to authenticate the ILS Integrity Tree root value.

[2] We envision that trusted CAs would sign these revocation messages.

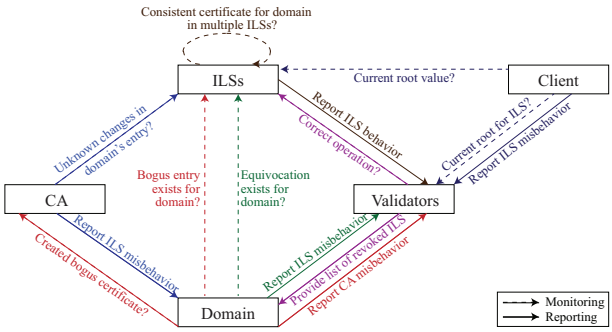

Figure 4: Checks and balances among AKI entities. In AKI, each entity monitors other entity for misbehavior detection and reports to other entities.

in case the signed records are inconsistent with the ILS's state. In the absence of a compromised ILS private key, a validator cannot perform a slander attack, as it cannot forge signatures that would incriminate the ILS for malicious behavior.

Validation by domain owners. Prior to initial registration of a certificate to an ILS, the domain first ensures that a CA has not created a bogus certificate for that domain by checking the ILS as follows: the domain owner queries the ILS for entries that are lexicographically adjacent to its entry in the Integrity Tree. After confirming that no entry exists for the domain, the domain owner registers its certificate with the ILS.

Validation by clients. A client browser obtains ILS and validator information from the domain's web server. The browser may also check ILSs and validators directly. In case of any inconsistent information, the client will report such misbehavior to validators for further dissemination.

5. SECURITY ANALYSIS

In this paper, we provide an informal analysis of the AKI architecture. We are working on a formal analysis, which we will publish in a paper currently under preparation.

The main security property we aim to achieve is the prevention of successful impersonation of a victim server. More concretely, given a domain S with a certificate C_S, an adversary M attempts to impersonate S to a client C during the TLS connection establishment. The attack succeeds if C has a connection with M while believing that the connection is with S.

We assume that S uses an AKI certificate, and that C uses an AKI-enabled browser. We argue that AKI is resilient against attackers that compromise different entities' private keys.

***M* compromises *S*'s private key.** Assuming S detects the compromise, it can immediately revoke the key. However, M can use the key until time t_0 + `ILS_TIMEOUT` period, where t_0 is the time when the certificate had been revoked at the ILS. The domain's selection of `ILS_TIMEOUT` depends on its choice of tradeoff between availability, overhead, and duration of impersonation after compromise.

To re-instantiate a key after compromise, there is an unavailability period (`COP_UNLINKED` as specified in the certificate) in case the domain lost access to its private key. However, if the domain owner still has access to the private key, it can obtain trusted CA signatures for its new key, sign it with its old key, and immediately obtain an ILS confirmation that will enable use of the new key.

***M* compromises CAs' private keys.** As long as fewer than `CA_TH` keys of trusted CAs in `CA_LIST` are compromised, there is no impact on browsers who contact the trusted ILSs. The `CA_LIST` evicts untrusted CAs from the set of potential weak links. Given a small well-selected list of trusted CAs, it is highly unlikely that more than a threshold number are compromised.

Even in case more than the `CA_TH` number of CAs are compromised, a newly registered key will have to cool off during the prolonged period (`CA_UNLINKED`) as the fraudulent certificate is not linked to the previous one (as we assume that S's private key was not compromised in this case). Such an extended time should leave sufficient time for S to detect and react to the impersonation attempt, without suffering any compromise (thanks to trusted CAs who watch out for ILS entry changes). If S's private key was compromised in addition, M can impersonate S during the entire `CA_UNLINKED` period. However, this case is exceedingly unlikely, as several well-selected CAs and the domain's private key all need to be compromised at the same time.

An adversary can also contact a different ILS to register an AKI certificate for a victim domain whose AKI certificate is already registered at a legitimate ILS. If ILSs coordinate their name spaces, as we discuss in Section 6, this attack is prevented.

***M* compromises ILSs' private keys.** We consider three attacks: equivocation with multiple Integrity Trees, fake entries for S.com in same Integrity Tree, and forged ILS confirmations. As we argue in the following analysis, a compromised ILS is not sufficient to perform a MitM attack on a server with an existing AKI certificate.

