Effective and Practical Improvements to the Web Public-Key Infrastructure

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Stephanos Y. Matsumoto

B.S., Departments of Mathematics and Computer Science, Harvey Mudd College
M.S., Department of Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May 2019

For Erika, my light and my Joy. May you rest in peace.

Acknowledgements

I would first like to thank my advisor and chair of my thesis committee, Bryan Parno, who provided unwavering support and guidance during my doctoral studies. Bryan's systematic approach to solving problems, his attention to detail, and his knack for proposing just the right solution when I got stuck were all tremendously valuable contributions, both to my development as a researcher and to my research itself. I also want to thank Raphael Reischuk and Paweł Szałachoswki, my mentors and collaborators at ETH Zürich. Their discussions, feedback, and guidance helped me hone my technical skills and my research workflow, even at the most difficult or stressful of times.

I also want to thank my thesis committee: Bryan Parno, Virgil Gligor, Vyas Sekar, Matthew Smith, and Tudor Dumitraș. Their feedback and guidance over the past several years helped me develop my thesis into a work of which I am proud. I am also grateful to Nicolas Christin, who gave me the opportunity to co-teach a new course on cryptocurrencies and helped me develop my teaching skills. Nicolas also helped me to broaden my research horizons with opportunities to collaborate on projects in the cryptocurrency space.

Throughout my studies, I was fortunate to receive support from the National Science Foundation under a Graduate Research Fellowship (grant numbers 0750271 and 0946825) and by a CyLab Presidential Fellowship at Carnegie Mellon University. This work was also supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. These sources of support provided me with the freedom to explore a breadth of research areas, several of which became the foundation for this thesis.

I want to thank my family: my parents Sab and Sonah, as well as my sisters Mina and Erika. Each of you provided comfort when I was stressed, encouragement when I was down, guidance when I was lost, and above all love when I needed it most. Your support and prayers helped me throughout my studies, and I am both lucky and grateful for all that you have provided and sacrificed for me.

My friends, in Pittsburgh, in Zürich, and around the world, provided thought-provoking discussions, fun adventures together, and a feeling of home away from home. I especially want to acknowledge Can Akın, Marie Bachmayer, George Eleftheroudis, Bert Esterhuysen, Yared Haile Selassie, Jonathan Jacobson, Nina Kamčev, Mahdi Kooshkbaghi, Mohsen Malehmir, Luka Mališa, Siniša Matetić, Alejandro Monsech Hernández, Ben Mueting, Finn Schofield, Aude Spang, Michael Taylor, Remi Tobler, Maia Valcarce, Dan Valentine, Mary Van Vleet, Thilo Weghorn, Maverick Woo,

and Tim Yee.

Finally, and most importantly, I want to thank my wife Evelyn, who was the inspiration and encouragement for me to finish this thesis. Her organizational skills kept me from losing my mind during my busiest times, and she was there by my side in both my happiest and darkest times. I am forever grateful for her partnership, encouragement, and love.

Abstract

The Web public-key infrastructure (PKI) provides a mechanism to identify websites to end users for the purposes of encrypted communication. The security of the Web PKI primarily relies on certification authorities (CAs), trusted parties whose misbehavior can enable man-in-the-middle (MITM) attacks: the impersonation of websites to users, followed by the theft or modification of sensitive information. While many methods of addressing CA misbehavior have been proposed, no solution has been both effective and practical: able to protect websites users against CA misbehavior and to be easily deployed and used by all parties involved. Thus, despite more than two decades of research advances, the Web PKI remains largely vulnerable to misbehaving CAs.

In this thesis, I argue that we can use minimal changes to existing technology to build deployable solutions that reduce the rate of successful MITM attacks in the Web PKI. Specifically, I present three projects that exemplify effective and practical approaches to improving the Web PKI. In IKP, I use the Ethereum cryptocurrency and smart contract platform to build an insurance-like mechanism that disincentivizes CA misbehavior. In CAPS, I use two global monitoring and logging systems, CT and Censys, to build a system that strengthens the existing PKI against misbehaving CAs and enables the secure incremental deployment of new and improved PKIs for the Web. In SAINT, I use the SCION future Internet architecture to propose a PKI that unifies public-key authentication for naming, routing, and end-entity public keys in a federated environment, and identify challenges and desired properties in such an environment. Through this work, I provide a first step towards making a more resilient Web PKI a reality.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 (In)Secure Communication in the Modern Web

Users and services in the modern Web send increasingly valuable data to one another. For example, users today consider it rather unremarkable to send login credentials to view emails with sensitive information, transmit financial information such as credit card details to complete an online purchase, or send highly personal search queries to a website. Due to the sensitivity and importance of this communication, users and services require *confidentiality* and *integrity*, which guarantee that only the communication endpoints can read or write the transmitted information, respectively. In the modern Web, communication is secured using the HTTPS protocol [112], which relies on the *Transport Layer Security (TLS)* protocol [113] to provide confidentiality and integrity.

While TLS is necessary to secure communication between a user and a service, it is not sufficient to do so. On its own, TLS does not provide *public-key authentication*, that is, it does not guarantee that a public key is bound to (i.e., associated with) identity information such as a domain name. Without this information, a malicious party could mount a *MITM attack*: if Alice and Bob are communicating with one another, Mallory could masquerade as Bob to Alice and vice versa. Both Alice and Bob have confidentiality and integrity in their communication with *Mallory*, but not in their communication with each other: Mallory can read and change the messages between them as she pleases. Thus in a MITM attack, not even the most ironclad guarantees of confidentiality and integrity can secure communication between Alice and Bob, as they are not directly communicating with one another.

To provide public-key authentication and thus prevent this type of MITM attack, the Web uses a *public-key infrastructure (PKI)*, which provides mechanisms for certifying and verifying the binding between a public key and identity information. In the Web PKI, trusted entities called *certification authorities (CAs)* check these bindings based on external information such as DNS records, legal business registrations, or an in-person meeting, and digitally sign *certificates*, which can be sent to Web clients. This process illustrates the fact that a CA that does not properly perform

this duty can issue a certificate that binds identity information to the wrong public key, thus enabling a MITM attack. Thus secure communication in the modern Web also depends heavily on the correct behavior of CAs.

Unfortunately, CAs have failed time and again, resulting in misissued certificates that provide opportunities for MITM attacks. These failures have occurred both at relatively small, localized CAs [56] and at the largest CAs in the market [121]. The misissued certificates themselves have been the product of (1) insufficiently diligent checks of real-world identity information [99], (2) poorly implemented certificate issuance software [128], (3) weaknesses of authority delegation in the Web PKI [83], or (4) deliberate subversion of the Web PKI for the monitoring of HTTPS connections [107].

In many of the above examples, the misissued certificates have been used to attempt MITM attacks on popular sites with large user bases, such as those owned by Microsoft and Google, and in some cases, it was possible for someone to mount a MITM attack on *all* HTTPS connections from a certain network. Worse yet, modern browsers trust over 1,500 CAs public keys spanning more than 600 organizations; any of these CAs has the authority to misissue a certificate for, and thereby enable a MITM attack on, nearly any site on the Web. These examples show that CA failures are a systemic problem with potentially disastrous consequences for secure communication in the Web.

1.2 Advances and Shortcomings in Public-Key Infrastructures

There has been a great deal of previous work that aims to mitigate the effects of CA failures and misissued certificates on the Web PKI. Much of this work takes one of three main approaches:

1. Allow misissued certificates to be quickly detected so the responsible CAs can be notified and, for popular sites with valuable information, so Web browser vendors can take action to prevent the use of the certificate [85].
2. Impose additional rules for the issuance or verification of certificates to prevent certain misissued certificates from being used in successful MITM attacks [73].
3. Introduce additional entities that provide information used to corroborate the binding between a public key and identity found in a CA-issued certificate [130].

As I describe in Chapter 2, these approaches have collectively strengthened the Web PKI and, in some cases, provided strong formal guarantees of the security of the certificate issuance process against misbehaving trusted parties [30].

Despite the advancement of previous work, however, the Web PKI still lacks a sufficiently practical and effective solution to MITM attacks. The most widely-deployed solutions do not reduce CA misbehavior or MITM attacks, and in fact often fail to even disincentivize CA misbehavior. Proposals that can prevent MITM attacks often have non-viable deployment strategies: they offer only modest security improvements if deployed gradually by participants, greatly increase the complexity of interaction among trusted parties such as CAs, or make significant changes to the operation of domains or CAs. While a small number of these proposals have been deployed, they have been error-prone in

practice, preventing successful public-key authentication *and even recovery from error* in the face of misconfiguration or the loss or compromise of a private key. In short, a major problem plaguing the Web PKI is that *there is currently no practical, deployable solution that reduces the occurrence and possibility of MITM attacks in the Web PKI.*

1.3 Thesis Overview

To address the above problem, I show in this thesis that **we can use minimal changes to existing technology to build deployable solutions that reduce the rate of successful MITM attacks in the Web PKI**. To substantiate this claim, I have designed three systems that strengthen the Web PKI while avoiding major shortcomings of previous work. Below, I describe each system and its contribution to my thesis.

1.3.1 IKP: Realigning Incentives for the Web PKI

Despite an epidemic of CA failures over the years, CAs have rarely suffered major consequences, despite putting major sites and their users at risk of MITM attacks. This inconsistency is partially due to the reluctance of browser vendors to distrust misbehaving CAs, as this removal of trust would force sites with legitimate certificates from the misbehaving CAs to obtain certificates from another CA. Due to the time, effort, and uncertainty in reporting potentially misissued certificates, it is also unlikely that a user or service that encounters a suspicious certificate will publicize it, reducing the chances that a misbehaving CA will be caught. The result is that while CAs could simply invest more in security to minimize the risk of misissuance (and thereby the occurrence of MITM attacks), they do not do so.

In IKP, I extend previous work that proposes the use of *certificate policies*, criteria specified by the site (hereafter *domain*) about which certificates should be considered authorized [126]. Specifically, I generalize certificate policies as programs that take a certificate as input, allowing IKP to use any part of a certificate to determine if the certificate is authorized. I use these policies to build an insurance-like mechanism in which upon proof of misbehavior, which can be submitted by anyone, CAs must immediately compensate domains for the incurred security risk and provide a bounty to the finder of the certificate. I then implement this system in the cryptocurrency Ethereum [131], which provides both the smart contracts used to express policy and insurance terms and a financial framework to handle payments.

IKP reduces MITM attacks in the Web PKI by realigning incentives: it imposes financial consequences on misbehaving CAs, simplifies the certificate reporting process, and publicizes information such as rewards, penalties, and liability to provide an economic signal to clients and domains. Specifically, IKP guarantees that in the absence of out-of-band payments, (1) a rational CA is better off joining IKP and correctly issuing certificates than not joining IKP or misissuing certificates, (2) a rational entity is better off reporting a misissued certificate than not reporting it or incorrectly reporting a certificate as misissued, and (3) no collusion of parties can collectively result in a profit from

the misissuance of a certificate. These incentives serve to reduce the rate of misissued certificates and thus the rate of MITM attacks, and also provide a means of punishing misbehaving CAs without resorting to complete distrust of those CAs.

IKP is provides a straightforward deployment plan: the main functionality of IKP can be deployed as a set of smart contracts, and as an insurance-like mechanism, IKP does not change existing operations in the Web PKI. Furthermore, while building IKP on Ethereum means that we implicitly trust the consensus of Ethereum miners, the deployment of IKP does not introduce a new trusted party because all functionality is implemented as smart contracts whose code and execution is recorded publicly in the Ethereum blockchain.

1.3.2 CAPS: Deployably Increasing Web PKI Resilience

Previous work at improving the security of the Web PKI has been effective *or* practical, but not both. By *effective*, I mean that a system can preemptively stop MITM attacks even in the face of multiple misbehaving trusted parties, and by *practical*, I mean that a system has a viable deployment strategy and offers a realistic scheme for recovery in the face of domain misconfiguration or lost private keys. For example, while IKP is a practical system, it only provides a way to find and punish misbehaving CAs, and does not seek to prevent MITM attacks as they occur.

While previous work has succeeded in designing solutions that can provably prevent MITM attacks in the face of misbehavior by multiple CAs [30], such solutions have notable weaknesses under realistic deployment models. Because the entire Web is unlikely to adopt any solution overnight, any improvement to the Web PKI must assume an *incremental* deployment model, that is, a model in which the existing Web PKI and any improvements (or entirely new PKIs) coexist for a likely long period of time. In such a model, *downgrade attacks* are possible: consider an attacker Mallory who has obtains a misissued certificate for Bob's domain. Using a downgrade attack, even if Bob has deployed a PKI improvement, Mallory can convince a user Alice that Bob's site has not deployed the improvement, and thus carry out a MITM attack. Thus the integrity of a domain's deployment status itself must be protected just as much as public-key certificates, a challenge that I call the *signaling problem*.

Moreover, for security purposes, this previous work, as well as solutions that have been deployed in the Web [58], make it difficult for domains to change their public keys. Unfortunately, in the event a domain configures the wrong public key, or loses its private key due to catastrophic failure or compromise, this difficulty in changing keys renders the domain inaccessible. In particular, existing solutions require the domain to wait for a specified "cool-off period" or contact one or more trusted authorities, who are typically slow to react. At the same time, making it easy to change a domain's public key seems to run counter to the concept of effectively preventing MITM attacks. This problem of preventing MITM attacks in the face of multiple misbehaving trusted parties while simultaneously maintaining the ability to easily change public keys is a challenge I call the *recoverability problem*.

In CAPS, I address the signaling problem by leveraging the fact that global certificate monitoring systems [85, 52], many of which are referred to as *public logs*, keep track of nearly all certificates accepted by major Web browsers. In particular, I design *log aggregators*, who maintain a list of this global view over time to determine which sites have deployed HTTPS. Moreover, I introduce a simple certificate policy in which domains obtain multiple certificates for the same public key in order to denote that certain public keys bound to their name should be preferred over others. This information is sent in a succinct encoding to client browsers. The actual certificates are sent in an extension to the standard TLS handshake [55], making the scheme backwards-compatible with the existing Web PKI.

CAPS is effective in reducing the rate of MITM attacks, both in the current Web PKI and during the deployment of a new PKI. The way in which CAPS addresses the signaling problem prevents downgrade attacks, both from a PKI improvement to plain HTTPS and from HTTPS to HTTP (which would bypass the PKI improvement entirely). Furthermore, the way in which CAPS addresses the recoverability problem allows each domain to gain n certificates for the same public key, preventing against MITM attacks for that domain in spite of $n-1$ misbehaving trusted entities such as CAs. CAPS is also practical because it leaves the existing certificate issuance process unchanged. This design choice means that CAs do not need to do anything for clients and domains to begin using CAPS. Furthermore, by simplifying the policy to only consider how many certificates a domain has for each public key, recovering from misconfiguration or a lost private key is as simple as gaining n certificates for a new public key.

1.3.3 SAINT

Even though IKP and CAPS provide deployable solutions to reducing the rate of MITM attacks in the current Web PKI, they implicitly assume a CA ecosystem almost identical to that of today: browsers and operating system (OS) vendors provide a set of public keys of trusted *root CAs*, which delegate authority to other CAs, and so on. Each of these CAs can issue certificates for any site in the Web, and though previous work has endeavored to mitigate MITM attacks that stem from this global authority [73], the use of a near-globally common set of near-unrestricted roots of trust has proven problematic. Unfortunately, simply moving to a world in which different users may have different roots of trust comes with its own challenges, the most notable being how two users with different roots of trust can authenticate one another's public keys.

In SAINT, I address this challenge by noting that in a world with distinct "realms" of trust (which I refer to as isolation domains because they can isolate the failures within their realm from others), each with its own root CAs and other trusted entities, each realm must still be able to communicate with the others using a series of internetwork connections found via a routing algorithm. The inherent business or political relations underlying these routes provides a natural starting point to build a graph of trust among these ISDs. The public keys of each ISD's trust roots can be sent along these routes as well, providing relatively rapid updates of trust root information. I then show that this design

limits the scope of a misbehaving root of trust (or delegated trusted entity) while allowing users to authenticate each other's public keys, even when they are physically present in a different ISD from where their trust roots are located. I then instantiate this design using the SCION future Internet architecture [134].

SAINT provides a viable deployment strategy based on existing technology, both by considering the existing basis of routing relationships and by presenting an instantiation with a future Internet architecture whose deployment is ongoing and increasing [28]. However, SAINT primarily aims to anticipate the future of the Web PKI and distill the fundamental properties required for authentication in a holistic and federated enviroment. SAINT is thus orthogonal to IKP and CAPS: it does not aim to reduce MITM attacks in the current Web PKI, and it can be augmented with IKP and CAPS to provide greater resilience against misbehaving CAs. SAINT also demonstrates how a future Internet architecture can add a further layer of recovery to the Web PKI: while CAPS makes it easy for a domain to change its public key in the face of misconfiguration, SAINT allows a trust root (e.g., a CA) to do the same.

1.3.4 Contributions

To summarize, in this thesis I make the following contributions:

- In designing and implementing IKP, I show that by leveraging smart contracts, we can realign incentives to deter and detect misissued certificates without changing the existing Web PKI.
- In designing and implementing CAPS, I show that by leveraging public certificate logs, we can increase the resilience of the Web PKI against misbehaving CAs with minimal changes to existing entities and with recovery from misconfiguration or lost private keys.
- In designing SAINT, I show that by leveraging the business relationships inherent in internetwork routing, we can maintain the resilience of a PKI, even in a federated model without common roots of trust.

Together these contributions show that minimal changes to existing technology can indeed create practical, effective improvements to the Web PKI.

Chapter 2

Background and Related Work

In this chapter, I provide an overview of the Web PKI literature related to my work in the remainder of the thesis. Rather than providing a comprehensive treatment of the literature in this space, my goal is to provide a foundation of, and context for, the insights and techniques I describe in later chapters.

I begin by briefly explaining foundational features of the current Web PKI. I then describe technical approaches seen in the literature that aim to improve the security of the Web PKI. I organize these approaches into three classes: (1) those that aim to detect instances of potential CA misbehavior, (2) those that augment the existing PKI, leaving current CA operations unchanged, and (3) those that change or replace existing CA operations. I then describe primarily non-technical efforts to understand and realign incentives in the Web PKI as a means of reducing the rate of CA misbehavior. Finally, I describe PKIs beyond that of the Web, along with the challenges they face, how these challenges compare to those of the Web PKI, and proposed solutions to these challenges.

2.1 Overview of the Web PKI

I begin with an overview of four foundational facets of the Web PKI. I first describe the information contained in public-key certificates. Next, I describe the CA ecosystem, including the certificate issuance process, the CA hierarchy, and organizational bodies. I then list instances of CA failures observed over the years, along with the causes and impacts of these failures, where they are known. I conclude with a discussion of certificate revocation. While revocation is not a focus of my thesis, it is an example of a relatively simple fix to CA misbehavior that is rarely used due to various hurdles, both technical and non-technical.

I do not intend for this section to constitute a detailed overview of the entire Web PKI. For more thorough coverage, I refer the reader to works such as those by Gutmann [63], Clark and van Oorschot [44], or Yu and Ryan [132].

2.1.1 Public-Key Certificates

The format for public-key certificates used in the Web PKI follows Version 3 of the ITU-T X.509 standard [5], typically abbreviated X.509 v3, with additional Web-specific clarifications in an IETF RFC [46]. According to these standards, a public-key certificate contains a minimum of (1) metadata, such as the version of the certificate standard and a CA-issued serial number, (2) the validity period of the certificate, (3) identity information of the issuer (CA), (4) identity information of the subject (domain), (5) the public key of the subject, and (6) a signature by the issuer. The subject identity information is typically given as a DNS name, which, for example, allows a client connecting to a site to verify that the public key in the certificate is indeed bound to the DNS name of that site.

X.509 v3 certificates support extensions, which provide additional information that clients can use during identity verification. In the Web PKI, widely-used extensions include Basic Constraints (which allows delegation of CA powers), Subject Alternative Name (which allows multiple names to be bound to the same public key), Server Name Indicator (which allows multiple HTTPS sites to reside at the same IP address), and Authority Key Identifier (which indicates a fingerprint of the issuer's private key) [46, 55]. Perhaps the most influential extension has been the Certificate Policies extension [46], which provides support for *Extended Validation (EV) certificates*. These certificates represent additional, stringent identity checks performed by the CA during issuance. Web browsers only recognize EV certificates issued by a small subset of trusted CAs [20].

2.1.2 The CA Ecosystem

When issuing certificates, CAs are responsible for ensuring that the information in the certificate is correct. Typically, a domain sends the CA a *certificate signing request (CSR)*, which is effectively the certificate that should be issued to the domain, but self-signed to show that the domain has control over the private key corresponding to the public key in the certificate [106]. The CA further verifies that the requesting entity controls the domain name(s) in the certificate [19]. For example, the CA may send a confirmation email to a specified address at the requested domain name, or, in the case of an EV certificate, send an employee to physically meet with a representative of the requesting entity [20]. Each CA has a *certification practice statement (CPS)*, which is a legal, non-technical document used to describe how the CA verifies the subject identity and the procedural steps for issuing certificates.

As mentioned in Section 2.1.1, the Basic Constraints extension allows for the delegation of CA powers. This delegation system has led to a hierarchy of CAs in the Web PKI that begins with the set of root CAs public keys configured by browser or OS vendors. The hierarchy of CAs stems from business rather than technical considerations; in almost all cases, a CA can issue a certificate for any domain in the Web, greatly increasing the fragility of the Web PKI [109].

The standards for CA operations are set by the CA/Browser Forum, a collection of Web browser vendors and CAs.

The CA/Browser Forum has produced two major standards: a set of Baseline Requirements that all trustworthy CAs issuing public-key certificates for the Web should follow [19], and a set of EV Guidelines that all CAs should follow when issuing EV certificates [20].

2.1.3 CA Failures

Unfortunately, despite the standards and requirements above that govern CA operations, CAs have failed repeatedly, misissuing certificates that are often subsequently used in MITM attacks. As I describe below, these misissuances have resulted from a diverse range of failures in the CA operational process and have affected both large and small CAs across the world. The diversity of these CA failures shows that certificate misissuance is a systemic problem in the Web PKI.

One way in which CAs have misissued certificates is to perform insufficiently stringent identity checks on a subject requesting a public-key certificate. For example, in 2001, VeriSign mistakenly issued code-signing certificates to someone who claimed to be a Microsoft employee [99]. In 2009, a researcher obtained a certificate for `live.com` by using the publicly-available email address `SSLCertificates@live.com` [136].