If M attempts to perform equivocation (i.e., create a shadow Integrity Tree with malicious entries and then answer queries from either tree depending on the querier), validators can readily detect this since they need to sign off the Integrity Tree root value in each time period for clients to accept the ILS information. If an ILS provides different information to different validators, then domains can detect this as they contact different validators and disseminate the information amongst validators. Since ILSs need to sign off the Integrity Tree root value, all the operations performed since the previous Integrity Tree need to be correct, thus, the ILS has very limited opportunity to manipulate the tree, e.g., removal and introduction of a fake certificate would not be possible unless it follows the correct policy of the certificate.

The fact that the Integrity Tree root value prevents equivocation even helps in the case when M compromises CAs' private keys in addition to the ILS's. If M attempts to re-register a new key, S can detect this behavior (as we discuss in Section 4.1) and raise an alarm. If M attempts to provide different answers to S's queries, it would need to create different Integrity Trees within one time period, which can be detected as described in the previous paragraph.

Another attack for M would be to attempt to create two different entries for S.com at different places in the Integrity Tree, one for the legitimate S.com and the other for a fraudulent S.com. In this scenario, M would provide the legitimate response to S's queries, and a fraudulent certificate for other queries. Fortunately, this case is easily detectable by the validators, as the leaf nodes would not be in sorted order. Placing the two leaf nodes next to each other will be detected by S, when it also queries for the leaf nodes that are adjacent to its certificate in the Integrity Tree.

Another case is where M misuses the ILS's compromised private key to sign fake ILS registration confirmations, which would enable a freshly generated and initially registered AKI certificate to be immediately used and trusted by AKI-enabled browsers. The proof of absence that is part of the registration confirmation though prevents the ILS from creating such confirmations for names that are already registered. Since domains register the ILS registration confirmation with validators, the absence of the name in the next Integrity Tree can be detected.

6. DISCUSSION

Censorship resilience. Corporations/governments may want to eavesdrop on all employees'/citizens' communication. More specifically, corporations/governments can set up their own CA and ILS that create fake certificates. In such a case, users can opt out by installing legitimate CAs and ILSs as roots of trust. Moreover, malicious behavior can be easily collected and demonstrated to the world.

Absence of ILS information during incremental deployment. Similar to Extended Validation (EV) certificates, absence of ILS information may not raise any suspicions. To prevent an attack where a non-AKI certificate is used to attack a domain that is using an AKI-certificate, we require browsers to contact ILSs trusted by the browser to validate the absence of AKI information.[3] In the absence of an ILS response, the browser needs to abort the connection. Note that no additional latency is required for deploying sites, since they provide the ILS information during the SSL handshake; only legacy domains and attackers have additional latency. Hence, this additional latency provides a positive incentive for adoption at domains.

In some legitimate environments, ILSs may not be reachable, for example paywalls at airports or hotels do not permit any external connections until the user has authenticated, paid, or accepted the terms of service. A challenge then is: how can the browser verify the non-AKI certificate of the paywall service without access to the ILSs? In this case, geographically-linked certificates [17] can be used, or the paywall obtains an AKI-certificate.

In case of legacy certificates, a browser would need to contact several ILSs and could only proceed when receiving a negative response from all ILSs, which would increase latency considerably, deterring initial AKI adopters. To speed up this process, ILSs could sign a Bloom Filter representing the set of entities in its Integrity Tree. Browsers would then only contact an ILS if the entity is in the ILS's Bloom Filter. Since Bloom Filters have a tunable false-positive rate, the browser may unnecessarily contact the ILS for a small fraction of legacy domains. Unfortunately, freshly registered certificates are not yet part of the Integrity Tree, thus they would not be part of the Bloom Filter either (unless the Bloom Filter is generated and distributed frequently), so a malicious legacy certificate could be used to impersonate a site until it is properly added to the Integrity Tree. Since such Bloom Filters would only be needed during initial phases of AKI adoption, we feel that the efficiency-security tradeoff is worthwhile, especially for short ILS update periods.