CAs have also misissued certificates by incorrectly configuring their signing infrastructure, often by using test configurations in production. For example, in 2013, the CA TURKTRUST mistakenly issued two CA certificates to entities that should have received a leaf certificate [81]. TURKTRUST claimed that this was due to certificate issuing profiles that were intended for testing but were used in production [128]. In 2015, Symantec, the CA with the largest market share at the time, issued unauthorized certificates for several domains [123]. An audit revealed that Symantec had issued 164 certificates for over 76 domains and almost 2,500 certificates for unregistered domains [121]. As a result of this failure, Google made the decision to remove Symantec as a trusted root CA from Google products [122].

Some certificate misissuances have resulted from compromises of a CA's network or administrative accounts. In 2011, a reseller account for the CA Comodo was compromised and used to issue fraudulent certificates for Google, Yahoo, Mozilla, and various Microsoft sites [3]. Later in 2011, the internal network of the CA DigiNotar was breached, resulting in more than 200 unauthorized certificates for more than 20 different domains [68]. Due to the way that DigiNotar subsequently handled the breach, Mozilla removed DigiNotar as a trusted CA [104] and the CA later declared bankruptcy [4].

One particularly dangerous CA failure is when CAs misissue certificates *on the fly* to mount MITM attacks. These attacks often arise from good intentions, such as using privately configured CAs to prevent data exfiltration from networks that handle sensitive information, but instead enable MITM attacks against any client that receives the misissued certificates, even those outside the intended networks. For example, in 2012, TrustWave sold a CA certificate to an enterprise client who used the corresponding private key to sign fraudulent certificates for any site accessed from the

client network [107]. In 2013, an intermediate CA rooted in the French CA ANSSI used its certificate to surreptitiously intercept and inspect encrypted traffic from a network [82]. In 2015, an Egyptian CA called MCS Holdings installed its CA private key in a MITM proxy device that could be used to intercept arbitrary connections [83].

Some CA failures have been the result of deliberate, malicious behavior rather than an error. For example, in 2014, Lenovo shipped some of their products with the Superfish software preinstalled [8]. Superfish installed a root certificate on victim machines that allowed the software to generate certificates for MITM attacks on the fly for any site a user was visiting [61]. While Superfish was intended to inject ads into HTTPS webpages, it allowed the interception of all encrypted Web communication from affected devices. Sennheiser was the center of a similar scandal in 2018, installing a root certificate as part of its management and control software for its audio products [77]. Finally, the CA WoSign was slated for removal from major browsers for two major failures [89]: 1. it was found backdating its certificates in response to browser deprecation of certificates issued with SHA-1 hashing in the signature, and 2. it did not disclose the fact that it had acquired StartCom, another relatively large CA. A salient insight from these deliberate attacks is that CA misbehavior can be *profitable*, and thus this systemic failure of CAs is a phenomenon that will likely continue to occur in the future.

2.1.4 Certificate Revocation

A certificate, even a misissued certificate, is considered valid until its expiration, and thus a misissued certificate can continue to be used for MITM attacks after it has been discovered. One method of mitigating these attacks is through *certificate revocation*, that is, the invalidation of a certificate before its expiration. In the current Web PKI, a certificate can only be revoked by the issuing CA via a signed message, and thus a CA that has deliberately issued an unauthorized certificate is unlikely to revoke the certificate even if discovered.

Unfortunately, the distribution of these revocation messages has proven to be a major challenge in the Web PKI. The earliest approach to distributing revocation information was through *certificate revocation lists (CRLs)*, in which each CA periodically published lists of serial numbers of revoked certificates, along with the time at which each certificate was revoked [69]. Unfortunately, these lists can only grow, and with the scale of certificates being issued in the Web, even distributing only the differences from the previous list has proven problematic for CAs. Another approach is *short-lived certificates*, in which CAs issue certificates with short validity periods, minimizing the window of time in which a misissued certificate can be used [114]. CAs have been seemingly reluctant to use this approach, perhaps due to the far more frequent certificate issuances that are required. Yet another is approach is the *Online Certificate Sptatus Protocol (OCSP)*, in which clients can contact an additional OCSP server to check whether a certificate has been revoked [116]. The additional latency incurred by this check can be minimized by using OCSP stapling [55], an extension to the TLS protocol in which the client requests this information from the domain itself.

Recent improvements in certificate revoation have focused on making the distribution of revocation information quicker and more comprehensive. For example, the Chrome browser uses CRLsets [80], which push CRLs containing misissued certificates for the most frequently visited sites to clients using the browser's update mechanism. This solution, however, clearly cannot scale to the entire Web. Thus CRLite [84] proposed to use filter cascades [115], a data structure based on Bloom filters [35], along with the global view of Web certificates provided by monitoring solutions (described in Section 2.2), to compactly represent the set of all revoked certificates in the Web.

Despite these improvements, certificate revocation remains a challenging problem in PKIs. With the diversity of approaches to revocation still in use, clients must obtain the revocation method used as well as the information necessary to check the revocation status of the certificate.

2.2 Detecting Potential CA Misbehavior

I now describe work that aims to decrease the incidence of MITM attacks by detecting instances of potential CA misbehavior. Approaches in this area range from simply publicizing certificate issuances to using external information to highlight possible MITM attacks with varying levels of confidence. Thus, the work in this area focuses on finding possibly misissued certificates and does not necessarily propose steps to prevent MITM attacks using these certificates.

Many of the approaches used are simple and easy to deploy: they rarely introduce new parties or change existing protocols for certificate issuance, and are often implemented as browser extensions that can be installed and used immediately. However, these solutions also suffer from serious drawbacks: (1) they often cannot detect a misissuance with certainty, leading to false positives, and (2) because they focus on detection, there is usually a window of time in which a misissued certificate can be used for MITM attacks.

In this section, I describe three major approaches to detecting CA misbehavior: (1) global certificate monitoring, (2) using past CA behavior, and (3) using domain-provided information.

2.2.1 Global Certificate Monitoring

Global certificate monitoring systems aim to find potential misbehavior by publicizing large numbers of certificates. Censys [52] is a search engine that leverages large-scale scanning of the Internet to find certificates, primarily by using ZMap [53], which can scan the entire public IPv4 address space in around 45 minutes. One data set that Censys provides is the collection of certificates received from TLS handshakes, some dating back to 2013. Google's Certificate Transparency (CT) project [85] uses *public logs*, entities that maintain timestamped records of certificates issued by browser-trusted CAs. The integrity of these records can be proven using a system based on Merkle Hash Trees [47], a data structure first used for an efficient digital signature algorithm [98]. Clients using CT do not accept a certificate unless it is accompanied by one or more *signed certificate timestamps (SCTs)*, which prove that a public log has

recorded the certificate. Thus any certificate used in a successful MITM attack on a client using CT can be detected as potentially misissued by someone auditing the public logs. Because Censys and CT provide a near-complete view of the known Web certificate ecosystem, they have formed the basis of a variety of studies [129] and improvements to the Web PKI [84].

2.2.2 Using Past CA Behavior

CAge [73] aims to find potential CA misbehavior by examining the top-level domains (TLDs) in certificates issued by a CA. In particular, a certificate is flagged as suspicious if the CA has never issued another certificate for a subject with the same TLD. While this approach would have prevented several past instances of CA misbehavior, it suffers from the possibility of false positives, particularly for new CAs. As another approach, Perl *et al.* find that few root CAs trusted by browsers and OSs have ever issued certificates, and recommend distrusting these unused CAs for HTTPS, thus flagging *any* certificate issued by these CAs as suspicious [108]. While this approach reduces the attack surface of the Web PKI, it would not have prevented any previous instances of CA misbehavior, as these all involved CAs who had issued certificates before.

2.2.3 Using Domain-Provided Information

For a given certificate, the domain named as the subject is the only entity who knows whether the issuance of the certificate was authorized. Therefore, the domain can provide valuable information for finding potential misissuances by specifying criteria for certificates it has authorized. One way of doing this is through the *Certification Authority Authorization (CAA)* record in Domain Name System (DNS) [64], which allows a domain to specify a set of CAs authorized to issue certificates for that domain. Before issuing a certificate for the domain, a CA checks this record to ensure that it has been explicitly authorized to issue the certificate. While CAA is becoming more widely used due to its simplicity, this approach suffers from two major flaws: 1. it implicitly relies on DNSSEC [23] to protect the integrity of the CAA record itself, and 2. a malicious CA can simply ignore the record, thus relying on external entities to monitor both certificates and CAA records to find misissued certificates.

2.3 Augmenting CA Authentication

In this section, I describe approaches that aim to reduce MITM by augmenting the existing CA ecosystem. Thus these approaches leave the certificate issuance process unchanged, and in general, add extra information that can be used to corroborate the information in certificates issued by CAs. The client then uses this extra information to determine whether or not to accept a certificate presented by a domain.

The extra information in these approaches is typically distributed through publicly accessible locations such as an external server. The information may be certified by the domain or third party, or simply assumed as *trust on first use (TOFU)*. As with OCSP, the distribution of this extra information and the deployment of these approaches comes with challenges. All of these approaches face the signaling problem, that is, the challenge of proving to the client that a domain has deployed a new approach (and thus has distributed extra information for the verification of a certificate). If the information is distributed by a third party, then an adversary can block connections to this third party, forcing the client to choose between simply accepting a domain's certificate (thus facing a potential MITM attack) or rejecting the connection (thus rendering the domain inaccessible). For TOFU solutions, some clients face the risk of MITM attacks between them and a domain if receiving an unauthorized certificate on their first connections to that domain.

2.3.1 Domain-Certified Information

As described in Section 2.2.3, domain-certified information has proven useful in solutions because only the domain knows which certificates or public keys it has authorized to be bound to its name. Clients can use this information to reject suspicious certificates, thereby preventing a successful MITM attack. For example, the DNS record created for CAA (also described in Section 2.2.3) can be used at verification time to check whether the domain authorized the certificate.

In other systems, domains provide public keys whose corresponding private keys are used during certificate verification. For example, in PoliCert [126], domains specify policies that include criteria such as authorized CAs or public keys, maximum certificate chain length, and the use of wildcard certificates. These policies can be enforced on DNS subdomains, offering some protection during incremental deployment.

Solutions that use domain-certified information face the risk of misconfiguration. For example, a domain that misconfigures its policy in PoliCert may become unable to quickly update its policy, its keys, or its certificates, instead relying on a longer "cool-off period" imposed by the system, as described in AKI [75].

2.3.2 Externally-Certified Information

The use of externally-certified information (i.e., not certified by the domain or CA) avoids the misconfiguration pitfalls of using domain-certified information. Externally-certified information can also prevent MITM attacks in a relatively common scenario: an adversary obtains an unauthorized certificate for a domain and uses this certificate to carry out targeted MITM attacks on certain clients. Thus solutions that rely on externally-certified information typically assume that a MITM attack only targets certain clients and then relies on a global view to prevent these attempted attacks.

In Perspectives, a system of *notary servers* periodically establish connections with domains and note the public keys presented by these domains. Clients can then contact some of these notary servers in parallel with establishing

a TLS connection with a domain, and only proceed with the connection if a threshold number of notary servers corroborate the public key that the domain presents to the client [130]. In Convergence, clients' queries to these notary servers (i.e., which domains' public keys are being requested) are kept private: clients forward encrypted queries to a different notary server, who then forwards this to the desired notary and relays the result back to the client [91].

Solutions that use externally-certified information suffer from several critical shortcomings. These solutions require the use of several external sources of information, yet there are generally a low number of providers of this information in practice. In part, this is because there are no incentives for these external entities to operate; they are often encouraged to do so for the greater good of the system, but receive no compensation for the resources they use during operation. Additionally, some entity must typically be trusted to maintain a list of the providers of this information. For example, a browser extension for Perspectives may include a list of suggested notary servers, and in CT (described in Section 2.2.1), Google maintains a list of trusted public logs.

2.3.3 Trust on First Use

TOFU solutions are typically simple, yet provide protection against MITM attacks in many common scenarios. In TOFU solutions, the client assumes that its first connection with a domain is benign, that is, not an attempted MITM attack. With this assumption, a client can store the fact that a domain uses a certain public key or CA in its authorized certificates. A similar system called HSTS [66] was used to signal to clients that they should connect to certain domains over HTTPS.

In terms of protecting against MITM attacks, HTTP Public Key Pinning (HPKP) stores public keys for each domain on a TOFU basis for a period of time determined by the domain [58]. This information is sent through HTTP, and clients connecting to the domain in the future additionally validate that the domain's presented public key matches the pinned one. The pinned public key can also be used for subdomains. Even if a misbehaving CA issues an unauthorized certificate for a domain, HPKP can protect against MITM attacks for all clients that have connected to that domain before and still have the domain's legitimate public key pinned. Unfortunately, solutions such as HPKP will render a domain inaccessible if the domain loses access to the private key corresponding to its pinned public key. Moreover, an adversary can carry out a *hostile pinning* attack: by pinning clients to an unauthorized key for a long duration, the adversary can continue to mount MITM attacks for all such clients even after the misissuance has been discovered. To mitigate this, solutions such as HPKP typically set a maximum on how long any key can be pinned.

2.4 Replacing CA Authentication

In this section, I focus on approaches that aim to reduce MITM attacks by replacing traditional CA-based authentication or certificate issuance systems. Some approaches that simply aim to augment existing CA-based authentication,

such as Perspectives or Convergence can be extended to replace traditional CA authentication, but currently still rely on CA-issued certificates. The approaches I describe here are largely not compatible with the existing PKI.

Solutions that replace the existing CA system provide strong protection against MITM, either because they make it difficult for a single compromised CA to enable a MITM attack, or they replace CAs entirely. Some solutions have even gone as far to provide formal proofs of their increased security. However, to date, none of these solutions have seen widespread deployment due to the massive operational changes required for clients, domains, and CAs. Additionally, some solutions make larger changes that severely disrupt deployment; for example, changing the namespace from the existing DNS presents challenges for migrating existing domains to the new namespace.

Below, I describe the two major classes of solutions that aim to replace CA authentication: those based on CAs (but with a different issuance protocol) and those based on trusted entities that are not traditionally CAs.

2.4.1 CA-Based Solutions

To my knowledge, all of the CA-based solutions in the literature rely on public logs similar to those in CT. In AKI [75] and its successor ARPKI [30], policy information similar to that used in PoliCert conveys information including which CAs are authorized to issue certificates for a domain. However, in AKI and ARPKI, this information is included in the certificates themselves. Because certificates in both AKI and ARPKI rely on signatures by multiple CAs, the certificate issuance process must be changed, though AKI offers a slightly weaker but backwards-compatible solution by leveraging X.509 extensions. The security guarantees provided by ARPKI are strong and formally proven: ARPKI can prevent MITM attacks in the presence of n compromised trusted CAs, where n is a systemwide parameter. Unfortunately, the deployment of ARPKI requires n CAs to cooperate (and in doing so, deploying ARPKI). Moreover, both AKI and ARPKI face the signaling problem (described in Section 1.3.2.

2.4.2 Non-CA-Based Solutions

Similarly to solutions discussed in Section 2.3, solutions that aim to replace CAs with other sources of information about a domain's public key distribute this information via TOFU mechanisms, the domain, or external entities.

In Trust Assertions for Certificate Keys (TACK), CAs are replaced with a TOFU mechanism very similar to that of HPKP [92]. Rather than using this mechanism to pin the public keys used in the TLS handshake, a domain specifies a *TACK Signing Key (TSK)* that certifies all keys used in TLS handshakes with that domain. This effectively makes a domain its own CA, and the pinning mechanism is used to establish this "CA key."

In DNS-based Authentication of Named Entities (DANE), domains use DNS records to communicate information about their public keys [67]. Domains create a record with type TLSA which contains information, including public keys or CAs to expect from the domain. Thus DANE is compatible with the existing CA system, but also provides a

way to "pre-pin" a public key to a domain. DANE offers strong protection against MITM attacks, since an attacker would have to alter the TLSA record to succesfully complete any MITM attempt, and moreover, domains in DANE that lose access to a private key can easily change their own TLSA record to avoid becoming inaccessible to clients. However, DANE implicitly relies on DNSSEC to protect against MITM attacks, and thus its deployment implicitly relies on that of DNSSEC as well. Because DNSSEC is not yet widely deployed, the deployment of DANE has similarly narrow in scope.

Domains can also use a variety of external entities to convey public key information. Sovereign Keys takes a similar "pre-pinning" approach to DANE, but rather than store the information in DNS, introduces a new entity called a *timeline server* to store this information [57]. The timeline server is a precursor of public logs in CT and maintains an append-only list of bindings between a domain's name and a *sovereign key* used to certify all of the domain's public keys. In Namecoin[1] and Blockstack [22], a domain's public keys are stored in a blockchain. As with DANE, these solutions rely on the deployment of their underlying technologies and thus have not seen widespread use.

Namecoin and Blockstack, which introduce a new namespace, face the challenge that particularly meaningful or valuable names in DNS may be claimed in the new namespace by entities other than those that own the name in DNS. Indeed, a majority of the names in Namecoin seemed to be held by speculators, and only a few names pointed to sites with substantive content [72]. Sovereign Keys proposed using certificates or DNS records to establish TSKs, which could also work in Namecoin and Blockstack, but this approach may then increase the occurrences of certificate misissuance in the Web PKI, as malicious parties attempt to gain control of potentially valuable names in a new namespace.

2.5 Realigning Incentives in the Web PKI

While the previously described solutions have focused on technical approaches to improving the Web PKI, there have also been primarily non-technical measures that aim to understand and improve the underlying incentives of the Web PKI. By designing mechanisms that decrease the benefit of a misissued certificate or increase the cost of successfully mounting a MITM attack, we can naturally discourage CA misbehavior, and by decreasing the cost and work required to deploy these mechanisms, we can naturally encourage deployment.

The need to realign incentives in the Web PKI has been documented in the literature. Economic study of the Web PKI has shown that incentives are woefully misaligned, both from a market and a security perspective [26]. Moreover, while simply revoking misissued certificates or misbehaving CAs can prevent MITM attacks, these approaches have not been adopted due to insufficient incentives. PKISN is a system motivated by the "collateral damage" that occurs when legitimate certificates issued by a misbehaving CA are invalidated because the CA was revoked as a trusted

[1] `https://namecoin.info`

entity [125].

While the literature on incentives in the Web PKI is somewhat scant, it has produced several valuable observations regarding the viability of improvements to the Web PKI. An early analysis of metrics for public-key authentication proposed the use of CA-offered insurance as an incentive-oriented solution to evaluating confidence in a public-key certificate [111]. My work on Certificates-as-an-Insurance (CaaI) examined this idea in the context of the Web PKI, proposing the use of an insurance-like scheme to realign misaligned incentives [93]. As motivation for this approach, I observed that CAs largely act as the arbiters of public-key validity in the Web without suffering the consequences of MITM attacks, further corroborating the lack of incentives in the Web PKI. Regarding deployment, I qualitatively analyzed the mutual influences among clients, domains, CAs, and browser vendors in the Web PKI, and concluded that deployment of improvements to the Web PKI, particularly those using public logs, is best driven by browser vendors and CAs [96]. These observations provide support for the idea that realigning incentives can be beneficial for both security and deployment in the Web PKI.

2.6 PKIs Beyond the Web

While I have focused on research challenges in the Web PKI, other applications fundamental to the Web rely on PKIs. The technical challenges in these PKIs offer valuable insights into general PKI design, and highlight problems that we may have to consider as the Web PKI continues to evolve. Therefore, in this section I describe PKIs for several other technologies: naming, routing, end-entity public-key authentication (hereafter simply *end-entity authentication*), and code signing. I also discuss the challenges of unifying multiple PKI, primarily by describing SCION, a future Internet architecture whose public-key authentication component seeks to unify PKIs for naming, routing, and end-entity authentication.

2.6.1 Naming

By default, records in DNS are not authenticated and can thus be forged by attackers such as malicious DNS resolvers in a class of attacks called *DNS hijacking*. As with the hostile pinning attack described in Section 2.3.3, DNS hijacking can be particularly vexing because DNS responses are typically cached for performance, meaning that a hijacked resource record could persist long after the attack is discovered, an attack called *DNS cache poisoning*. In an effort to prevent these attacks, DNSSEC [23] provides new resource records [25] and modifies the DNS protocol [24] to protect the integrity of DNS records for domains. These records include signatures on sets of resource records as well as delegation of signing authority to others. Similarly to verification in the Web PKI, clients check a chain of signatures on resource records, anchored at the *DNSSEC root key-signing key (KSK)*.

The nature of DNS has played a crucial part in the design of DNSSEC, which has highlighted several important challenges of designing and deploying a PKI for naming. Because the Internet Corporation for Assigned Names and Numbers (ICANN) manages the DNS root zone, it also manages the DNSSEC root KSK. Thus DNSSEC follows the *monopoly trust model*, in which a system has only a single root of trust [109]. While ICANN, previously controlled by the US Department of Commerce, has taken steps to diversify both its governance [21] and control of its root key [51], the root KSK remains a single point of failure for DNSSEC. Because naming inherently aims to present a globally consistent mapping to clients, it is likely that any PKI for naming will face the challenge of protecting such a root key and providing mechanisms to recover from catastrophic loss or compromise of this key.

In both design and practice, DNS has also influenced the major challenges in the deployment of DNSSEC. Deploying DNSSEC is as simple as creating the appropriate resource records. However, the hierarchy of delegation from the DNSSEC root KSK follows the hierarchy of DNS zones, meaning that only zones whose parent zones have deployed DNSSEC can themselves serve DNSSEC records. While much of the global TLDs in DNS have been signed and thus support DNSSEC in theory [119], only a tiny fraction of DNS records are actually validated with DNSSEC [43]. Moreover, the use of DNS registrars to manage DNS and DNSSEC, as well as the massive outsourcing of DNS nameservers [110], have made support for DNSSEC mediocre at best, both for servers and clients [42].

2.6.2 Routing

Internet routing is secured by BGPsec [87], which uses chains of signatures to protect the integrity of internetwork routes. BGPsec uses the *Resource Public Key Infrastructure (RPKI)* to identify the networks, or *ASes*, that create and distribute these routes, and to certify ownership of public IP address space [86]. Specifically, the RPKI uses *resource certificates* [70] that follow the X.509 standard and bind a public key to a set of IP addresses or to an *AS number (ASN)* via certificate extensions [88]. As with DNSSEC, the root of this PKI is a branch of ICANN called the Internet Assigned Numbers Authority (IANA). Because RPKI also follows the monopoly trust model, its root key is similarly a single point of failure for the system.

RPKI also suffers from challenges unique to routing PKIs. To verify routes and resource certificates, ASes must fetch additional information from RPKI repositories. However, to reach these repositories, ASes must use routes that may be unverified, leading to a circular dependency. This problem is exacerbated by the risk of *whacking*, which occurs when revocations of resource certificates invalidate entire subtrees in the RPKI hierarchy [45]. The perils of the hierarchical nature of RPKI and its use for routing (and thus for communicating with other entities) illustrate that routing PKIs must pay particular attention to revocation and updates of information.

2.6.3 End-Entity Authentication

While the Web PKI is the most widely used PKI for end-entity authentication, it overlooks some challenges in designing a PKI for end-entity authentication. For example, CONIKS [97] presents desirable properties of *identity providers* such as those that certify users' public keys in end-to-end messaging applications: 1. ability to monitor keys bound to one's own name, 2. privacy with respect to other users of the same identity provider, and 3. privacy with respect to how many users a provider has. Thus in a system such as CONIKS, many of the solutions specific to the Web PKI, such as those that rely on a global view of all certificates, may not be feasible. Moreover, end-entity authentication may see a future in which many identity providers exist, leading to a system in which a user may have public keys at many such providers and must "link" these keys in some way.

Unfortunately, work on negotiating authentication among a federated system with multiple namespaces and providers has stagnated in recent years. Previous approaches have included constructing a path of authentication through namespaces [34], using delegated authority to reason about a subject's authority [79], and strategically placing peer links of trust between hierarchies [60].

2.6.4 Code Signing

Code signing is used primarily by OS vendors to certify the identity of software publishers, thus allowing users to have greater confidence in the software shipped by these publishers. Code-signing PKIs are thus hierarchies of certificates rooted at a OS vendor. Unfortunately, these PKIs have long been understudied and poorly understood. Recent work has endeavored to study security incidents that arise from incorrect behavior in these code-signing PKIs [74], but this comes with challenges: in particular, there is no widely-known set of "root code-signing certificates" and no known way of scanning a network device to find code-signing certificates.

2.6.5 Unifying PKIs

Scalability, Control, and Isolation On Next-generation networks (SCION) is a future Internet architecture that aims to improve interdomain routing in the Internet [134]. SCION divides the Internet into routing units called *trust domains (TDs)*, and uses this division to prevent routing failures or misbehavior in one TD from affecting routing in other TDs. While SCION originally aimed to provide only a PKI for routing messages, it now aims to provide public key authentication for naming and end-entity authentication as well, unifying the trust roots of these PKIs into one system [28]. Unfortunately, there are significant challenges to unifying PKIs for naming, routing, and end-entity authentication, because as described above, these PKIs have drastically different features and requirements. In particular, the mechanisms for updating or revoking the public keys of trust roots will need to consider the frequency and latency requirements of all of these PKIs. Moreover, the division of the Internet into TDs means that a unified PKI for SCION

will be a federated system in which each TD will have its own trust roots and must bridge their authentication systems in some way. I will describe such a system in greater detail in Chapter 5.

Chapter 3

Reducing MITM Attacks through Decentralized Automated Incentives

In this chapter, I describe how we can reduce MITM attacks by realigning incentives in the Web PKI. In particular, I explore how we can encourage CAs to avoid misissuing certificates and how we can more quickly find misissued certificates. As I describe in Section 1.3.1, detectors of misissued certificates are not incentivized to do so, and even when such a certificate is publicized, distrusting the CA is the only way, however disruptive, of preventing MITM attacks. Thus I explore how we can offer incentives for, and automated handling of, reports of misissued certificates, to ensure that misbehaving CAs are more reliably caught and punished. This endeavor prompts several important questions. How can we formally define what it means for a CA to behave correctly? What incentives can we offer CAs and detectors? What mechanisms are necessary for automating the handling reports of misbehavior, and what benefits does automation provide?

As a first step towards answering these questions, I propose the **Instant Karma PKI (IKP)**, an automated platform for defining and reporting CA misbehavior. IKP incentivizes CAs to correctly issue certificates and detectors to quickly report unauthorized certificates [94]. IKP allows domains to specify policies that define CA misbehavior, and CAs to sell insurance against misbehavior. As part of the design of IKP, I propose a formal model for incentive analysis to show that IKP provides positive financial incentives for CAs and detectors and punishes misbehaving CAs. I further show that with our incentive structure, rational CAs cannot profit financially from misbehavior, even when colluding with other domains or detectors.

Concretely, for the TLS Web PKI, IKP allows participating HTTPS domains to publish *domain certificate policies (DCPs)*, policies that specify criteria that the domains' TLS certificates must meet. Any violation of these policies constitutes CA misbehavior. IKP allows participating CAs to sell *reaction policies (RPs)* to domains, which specify financial transactions that execute automatically when an unauthorized certificate is reported. Domains affected by

the certificate, the detector, and the CA receive payments via these transactions. The payment amounts are set such that CAs expect to lose money by issuing unauthorized certificates, and detectors expect to gain money by reporting unauthorized certificates. Information about CA misbehavior and RP offerings are public, allowing domains to use this information as an indicator of how likely a CA is to maintain high security and thus protect against unauthorized certificate issuance.

In summary, I make the following contributions in IKP:

- By building on certificate policies of previous work, I show through my design and implementation of DCPs that IKP can detect many different types of certificate misissuance.
- By modeling and analyzing RP payouts under different scenarios of misbehavior, I show that IKP provides incentives for rational entities that discourage misbehavior such as the misissuance of certificates, even in the face of collusion.
- By implementing IKP in Ethereum, I show that we can provide these incentives without having to trust a centralized third party or waiting for manual handling of certificate reports.
- By evaluating the cost of computation in Ethereum, I show that the deployment and operation of IKP is inexpensive compared to the cost of existing certificates.

I begin this chapter by formulating the problem that I aim to solve with IKP, and explain the design of IKP in detail. I then present an analysis of the incentives in IKP. To show how IKP can be realized in practice, I first present a background on cryptocurrencies and smart contracts, and then describe an Ethereum-based instantiation of IKP. I then evaluate the feasibility of deploying IKP by analyzing my implementation as well as data from the Web PKI market. I end with a discussion of limitations and future work of IKP, as well as lessons learned from designing this system.

3.1 Problem Definition

In a nutshell, the goal of IKP is to provide incentives for correct CA behavior (*i.e.*, due diligence when issuing certificates) and automation in processing reports of unauthorized certificates from detectors. To achieve this goal, we must design a system that can 1. *define* CA misbehavior, 2. *evaluate* whether a given certificate constitutes misbehavior according to the above definition, 3. *specify* reactions and payments that will occur in response to CA misbehavior, 4. *process* reports from detectors regarding unauthorized certificates, and 5. *execute* these reactions and payments automatically after a CA has misbehaved. Achieving these goals allows us to deter CA misbehavior by choosing payments that provide the appropriate incentives for correct CA behavior and for reporting unauthorized certificates. These incentives also increase the number of entities monitoring CAs and thus the probability that an unauthorized certificate is quickly detected. Automatic execution of reactions and payments ensures "instant karma" in IKP: detectors quickly receive rewards and CAs quickly receive punishment.

3.1.1 Desired Properties

A system achieving the above goals should have at least the following properties:

- **Public auditability**: all information required to detect an unauthorized certificate is publicly accessible.
- **Automation**: once CA misbehavior has been reported and confirmed, reactions should automatically proceed without requiring additional information or authorization.
- **Incentivization**: entities that expose CA misbehavior have a positive expected return on investment (ROI).
- **Deterrence**: CAs have a negative expected ROI for issuing an unauthorized certificate for a domain, regardless of the entities they collude with.

As secondary goals, the system should achieve *decentralization* (*i.e.*, the absence of a central trusted entity in the system) and *MITM prevention* (*i.e.*, the rejection of all unauthorized certificates by clients).

3.1.2 Adversary Model

Our adversary's goal is to issue a rogue certificate while maintaining a positive expected ROI. The adversary may access the long-term private keys of one or more CAs (and can thus issue arbitrary certificates from these CAs), as well as those of colluding domains. The adversary may take any action within the PKI (*e.g.*, issuing/revoking certificates) or within IKP (*e.g.*, issuing RPs or reporting certificates) to obtain a net positive ROI among all entities it controls. We assume that the adversary cannot break standard cryptographic primitives, such as finding hash collisions or forging digital signatures. The adversary also cannot compromise the private keys of arbitrary domains. In our blockchain-based instantiation, we further assume that the adversary cannot control a majority of hashing power in the blockchain network.

3.2 Overview

In this section, we provide an overview of the key features of IKP. We begin by introducing its main components, and then describe the main functions of the system.

3.2.1 Architecture

IKP is an extension of the standard TLS architecture, and thus as in TLS, CAs issue certificates to domains, whose servers carry out TLS handshakes with clients. As shown in Figure 3.1, IKP introduces two new entities: the *IKP authority* and *detectors*.

The IKP authority is responsible for the core functionality of IKP. Specifically, the IKP authority maintains information on CAs such as identifiers (e.g., DNS names), public keys to authenticate to the IKP authority, and financial account information at which to receive payments. The IKP authority also stores *domain certificate policies (DCPs)*,

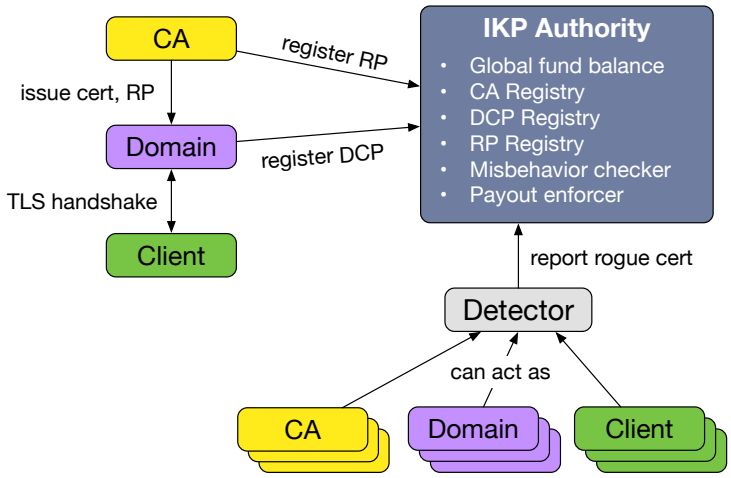


Figure 3.1: Overview of the entities and functions in IKP.

which are provided by domains and can be used to computationally determine whether a given certificate is authorized for a domain, and *reaction policies (RPs)*, which specify automatic reactions that occur if an unauthorized certificate is reported. The IKP authority is responsible for executing these reactions. The IKP authority also maintains a balance called the *global fund*, which can send and receive payments in IKP.

Detectors are responsible for reporting suspicious certificates to the IKP authority. They monitor certificates issued by CAs, and report any certificates they deem to be unauthorized. Any entity, be it a CA, domain, or client, can detect and report CA misbehavior. Each detector must have a financial account at which it can receive rewards for successfully reporting an unauthorized certificate.

Entities in the standard TLS architecture have additional responsibilities. CAs who have registered with the IKP authority can issue RPs, thus acting as a sort of "insurer" against CA misbehavior. Domains register DCP with the IKP authority, providing a public policy that defines CA misbehavior (*i.e.*, issuing an unauthorized certificate). While Figure 3.2 shows intuitive examples of a DCP and a RP, the logic of both DCPs and RPs is determined by machine-understandable policies specified by the domain and by the CA, respectively, providing flexibility in addition to automation and financial incentives.

3.2.2 Operation

We now summarize the actions that occur in IKP, some of which are shown in Figure 3.2.

CA registration A CA registers its information with the IKP authority. Specifically, the CA registers its identifier, financial account information, one or more public keys, and an update policy as shown in Table 3.1. To update its registration, the CA must provide signatures on the update with a threshold number of its update private keys.

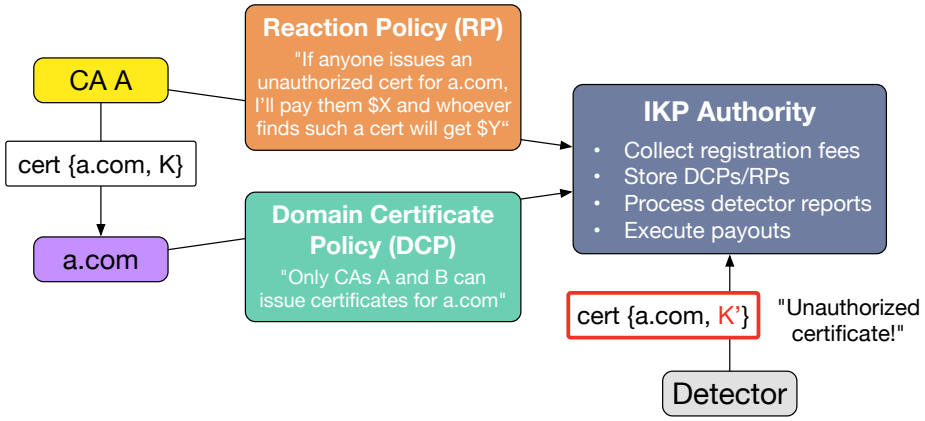


Figure 3.2: Sample interactions between entities in IKP. As in Figure 3.1, yellow denotes a CA and purple denotes a domain.

Table 3.1: Explanation of fields in a CA registration.

Field	Use
CA name	identify CA
Valid from	specify start period of information validity
Payment account	receive payments for CA
Public keys	list of CA's public keys
Update keys	authorize updates to this information (default: empty)
Update threshold	threshold of signatures required for updates (default: 1)

Domain registration A domain registers a DCP with the IKP authority. Specifically, the domain registers its DNS name, one or more public keys, financial account information, and a *checker program* that decides whether a given certificate is authorized for the domain.

RP issuance A registered domain negotiates the terms of a RP with a registered CA. The RP contains the domain name, CA identifier, validity period, a reference to the domain's DCP, and a *reaction program* that contains the payments that occur in response to CA misbehavior. The domain pays the CA to issue the RP, with the IKP authority acting as a mediator to ensure a fair exchange.

Certificate issuance A domain obtains a certificate from a CA. The CA does not have to be the same one that issued the domain's RP, and does not need to have registered with the IKP authority. Thus certificate issuance occurs in the same way as in TLS.

Misbehavior report A detector sends evidence of CA misbehavior (usually an unauthorized certificate) and its financial account information to the IKP authority. The detector must pay a small *reporting fee* to prevent detectors

from reporting all certificates they see. We also use a commitment scheme to prevent frontrunning of detector reports. The IKP authority runs the checker program on the certificate to determine whether the certificate is authorized.

Reaction If a reported certificate is unauthorized, the IKP authority triggers a reaction by running the reaction program specified in the domain's RP. The reaction program usually executes financial transactions, which are sent to the financial accounts of the CA, domain, and detector as appropriate.

The use of checker programs and reaction programs provide expressivity and extensibility to policies and reactions in IKP. As we describe in Sections 3.3 and 3.4, DCPs can provide features such as CA whitelisting, public-key pinning, and short-lived certificate enforcement, while RPs can provide financial payouts to parties beyond the CA, domain, and detector.

3.3 Domain Certificate Policies (DCPs)

In this section, we take an in-depth look at domain certificate policies. In particular, we describe the features and format of DCPs, and present several examples of DCPs that enable various useful defenses against CA misbehavior. We conclude this section by describing the relevant operations for registering and updating DCPs.

3.3.1 Design Principles

We begin by describing the fundamental principles on which we base our design for DCPs. In particular, we identify three main design principles: (1) policies are domain-specified, (2) policies offer sufficient expressiveness, and (3) policy information is public, authenticated, and consistent. These principles help ensure that we can use DCPs to determine *certificate authorization* (*i.e.*, whether a certificate is considered authorized or not for a given domain) securely and effectively.

1) Domain-specified policies The information used to determine certificate authorization is specified by that domain itself. We observe that only domains know with certainty which certificates they have and have not authorized. Therefore, to enable others to deem certificates *unauthorized* as opposed to simply *suspicious*, domains must specify policies governing their certificates. By adhering to this principle, we can ensure that any entity with a domain's policy information can be a detector and find unauthorized certificates for that domain.

2) Policy expressiveness The information used to determine certificate authorization is expressed in a Turing-complete language and can thus represent arbitrarily complex policies. Proposed certificate policies in the literature [75, 126] allow domains to specify only a small set of parameters (e.g., governing how their certificates should be

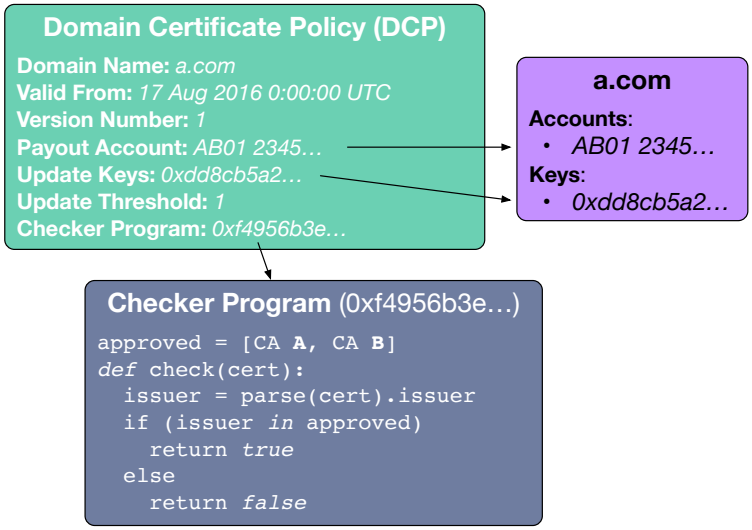


Figure 3.3: A sample DCP with a checker program written in pseudocode.

Table 3.2: Explanation of DCP fields.

Field	Use
Domain name	identify domain for which the policy is active
Valid from	specify start period of DCP's validity
Version number	identify version of this domain's DCP
Payment account	receive payments for domain
Checker program	implement the DCP's certificate policy
Update addresses	(default empty) authorize DCP updates
Update threshold	(default 1) thresh. of signatures required for DCP updates

verified or how errors in the TLS handshake should be handled). These policies cannot be changed in a backwards-compatible way without upgrading all client browsers and possibly all existing domain policies. Moreover, such policies do not enable the automation of reaction to CA misbehavior. IKP provides a general format for DCPs by allowing domains to specify executable code that determines whether or not a given certificate is authorized and specifies concrete reaction to misbehavior.

3) Public, authenticated, and consistent information The information used to determine certificate authorization is stored in a publicly accessible location, is globally consistent, and its authenticity can be verified by the public. Publicly accessible information ensures that all potential detectors can find unauthorized certificates using a domain's policy information. Globally consistent information ensures that all potential detectors see the same policy for a domain and can thus determine with certainty whether a certificate for that domain is authorized.

3.3.2 DCP Contents

We now describe the contents of a DCP. Figure 3.3 shows a sample DCP, and Table 3.2 describes the fields of a DCP. In short, a DCP contains identifying information for the domain, (its DNS name and financial account information) and for the policy (the `Valid From` and `Version Number` fields). A DCP also contains the policy itself, namely, the threshold of signatures required to authorize changes to the DCPs (the update keys and update threshold) and the checker program.

The `Valid From` and `Version Number` fields of a DCP are used in part to help determine whether or not a certificate constitutes CA misbehavior. In particular, each RP is tied to a specific version of a domain's DCP, and a given certificate only triggers a RP if 1) the certificate's validity period began after the DCP's `Valid From` time, 2) the RP's `Version Number` field matches that of the DCP, and 3) the checker program deems the certificate unauthorized (as described below). Because the DCP defines misbehavior by the output of the checker program, an update to a DCP only increments the version number if the checker program is changed. This prevents a domain from having to renegotiate a RP for changing DCP fields unrelated to its policy, such as its financial account information.

The update keys and update threshold protect a domain against the loss or compromise of a private key. We allow a domain to update its DCP by authorizing the update with signatures from a threshold of its update keys. Because DCPs are crucial to determining CA misbehavior, domains should protect against unauthorized updates with a sufficiently high update threshold. Our recovery system is not foolproof; a domain is ultimately responsible for managing its own recovery addresses. However, our approach provides a tunable level of security and recoverability for each domain. In order to guard against a mass loss or compromise of its private keys, a domain can store some of its private keys offline, with trusted peers, or even with a large group of authorities such as one provided by the CoSi protocol [124].

3.3.3 Sample Checker Programs

We now present example checker programs in IKP. These examples represent a range of existing proposals to improve the TLS PKI and demonstrate the flexibility of IKP's checker programs. For the following examples, we assume the use of X.509 v3 certificates [46], but we note that checker programs can define their own formats or handle multiple formats, allowing different certificates formats to coexist in IKP. We also assume access to a method to parse a certificate and extract the contents of its fields.

CA whitelisting A checker program can enforce the use of certain CAs by extracting the `Issuer Name` field of the certificate and checking whether the issuer is on a whitelist of CA names. In order to enforce the use of a specific set of CA keys, the checker program can instead extract the `Authority Key Identifier` extension for X.509 and check the identifier against a whitelist. In either case, the program first defines a whitelist and then performs the appropriate

check.

Public key pinning A checker program can implement a form of public key pinning by extracting the `Subject Public Key Info` field of the certificate and checking this key against a whitelist. Similarly to above, the program defines the whitelist and performs the appropriate check. We note that unlike other key pinning solutions, no trust on first use is necessary because DCPs are public and consistent and thus the client can simply check the domain's DCP for the key pins.

Short-lived certificates A checker program can enforce the use of short-lived certificates [127] by checking that a certificate's validity period does not exceed a given maximum value. This can be done by extracting the `Not Before` and `Not After` fields from the certificate and calculating the time difference to determine the length of the certificate's validity period, and checking that this length is less than a specified maximum allowable value.

Wildcard restrictions A checker program can prevent the use of wildcard certificates by simply extracting the `Subject Name` field and checking that the wildcard character `*` does not appear.

Certificate Transparency A checker program can implement criteria similar to those of Certificate Transparency [85] by checking for proof that the certificate has been publicly logged. The checker program first defines a list of trusted logs. The program can then query the logs directly or take a proof from a trusted log as input in addition to the certificate itself.

Combining checker programs An additional benefit of public consistent DCPs is that domains can see other checker programs and model their own from these programs. We additionally allow domains to call other checker programs. This feature allows a domain to write a checker program that simply calls a set of checker programs, thus allowing the domain to combine existing policies. For example, a domain can specify that all criteria in the called checker programs must be fulfilled by requiring that all referenced programs deem a certificate authorized, or specify that some threshold of referenced programs must do so by counting the number of referenced programs that deem the certificate authorized.

3.3.4 DCP Operations

We now describe relevant operations for a DCP. Specifically, we cover the initial registration of a domain's DCP as well as the update process.