Globally consistent registration. Ideally, all global ILSs coordinate registration and provide one global name space, preventing the same name to be registered at different ILSs with different certificates. However, global coordination is challenging to implement in practice, as different global entities mutually distrust each other. An example is where a rogue CA issues a bogus certificate for `A.com`, presumably the CA and the requester of `A.com` are in a different legal region from the legitimate owner of `A.com` preventing the conflict to be locally resolved through legal means. In such cases of inconsistent registrations, the CAs local to the registered name of `A.com` obtain precedence in determining the correct certificate. If the foreign ILS does not unregister the conflicting entry, it loses its credibility and will be subsequently ignored. It is the task of the validators to document, store, and disseminate such incriminating ILS information.

The SCION architecture [34] offers isolation among Trust Domains with uniform legal environments. By defining trusted CAs, ILSs, and validators within each Trust Domain, and by binding DNS name spaces to Trust Domains, SCION greatly simplifies consistency issues, as it is clear which CAs, ILSs, and validators are responsible for the certificate validation. Within a Trust Domain, these entities can coordinate to prevent registration of rogue certificates.

Usability. Prior work has shown that users ignore and click through certificate warnings [31]. However, AKI can identify real attacks and completely block users from proceeding without an option to click through.

Interaction with HSTS. AKI does not address SSL stripping [23] attacks where an adversary actively rewrites HTTPS links to HTTP links while eavesdropping HTTP traffic. Against such attacks, websites can force clients to always use HTTPS by declaring the HTTP Strict Transport Security (HSTS) header [16]. Furthermore, HSTS treats certificate warnings as fatal errors as does AKI. AKI extends HSTS by requiring valid ILS information for AKI certificates.

7. REALIZATION IN PRACTICE

To demonstrate the feasibility of AKI in a real-world setting, we built a prototype as a proof-of-concept system. For testing, we created a CA with OpenSSL. We pre-installed the CA and ILS root certificates on our servers and clients. We implemented an ILS server in Python that maintains an AKI Integrity Tree. The Integrity Tree node hashes were computed with SHA-256, and the root node was signed with RSA-2048. We used Coordinated Universal Time (UTC) to define precedence for domain to key mappings and to audit timeline integrity. We configured a stock Nginx HTTP server to serve our AKI-certificates, which are basically X.509 certificates with custom AKI extension fields (described in Section 4.1). We implemented our AKI client in the Chromium web browser.

ILS proof stapling. To deliver fresh ILS proofs for AKI-certificates to clients, we explored the following options. One option is to let CAs embed the ILS proof in the certificate itself, by inserting it into a certificate extension. However, once the certificate file is updated with a time-bounded ILS proof, the hash of the updated certificate would not match the original certificate recorded in the ILS Integrity Tree. To avoid this issue, another option is to let servers send the ILS proofs over the TLS handshake, utilizing a TLS extension. An alternative is to provide the ILS proof in a separate dummy certificate, appended to the leaf of the server's certificate chain. In our prototype, we sent ILS proofs via TLS extensions, given that Nginx currently supports the TLS `Certificate Status Request` extension (primarily used for OCSP stapling). This allows our server to deliver ILS proofs as a stapled response over the TLS handshake to clients without modifying Nginx. The server could use a side-loaded script to periodically fetch fresh ILS proofs and load them into Nginx. We modified the Chromium browser to extract the embedded ILS proofs via the TLS `Certificate Status Request` extension, and validate the ILS proofs.

Performance cost. AKI induces no round trip latencies (no extra network requests) to the TLS handshake. However, AKI increases the TLS handshake message size by roughly a kilobyte due to ILS proof stapling (assuming millions of domain names registered to the ILS). The ILS proof is composed from the following constituents: a list of authentication node hashes (32 bytes per node), a timestamp (4 bytes), and a root node signature (256 bytes). Further, the server's certificate size is slightly increased (by roughly 40 bytes) due to the additional custom X.509 extension.

[3] In AKI, ILSs can provide proofs of non-existence by providing the authenticated Integrity Tree leaf nodes before and after the point where the domain in question would be located at (since all the nodes are sorted in lexicographic order).

We measured the client validation processing time in Chromium on a machine with 2.26 GHz dual-core CPU and 4 GB RAM, tested with a million domains registered at the ILS. The overall AKI processing time averaged 990 μs (median = 936 μs). Specifically, the RSA verification step averaged 880 μs (median = 831 μs), while the Merkle verification step only averaged 95 μs (median = 87 μs). The overall AKI processing time is relatively small, especially compared to other approaches, such as Perspectives and Sovereign Keys that require several network round-trips to communicate with servers.[4] Another advantage of using Integrity Trees that are relatively infrequently updated means that RSA verifications on the client side are amortized, as we envision that many domains will use the same set of ILSs, hence the root of the Integrity Tree will remain the same for numerous sites. Consequently, clients mostly only perform efficient hash tree verifications and only rarely perform signature verifications.

8. RELATED WORK

Several proposals to detect malicious key changes have been proposed, such as Monkeysphere Web-of-Trust for SSL [3], the EFF's SSL observatory [13], and Certificate Patrol [4]. These proposals make it easy to detect key changes, but it is difficult to distinguish legitimate key changes from attacks.

Langley et al. [5] implemented a public key pinning mechanism in Google Chrome. The browser vendor maintains a list of trustworthy public key(s) associated with each site. Public key pinning provides similar security benefits to AKI by preventing certificates signed by rogue CA from being accepted by the browser. Typically the keys of trusted CAs are pinned, allowing for an orderly transition from one certificate to the next. To address the scalability challenges of a browser vendor maintained database, the Public Key Pinning Extension for HTTP [6] generalizes this mechanism to an HTTP header that allows a server to declare the keys that can be used in the future for that domain name. Choosing a pin duration that is too long risks a lengthy period of unavailability for the site. Furthermore, if the user is visiting the site for the first time on a device or the pin has expired, no protection is provided. By contrast, AKI provides protection on the first visit to the site.

Marlinspike and Perrin propose Trust Assertions for Certificate Keys (TACK) which pins public keys generated by the domain owners themselves [25]. More specifically, a server generates a TACK key pair, and use the TACK private key to sign the TLS public key. The TACK public key and the signature form the TACK, which clients can see in the TLS extension field, and clients "pin" the domain's TACK public key after observing the consistent TACK multiple times. Although TACK aims at removing complete trust in CAs, TACK relies on frequent visit patterns by clients to pin the domain's public key, resulting in a long initial unavailability period for *every* site. Furthermore, if a certificate becomes compromised and the pin is still inactive, the client must delete the observed TACK information. In contrast, AKI provides no initial unavailability period for any site, providing protection on the first visit to the site.

Short-lived certificates [32] in conjunction with browser vendor maintained Certificate Revocation Lists (CRLs) can reduce the impact of key compromise. This requires servers to provide certificates with a short validity lifetime and update them from the CA on a daily basis. Short-lived certificates provide similar security benefits to OCSP while eliminating the need for an online check during the HTTPS handshake. However, unlike AKI, they rely on browser vendors to somehow detect certificates that are issued by compromised CAs and block them using a browser vendor maintained blacklist.

DNS-based Authentication of Named Entities (DANE) securely binds certificates with domain names using Domain Name System Security Extensions (DNSSEC), enabling domain holders to assert certificates without reference to CAs [8]. However, the security of DANE heavily relies on the security of DNS operators.

In the following section, we perform an in-depth comparison of all the closely related certificate validation infrastructures.

9. THEORETICAL COMPARISON

In this section, we compare AKI with other proposals with respect to security, availability, and efficiency metrics. One of the contributions of this paper is to establish a set of metrics for comparison, which we present in the following subsection.

9.1 Evaluation Metrics for Comparison

Security metrics. The main security metric is Duration of Compromise (DoC): given the compromise of a private key,[5] for how long can a domain be impersonated? This metric can be specified with respect to the following cases:

- DoC after a trusted CA's private key compromise: This case also covers compelled certificates [28].
- DoC after untrusted CA's private key compromise: This case is important for AKI, where a domain defines trusted and untrusted CAs.
- DoC after trusted public log server's private key compromise: To avoid a proliferation of cases, we do not consider untrusted log server's private key compromise, as it is a strictly weaker attack scenario.
- DoC after domain's private key compromise: This metric measures the DoC, for how long an adversary can misuse the captured private key. We define the DoC as the duration during which a key is revoked or the domain bootstraps a new key which invalidates the old key, whichever is earlier.

Security guarantees of new systems can sometimes be circumvented due to required compatibility issues with legacy systems. For example, Extended Validation (EV) certificates or OCSP information in a certificate are both optional extensions, and their absence does not raise any suspicions. Therefore, even if an entity obtains an EV certificate and uses OCSP, an adversary can still obtain a fraudulent non-EV certificate without OCSP extensions that will enable MitM attacks. To measure the security of public key validation infrastructures during incremental deployment, we propose the following metric:

- Protection during incremental deployment: This is a binary measure to characterize whether any security is offered while compatibility with legacy systems needs to be ensured.