DCP registration A domain D requests to initially register its DCP in the blockchain by sending a message to the IKP authority containing its DNS name, the contents of its initial desired DCP, and information to authenticate

itself to the IKP authority. Specifically, to authenticate itself, *D* provides a signature on its name and DCP with 1. its DNSSEC [23] private key, as well as a DNSSEC signature chain to the ICANN root zone key, or 2. its TLS private key, as well as a certificate chain from the corresponding public key to a root CA key. This authentication method, which we call the use of a *bootstrap proof*, provides a way for *D* to show control over its identifier and public key by leveraging an existing PKI. Because IKP is tied to TLS and hence to DNS names, we can use bootstrap proofs to protect DCP squatting by malicious entities that do not own the names they claim.

It is safer to use DNSSEC-based bootstrap proofs, as DNSSEC has had far fewer compromises than TLS and only requires a single root key to be stored by the IKP authority, However, in a measurement we conducted using data from Censys [52], we found that only 649 of the top 100,000 most popular domains use both DNSSEC and HTTPS. Therefore, few domains will be immediately able to reap the benefits of using DNSSEC-based bootstrap proofs, though a wider deployment of DNSSEC will make this option more readily accessible to domains.

Using TLS-based bootstrap proofs requires the IKP authority to select a list of accepted root CA keys, and also runs the risk that an unauthorized certificate can be used to register a DCP. To address the first problem, the IKP authority could simply select a set of 28 root certificates which are present in most popular desktop and mobile operating systems and web browsers [108]. To address the second problem, we can allow domains to override an existing registration by submitting multiple independent bootstrap proofs. This approach makes registration easy for most domains, but allows a domain whose registration is stolen by an adversary with an unauthorized certificate to recover by obtaining an additional certificate.

We note that bootstrap proofs can make it more difficult for legitimate domains to register themselves with the IKP authority, and are not foolproof. However, given the crucial role DCPs play in IKP, we need to protect them from being easily claimed and held by adversaries. We also do not envision bootstrap proofs as a long-term solution, as they are based on PKIs that suffer from the problems that we aim to solve with IKP. We can instead configure the IKP authority such that as deployment increases, the bootstrap proof requirement can be relaxed or eliminated.

Updates A domain *D* can update its information by sending a transaction to the IKP contract with its new DCP or registration and signatures from a threshold number of its update keys. The IKP authority verifies each of these signatures, checks that the number of signatures is at least the threshold number required by *D*'s current DCP, and if so, updates *D*'s DCP in its registry. Recall that the IKP authority only increments the version number of *D*'s DCP if the checker program changes.

3.4 Reaction Policies (RPs)

In this section, we take an in-depth look at reaction policies. In particular, we begin by explaining the principles behind the design of RPs, and describe the contents of RPs. We then describe payout reaction programs, which provide financial incentives in IKP. We conclude this section by describing the relevant operations for issuing RPs, selecting the relevant RP for a domain, and executing a RP.

3.4.1 Design Principles

We begin by describing the design principles upon which we base our design of RPs. In particular, we identify three main design principles for RPs: (1) certificate-independence, (2) policy-adherence, and (3) single-use. These principles help ensure that reactions to misbehavior do not cause perverse incentives or unintended consequences. We next discuss the three principles in detail.

1) Certificate-independence A RP should be decoupled from public-key certificates. Like certificates, RPs are negotiated between CAs and domains. However, certificates and RPs are independent: CAs issue RPs in addition to certificates, and therefore domains can obtain certificates and RPs from different CAs. RPs provide a relying party with a measure of confidence in a domain's certificates, and serve a fundamentally different role from certificates in the IKP ecosystem. In particular, a RP protects a domain against any unauthorized certificate issuances during the lifetime of the RP.

2) Policy-adherence A RP should be bound to a specific policy for a domain. In particular, since a DCP may change over time, a RP should represent a reaction to violations of a specific *version* of a domain's DCP. Binding a RP to a specific DCP version ensures consistency between the *criteria* for certificate authorization and the *reaction* to the violation of those criteria. This principle also implies that a domain must have a DCP before obtaining a RP.

3) Single-use A RP should be limited to a single instance of CA misbehavior. Because RPs may execute financial payments for which funds must be available, enforcing single-use RPs helps ensure the availability of such one-time resources for each instance of misbehavior. Single-use RPs also prevent adversaries colluding with domains or detectors from repeatedly triggering a RP to obtain payouts. Thus each time a CA issues a certificate that violates a domain's DCP, one of the domain's RPs is triggered and then terminated. We note that domains can have multiple RPs at a given time to protect against multiple instances of CA misbehavior. However, we anticipate that in the vast majority of cases, a domain will only have a single RP at a given time.

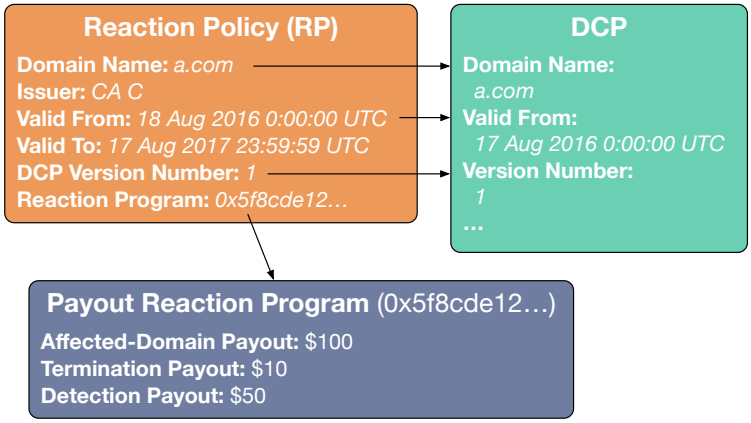


Figure 3.4: A sample RP with a payout reaction program. The domain name and version number in the RP must match those of the DCP, and the start of the RP's validity must be after that of the DCP.

Table 3.3: Explanation of RP fields.

Field	Use
Domain name	identify domain for which the RP is active
Issuer	CA who issued the RP
Valid from	specify start period of RP's validity
Valid to	specify start period of RP's validity
Version number	version of domain's DCP used to trigger RP
Reaction program	implement a response to CA misbehavior

3.4.2 RP Contents

We now describe the contents and format of RPs. Figure 3.4 shows the format of a sample RP, and Table 3.3 describes each field of a RP. Like a DCP, a RP contains identifying information for the domain as well as for the issuing CA. A RP also specifies a validity period and identifies the version of the domain's DCP for which it is active. Finally, the reactions that take place are specified as an address to a contract.

A *reaction program* contains code that can be executed by the IKP authority when a certificate meeting certain criteria is reported and the relevant domain's checker program deems the certificate to be unauthorized. As described in Section 3.4.3, we expect reaction programs to execute financial transactions. After a reaction to CA misbehavior is triggered via a reaction program, the RP containing the reaction program is destroyed.

A reaction program defines the following three methods: (1) `trigger`, which executes when an unauthorized certificate is reported for the domain named in the RP, (2) `terminate`, which executes upon request from a domain whose CA issued an unauthorized certificate, and (3) `expire`, which executes upon request from a CA after the RP has expired.

We note that a RP has a start and end time for its validity, rather than only a start time as a DCP does. A RP, like a certificate, has a limited validity period, but can be prematurely terminated if the issuing CA misbehaves. If a RP is

terminated for any reason, the specified amount of funds is split between the domain and the issuing CA based on the fraction of the RP's validity period that has passed. The exact payouts are detailed below.

3.4.3 Payout Reaction Programs

We now provide a framework for payout reaction programs, which specify a series of financial payments that execute in response to CA misbehavior. Financial payments are important to achieving incentivization, since financial payments be quantified and analyzed. Our goal in designing a framework for this class of reaction programs is to provide a general model for who should receive payments under different circumstances of misbehavior.

We identify three relevant parties who may receive payments if a method from a payout reaction program is executed: (1) the domain, which we denote by D, (2) the certificate-issuing CA, which we denote by C, and (3) the detector, which we denote by d. As Figure 3.4 shows, a payout reaction program specifies three payouts: *affected-domain payouts*, *termination payouts*, and *detection payouts*. To ensure that the IKP authority has a sufficient balance for these payouts, an amount E is sent to the global fund when the RP is issued.

Affected-domain payouts The affected-domain payout (written a) is paid to domain D in the event that a registered CA issues an unauthorized certificate in D's name. The payout compensates D for the security risk it incurs by having an unauthorized certificate that could be used in a MITM attack. The domain does not receive this payout in case of misbehavior by an unregistered CA.

Termination payouts The termination payout (written t) is split between domain D and CA C if D terminates the RP. The termination payout compensates D for lost trust in C after its misbehavior and contributes to the costs of obtaining a new certificate and/or RP. The split of the termination payout between D and C is proportional to the amount of time left in the RP's validity. To ensure that D receives some minimum amount of funds, we set a systemwide parameter τ that D is guaranteed to receive. Letting $\alpha \leq 1$ denote the fraction of the RP's remaining validity, we then have

$$t_D = \alpha \cdot (t - \tau) + \tau \tag{3.1}$$

Because $0 \leq \alpha \leq 1$, we see that t_D is bounded by

$$\tau \ \leq \ t_D \ \leq \ t \tag{3.2}$$

We note that although C does receive funds from the termination payout in spite of its misbehavior, we show in Section 4.2 that C loses funds compared to if it had behaved correctly.

Detection payouts The detection payout (written δ) is the amount paid to whomever reports an unauthorized certificate to the IKP contract. The payout provides an incentive for entities to monitor CA operations in search of unauthorized certificates. Domains can negotiate their own detection reward; high-profile domains may choose to specify a higher detection payout than domains for which security is less important. The RP specifies the detection payout for misbehavior by a registered CA. If a detector reports misbehavior by an unregistered CA, the detector instead receives a smaller payout amount m. This reduced payout deters a collusion attack that we describe in Section 4.2.

3.4.4 RP Operations

We now describe relevant operations for a RP issued in IKP. Specifically, we cover RP issuance as well as the scenarios in which each of the reaction program's three methods (`trigger`, `terminate`, and `expire`) are executed.

RP issuance When a domain D wants to purchase a RP from a CA C registered in IKP, the two parties first agree on the terms of the RP or certificate out of band. In particular, for a RP with a payout reaction contract, D and C negotiate the payouts $a\,t$, and δ, as well as the price ρ of the RP. IKP sets two constraints on the amounts that must hold:

$$t < \rho < a + \tau \tag{3.3}$$

$$m < \delta \tag{3.4}$$

These constraints are justified in Section 4.2.

Once C and D have agreed on the terms of the RP, we must ensure that a domain who purchases a RP in IKP obtains what it agreed on with the CA, and conversely, that the CA receives the appropriate payment for the RP that it has issued. We can achieve such a fair exchange by having the IKP authority act as a third-party escrow.

Specifically, D sends the payment for the RP or certificate to the IKP authority, along with the hash of the RP or certificate and C. In turn, C creates and sends the RP or certificate to the IKP authority. To ensure that the IKP authority has enough funds to pay out the appropriate parties, C may also need to send additional funds to the IKP authority (see Section 3.7).

The IKP authority then verifies that 1. the RP or certificate hashes to the value provided by D, 2. the amount of funds that C has sent over (if necessary) is sufficient to ensure that the global fund will be able to send the payouts in case of misbehavior, and 3. the terms of the RP meet the constraints described above. If any of these criteria do not hold, then the domain's fee ρ is returned and the issuance is canceled. If all of these criteria hold, ρ is transferred to C and any funds sent with C's message are transferred to the global fund.

Domain RP selection The IKP authority maintains a mapping between domains and a list of their currently-active RPs. When a domain purchases a new RP, the IKP authority adds the new RP to the domain's corresponding list ordered by the validity ending time. When misbehavior is reported, the IKP authority triggers the appropriate reaction in the first policy in the list. This scheme ensures an unambiguous reaction to an instance of CA misbehavior while also triggering the RP that expires the soonest.

RP trigger If C is found to have issued an unauthorized certificate for a domain D, then the `trigger` method of D's RP is automatically executed. For payout reaction programs, D receives the affected-domain payout a and its share termination payout t_D, the detector receives the detection payout δ, and C receives its share of the termination payout $t - t_D$. The IKP authority then removes the RP from the list of D's RPs. The IKP authority also records the time at which a detector reported misbehavior by C to handle the termination case below.

RP termination If C is found to have misbehaved, any domain D that has a RP issued by C can prematurely terminate the RP. To do so, D sends a message to the IKP authority with the RP it wishes to terminate. The IKP authority checks that the RP's validity began before C's last misbehavior and that the RP has not yet expired, and if so, executes the `terminate` method. In this case, D receives its share of the termination payout t_D and C receives its share $t - t_D$.

RP expiration Once the validity period for a RP belonging to a domain D has ended, the IKP authority simply removes the RP from the list of D's RPs. Because doing so can reduce the liability of the issuing CA C, the IKP may also note the reduction in liability and return funds as necessary to C's payment account.

3.5 Analysis

In this section, we analyze the design of IKP. In particular, we model the incentives of each entity in the IKP ecosystem by considering the flow of payments among entities for each operation (such as RP issuance). Using this model, we demonstrate two important guarantees that hold in IKP:

1. **Incentives** for DCP compliance and misbehavior reporting: issuing a certificate that complies with a domain's DCP or reporting a certificate that violates a domain's DCP results in a higher payout than alternative actions.
2. **Disincentives** against misbehavior and collusion attacks: falsely reporting a valid certificate as unauthorized or issuing a certificate that violates a domain's DCP does not result in a profit for a misbehaving detector or CA, respectively, regardless of who the detector or CA colludes with.

In the course of our analysis, we derive constraints on RP terms that must hold for the above properties to be true.

Table 3.4: List of payments sent for each event. D represents the domain, R is the CA that issues the RP, C is the CA that issues the certificate, d is the detector, and F is the global fund. E represents the amount sent to the IKP authority by R.

Event	From	To	Amount
Register CA	C	F	r_C
Register domain	D	F	r_D
Issue reaction policy	D	F	ρ
	R	F	E
	F	R	ρ
Expire reaction policy	F	R	E
Terminate reaction policy	F	D	t_D
	F	R	$E - t_D$
Report false misbehavior	d	F	m
Report internal misbehavior	d	F	m
	F	D	$a + t_D$
	F	d	δ
	F	R	$E - t_D$
	C	F	$a + \delta$
Report external misbehavior	d	F	m
	F	D	t_D
	F	d	m
	F	R	$E - t_D$

3.5.1 Model

We begin by analyzing the payments that occur within the model described by the constraints in the previous sections. Table 3.4 summarizes the payout amounts for each action in IKP. For most of our analysis we consider a single RP lifetime and certificate issuance, and use the following notation:

- D denotes the domain for whom a (possibly unauthorized) certificate is issued,
- R denotes the CA that issues the RP to D,
- C denotes the CA that issues the certificate to D,
- d denotes a detector who can choose whether or not to report the certificate as unauthorized, and
- F denotes the global fund.

Note that R and C may be the same entity, and d may be any entity (even one of D, R, or C).

To demonstrate the two central properties above, we consider two scenarios of CA misbehavior. In the first scenario, which we call *internal issuance*, the certificate-issuing CA C is registered in IKP and in the second scenario, which we call *external issuance*, C is not registered in IKP. Considering these scenarios separately simplifies our analysis below. In both scenarios, C issues a certificate to D, and can choose whether or not to issue a certificate that complies with D's DCP or not. Detector d can then choose whether to report the certificate as unauthorized or not.

For each case, we consider the payments made in the series of events that must have occurred and can determine the net reward of each entity by summing the payments it received and subtracting the sum of the payments it made. We note that we do not consider payments made outside of IKP, as we cannot track or constrain these transactions.

Given our model, we can prove the following properties in the two scenarios, assuming rational entities:

- *Compensation* of domains affected by misbehavior: a domain with a DCP for whom an unauthorized certificate is issued should receive a higher net payout after a successful report.
- *Rewards* for successful detectors: a successful misbehavior report results in a higher net payout for the detector than an unsuccessful report or no report at all.
- *Deterrence* of internal misbehavior: a CA that has registered in IKP and issued an unauthorized certificate for a domain has a negative net payout.
- *Collusion-proofness* for external misbehavior: in the second scenario, a CA that has not registered in IKP cannot collude with any set of other entities to gain a positive net reward from issuing an unauthorized certificate.

The last property highlights the need to consider *collusion attacks* in IKP. In particular, we must verify that a misbehaving C cannot collude with other entities and sum their net rewards to gain a profit. We observe that C will only collude with entities that receive a positive net payout on their own, but can purposely misbehave in order to trigger RP payouts. To ensure that no possible collusion can result in a profit for C, we sum the rewards of all positive-reward entities with those of C to find the maximum profit that C can receive.

In our analysis, we assume that the CA R (who issued the RP to D) has registered in IKP, and that the domain D has registered a DCP. We do not consider these operations in our analysis due to the fact that they occur once and thus should not factor into the analysis of an individual RP's lifetime, which may occur (with its costs) many times.

3.5.2 Scenario #1: Certificate Issuance inside IKP

For the first scenario, we consider whether or not C misbehaves by issuing a non-compliant certificate, and whether or not a detector d reports this misbehavior. We assume that the issuance has taken place and the appropriate payments have been made. We observe that if no misbehavior is reported, then the RP will eventually expire, regardless of whether C misbehaves. Thus we consider only three cases: (1) no detector reports misbehavior, (2) C issues a compliant certificate and detector d reports it, and (3) C issues a rogue certificate and detector d reports it.

The first section of Table 3.5 shows the results for this scenario, that is, how much is paid out to the involved entities, according to Table 3.4, aggregated into the three cases.

Regarding the affected domain D, we observe that in the case of reported misbehavior, D receives an additional $a+t_D$ than it would if no misbehavior was reported. In order for D to profit, we require $\rho < a+t_D$. Since we know from Equation 3.2 that $t_D \geq \tau$ and since we want a positive compensation for all values of t_D, we set the slightly

stronger constraint

$$\rho < a + \tau \tag{3.5}$$

thus ensuring compensation of affected domains.

Regarding detector d, we observe that if d reports misbehavior correctly, it receives $-m + \delta$. To ensure that a successful report is rewarded, the quantity $-m + \delta$ must be positive, and thus we set the constraint

$$m < \delta \tag{3.6}$$

which thus ensures rewards for successful detectors (and unsuccessful detectors simply lose the reporting fee m).

Regarding CA C, we make a case distinction as follows. We first consider the case $C = R$, that is, the same CA has issued an RP and a certificate to domain D. In this case, we observe (after summing up the entries for R and C) that for the CA to lose money due to a possible misbehavior, we require $\rho < a + t_D + \delta$. Again, since $t_D \geq \tau$ and we want a loss of money for all possible values of t_D, we obtain the stronger inequality $\rho < a + \tau + \delta$ which ensures the deterrence of internal misbehavior for this scenario. However, this constraint is subsumed by Equation 3.5, which sets a tighter bound on ρ.

The case of $C \neq R$ is similar. The RP-issuing CA R should still profit since it has not misbehaved, and we thus require $\rho > t_D$. Because of $t \geq t_D$ from Equation 3.2, we simply let

$$\rho > t \tag{3.7}$$

and obtain a positive incentivization for R. Regarding the misbehaving CA C, we simply require the values a and δ to be positive, which is satisfied by definition.

Finally, to avoid collusion attacks in the first scenario, we consider the entities besides C receiving a positive reward. We observe that although both D and d profit in the case of misbehavior, if we sum the rewards of D, d, R, and C, the result is $-m < 0$, and thus a collusion between C and all other parties does not profit.

3.5.3 Scenario #2: Certificate Issuance outside IKP

In the external scenario, we assume that the certificate-issuing CA C does not register with IKP. We investigate, as before, whether or not C misbehaves, and whether or not d reports misbehavior. We again assume that domain D has purchased a RP from CA R, and we again observe that if no misbehavior is reported, then the RP expires and C's misbehavior status does not matter.

The second area of Table 3.5 shows the results for the external scenario, that is, how much is paid out to the involved entities, according to Table 3.4, aggregated into three cases as above.

Table 3.5: Rewards for each entity in different scenarios.

Entity	Unreported	Rep.+Behave	Rep.+Misbehave
Scenario #1: Certificate issuance by IKP-CA *C*			
D	$-\rho$	$-\rho$	$-\rho+a+t_D$
d	0	$-m$	$-m+\delta$
R	ρ	ρ	$\rho-t_D$
C	0	0	$-a-\delta$
F	0	m	m
Scenario #2: Certificate issuance by non-IKP-CA *C*			
D	$-\rho$	$-\rho$	$-\rho+t_D$
d	0	$-m$	0
R	ρ	ρ	$\rho-t_D$
C	0	0	0
F	0	m	0

We concentrate on the differences from the previous scenario. First, note that the misbehaving domain C is not punished and thus it is not deterred from misbehavior. The reason is that external misbehavior cannot be deterred from within IKP. On the other hand, the CA also does not receive any payments for good behavior if outside IKP. Similarly, detectors are not rewarded for spotting external misbehavior. Handing out rewards would eventually drain the global fund.

Regarding the affected domain D, we have to consider collusion attacks since a positive payout for an affected domain might incentivize a malicious external CA to collude with that domain. We thus consider again the entities besides C that make a positive reward: As C does not need to pay anything, colluding with any entity with a positive reward results in net profit. Colluding with R does result in a net profit, but the profit is less than collusion would yield if C behaved, and thus this is not a viable strategy for C. However, C can collude with the affected domain D if the reward $-\rho+t_D$ is positive. To avoid this, we set $\rho \geq t_D$, and since $t_D \leq t$ and we want this constraint to hold for all values of t_D, we obtain the stronger constraint $\rho \geq t$ which provides collusion-proofness in the external scenario. However, this constraint is subsumed by Equation 3.7. We note that this constraint does not imply that an affected domain is losing money. The domain receives the termination payout, which partially offsets the cost of the RP, and additionally benefits from the fast detection offered from having a RP. For the same reason, we additionally set the detector d's reward to m instead of δ in the case of external misbehavior (see Table 3.4), so that the detector's expected reward is zero (see Table 3.5).

We observe that honest CAs issuing RPs benefit from joining IKP, as they receive rewards in any case. We further stress that domains have no financial loss when joining IKP and purchasing RPs since they receive positive compensation for internal misbehavior and offset their loss in case of external misbehavior. We also observe that the constraints set by Equation 3.5, Equation 3.6, and Equation 3.7 can be easily satisfied by C, who can select ρ, a, δ,

and t based on the values of the constants τ and m. We explore realistic values for these parameters in Section 3.8.2.

3.6 Cryptocurrency Background

In this section, we provide a brief overview on blockchain-based cryptocurrencies, which we use to instantiate IKP. In particular, we describe the fundamental principles underlying blockchains through Bitcoin, and then describe Ethereum (which we use to implement IKP) and the advantages it offers over Bitcoin. For further details on all issues related to blockchains, we refer readers to a more complete view of decentralized cryptocurrencies [36].