Finally, we measure privacy of client requests.

- Connection privacy: information about a client is not leaked to entities other than the contacted domain.

Availability Metrics. The main availability metric we use is Duration of Unavailability (DoU) of a domain's certificate after various

[4]A previous study indicates that round-trip latencies for OCSP lookups cost a mean of 497 ms with a median of 291 ms in real-world deployments [29].

[5]In a private key compromise, the key is disclosed to the adversary. Depending on the attack, the legitimate owner may still possess the key.

Table 1: Comparison of different public-key validation infrastructures based on the security, availability, and efficiency metrics. Entries in bold red font indicate major disadvantages of the corresponding scheme. Server* stands for the ILS, DNS, Notary, or OCSP responder server, depending on which scheme is used. Δ_U corresponds to the public log servers' update interval, which is in practice on the order of one hour. Section 9.2 describes our methodology for filling in the entries.

	CA+CRL	CA+OCSP	SLC	Key Pinning	TACK	DANE	Perspectives Convergence	SK	CT	AKI
Security										
Trusted CA compromise (compelled certificate) DoC	**days**	**days**	**days**	**days**	N/A	hours	0	0	0	0
Untrusted CA key compromise DoC	**days**	**days**	**days**	0	N/A	0	0	0	0	0
Trusted Server* key compromise DoC	N/A	0	N/A	N/A	N/A	**days**	**days**	**days**	**days**	0
Domain key compromise DoC	hours	min	days	days	**<month**	hours	day	min	∞	ILS_TIMEOUT
Protection during incremental deployment	N/A	**N**	**N**	Y	**N**	**N**	Y	Y	Y	Y
Connection privacy	Y	**N**	Y	Y	Y	Y	**N**/Y	**N**	Y	Y
Availability										
Initial registration DoU	0	0	0	0	**days**	hours	**days**	0	0	0
Planned key update DoU	0	0	0	0	**days**	hours	0	0	0	0
Unplanned key update DoU	0	0	0	0	**days**	hours	**days**	0	0	0
CA compromise DoU	**days**	**days**	**days**	**days**	N/A	**days**	0	0	0	0
Server* compromise DoU	**days**	**days**	N/A	N/A	N/A	**days**	**days**	**days**	**days**	up to 1 day
Domain compromise DoU	min	min	min	min	**<month**	hours	days	0	Δ_U	COP_UNLINKED
Efficiency										
Number of additional servers required	0	$O(C)$	$O(D)$	0	0	0	$O(C)$	$O(C)$	$O(D)$	$O(D)$
Additional latency for TLS connection setup	0	**RTT**	0	0	0	0	**RTT**	**RTT**	0	0
Additional bandwidth for TLS connection setup	0	0.1KB	0	0	0.1KB	0.1KB	**10KB**	1KB	1KB	1KB

system events. The shorter the DoU, the more available the system. We do not consider DDoS attacks in this paper; thus, we assume the general availability of all servers and communication networks.

- DoU after initial registration: This metric measures the time duration until the registered certificate becomes valid.
- DoU after planned key update: This metric measures the duration when an updated key becomes valid, which was planned to replace the current key.
- DoU after unplanned key update: In case of unplanned events such as losing a domain's private and backup keys, this metric measures the time duration when the updated key becomes valid.
- DoU after trusted CA's private key compromise: After a trusted CA's key becomes compromised, a domain's certificate may also become invalid. This metric measures the time to acquire a new certificate using the CA's new key.
- DoU after a trusted log server's private key compromise: A log server's public key compromise leads to invalid log entries. This metric measures the time to recover a log server's private key.
- DoU after domain's private key compromise: This metric measures the time until the domain's certificate becomes available with a new private key.

Efficiency Metrics. Below is a list of metrics to measure the efficiency of certificate infrastructures:

- Number of additional servers required: This metric measures how many additional infrastructure servers are required, expressed as an order in the number of new connections established. For example, if C connections are established to D different domains, would we require $O(C)+O(D)$ additional servers, $O(D)$, or even $O(1)$?
- Additional latency to establish a secure connection: Compared to standard TLS, what additional latency would be required for a secure connection with the proposed scheme?
- Communication overhead: This metric measures the additional network overhead incurred for establishing a secure connection.