3.6.1 Blockchain Principles and Bitcoin

At a high level, decentralized cryptocurrencies such as Bitcoin [101] are public ledgers created and maintained through decentralized, peer-to-peer consensus. These ledgers are most commonly implemented as *blockchains*, chains of *blocks* linked by hash pointers to the previous blocks and containing lists of transactions. This structure provides full history of all past transactions and prevents the transactions from being retroactively modified.

Bitcoin implements transactions with a small, limited-capability scripting language called Script [6]. The use of Script enables a wider range of transactions such as paying to any account, to no account (thus destroying the coin), or to the first account to solve a puzzle. Script is deliberately non-Turing-complete because nodes must process Script to verify transactions and malicious Script transactions could otherwise cause nodes to loop forever. Script can also be used to store non-financial transactions in the ledger, such as a key-value store (used by proposals such as Namecoin [2] to implement a DNS-like system).

Most blockchains grow through the *mining* process, in which nodes in the network race to find a value v that, when hashed with the hash of the previous block and the transactions since that previous block, results in a hash value of a certain form [27]. In Bitcoin, the hash must be of a certain form (i.e., the computed hash value must be smaller than a target value tuned) so that a new block is found approximately every ten minutes. Using a cryptographically secure hash function requires a brute-force search to find v, making mining a proof-of-work scheme [54]. A node or *miner* is incentivized by the *block reward*, a set amount of currency given to whomever extends the blockchain by recording the new transactions, finding v, and then broadcasting the new block.

Because multiple miners may find v at different times, the blockchain can *fork*, resulting in different versions of the blockchain. Miners decide on which version to mine on by *Nakamoto consensus*: each miner picks the chain with the greatest length. Though multiple chains may be tied for the longest, one of the chains will eventually become longer than the others due to the probablistic nature of mining. Nakamoto consensus also ensures that an adversary cannot fork from a much earlier block, as the adversary would have to mine enough blocks to outrun the current longest chain, which becomes more difficult the earlier the desired block is.

The security of blockchain-based cryptocurrencies relies on the fact that no entity controls a majority of the hashing power of the network. Otherwise, that adversary can reverse previous transactions (called a *double-spending attack*) or selectively suppress transactions by outpacing the rest of the network's mining power in the long run. Controlling the network in this way is commonly called a *51% attack* in the blockchain community, though recent work has shown that with pathologically malicious behavior, controlling a smaller percentage of the hashing power is sufficient to double-spend or to suppress transactions [59, 117, 65, 102]. Blockchain proponents argue that such an attack is unlikely to be sustained because doing so would devalue the currency as the network loses trust in the reliability of the currency.

3.6.2 Ethereum

Ethereum [131] generalizes the ideas behind blockchains and Bitcoin Script, enabling the storage of arbitrary state and Turing-complete computation in the blockchain. Transactions in Ethereum represent computations in the *Ethereum Virtual Machine (EVM)*, and the language used for these computations – in contrast to Bitcoin's Script – is Turing-complete. To deter malicious transactions that cause nodes to carry out expensive or nonterminating computations, the sender of a transaction must send *gas*, additional funds that compensate miners for their computational and storage costs when executing the transaction. Operations in the EVM are priced in units of gas and each transaction specifies a gas price, offering a tuneable incentive for miners to execute the transaction. Ethereum thus offers a richer computational environment than Bitcoin does.

Code in Ethereum is stored in *smart contracts*, autonomous accounts that run their code upon receiving a transaction. A contract maintains its own data storage and balance, access to both of which is governed completely by its code (though all contract data and balances can be publicly read on the blockchain). Contracts allow for the creation of autonomous agents whose behavior is entirely dependent on their code and the transactions sent to them, thus providing functionality comparable to that of a centralized party in a transparent, decentralized manner. This benefit has been utilized in such ambitious efforts such as *Decentralized Autonomous Organizations (DAOs)*, which aim to automate governance of a central entity using decentralized smart contracts [71]. Ethereum thus offers the possibility of decentralized trusted entities, a feature not possible in Bitcoin.

3.7 IKP in Ethereum

We now describe our instantiation of IKP in Ethereum, whose architecture is shown in Figure 3.5. We instantiated the IKP authority as a smart contract called the *IKP contract*, thus providing a decentralized authority that does not need to be trusted. Ethereum also provides a natural computation platform for checker and reaction programs, and its cryptocurrency Ether can be used as the currency for financial payments made in IKP.

However, Ethereum had two limitations that made instantiating IKP difficult:

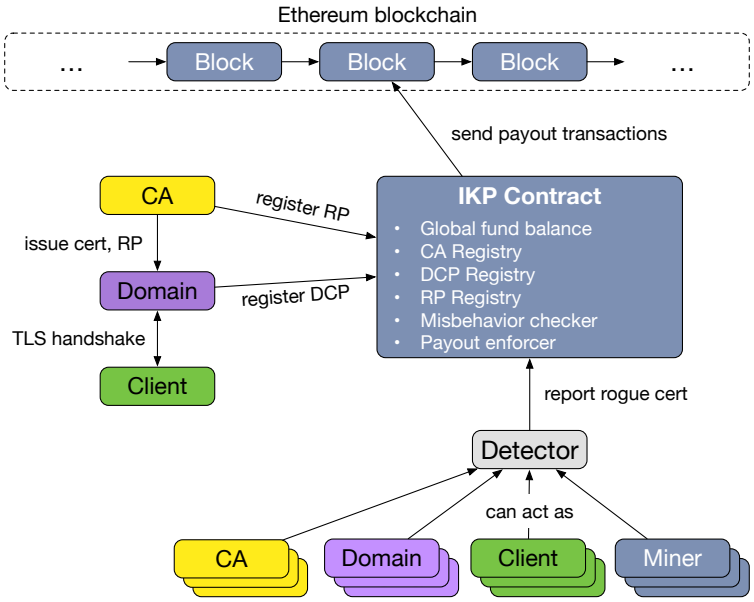


Figure 3.5: IKP architecture in our Ethereum instantiation.

1. *Necessary solvency*: CAs need to pay enough into the global fund to cover reaction payouts resulting from their own misbehavior.
2. *Report frontrunning*: detector reports (and the corresponding detection payouts) can be stolen if an entity such as a miner submits a detector's report as its own.

Both of these required slight modifications to the centralized version of IKP.

In this section, we first describe the general changes we made to IKP. We then describe the techniques we used to ensure solvency and to prevent frontrunning.

3.7.1 Modifications for Ethereum

Because all payments in our instantiation of IKP will be in Ether, the financial account information of CAs, domains, and detectors are simply Ethereum addresses used to send and receive Ether transactions. Since Ethereum addresses also represent public keys, we also use addresses as update keys for CAs registrations and DCPs. Using Ethereum addresses in this way allows us to take advantage of the built-in signature verification support for messages from addresses (*i.e.*, transactions). We note that CA public keys (described in Table 3.1 are not Ethereum addresses, as they represent public keys used to verify certificate signatures.

All messages to the IKP contract are sent as transactions with the appropriate funds and parameters. This includes registration messages for CA information and for DCPs, messages sent for RP issuance, detector reports, and messages sent to terminate a RP. Because Ethereum requires each entity to pay the gas costs of the computation that results from the transaction, the incentives discussed in Section 4.2 may not exactly hold. However, we note that the current

maximum that an entity would pay for a transaction is around 4 million gas, which is currently worth $3.14 USD at the standard price, and thus with current certificate prices, the gas costs are unlikely to make a significant difference in the incentives.

Checker programs and reaction programs are also implemented and stored as smart contracts, which we call *check contracts* and *reaction contracts*, respectively. Because each contract is its own account, referencing a checker program or reaction program in a DCP or RP respectively can be done by simply storing the address to the relevant check contract or reaction contract. Similarly, combining checker programs or reaction programs can be done by calling check contracts or reaction contracts by address.

3.7.2 Ensuring Solvency

To ensure solvency, we need to first show that CAs pay enough into the global fund to cover any reaction payouts that may occur if they misbehave. We can achieve this by having the IKP contract maintain a balance for each CA, keeping track of the payments that come in from the CA (most often from RP issuance). Each CA must maintain a certain minimum balance (called the *solvency threshold*) in order to issue new RPs. We define the solvency threshold for a CA C as the sum of the maximum affected-domain payout, the maximum detection payout, and the sum of all termination payouts, computed over all of C's currently active RPs. This threshold ensures that for any single instance of C misbehaving, all of C's RP customers and the detector will receive their appropriate payouts.

When C initially registers, it must pay a *registration fee* r_C, which prevents frivolous CA registration. When C wants to issue a new RP, it must provide sufficient funds to maintain its solvency threshold. However, C can also add money to its balance without issuing a RP or add more than is necessary when issuing a RP. Exceeding the solvency threshold may attract potential customer domains by giving them greater confidence that C will be held accountable in case of misbehavior. If C issues multiple unauthorized certificates and drops below its solvency threshold, it may not have enough funds for all of its payouts. In this case, C's registration fee r_C is used towards the payout amount until its balance is depleted. For the remaining payout amount, the IKP contract records the debts and the entities owed, and this record can be used as a basis for legal action against C. Thus while IKP cannot provide full protection in all cases, it improves upon the existing ecosystem by providing some automatic reactions, and only requiring manual intervention in extreme cases.

The IKP contract stores metrics for each registered CA, namely, the total payout value of the CA's current RPs, the time of the CA's last misbehavior, the total number of RPs the CA has issued, and the total number of instances of misbehavior for the CA. These metrics can help domains evaluate whether or not a CA is trustworthy.

When choosing a CA from whom to purchase a RP or certificate, we note that a domain can query the CA's balance and its outstanding liabilities (the sum of all payouts in all of its payout reaction contracts). This provides

```
1:  procedure PROCESS_REPORT
    Input: detector address d, certificate 𝒞
2:      D ← get subject name from 𝒞
3:      DCP_D ← lookup D in DCP map
4:      CC ← get check contract address from DCP_D
5:      if !CC.check(𝒞) then
6:          RPL_D ← lookup RP list for D
7:          RP ← get reaction contract address from RPL_D[0]
8:          RP.trigger(d)
9:          delete RP from RPL_D
10:     end if
11: end procedure
```

Algorithm 1: IKP contract handling a misbehavior report.

the domain with a measure of confidence of how solvent the CA is in case of misbehavior. Moreover, the outstanding liability amount also serves to provide the domain with a measure of the CA's own confidence in its security of issuing certificates.

3.7.3 Preventing Frontrunning

To report misbehavior, a detector needs to send an unauthorized certificate to the IKP contract. However, we must ensure that misbehavior reports (each containing an unauthorized certificate) cannot be stolen via frontrunning by blockchain miners. We achieve this by using a protocol similar to the domain registration protocol of Namecoin [9] to report misbehavior: a detector d first sends a "pre-report" containing the reporting fee and a commitment hash $H(\mathscr{C}\|s)$ to the IKP contract, where $\mathscr{C}$ is the certificate to report and s is a secret known only to d. After waiting for a certain number of blocks, d opens the commitment by sending $\mathscr{C}$ and s to the IKP contract. A miner or other entity that sees a pre-report does not know s and hence cannot determine what $\mathscr{C}$ is until d opens the commitment. Because reporting misbehavior requires waiting for a set number of blocks, frontrunning is not possible.

Upon receiving the detector's report, the IKP contract checks that the certificate and secret sent by d matches the committed value sent earlier. The contract then carries out the check shown in Algorithm 1. If the check contract returns deems $\mathscr{C}$ unauthorized, the IKP contract triggers the reaction contract for the oldest of the domain's RPs. We note that in addition to the reporting fee, a detector d must also pay the gas costs for the work performed by the IKP contract.

3.8 Evaluation

In this section, we investigate the technical feasibility and real-world challenges of IKP in today's blockchains. In particular, we detail our prototype implementation in Ethereum, and describe why the current limitations of Ethereum

Table 3.6: Cost of various IKP operations.

	Approximate Cost			Approximate Cost	
Operation	**Gas**	**USD**	**Operation**	**Gas**	**USD**
Verif. cert.	31 012	$0.0238	Bootstrap proof	681 731	$0.5232
Register CA	91 400	$0.0701	Register DCP	152 579	$0.1171
Update CA	34 656	$0.0266	Update DCP	181 226	$0.1391
Order RP	49 024	$0.0376	Pre-report cert	63 951	$0.0491
Create RP	226 892	$0.1741	Report cert	149 284	$0.1146
Terminate RP	99 461	$0.0763	Send payouts	107 962	$0.0829
Expire RP	39 823	$0.0306	CA Balance	39 716	$0.0305
IKP Contract Creation				1 660 319	$1.2742

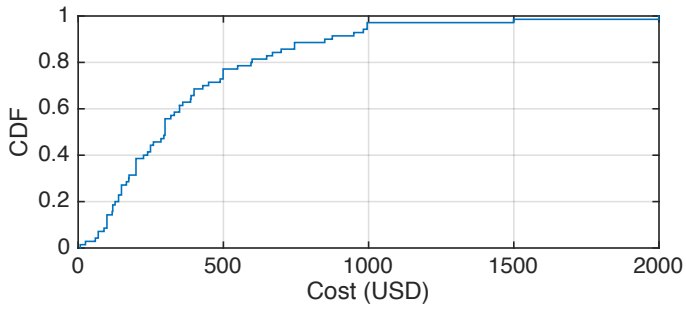
make a full-fledged deployment of IKP challenging. We also analyze real-world CA data to determine reasonable quantities for systemwide parameters based on existing prices.

3.8.1 Prototype Implementation

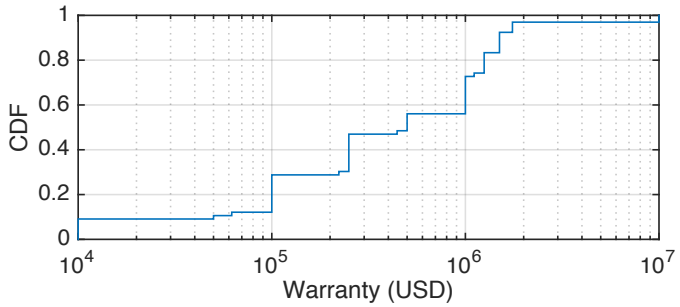
We implemented IKP in 290 lines of Solidity, a high-level Ethereum language that resembles JavaScript. Our code is available at `https://github.com/syclops/ikp`. We faced numerous challenges during our implementation. In the current version of Ethereum, full X.509 certificate parsing is prohibitively expensive, exceeding the current maximum limit on gas allowed by a single transaction. Accordingly, for the purpose of check contracts, we had to resort to leveraging the DER-encoded format [7] of the certificates, recursively extracting type-length-value encoded byte strings and finding the desired object identifier (OID) such as the domain's common name (usually defined as its DNS name).

Additionally, the current version of Ethereum does not support RSA signature verification, which hindered our effort to determine the approximate cost of operations in IKP. We overcame this obstacle by using a modified version of the JavaScript-based Ethereum virtual machine [31]. The modification adds RSA verification and sets its cost to be 200 gas; for comparison, the cost of verifying an ECDSA signature using the secp256k1 curve costs 3000 gas. We obtained a roughly similar ratio of running times in comparing signature verification between these two algorithms on our own machines. While RSA verification is not officially part of Ethereum, support for signature algorithms other than ECDSA has been considered [32] and is currently planned for future versions of Ethereum [38].

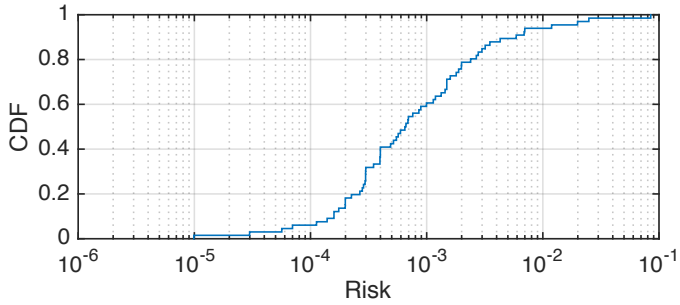
To measure the approximate costs of running various IKP operations, we ran the functions of our prototype implementation in a test Ethereum network. We measured the approximate computational steps (in Ethereum's *gas*) and approximate cost (in US dollars) for creating the IKP contract and for each operation supported by the IKP contract. To convert the cost in gas to USD, we used the current standard price of 1.8×10^{-8} Ether $\approx 7.67 \times 10^{-7}$ USD per unit of gas. For the purposes of testing, we assumed that all strings (used for domain and CA names) were a maximum of



(a) Distribution of 1-year certificate costs.



(b) Distribution of certificate warranty amounts.



(c) Distribution of calculated risk amounts.

Figure 3.6: Empirical CDFs of certificate costs, warranties, and assessed risks from the most popular CAs [17].

32 bytes, and that the public keys for certificate verification were 2048-bit RSA keys.

Table 3.6 shows the costs of various operations in gas and USD. We observe that by far the highest cost in the system is for checking a bootstrap proof. Much of this cost comes from simply handling data that is the size of a standard 2048-bit certificate, since we can also see that the cost of verifying an RSA-signed certificate is relatively low. However, since we are dealing with amounts (under $1 USD) that are drastically smaller than the cost of most certificates, we can conclude that barring large fluctuations in the gas price, gas limit, or price of Ether, it is both technically and financially feasible to deploy IKP in the Ethereum blockchain.

Table 3.7: Risk upper-bounds inferred from CA certificate and warranty amounts (in US dollars) from CA websites.

CA	Certificate	Cost	Warranty	Risk
Highest-Risk				
GlobalSign [12]	Wildcard	\$849	\$10,000	8.49e−2
GlobalSign	DomainSSL	\$249	\$10,000	2.49e−2
StartCom [16, 103]	Ext. Validation	\$199	\$10,000	1.99e−2
StartCom	Org. Validation	\$119	\$10,000	1.19e−2
Entrust [14]	Wildcard	\$699	\$100,000	6.99e−3
...	...	...	...	...
Certum [10]	Commercial SSL	\$25	\$222,000	1.13e−4
Starfield [15]	Standard SSL	\$7	\$100,000	7.00e−5
Comodo [11]	EV SSL	\$99	\$1,750,000	5.66e−5
IdenTrust [13]	Multi Domain SSL	\$299	\$10,000,000	2.99e−5
IdenTrust	Standard SSL	\$99	\$10,000,000	9.90e−6
Lowest-Risk				

3.8.2 CA Certificate Offerings

To get an estimate of sample RP payout values, we collected data from the most popular CAs. In particular, we examined each of the standard TLS certificate offerings of the 20 CAs with a market share of at least 0.1%, representing 99.9% of all TLS certificates on the Web [17]. For each certificate, we noted the cost of a 1-year certificate (ignoring discounts for purchasing multi-year certificates) and the relying party warranty provided with the certificate. In total, we examined 70 certificate offerings across 18 CAs (Deutsche Telekom did not specify a warranty amount, and Let's Encrypt does not offer a warranty because its certificates are free). For each certificate available for purchase, we also calculated the risk as the price divided by the warranty. We note that this is an upper-bound for the actual risk that the CAs face.

Figure 3.6 shows the CDF for the cost, warranty, and calculated risk of each of these certificates. Of the certificates we examined, the prices ranged from \$7 (Starfield's Standard SSL) to \$1999 (Symantec's Secure Site Wildcard), and the warranty amounts ranged from \$10k to \$10M. Some of these warranties, however, had caveats; for example, IdenTrust, who offers a \$10M warranty, stipulates that each transaction is covered to a maximum of \$100k and each relying party is covered to a maximum of \$250k. As shown in Table 3.7, the risk for each certificate varied widely, ranging from around 0.001% up to almost 8.5%.

To set sample RP values, we can conservatively estimate the risk of a CA to be 10%; thus the affected domain payout could be 10 times the RP cost. Using the median cost of a certificate as a reference, we can estimate that a standard RP will cost $\rho = \$299$, and thus $a = \$2990$. Similarly, we can use the risk to estimate that 10% of RPs may be terminated early, and thus set the minimum termination payout as $\tau = \$29.90$. We can estimate the reporting fee to be a small but non-trivial amount, such as $m = \$5$. Given these values, we can see that the constraints from Section 4.2 are easily satisfiable, for example, $\rho = \$299$, $a = \$2990$, $t = \$150$, and $\delta = \$100$.

3.9 Discussion

In this section, we discuss the insights, various limitations and proposed future work of IKP.

Blockchain Weaknesses Blockchains have several weaknesses which have been demonstrated in practice. For example, mining pools controlled a majority of hashing power in the network before [37], allowing double-spending attacks and suppression of selected transactions. There are also attacks that can be mounted with less than half of the network's hashrate [59]. Moreover, there may be bugs in the IKP contract which could result in exploits such as the one that plagued the DAO in Ethereum [39], and check and reaction contracts may have bugs as well. Learning to write secure contracts is difficult [50], but we can build on existing work such as smart contract formalization [78] to make IKP more robust.

Compelled Certificates In this work, we did not explicitly attempt to defend against nation-states who can compel CAs to issue unauthorized certificates, as they are irrational adversaries with an effectively unlimited budget. However, client-side extensions (described below) can prevent MITM attacks even by such adversaries and record the certificate for out-of-band responses.

Deployment Benefits While detectors and miners can benefit financially in IKP, domains, CAs, and clients can also benefit from deploying IKP. Beyond RP payouts, domains can be quickly alerted to CA misbehavior because of detector payouts. IKP also protects against misbehavior by both internal and external CAs, and thus allows domains to have greater confidence in their CAs, particularly those with good proven reputations. CAs in IKP profit from good behavior, and selling RPs provides a value-added service by which CAs can compete with free certificate services such as Let's Encrypt [1]. Moreover, CAs can use IKP to prove a history of good behavior, attracting more business.

Protecting Clients In this chapter, we described ways to compensate domains affected by potential MITM attacks, but even with RP-based payouts, clients have no protection from the use of unauthorized certificates. To protect clients, we can extend the IKP authority to record each unauthorized certificate. A browser extension can then check this data during the TLS handshake or maintain a local copy of unauthorized certificates and reject any certificate that IKP has confirmed to be unauthorized. A browser extension could even contribute to this certificate blacklist, checking certificates the client sees against the relevant domain's DCP and reporting the certificate if the it violates the DCP.

In our Ethereum instantiation, the first browser extension could be implemented using *events*, which leverage the logging functionality of the Ethereum virtual machine. Events cause a logging opcode to execute in the Ethereum VM, storing information in the receipt of the transaction that generated the event [131]. Event information is not accessible

to contracts, but rather is designed for use in applications that can access the blockchain history. A third-party service or the clients themselves could then store the blockchain history to maintain the certificate blacklist.

The second browser extension could be implemented by sending certificates to the relevant domain's check contract. Because these checks do not modify any state, they do not cost any gas to execute, and can even be run locally. The certificates also do not need to be checked synchronously. If an unauthorized certificate is discovered, the browser extension could automatically carry out the pre-report and reporting steps. This automated reporting mechanism provides an incentive for clients to deploy IKP and further deters CA misbehavior by increasing the chance that an unauthorized certificate will be quickly detected.

Future Work We next plan to explore the following improvements to IKP. First, we plan to further investigate possible designs for check and reaction contracts, such as how a system such as Town Crier [133] could be used to allow these contracts to interface with real-world data. We also plan to implement our browser extensions described above. Finally, we plan to leverage work in mechanism design [105, 29] to formally verify the incentive structure of IKP.

Chapter 4

Smoothly Transitioning to a More Resilient Web PKI

In this chapter, I describe how we can reduce MITM attacks during the deployment of an improvement to the Web PKI. In particular, I explore how we can overcome the signaling problem and the recoverability problem (described in Section 1.3.2) to prevent downgrade attacks and make MITM attacks more difficult while avoiding pitfalls of misconfiguration. In doing so, I tackle several important questions. First, how can we effectively prevent downgrade attacks without imposing an undue burden on clients, domains, or CA? Second, how can we simultaneously achieve greater confidence in the public key presented by a domain while allowing domains to easily change this key?

To address these questions, I propose **Certificates with Automated Policies and Signaling (CAPS)**, a system that leverages existing infrastructure in the Web PKI to signal the deployment of HTTP over TLS (HTTPS) and Web PKI improvements and to protect against CA misbehavior with secure yet easily changed policies. CAPS uses information from public logs and certificate databases to determine which domains have deployed HTTPS; the presence of a certificate indicates such a domain. The set of these domain names can be represented in a compact data structure, providing much greater coverage of domain names than existing approaches. Without this bit of information, TLS stripping [90] can occur, causing the client to ignore even the existing PKI. With the use of this bit, the remaining signaling information can be communicated as an extension to the TLS handshake. CAPS also uses log information to determine one or more *authoritative* public keys for a domain, that is, keys that should be trusted by clients to identify domains and to secure policies in systems such as PoliCert [126]. I show that together, these two components of CAPS are sufficient to show that a MITM attack cannot occur as long as the TLS handshake is secure and as long as fewer than n trusted parties (where n is set on a per-domain basis) are compromised.

Because all known instances of CA misbehavior up to this point have involved a single misbehaving CA, CAPS allows domains to use multiple independent certificate chains to authenticate a single public key. Then, CAPS can use log information to determine the maximum number of certificate chains for a given key, and signal the *number* of such chains that clients should expect. This protects against any fewer number of misbehaving trusted parties while

allowing the domain to easily change its authoritative public key by gaining another set of independent certificate chains. Moreover, this can be done automatically by simply monitoring logs, meaning that domains do not have to consult with CAs or logs to change their authoritative key.

Finally, we observe that such a strongly authenticated, authoritative public key for a domain is a powerful force: it can be used on its own to provide greater confidence in a site's identity, or it can be used to authenticate richer policies as proposed in prior work [30, 126]. CAPS can thus simplify the deployment of these proposed systems, whose existing deployment and certificate issuance strategies rely on complex coordination among domains, CAs, and public logs to certify these policies. CAPS also enhances the recoverability of these systems, which, as originally proposed, require waiting for days to replace a policy if the corresponding private key is lost or compromised.

In summary, I make the following contributions in CAPS:

- By leveraging a global view of the certificate ecosystem and extending previous work in data compression, I demonstrate that we can prevent downgrade attacks for the vast majority of known domains with a moderate storage and memory overhead for clients.
- By again leveraging a global view of the certificate ecosystem and extending previous theoretical work on evaluating public-key authentication, I show that a simple certificate policy, based on the minimum number of CAs that must be compromised, can provide resilience to CA misbehavior while enabling easy recovery in case of misconfiguration.
- By implementing the above as well as a custom TLS extension, I show that CAPS can be deployed with minimal changes to domains and only moderate overhead for clients.

I begin this chapter by describing the design of CAPS, in particular, the design of the signaling mechanism and the policy database. I then analyze the security of CAPS and informally show its security as well as its potential weaknesses. I then evaluate CAPS and estimate its cost on clients in terms of storage, latency, and bandwidth. I conclude the chapter with a discussion of orthogonal security issues to consider when deploying CAPS, how CAPS can be used in conjunction with other systems, and future work.

4.1 Design of CAPS

Following an overview of CAPS's design (Section 4.1.1), we describe CAPS in detail, including how it signals which domains have deployed HTTPS and improvements to the Web PKI (Section 4.1.2), how it provides stronger public-key authentication over the existing Web PKI (Section 4.1.3), and how clients establish secure end-to-end connections with servers (Section 4.1.4). We conclude by describing how CAPS thus enables the bootstrapping of more advanced policies (Section 4.1.5).

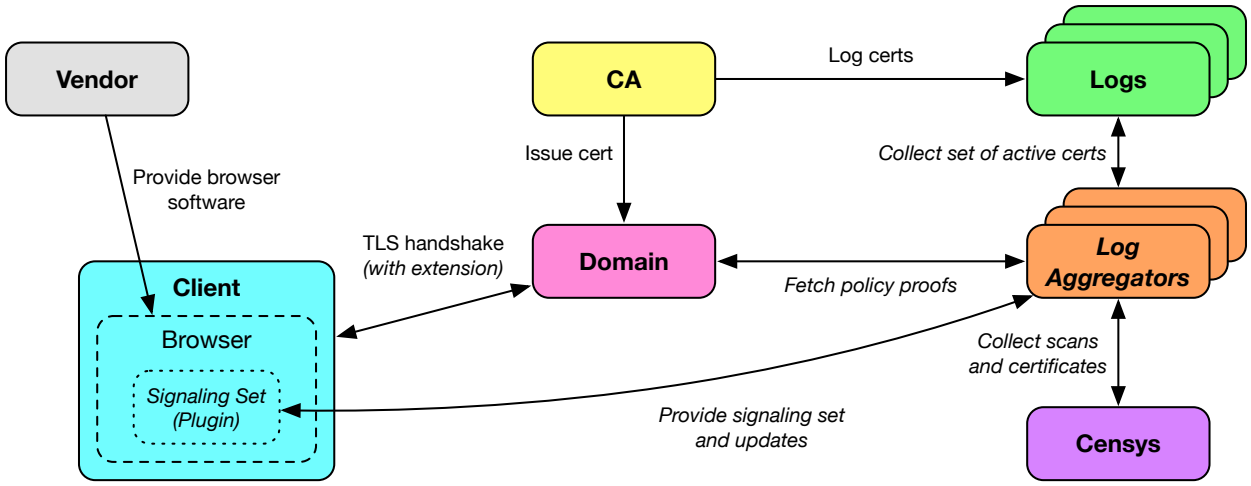


Figure 4.1: Overview of CAPS architecture (log auditors and monitors not shown). Dotted lines denote the browser and its components, and italic text denotes new entities or actions in CAPS.

4.1.1 Overview

Goals CAPS primarily aims to enable a smooth transition from the Web's existing PKI to an *improved PKI* (which can range from an extension of the existing PKI to a new PKI altogether). We assume that during this transition, both the existing and improved PKIs will coexist, and that the improved PKI will make MITM attacks more difficult to carry out. Hence, CAPS must prevent downgrades to the old PKI. More precisely, if a client and server both support the improved PKI, then when they perform a handshake, they should negotiate a session key based on the domain's public key as certified in the improved PKI. As secondary objectives, we also seek to prevent domains from becoming inaccessible due to misconfiguration, private key loss, or private key compromise, and to minimize the changes to existing interactions between clients, domains, and CAs.

Adversary Model In designing CAPS, we consider an adversary with the goal of successfully mounting a MITM attack. We assume that the adversary has access to the signing keys for n CAs and can thus issue and revoke arbitrary certificates under these keys. We also assume that the adversary has full control of the network during the handshake protocol; that is, the adversary can intercept, drop, or modify all messages sent among all entities described below. We assume that the adversary cannot mount a MITM in the improved PKI, or break standard cryptographic primitives.

Architecture Figure 4.1 illustrates the CAPS architecture and how CAPS achieves the goals stated above. Since CAPS transitions from the current Web PKI, it necessarily includes the entities in the current PKI:

- *Domains* serve webpages to clients. Each domain has a name such as `example.com`.
- *CAs* issue certificates to domains. Each certificate binds a set of names to a single public key.

- *Clients* connect to domains over HTTP or HTTPS, and in the latter case, verify the binding between a domain's name and public key.
- *Browser/OS vendors* (hereafter simply *vendors*) provide the software by which clients connect to domains and verify domains' certificates.
- *Public logs* maintain a publicly auditable, append-only database of certificates, such as those used in CT.

CAPS introduces a new entity, the *log aggregator*, that uses publicly available data to maintain a database of domains that have deployed HTTPS and/or the improved PKI. As our figure shows, there may be multiple independent log aggregators. While throughout this section we assume that the log aggregator is separate from other entities and that there is a globally accepted set of known and trusted log aggregators, in Section 4.4 we discuss how we can relax these assumptions, namely, by arguing that browser vendors should take on this responsibility.

As shown in Figure 4.1, many of the interactions between entities in the current PKI remain the same in CAPS. CAs issue certificates to domains and log newly-issued certificates as they do in the current PKI (with CT); clients and domains establish an encrypted communication channel through the TLS handshake, and vendors provide clients with browser software.

To prevent an adversary from forcing an HTTP connection (thus bypassing TLS completely), the log aggregators use data from public logs to construct a *signaling set*, which succinctly represents the set of all domains that deploy HTTPS. The log aggregators build this set by downloading the set of all currently valid (i.e., non-expired) certificates from the logs and extracting all domains named in these certificates. The log aggregators then make this set, as well as updates to this set over time, available to client browsers. With the signaling set, a browser making a request to a server can first check whether or not the server has deployed HTTPS.

To allow domains to certify their public keys more securely than in the current PKI, log aggregators use data from public logs to construct a set of *CAPS policies*, which allows each domain to establish an *authoritative public key* in the *curent* Web PKI. CAPS policies take a simple and intuitive approach: *treat whichever public key is backed by the most independent certificate chains*[1] *in the current PKI as authoritative in the improved PKI*. A domain wanting to increase client confidence in one of its public keys can then simply obtain more independent certificates for that key, and the log aggregators will update their CAPS policies for that domain.

Intuitively, log aggregators simply send a signed pair $(name, c)$ where *name* is a domain name and c is the *CAPS policy value*, the number of independent certificate chains backing an authoritative key for *name*.[2] To prevent an adversary from downgrading handshakes in the improved PKI to a handshake in the existing Web PKI, each log aggregator indicates which domains in its signaling set have a key policy value greater than 1.

[1] This term means that the certificate chains share no public keys except at the leaf.

[2] *name* may have more than one authoritative key; if public keys K_1 and K_2 are both backed by three independent chains, for example, then both of their public keys are treated as authoritative for *name*.

4.1.2 Building the Signaling Set

The signaling set represents 1. a set of fully-qualified domain names (FQDNs) (hereafter names) known to have deployed HTTPS (by virtue of having a known public-key certificate for the Web PKI), and 2. the subset of names that have adopted the improved PKI (by virtue of having multiple independent public-key certificates for the same public key in the Web PKI). Formally, the signaling set is a pair $(S_{\mathsf{HTTPS}}, S_{\mathsf{MC}})$ where S_{HTTPS} and S_{MC} are unordered sets of valid[3] names in ASCII and $S_{\mathsf{MC}} \subseteq S_{\mathsf{HTTPS}}$. The set supports a query operation, formally defined as $\mathsf{query} : \Sigma^* \to \{\mathsf{no\_https}, \mathsf{one\_cert}, \mathsf{multi\_cert}\}$ where Σ^* is the set of all ASCII strings and no_https, one_cert, and multi_cert are values indicating whether a string is a name known to have no HTTPS certificate, one certificate, or more than one independent certificate, respectively. Specifically, query must satisfy

$$\begin{aligned}
\mathsf{query}(\mathit{name}) = \mathsf{no_https} &\iff \mathit{name} \notin S_{\mathsf{HTTPS}} \\
\mathsf{query}(\mathit{name}) = \mathsf{one_cert} &\iff \mathit{name} \in S_{\mathsf{HTTPS}} \setminus S_{\mathsf{MC}} \\
\mathsf{query}(\mathit{name}) = \mathsf{multi_cert} &\iff \mathit{name} \in S_{\mathsf{MC}}
\end{aligned}$$

for all $\mathit{name} \in \Sigma^*$.

To build its signaling set, a log aggregator must first determine S_{HTTPS} and S_{MC}, which it can achieve using the set of all certificates in the Web PKI. The log aggregator maintains a database of current certificates by using information from public sources, namely, 1. public logs that collect Web certificates, 2. CRLs such as those published by CAs [46] or browser vendors [80, 62], and 3. revocation information retrieved from OCSP responders [116]. The log aggregator updates this database regularly (e.g., each day), thus maintaining a list of certificates valid on a given day.[4]

The log aggregator obtains the names from the current and valid certificates in the databaseand converts internationalized domain names to ASCII using the IDNA conversion process [76]; the resulting set of distinct names is S_{HTTPS}. The log aggregator also analyzes the certificate chains in this set to determine S_{MC}, a procedure we describe in Section 4.1.3. The log aggregator then creates a representation of the signaling set and makes it available to client browsers.

Because the signaling set must be available to each client that supports the improved PKI and may contain all names in DNS, the log aggregator must succinctly represent this set to minimize the bandwidth and storage burdens on each client. However, simply minimizing the storage burden is insufficient; clients must store the signaling set in memory to minimize connection latency overhead. Thus the log aggregator must also represent this set in such a way that clients can query the set with minimal latency and memory.

[3] A valid name is a Unicode or ASCII string up to 253 bytes in length overall, and has no label longer than 63 bytes [100]. We further added the requirement that the name has a TLD that is a current global TLD according to ICANN.

[4] A certificate is valid on a given day if the signature on the certificate is valid and if that day falls in the certificate's validity period as defined by its `notBefore` and `notAfter` fields [46].

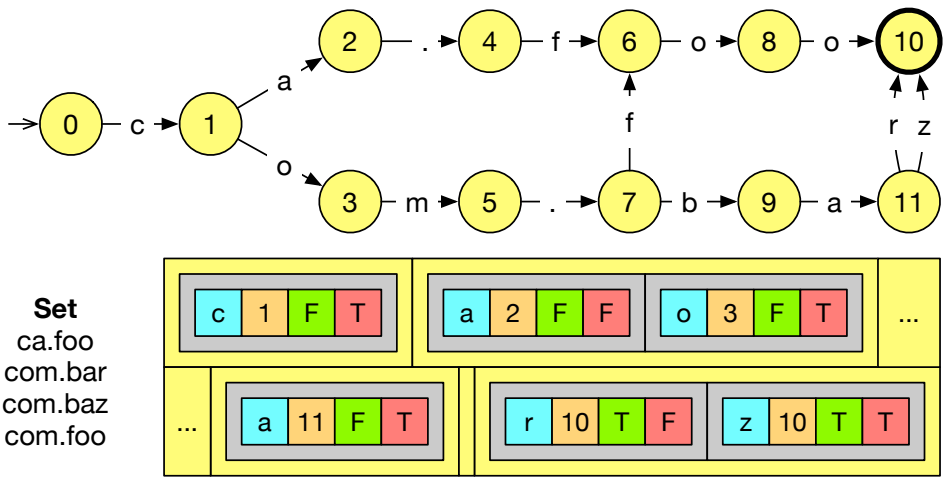


Figure 4.2: A simple DAFSA with one-character transitions, along with the set of strings it represents and its succinct representation as a vector of variable-length bitstrings. In this representation, each edge (gray) consists of a label (blue) and next state (orange), along with flags indicating whether the next state is an accept state (green) and whether the edge is the last one outgoing from this state (red).

We considered several approaches when determining how to represent the signaling set. A Bloom filter [35] supports efficient set membership queries, but has false positives (which for this application would result in flagging a site as an attack when there is none) and for the scale of data we considered, resulted in too large of a storage burden (Section 4.3.1). A filter cascade [115] would eliminate false positives, but has the virtually impossible prerequisite of knowing all names in the DNS namespace, which requires the cooperation of all TLD operators (including many national governments). Finally, using an existing data compression utility (e.g., zpaq) with aggressive compression parameters could minimize the required storage space, but would also require decompression each time a client tried to connect to a site whose HTTPS deployment status was not known, resulting in significant latency overhead (Section 4.3.3).

In our log aggregator prototype, we ultimately chose to represent the signaling set as a data structure known as a *deterministic acyclic finite state automaton (DAFSA)*, which succinctly stores a set of strings and supports efficient membership queries in this set, and can be efficiently constructed with minimal size [48]. As shown in Figure 4.2, a DAFSA takes advantage of both common prefixes and suffixes that appear in a set of strings; since such patterns are frequent in a large set of names, much of the redundancy in S_{HTTPS} can be removed with this approach. Additionally, DAFSAs can also be represented succinctly, using an approach we summarize below [49]. Given the characteristics of our input set as described in Section 4.3.2, we make two additional design changes to our DAFSA representation that further reduce its size.

Formal DAFSA Definition To precisely describe these changes, we begin by presenting a formal framework to describe DAFSAs. Formally, a DAFSA is a tuple $(\Sigma, S, s_0, \delta, F)$ where 1. Σ is a set of possible *input symbols*, 2. S is

a set of *states*, 3. s_0 is an *initial state* where $s_0 \in S$, 4. $\delta : S \times \Sigma \to S$ is a partial function called the *state transition function* that maps a state-symbol pair to a new state, and 5. F is a set of *accept states* where $F \subseteq S$. The DAFSA also has the restriction that the state transition function is *acyclic*, that is, there is no sequence of states and symbols $(s_1, \ldots, s_n), (\sigma_1, \ldots, \sigma_n)$ where $\delta(s_i, \sigma_i) = s_{i+1}$ for each i where $1 \leq i < n$ and $\delta(s_n, \sigma_n) = s_1$.

We represent queries to the signaling set within the DAFSA as follows: let S contain a symbol μ. Then, if there exists sequence of states and symbols $(s_1, \ldots, s_n), (\sigma_1, \ldots, \sigma_{n-1})$ that satisfies 1. $\delta(s_i, \sigma_i) = s_{i+1}$ for each i where $1 \leq i < n$, 2. $s_1 = s_0$, 3. $\sigma_1 \| \ldots \| \sigma_{n-1} = \textit{name}$, and 4. $\delta(s_n, \mu) \in F$, we say that $\mathsf{query}(\textit{name}) = \mathsf{multi\_cert}$. If no such sequence exists, but one satisfying the first three conditions and $s_n \in F$ exists, then $\mathsf{query}(\textit{name}) = \mathsf{one\_cert}$. Otherwise, $\mathsf{query}(\textit{name}) = \mathsf{no\_https}$.

DAFSA Representation We begin by building the DAFSA as described in previous work [48]. Though this previous work assumes transitions based on a single character, we consider the possibility of multi-character symbols and thus our symbol set $\Sigma^{\leq 253} = \bigcup_{i=0}^{253} \Sigma^i$, where Σ is the set of all ASCII characters allowed in domain names.[5]

We succinctly represent this DAFSA as a bitvector by following the high-level approach of previous work [49]. Intuitively, we represent the DAFSA as a sequence of state encodings, which mostly consist of outgoing transition encodings. By our definition of δ, for a state s, if $\delta(s, \sigma) = t$, then each outgoing transition must be represented by an encoding of the label σ and the destination state t. We observe that in this construction, the overall number of outgoing transitions, as well as the size of the representation of each transition's label and destination state, strongly influence the size of the final bitvector. We thus focus on leveraging patterns in the underlying data to minimize the size of the DAFSA representation.

Path Compaction We extend the design of prior work with *path compaction*, which minimizes the DAFSA representation by reducing the overall number of transitions. Intuitively, path compaction removes a connected set of states from the DAFSA and replaces transitions into or out of this set with transitions equivalent to paths through the set. As we formalize below, we can model this process as the transformation of one DAFSA into another and use this model to determine how we should select a set of nodes to ensure a minimal DAFSA representation.

Given a DAFSA $(\Sigma, S, s_0, \delta, F)$, we define a *walk* between s_1 and s_m to be a sequence of alternating states in S and symbols in Σ, written $(s_1, \sigma_1, \ldots, s_m)$, where for all i where $1 \leq i < m$, either $\delta(s_i, \sigma_i) = s_{i+1}$ or $\delta(s_{i+1}, \sigma_i) = s_i$. A walk where $\delta(s_i, \sigma_i) = s_{i+1}$ for all i where $1 \leq i < m$ is called a *path* from s_1 to s_m. We say that a set of states $T \subseteq S$ is a *connected component* if either $T \subseteq F$ or $T \cap F = \varnothing$, and for any two states $t, u \in T$, there exists a walk between t and u. The *upstream states* of a connected component T, written $U(T)$, is the set of all states $s \in S \setminus T$ for which there exists a state $t \in T$ and a symbol σ where $\delta(s, \sigma) = t$. The *downstream states* of T, written $D(T)$, is the set of

[5] Recall that DNS names can be a maximum of 253 characters.

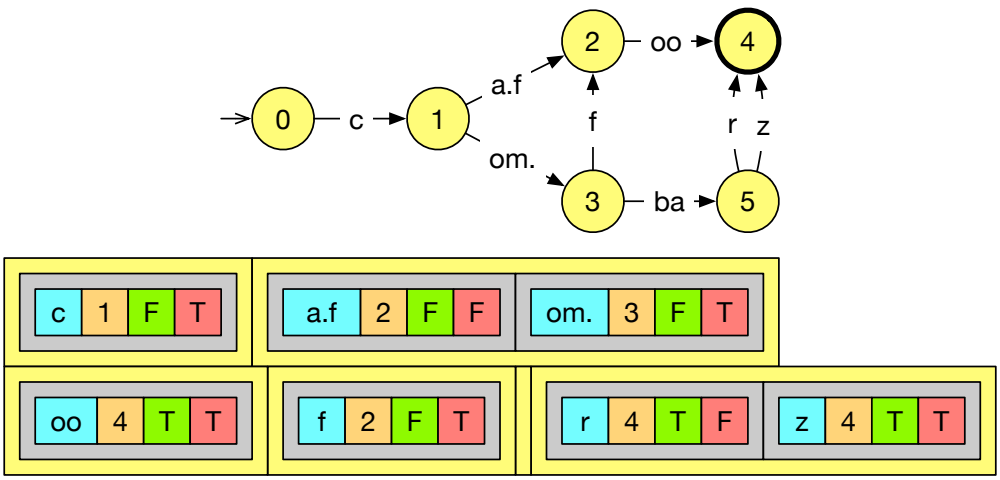


Figure 4.3: The DAFSA from Figure 4.2 with its long edges compacted. The binary representation is more succinct than its counterpart.

all states $s \in S \setminus T$ for which there exists a state $t \in T$ and a symbol σ where $\delta(t,\sigma) = s$. A path *through* a connected component T is a path $(s_1, \sigma_1, \ldots, s_m)$ where $s_1 \in U(T)$, for all i where $1 < i < m$, $s_i \in T$, and $s_m \in D(T)$.