In addition, we will evaluate schemes based on their ability for domains to select their trust perimeter (with respect to CAs and public log servers), as well as providing flexibility for certificate policies, such as specification to achieve different tradeoffs between availability and security metrics as defined above.

9.2 Comparison of Approaches

Based on our metrics, Table 1 compares AKI with the following proposals: CA + CRL [7], CA + OCSP [26], Short-Lived Certificates (SLC) [32], Key Pinning [5], TACK [25], DANE [8], Perspectives (P) [33], Convergence (C) [1], Sovereign Keys (SK) [9], and Certificate Transparency (CT) [21].

We now discuss the methodology we used to fill in the table. For many of the catastrophic failures, such as compromise of a trusted CA or ILS private key, we assume that a software update is required to revoke the old key and setup a new key. We assume that such a software update is secure, and can be completed within a few days for most users.

Security. For the "Trusted CA compromise (compelled certificate) DoC" metric, we assume that it will take days to push out a CA root certificate revocation message through a browser update, which was the method used to revoke DigiNotar's certificate after the compromise [30]. While some browsers use CRLs to revoke CA keys (e.g., Google Chrome), most browsers still require a software update. OCSP unfortunately does not help in this case, since CAs do not use OCSP to validate their root certificates. Similarly in the case of SLC and DANE, a browser update is required to revoke the CA key. Also in the case of Key Pinning, a browser update is required to remove the pin. Since P/C/CC, and TACK do not rely on CAs, the DoC is 0. Audit-log based schemes also protect from this case, preventing even a trusted CA from registering a new bogus certificate (the security analysis in Section 5 presents this case in more detail for AKI).

For the following properties, we explain the metrics in a less verbose manner. For the "Untrusted CA key compromise DoC" metric, the impact is less than in the previous case. In particular for DANE, the adversary cannot impersonate the domain which was possible in the trusted case.

For the "Trusted Server key compromise DoC," we consider that the ILS/DNS/Notary/OCSP responder server's private key is compromised, resulting in a severe disruption for several approaches. Since no additional third parties exist in CA + CRL, SLC, and Key Pinning, this case is N/A for those schemes. Since a compromised

OCSP server's private key does not enable creation of a fake key, DoC is 0. On the other hand, if the TACK key is compromised, recovery can take up to 30 days, depending on the domain's parameter setting. In the case of a compromised notary key in P/C/CC, we assume that a software update would require days to be fully deployed, during which time attacks are feasible. SK and CT would also require a software update, requiring days for full deployment. In AKI, a malicious ILS or malicious validator alone is insufficient to mount an attack, as we discuss in Section 5.

For the "Domain key compromise DoC" metric, we assume that browsers download CRLs every few hours; thus, the DoC for CA + CRL is on the order of hours. For SLC, it may take a few days for the certificate to expire. In TACK, it may require up to a month to have clients switch to a new key. For DANE, it may require hours until DNS entries time out and get replaced by new entries with the updated key information. In P/C/CC, depending on the client configuration, it can take days for an updated key to be consistently observed. Although the online validation of SK revocation is very fast, CT will require more time since stale validation information may be served by the adversary. For AKI, validation information is valid during domain-selected time `ILS_TIMEOUT`, which is on the order of several hours to one day, until the key is revoked.

For "Protection during incremental deployment," OCSP, and SLC offer no security, since an adversary can create a legacy certificate without any of these extensions which clients would accept. In TACK, a rollback to a compromised certificate attack is possible at the onset, when the TACK pin is not yet set up. For DANE, DNS responses may be rolled back to non-signed DNS replies. P/C/CC, SK, CT, and AKI all perform an online lookup for the case of a legacy certificate, which will reveal the legacy certificate.

"Connection privacy" is not provided by OCSP, Perspectives, and SK, as the client performs an online lookup for each certificate. Convergence, however, uses a blinding step during lookup.

Availability. "Initial registration DoU" requires several days for TACK and P/C/CC to confidently learn a new entry. In DANE, the current DNS entry needs to time out for the updated DNS entry to become available, which we estimate to take hours in the common case.

For "Planned key update DoU," we consider an optimization we discuss in Section 3, where domains pre-register a key with the notary servers, thus avoiding activation latency.

For "Unplanned key update DoU," we assume that P/C/CC use a configured policy where a key has to have been consistently observed for several days for clients to trust the key.

For "CA compromise DoU" and "Server compromise DoU," we assume that several days are required to recover and roll out new root keys. In key pinning, we assume that one day is required to push out a new software version with a new key. "Domain compromise DoU" indicates the delay required to register a new key.

Efficiency. For the metric "Number of additional servers required," we specify D for the number of domains and C for the number of connections established per day. For example, $O(D)$ indicates that the number of additional servers needs to be proportional to the number of domains.

For the metric "Additional latency for TLS connection setup," we denote a round-trip time to a server by RTT, which includes server processing time. Since P/C/CC, SK, and OCSP also involve additional external connections, they can have a significant time overhead.

For "Additional bandwidth for TLS connection setup," we list the order of magnitude of additional bandwidth required to set up a TLS connection. For the case of SK, CT, and AKI, we assume that extra signatures are on the order of 256 Bytes, hash tree values are on the order of 32 Bytes, and that a hash tree has about 30 levels, resulting in about 2 KBytes of additional information, which is on the order of 1KB as listed in the table.

9.3 Observations

As is evident from Table 1, all the newer Certificate Validation Infrastructures handle the case of untrusted CAs or CA key compromise, dramatically increasing the security over the current certificate validation infrastructure.

For practical deployment, it is critical that the TLS connection establishment does not incur any additional latency. Consequently, the additional RTT incurred by OCSP, P/C/CC, and SK is problematic. Moreover, any system requiring $O(C)$ additional server infrastructure load is likely to incur excessive cost. Performing an online per-connection lookup to an external server also challenges privacy, as it may leak information about the connection to a third-party server.

Another important factor is that certificates become immediately usable after initial registration. However, TACK, and P/C/CC do not support this feature.

Overall, CT and AKI emerge with many desirable features. Compared to CT, the overhead of AKI is lower due to the different hash tree structure, AKI can tolerate compromise of an ILS or validator server, and AKI can rapidly validate the absence of an entry. Moreover, AKI supports key revocation and re-establishment, which is a desirable feature for a public-key validation infrastructure.

10. CONCLUSION

Protecting current PKIs against CA root key compromises is becoming a topic of critical importance, as the weakest-link security model of the current PKI system is clearly too weak to provide meaningful security for critical web communication.

To improve the resilience of public-key validation infrastructures to attacks, we design the Accountable Key Infrastructure (AKI), which combines an accountability infrastructure (providing checks-and-balances on server operations and misbehavior dissemination) with key revocation mechanisms. Our AKI architecture offers flexibility for entities to select a security policy for their certificates, enabling a tradeoff between availability and security. AKI also provides tangible deployment incentives that we anticipate will help to drive adoption.

11. ACKNOWLEDGMENTS

We would like to thank Cristina Basescu, Emilia Kasper, Ben Laurie, and Steve Matsumoto for their feedback on this paper and for insightful technical discussions.

This research was supported by CyLab at Carnegie Mellon, and by support from NSF under awards CCF-0424422 and CNS-1040801.

12. REFERENCES

[1] Convergence. `http://convergence.io/`.
[2] Perspectives Project. `http://perspectives-project.org/`.
[3] The Monkeysphere Project. `http://web.monkeysphere.info/`, 2010.
[4] Certificate Patrol. `https://addons.mozilla.org/en-US/firefox/addon/certificate-patrol/`, 2011.
[5] Public Key Pinning. http://www.imperialviolet.org/2011/05/04/pinning.html, May 2011.
[6] Public Key Pinning Extension for HTTP. http://tools.ietf.org/html/draft-ietf-websec-key-pinning-01, Dec. 2011.

[7] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Technical report, RFC 5280 (Proposed Standard) Internet Engineering Task Force, 2008.

[8] M. M. Correia and M. Tok. DNS-based Authentication of Named Entities (DANE). Technical report, Universidade do Porto, 2011–2012.

[9] P. Eckersley. Sovereign Key Cryptography for Internet Domains. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD.

[10] P. Eckersley. A Syrian Man-In-The-Middle Attack against Facebook. https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook, May 2011.