Path compaction consists of repeatedly 1. selecting a connected component T within the DAFSA, 2. calculating the estimated reduction in representation size from compacting the paths through T, and 3. if the change reduces the size of the DAFSA representation, removing these states from S and replacing the paths through T with an equivalent set of transitions. Specifically, when removing T, we transform the DAFSA $(\Sigma, S, s_0, \delta, F)$ to $(\Sigma, S \setminus T, s_0, \delta', F)$, where for all paths $(s_1, \sigma_1, \ldots, s_m$ through T, $\delta'(s_1, \sigma_1 \| \ldots \| \sigma_{m-1}) = s_m$.

Our goal is to select components that result in the greatest reduction in size. To determine the size reduction of removing a connected component, we must consider both the reduction in the number of edges in the DAFSA as well as any changes to the symbol and destination state entropy, as removing a component would cause the distributions underlying these values to change. To quantify this change, we define several helpful variables.

For a set $X \subseteq S$, let $\tau(X)$ denote the set of transitions that start or end in X, that is, the number of triples (s,t,σ) where $\delta(s,\sigma) = t$ and s or t (or both) is in X. For a connected component C, let $\pi(C)$ denote the set of paths through C. Then the change in the number of edges by removing C through path compaction is $|\pi(C)| - |\tau(C)|$. By examining the distribution of symbols in $\tau(S)$ and $\tau(C)$, as well as the concatenated symbols in $\pi(C)$ (which we can find via depth-first search from $U(C)$), we can compute the change in entropy in symbols and similarly for destination states, which we writes as ΔH_σ and ΔH_δ, respectively. If we know the original entropies H_σ and H_δ, then we can compute the difference in size between the two DAFSAs as

$$|\tau(S)|\,(\Delta H_\sigma + \Delta H_\delta) + (|\pi(C)| - |\tau(C)|)\,(H_\sigma + \Delta H_\sigma + H_\delta + \Delta H_\delta) \tag{4.1}$$

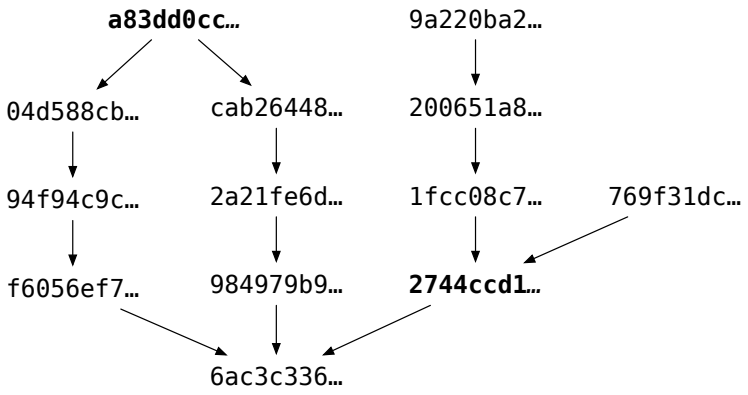


Figure 4.4: Sample certificate fingerprint graph. An arrow from A to B indicates that a certificate with fingerprint A is the authority public key for a certificate with fingerprint B. Though the leaf certificate `6ac3c336`... has four chains, only two of the chains are independent.

and only remove C if this quantity is negative.

We found several classes of components that, for our underlying set of domain names, provided substantial reductions in the size of the DAFSA representation. The first was to select what we call *isolated paths*, that is, paths of the form $(s_1, \sigma_1, \ldots, s_m)$ where for all i such that $1 < i < m$, s_i only had one incoming and one outgoing transition. Because building the DAFSA as described in prior work resulted in a significant number of isolated paths, performing path compaction with all such paths resulted can result in significant size savings. We also found that selecting a constant α and then selecting components consisting entirely of states that had one incoming transition and α outgoing transitions (or vice versa) yielded more modest but still significant size reductions for $\alpha = 2$ and $\alpha = 3$.

4.1.3 Building the CAPS Policy Database

The policy database represents a binding between a name and a policy, that is, the number of independent certificate chains that a client should expect during a handshake with a server corresponding to the name. To construct and maintain this database, each log aggregator must keep track of the certificates and chains active for a domain at any given time. The log aggregators again rely on the data from public sources for this information, which includes the data used to build the signaling set, as well as certificate chains observed by public logs.

A log aggregator uses this data to maintain an internal database with (many-to-many) maps of 1. certificates to names, 2. certificates to public keys, and 3. certificates to chains. Regular updates to this database (described in Section 4.1.2) ensure that the log aggregator has a list of currently valid certificates. The log aggregator can then use the database to construct a mapping of names to policies. Specifically, the aggregator creates a graph of certificate fingerprints as shown in Figure 4.4, and computes the *policy value*, the minimum number of CA public keys that must be compromised to fraudulently issue this certificate. The policy value can be computed using a straightforward

approach (e.g., computing a minimal vertex separator), allowing the log aggregator to easily construct a mapping of certificate fingerprints to policy values. The log aggregator can then perform a series of simple join operations to map each name to the maximum policy number associated with a certificate containing the name.

The log aggregator constructs this name-to-policy mapping each time it receives data from public sources (recall from Section 4.1.2 that this occurs at regular, scheduled intervals). Once it has created the mapping, the log aggregator certifies the name–policy-value pairs in this mapping by timestamping and signing them.[6]

With the policy databases of the log aggregators, a domain can provide the information necessary for clients to establish the domain's authoritative public key in the improved PKI. The domain periodically downloads the latest *policy proofs*, which are signed and timestamped name-policy pairs, from each of the log aggregators. For a name *name*, a policy value c, a timestamp ts, and a log aggregator A, we specify the policy proof to be

$$\mathscr{P}(A, name, c, \mathsf{ts}) = \left\{\mathsf{ts}, K_A, \mathsf{Sign}(K_A^{-1}, name \| c \| \mathsf{ts})\right\} \tag{4.2}$$

where K_A and K_A^{-1} denote respectively the public and private keys of A, and $\mathsf{Sign}(K, m)$ denotes a signature on m with private key K. The domain keeps these proofs for use in the handshake described below.

4.1.4 Connection Establishment

To establish a connection to a domain, a client first looks up the domain's name as a query to the signaling set. Due to the way that querying occurs in the DAFSA-based implementation of the signaling set, the client can observe the state transitions in the DAFSA and, if the symbols making up the domain name lead to a state that is not an accept state, the client can supply μ to determine whether that transitions to an accept state (and hence the domain has a policy value greater than 1). If this query returns $\mathsf{one\_cert}$ or $\mathsf{multi\_cert}$, then the client performs the CAPS handshake to establish a connection with the domain.

The CAPS handshake protocol allows a client to verify a domain's authoritative public key. The TLS handshake protocol [113] provides support for open-ended extensions (implemented in cryptographic libraries), and thus we designed our protocol as an extension within the existing TLS handshake. During the initial ClientHello message, the client includes a CAPS extension message, which consists of a single integer k, indicating a number of policy proofs that the domain should send back. We assume that a reasonable maximum value for k is enforced; for example, a domain may abort the handshake if k is beyond some acceptable maximum.

The domain then sends back a ServerHello message, which contains an extension message as a response to k. The extension message contains recent policy proofs from k distinct log aggregators, as well as a number of certificate

[6]For efficiency, the log aggregator can sign a subset of pairs each day, and have each client consider a signature valid for multiple update intervals. For example, the log aggregator could sign a third of the pairs each day, and clients could treat a signed, timestamp pair as valid for at least three days.

chains, which allows the client to verify the domain's policy value and the certificate chains that back this value. Specifically, a domain with hostname *name* and policy value c selects a set of log aggregators $\{A_1,\ldots,A_k\}$ and sends back

$$\{\mathscr{P}(A_1,name,c,\mathsf{ts}_1),\ldots,\mathscr{P}(A_k,name,c,\mathsf{ts}_k),\mathscr{C}_1,\ldots,\mathscr{C}_{c-1}\} \tag{4.3}$$

where ts_i is the timestamp of the policy proof from A_i and $\mathscr{C}_j$ is a certificate chain for *name*. In the extension message, the domain only sends $c-1$ certificate chains because the remaining chain will be sent in the ServerCertificate message of the handshake.

The client then checks that 1. the signature on each policy proof is valid, 2. the timestamp for each policy proof is sufficiently recent, 3. the name for each policy proof equals the domain's name, 4. the policy value for each policy is one more than the number of certificate chains sent in the domain's extension message, and 5. each certificate chain is valid as specified in the X.509v3 standard [46]. If the above checks pass, the client can then continue with the handshake, which requires the client to verify an additional certificate chain in the ServerCertificate message and perform all other checks required by the TLS handshake protocol.

4.1.5 Bootstrapping Advanced Policies

Once an authoritative public key for a domain has been established through the CAPS handshake, signatures made by the corresponding private key can be used to verify the binding between a domain and a richer set of policies. For example, in systems such as ARPKI [30] and PoliCert [126], these policies can specify a set of CAs that are authorized to issue certificate for the domain, or even make the domain a CA for itself by pinning specific public keys to the domain. In this way, the authoritative public key established in CAPS can be used to bootstrap confidence in these advanced policies while preventing downgrades to the old PKI.

This bootstrapping approach obviates the need for logs to directly store the policies, which can be quite large in these systems. Moreover, in CAPS, this bootstrapping can take place at any time during deployment, which means that in the case of a lost or compromised private key, a domain can simply obtain $c+1$ new, independent certificate chains. This contrasts with previous proposals, which often rely on heavyweight processes that involve manual CA intervention.

The authoritative public key can also be used directly, as a way of providing clients with more confidence in the current PKI.

4.2 Security Analysis

In this section, we analyze the security that CAPS provides. We begin with our main security claim and informally argue that it holds given our adversary model. We then describe two potential weaknesses of CAPS not covered by our security claim, and argue that countermeasures (some of which are currently used) make these weaknesses difficult or impossible to exploit.

4.2.1 Main Security Claim

In making our main security claim, we assume that the client requests k policy proofs and that the domain has a domain name of D and a policy value of c. We assume that both the client and domain follow the handshake protocol as described. We also assume that our adversary controls $k-1$ private keys of log aggregators and the private keys of $c-1$ CAs, and we assume that the adversary is the sole entity that controls the private key corresponding to a public key K_M. We assume that the adversary can intercept, suppress, replay, and modify any handshake message sent between the client and domain, and can sign any message it can construct with the information it has using its private keys. We also assume that the adversary cannot obtain a certificate binding D to K_M from any CA besides the ones it controls. Under these assumptions, we claim that the client will abort the handshake protocol if it receives any certificate chain containing K_M as the leaf public key. The truth of this claim guarantees the security of CAPS because the adversary cannot mount a MITM attack without the client completing the CAPS handshake based on a key that the adversary controls.

We first show that the adversary cannot convince the client that that the domain has a policy value other than c. Because the client requests k policy proofs for D, the client will abort the handshake if the extension data in the ServerHello message does not contain k independent policy proofs. Because we assume that the adversary only has access to $k-1$ log aggregator signing keys, we know that the adversary cannot use those keys to generate k independent proofs. Specifically, if the adversary sends a set of proofs and there are fewer than k valid proofs, or if any of them fails to prove that c is the policy value for D, then the client will abort the handshake.

Recall that the client determines c as the one plus the number of valid, independent certificate chains sent in the extension message. We show that the adversary cannot convince the client that the K_M is the domain's public key. From our assumption that the adversary can access the signing keys of $c-1$ CAs, we know that the adversary controls *at least* $c-1$ private keys. From this assumption, however, we also know that the number of *independent* certificate chains that the adversary can generate is *at most* $c-1$. It is straightforward to show this by induction on c, with $c=2$ as the base case.[7] If the adversary generates $c-1$ independent certificate chains for K_M and sends these chains with the policy proofs in the extension message, then the client will proceed with the handshake. However, the adversary

[7] Note that the $c=1$ case is equivalent to the current Web PKI (i.e., the domain only provides a single certificate chain).

will be unable to generate a final independent certificate chain to send with the ServerCertificate message, causing the client to abort the handshake at that point.

4.2.2 Potential Weaknesses

Our main security claim shows that under our assumptions, the adversary cannot mount a successful MITM attack on a client and domain. Therefore, we can conclude that the failure of log aggregators or CAs are not sufficient to enable a MITM attack. However, we did not describe the consequence of misbehavior by the other parties shown in Figure 4.1: the data sources (namely, public logs) or the browser vendor. We now show that while failures of these entities can have negative consequences for the performance or security of CAPS, we consider these cases out of the scope of this work because they are unlikely to succeed in practice or are catastrophic failures that can circumvent almost any practical defense.

Log aggregators rely on public logs to obtain a complete view of certificates in the Web PKI. However, given the open nature of many of these logs (i.e., the fact that anyone can submit certificates to them or otherwise make certificates available to them), an adversary may take advantage of this openness to force the log aggregators to increase their resource consumption. For example, an adversary who controls a CA signing key can issue millions of certificates and make them available to the logs, causing the log aggregators to store these certificates when updating their respective policy databases.

While CAPS provides no mechanism to stop this behavior, the issuance of these certificates produces non-repudiable proof of the adversary's behavior, and the requirement that newly-issued certificates be logged increases the likelihood of someone detecting this behavior. This detection would likely lead to consequences such as the revocation of the CA certificate by its issuing CA or by the browser vendor. Moreover, only an adversary with access to a CA signing key can carry out this attack; while Let's Encrypt does provide free certificates through an automatic protocol, it imposes a rate limit per domain based on the Public Suffix List,[8] thus requiring an adversary without access to a CA signing key to obtain many domain names to carry out this attack.

Adversary control of the browser vendor is a catastrophic failure that can enable a MITM attack under almost any circumstances, since this allows the adversary to completely control the client's experience with the Web. In this scenario, the adversary can arbitrarily deviate from the TLS or CAPS handshakes, insert its own root certificates, or display arbitrary pages. Even with CAPS, an adversary-controlled vendor can also intercept and modify the result of queries to the signaling set or rewrite HTTPS requests to HTTP requests, rendering CAPS's protection against downgrade attacks useless. While some countermeasures such as OS-level protections may protect the client, this scenario is severe enough that CAPS makes no security guarantees if it occurs.

[8] `https://publicsuffix.org/`

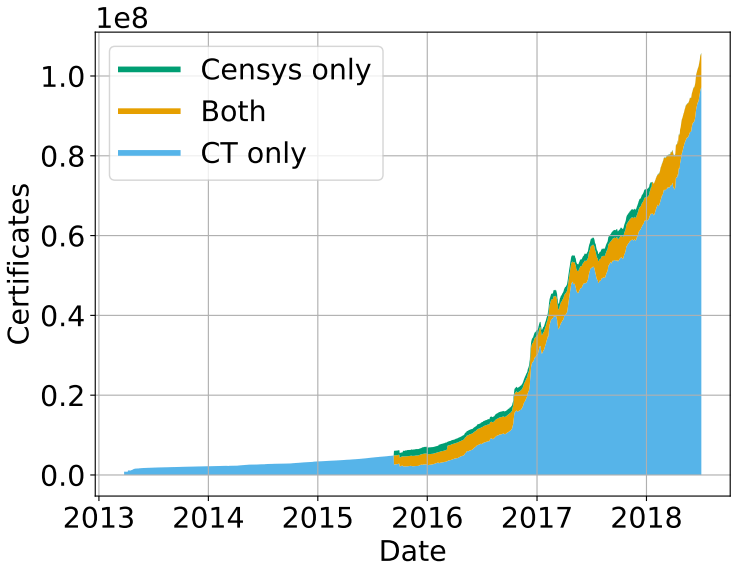


Figure 4.5: Certificates in the Web PKI as seen by Censys and CT.

4.3 Evaluation

In this section, we describe our evaluation of the performance and overhead of CAPS. We first describe the domains represented by the signaling set, including how we determined these domains. Next, we compare different approaches and optimizations for representing the signaling set. Finally, we describe the performance effects of CAPS on connection establishment.

4.3.1 Signaling Set Domains

Before building a representation of the signaling set, we need to determine which domains belong in the signaling set. Moreover, to determine the long-term viability of our solution, we also need to understand how the size of this set of domains may change in the future. Addressing both of these problems requires a complete and accurate view of the Web PKI, namely, the set of domains that are accessible over HTTPS.

We can obtain a view of the Web PKI using data from public logs, as we describe in Section 4.1. Specifically, we can obtain public-key certificates from sources such as Censys [52] and logs in CT [85]. From Censys, we collected 1,026 scans of the IPv4 address space over a period ranging from September 12, 2015 to July 3, 2018. From CT, we collected all entries from known CT logs that were not disqualified or unreachable as of July 3, 2018,[9] which totaled approximately 1.74B certificates from 26 logs over a period ranging from March 26, 2013 to July 3, 2018.

On each of these days, we consider an "active set" of certificates consisting of all certificates that were valid on that day and had an associated certificate chain rooted in one of the three major root certificate stores, determined

[9] `https://www.certificate-transparency.org/known-logs`

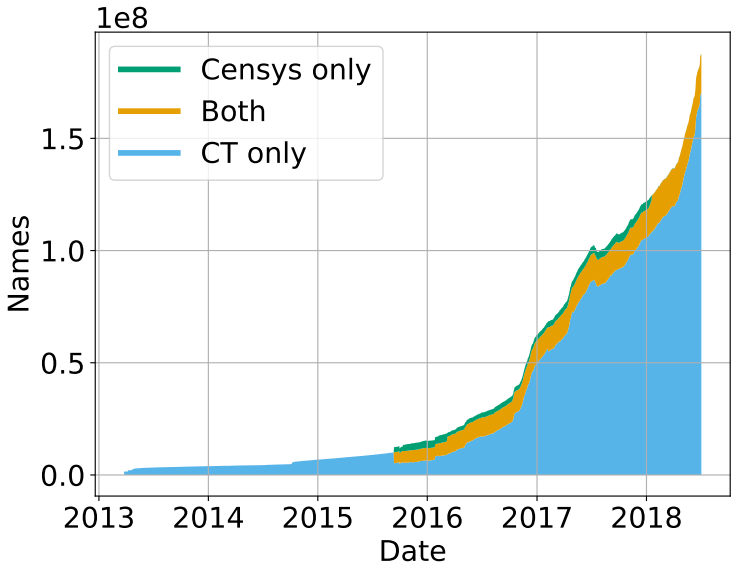


Figure 4.6: Names in the Web PKI as seen by Censys and CT.

by Apple, Microsoft, or Mozilla. In the Censys dataset, because we observed a great deal of churn (i.e., certificates disappearing and appearing in consecutive scans), we considered a certificate as in the active set from the time it was first observed in our data until its expiration. We then consider the number of unique, valid domain names, which we use to build the signaling set.

Figure 4.5 and Figure 4.6 shows the number of each observed by Censys, CT, and their overlap over time. Our results show that there are far more certificates (and consequently names and domains) observed by CT than by Censys. It is not clear whether the names seen in CT simply never deployed HTTPS in the wild or whether Censys was not observing certain certificates. However, to build a suitable signaling set, we performed a scan of port 443 using ZGrab [52] for all domain names we encountered and discarded any domain names that no longer respond. From this method, we isolated a set of 64,050,329 "valid" names that we used for testing.

4.3.2 Signaling Set Representation

As described in Section 4.1, my motivation for using a DAFSA-based representation of the signaling set was twofold: first, the representation has no false positives or negatives, and second, it can be searched in its compressed state, leading to a lower memory usage requirement for clients. To evaluate the effectiveness of these design decisions against other alternatives, I conducted an experiment to measure the space requirements for the signaling set in various representations, both on disk (when transmitted to the client) and in memory (when being used by the client during certificate verification).

In particular, I compared the plaintext representation of the signaling set (as of July 3, 2018) with a compressed

Table 4.1: Size of CAPS signaling set on July 3, 2018 (64,050,329 names) with various compression approaches. The representation size in memory is not shown.

Representation	**Size (MB)**
Plaintext	1,509
Bloom Filter (0.001% FP, best-case)	192
Bloom Filter (0.01% FP, best-case)	153
Bloom Filter (0.1% FP, best-case)	115
zpaq (method 1, 16 MiB blocks)	288
zpaq (method 5, 64 MiB blocks)	104
zpaq (method 5, 2048 MiB blocks)	104
DAFSA	185
DAFSA w/ path compaction (PC)	170
DAFSA w/ zpaq (method 5, 64 MiB blocks)	184
DAFSA w/ zpaq (method 5, 2048 MiB blocks)	145
DAFSA w/ PC, zpaq (method 5, 64 MiB blocks)	139
DAFSA w/ PC, zpaq (method 5, 2048 MiB blocks)	138

representation using Bloom filters, the compression utility zpaq,[10] and various configurations of my DAFSA-based representation. I also compressed the DAFSA-based representation using zpaq to find the size of the DAFSA-based representation on disk (as it would be used uncompressed in memory).

I specifically tested a Bloom filter with false positive rates of 0.001%, 0.01%, and 0.1%. Since the number of domain names is on the order of 100M [18], I expect that the number of false positives will be on the order of 1k, 10k, and 100k, respectively. I estimate that a false positive rate of 0.001% will be sufficient for most users. I tested zpaq using two compression methods, 1 and 5, where method 1 completes in a short amount of time (25 seconds) but compresses the input less while method 5 takes a long time (20 minutes) but yields excellent compression. Furthermore, with zpaq method 5, I tested with 64 MiB and 2048 MiB blocks, where larger blocks typically yield better compression. Finally, with my DAFSA-based representation, which extends previous work [49], I tested a plain encoding as well as an encoding of a path-compressed DAFSA, and compressed each of these encodings with zpaq method 5 using both block sizes.

The results are shown in Table 4.1. Bloom filters offer high compression even with a low false positive rate, and their in-memory representation is the same as their on-disk representation. However, they still have false positives, which would leadto those domains becoming inaccessible. While zpaq in any configuration does not have any false positives or false negatives and yields excellent compression, its in-memory representation of the signaling set would simply be the uncompressed set of domains, yielding a memory requirement of 1.5 GB. My DAFSA-based representation captures a "sweet spot" between these two alternatives, offering no false positives and, in the best case, an on-disk representation of just 138 MB with an in-memory representation of 170 MB.