[11] P. Eckersley. Iranian hackers obtain fraudulent HTTPS certificates: How close to a Web security meltdown did we get? https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https, Mar. 2011.

[12] S. Egelman, L. F. Cranor, and J. Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, 2008.

[13] Electronic Frontier Foundation. SSL Observatory. https://www.eff.org/observatory.

[14] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet Dossier. Technical report, Symantec Corporation, 2011.

[15] S. Haber and W. S. Stornetta. How to time-stamp a digital document. In *Advances in Cryptology, CRYPTO*, 1990.

[16] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard), Nov. 2012.

[17] T. H.-J. Kim, V. Gligor, and A. Perrig. GeoPKI: Converting Spatial Trust into Certificate Trust. In *Proceedings of the* 9^{th} *European PKI Workshop (EuroPKI)*, Sep 2012.

[18] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor. Transparent Key Integrity (TKI): A Proposal for a Public-Key Validation Infrastructure. Technical Report CMU-CyLab-12-016, Carnegie Mellon University, July 2012.

[19] B. Laurie and E. Kasper. Revocation Transparency. http://sump2.links.org/files/RevocationTransparency.pdf.

[20] B. Laurie and A. Langley. Certificate Authority Transparency and Auditability. http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf, 2011.

[21] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency certificate-transparency-draft. http://www.links.org/files/sunlight.html, Mar. 2012.

[22] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. http://tools.ietf.org/html/draft-laurie-pki-sunlight-07, Jan. 2013.

[23] M. Marlinspike. More Tricks For Defeating SSL In Practice. In *Blackhat*, 2009.

[24] M. Marlinspike. SSL And The Future Of Authenticity. http://blog.thoughtcrime.org/ssl-and-the-future-of-authenticity, Apr 2011.

[25] M. Marlinspike and T. Perrin. Trust Assertions for Certificiate Keys. http://tack.io/draft.html, May 2012.

[26] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. Technical report, RFC 2560 (Proposed Standard) Internet Engineering Task Force, 1999.

[27] P. Roberts. Phony SSL Certificates issued for Google, Yahoo, Skype, Others. http://threatpost.com/en_us/blogs/phony-web-certificates-issued-google-yahoo-\skype-others-032311, Mar. 2011.

[28] C. Soghoian and S. Stamm. Certified Lies: Detecting and Defeating Government Interception Attacks against SSL. http://files.cloudprivacy.net/ssl-mitm.pdf, 2010.

[29] E. Stark, L.-S. Huang, D. Israni, C. Jackson, and D. Boneh. The case for prefetching and prevalidating TLS server certificates. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[30] T. Sterling. Second firm warns of concern after Dutch hack. http://news.yahoo.com/second-firm-warns-concern-dutch-hack-215940770.html, 2011.

[31] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the USENIX Security Symposium*, 2009.

[32] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh. Towards Short-Lived Certificates. In *Web 2.0 Security and Privacy*, May 2012.

[33] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *Proceedings of USENIX Annual Technical Conference*, June 2008.

[34] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2011.

APPENDIX

A. MULTIPLE CA SIGNATURES

The X.509 certificate format was specified to contain exactly one CA signature. We discuss below two approaches to support multiple signatures for X.509 certificates.

Insert signatures into X.509 extensions. In X.509, the signature is computed from all fields in the certificate, except the signature algorithm identifier and the signature value itself. Therefore, we cannot simply inject X.509 extensions into a signed certificate, since the existing signature would not match.

Suppose that we want a total of N different CA signatures. A workaround is to (1) get $N-1$ CAs to independently compute signatures from the original certificate, (2) insert the $N-1$ signatures along with the additional signer information into custom X.509 extensions (instead of the signature field) of the original certificate, and then (3) get the last CA to sign the updated certificate (including $N-1$ signatures) and put the final signature into the signature field.

Send a dummy certificate over the TLS handshake. Without changing X.509 or existing CAs, the server can get signed certificates from each CA independently. However, sending multiple certificates for the same public key (signed by different CAs) is highly redundant. This overhead can be reduced if we extract the signatures along with the signer information, signature algorithm, etc., and store them into custom extension fields in an empty certificate at the end of an original certificate chain. An AKI-enabled browser can parse this dummy certificate for multiple signatures, while legacy SSL clients may ignore the dummy certificate and still be able to validate the certificate chain with a single CA signature.