[10]While I tested compression with other utilities, zpaq had the smallest size. Thus I used zpaq for the experiment to compare a near-best-case compression scenario.

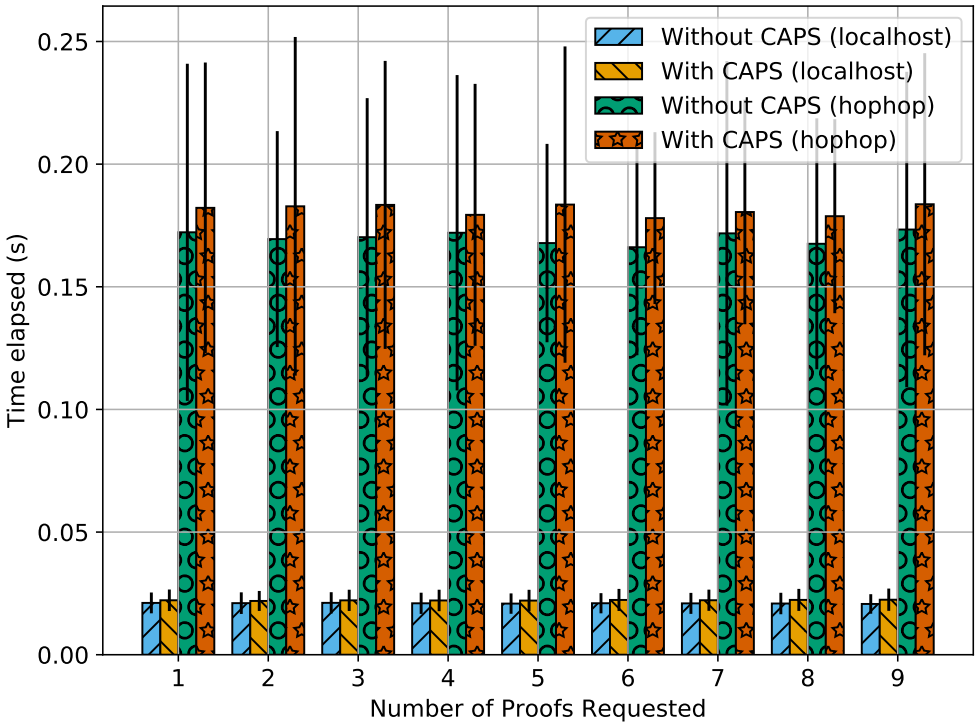


Figure 4.7: Connection latency based on the number of proofs requested.

4.3.3 Connection Establishment

To measure the performance of connection establishment in CAPS, we implemented the handshake as a custom TLS extension in the OpenSSL library. For concrete evaluation of this extension, we use nginx and curl with minor modifications to use our TLS extension.

Additionally, we constructed sample sets of domain names based on four parameters: 1. the number of proofs requested by the client during the ClientHello message (k), 2. the number of certificate chains the domain sends during the ServerHello message (c), 3. the average number of certificates per chain, and 4. the average size of each certificate chain. While varying each of these parameters, we measured the amount of extra data sent in the CAPS handshake, and the latency of the handshake both with and without the CAPS TLS extension.

We tested this both over the Internet (by connecting to a virtual private server with latency varying from 30 to 300 ms, referred to as *hophop*), as well as over the local loopback interface (referred to as *localhost*). The tests over the internet to *hophop* provide an indication of the effect of the extension on "real world" servers, which are usually accessed over WAN, whereas the *localhost* tests provide a lower bound on time added due to sending/receiving/processing extra data. A total of 15385 TLS connections were established for our testing: 5768 over hophop, 9617 over localhost.

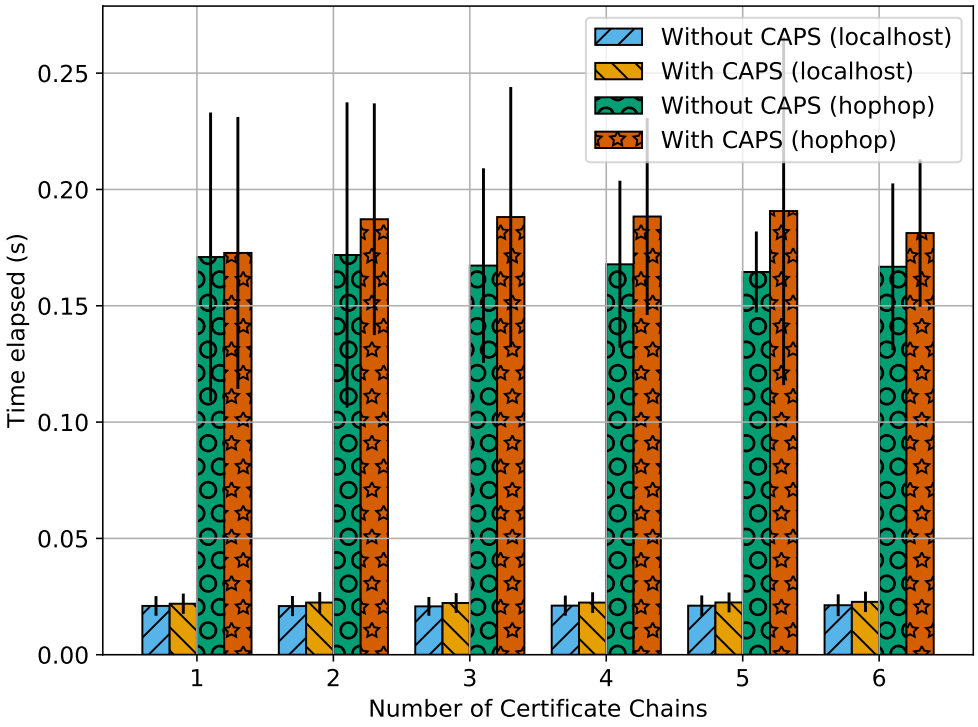


Figure 4.8: Connection latency based on the number of certificate chains sent.

Our results, shown in Figures 4.7, 4.8, 4.9, and 4.10, show that in comparison to the mean time elapsed, there is an approximately 5% increase in connection establishment time: an average of 11ms longer for hophop and 1.2ms for localhost. However, since our TLS extension does not add any extra round-trips to the handshake, the time added is small, especially in comparison to random measurement fluctuations, when we look at confidence intervals within one standard deviation from the mean (shown as error bars in the figures).

The extra data sent in the CAPS handshake is directly dependent on the size and number of certificate chains, as well as the number of proofs sent. In particular, the extra data sent from client to server is 1 byte, and the extra data sent from server to client is $(2 + (292 \times \#\text{proofs}) + (\sum_{\text{chain}} \texttt{sizeof}(\text{chain})))$ bytes. While this means that some connections may result in a great deal of extra data sent, we can expect that the vast majority of domains will not send additional certificate chains and the overhead will remain small.

4.4 Discussion

In this section, I discuss two issues to consider when deploying CAPS. In particular, I discuss security issues surrounding the integrity of the data sources used in CAPS. I also describe an avenue of further exploration that may reduce

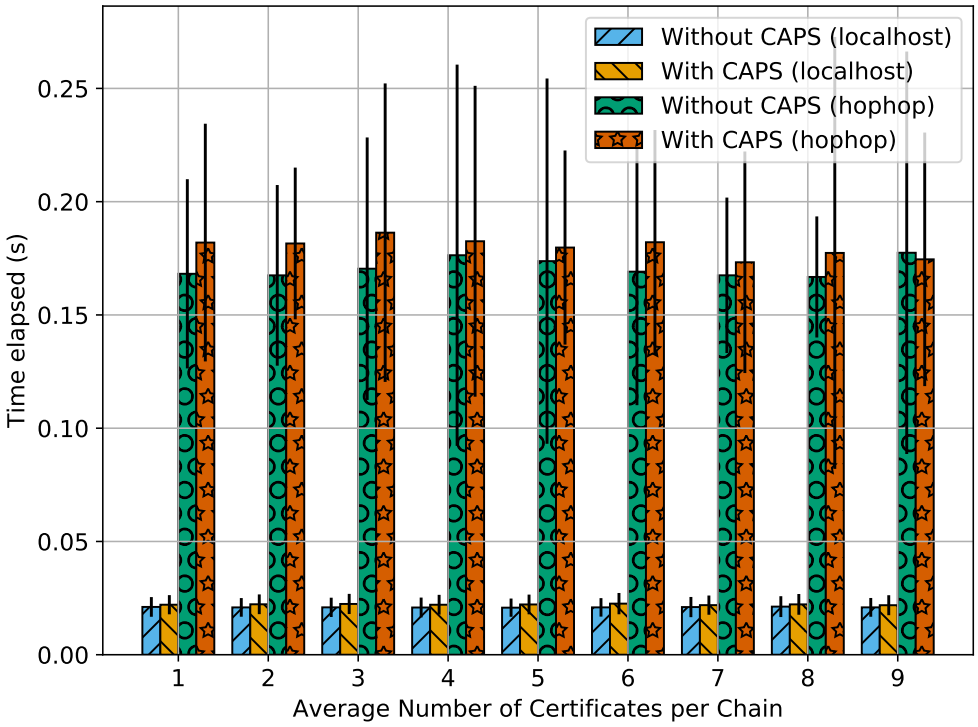


Figure 4.9: Connection latency based on the number of certificates in a chain.

the size of the DAFSA-based signaling set representation.

4.4.1 Data Source Integrity

CAPS relies on the integrity of the data sources (in our current prototype, Censys, and in the long term, the CT logs). Therefore, a CA who issues and publicizes an unauthorized certificate for a domain that does not deploy HTTPS can cause the domain to be included in the signal set and thus render the domain inaccessible. This fragility effectively allows an adversary to "poison the well" that CAPS relies on, and execute a denial-of-service attack of indefinite duration with just a single unauthorized certificate.

Unfortunately, recovering from such an attack requires immediate action. The affected domain could request that the CA revoke the certificate if the CA issued the certificate in error. If the CA deliberately misissued the certificate, the browser vendor could treat the certificate as revoked (though such action is unlikely unless the domain is popular). The affected domain could also upgrade to HTTPS, obtaining enough certificates to override the misissued certificate. In the event that the domain wishes to remain without HTTPS, we can allow domains to set a policy flag that signals that the domain will communicate over HTTP. The domain, however, must still send this policy and proof during

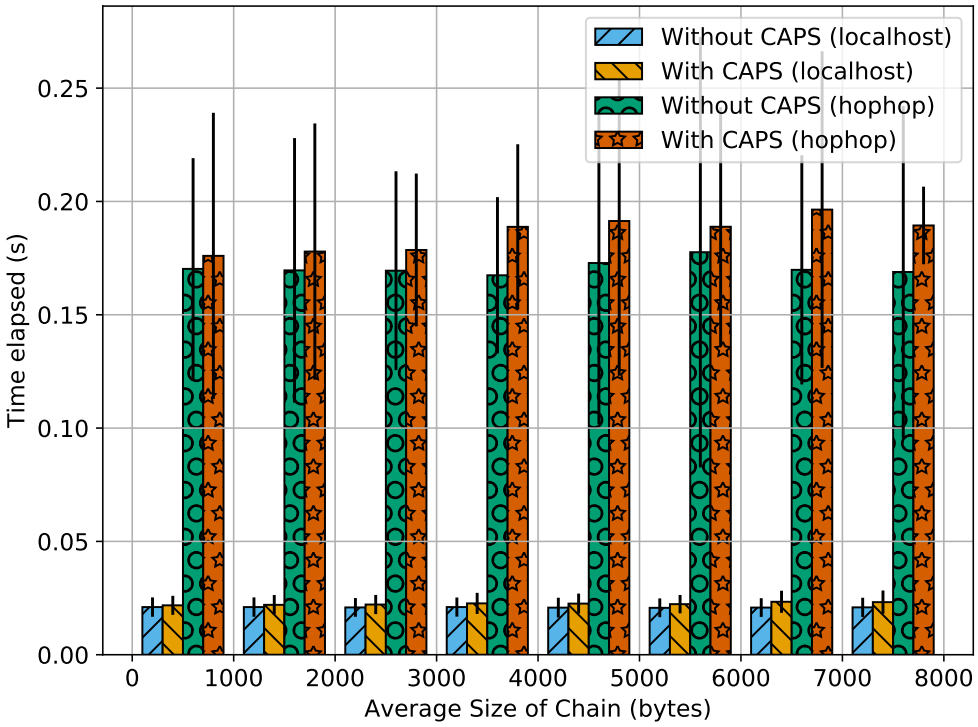


Figure 4.10: Connection latency based on the average size of the certificate chain.

connection establishment.

4.4.2 Transition Compaction

To further minimize the size of the DAFSA encoding, we can extend the design of prior work [49] using *transition compaction*, which aims to minimize the DAFSA representation by reducing the representation size of each transition. A transition in the representation consists of a symbol and destination state. In our testing, we found that both the symbols and destination states had a long-tailed distribution, so a variable-length encoding, particularly a Huffman code, could be used to further reduce the size of each transition. In practice, simply Huffman encoding each symbol may not be sufficient, as we also found that most symbols appear only once in the DAFSA, and thus encoding every symbol would actually increase the size of the representation. We may be able to add a signaling bit to indicate that a symbol is Huffman encoded and only encode symbols that appear more than once.

For the destination state, previous work examined the use of ordering the states of the DAFSA and indicating transitions to the immediate next state in the ordering using a special bit [49]. Finding an optimal ordering for such an approach is a hard problem, but there are several approximations that may work well. For example, we can consider

the difference between the source and destination states with respect to their positions in the ordering and encode the integer value of the difference (i.e., a number of states forward or backward in the representation) using variable-length integer coding. Again, finding the optimal ordering of states for this approach is difficult, with the problem being related to finding a vertex separator in a graph. However, we may be able to apply techniques for efficiently finding a vertex separator or an approximation thereof to our approach.

Chapter 5

Secure Public-Key Authentication in a Federated Web PKI

In this chapter, I describe how we can reduce MITM attacks in a broader range of PKIs. In particular, I explore how we can design a PKI that is resilient to compromised trusted parties in a holistic, federated environment. By *holistic*, I mean that a PKI can authenticate multiple types of information with different requirements; in this chapter, I specifically examine the authentication of names, routes, and end-entity public keys. By *federated*, I mean that a system consists of multiple PKIs that have different trust roots but certify the same type of information; for example, different companies may produce messaging applications that bind public keys to personal identities. This environment prompts several important questions. What challenges with respect to MITM attacks do PKIs face in a holistic and federated environment that they did not in the Web PKI? What properties should a PKI achieve in such an environment to minimize these attacks? How might we begin to transition the Web PKI to a holistic and federated system?

As a first step towards answering these questions, I propose the **Scalable Authentication Infrastructure for Next-generation Trust (SAINT)**, a proposal for a holistic and federated PKI [95]. SAINT considers a federated, holistic environment by imagining the Internet as a set of realms of authentication called isolation domains (ISDs), similar to TDs in SCION (described in Section 2.6.5). In SAINT, I tackle the problem of designing a unifying PKI for naming, routing, and end-entity public-key authentication, handled today by DNSSEC, RPKI/BGPsec, and the Web PKI, respectively.

I observe that PKIs in the current Internet follow one of two models: *monopoly* (the entire system has a single trust root) and *oligarchy* (the system has many trust roots of equal authority). Unfortunately, both of these models suffer from shortcomings. The monopoly model unrealistically expects users to trust a single global root, and while this is currently the case for systems like DNS, it is not clear that this state of affairs will persist indefinitely. The oligarchy model, on the other hand, gives all trust roots equal, global authority, leading to *weakest-link security*, as is the case in

the current Web PKI.

A holistic and federated PKI thus needs to provide several core guarantees. First, an *isolation* property should contain the damage that a compromised trust root can do. Moreover, users should have *trust agility* [91], the ability to easily select their trust roots. To recover quickly from misbehavior, the PKI should also support *update efficiency*. The PKI should allow authentication as in the current PKI with two related properties: *global verifiability* should guarantee that these trust roots can be used to authenticate any other entity, and *trust mobility* should enable trust roots to be used anywhere in the world.

In SAINT, I observe that routing information contains implicit trust information that can be used to bootstrap PKIs for naming and end-entity authentication. Using SCION, which provides a PKI for routing, I can instantiate SAINT in a way that provides isolation in the form of ISDs and quick updates of trust root information in the form of trust root configuration (TRC) files. I then show that the design and instantiation of SAINT provides the properties above.

While SAINT addresses problems with the PKI that arise in a holistic, federated environment, it does not aim to address the problems of deployably reducing or preventing MITM attacks *within* an ISD, since this problem can be addressed by deploying IKP (using the mechanism outlined in CAPS) within the ISD. Instead, SAINT focuses on MITM attacks enabled by exploiting the interconnection of PKIs for different purposes (such as naming and routing) or the use of different trust roots among communicating entities. We can thus use elements of IKP to further improve the resilience of SAINT to misbehavior or misconfiguration within an ISD, but for clarity, I will present SAINT without these features.

In summary, I make the following contributions in SAINT:

- By extending the concept of TDs in SCION, I propose my design of ISDs and show that it can isolate risks of MITM attacks caused by a compromised trust root.
- By utilizing SCION's routing infrastructure to distribute trust root information in the form of TRC files, I show that trust root information can be updated efficiently in a federated PKI environment.
- By building on the inherent business relations that underlie peering links among ISDs, I show that the separation of routing authentication from other types of authentication enable properties such as global verifiability and trust mobility in a holistic PKI environment.
- By implementing and evaluating a prototype of SAINT, I show that SAINT incurs only a moderate latency overhead.

I begin this chapter by describing the problem of designing a holistic PKI in greater detail. I then provide an overview of SAINT. I describe several aspects of SAINT in detail: ISDs, TRC files, cross-signing of trust roots, and the separation of authentication types. I then provide an in-depth example of public-key authentication in SAINT. I describe the implementation and evaluation of SAINT, along with a proposed deployment strategy. I finally conclude with a discussion of several non-technical issues and possible extensions to SAINT.

5.1 Problem Definition

Our goal is to design a global infrastructure allowing a user (Alice) to authenticate routes, names, and end-entity certificates (such as TLS certificates) for a server (Bob) in the Internet. In a global environment such as the Internet, Alice does not trust all trust roots in the environment, and she and Bob may not even have any trust roots in common. The trust roots may differ in functionality and scope: some may authenticate routes and others names, and some may be global and some local. We want to minimize global authority, allow trust agility and mobility while maintaining global verifiability, and allow Alice to evaluate the trustworthiness of information being authenticated.

Desired properties In order to effectively address the above authentication problems, a network architecture should have the following properties:

- **Isolated authentication.** The compromise of a trust root should be limited in scope. In particular, if Alice and Bob share trust roots for some information, no other trust root should be able to affect authentication of that information.
- **Trust agility.** Alice should have a clear, understandable choice over her trust roots. This choice should be easily modifiable at any time, with changes taking effect immediately (within seconds).
- **Trust mobility.** Alice's trust decisions should remain the same no matter where she is in the network. In other words, moving to different locations in the network should not require Alice to change her trust roots.
- **Global verifiability.** As in the current Internet, Alice should be able to authenticate any entity in the Internet that can be reached and has authentication information such as a name or end-entity certificate, even if its trust decision differs from hers.
- **Transparent authentication.** Alice should know when trust roots other than her own are certifying information that she verifies. In particular, for a chain of signatures Alice should be able to determine which trust roots are responsible for each signature.
- **Update efficiency.** Changes to trust root information (e.g., new keys and revocations) should take effect quickly (within minutes). In particular, Alice should be able to detect and obtain new information without requiring software updates.

Other assumptions In order for Alice to successfully verify authentication information, she must also be able to verify a set of trusted public keys which can then be used to bootstrap trust in other keys used during authentication. We therefore assume that users like Alice can verify an initial set of public keys through an out-of-band mechanism.

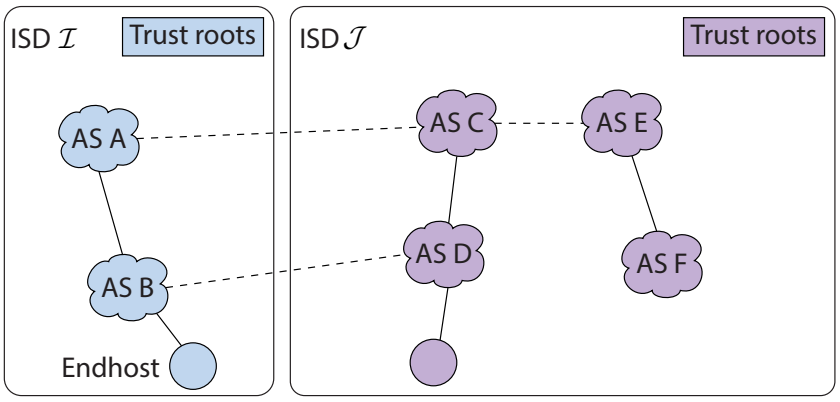


Figure 5.1: Two isolation domains with individual trust roots. The trust roots are illustrated in Figure 5.3. The solid lines indicate provider-to-customer relations; the dashed lines indicate peering links.

5.2 Overview

In this section, we highlight important features of SAINT. We provide intuitive explanations of how these features accomplish the desired properties mentioned in Section 5.1 and how they fit into the overall SAINT architecture.

5.2.1 Isolation Domains (ISDs)

The Internet consists of a diverse assortment of groups, or domains, each with its own set of trusted parties and individual policies regarding routing, naming and end-entity certification. We make these differences a central part of the SAINT architecture in order to achieve isolated authentication. SAINT groups hosts, routers, and networks into *ISDs*, such as those shown in Figure 5.1. SAINT's ISDs are inspired by SCION's trust domains [134] and leverage SCION's routing infrastructure. However, SAINT's ISDs provide additional authentication mechanisms for naming and end-entity certification.

An ISD is a collection of ASes with a common set of trust roots (see Figure 5.3). These trust roots manage authentication within their ISD, including management of routing, naming, and end-entity certification policies, but are not authorities outside of the ISDs. The structure of ISDs attempts to model existing trust relationships between humans by grouping those with similar trust decisions together and by protecting users from misconfigurations or breaches of trust outside these "circles of trust" where other trust decisions and policies hold. Thanks to ISDs, Alice can select her roots of trust and when communicating within her own ISD, she can stay protected from compromised trust roots outside of her ISD. Section 5.3 describes the structure of ISDs in more detail.

In practice, ISDs can represent groups of various scales, such as companies, conglomerates, or countries. ISD-level policies will vary greatly by the scale of the ISD, since corporate policies often contain much more detail than country-

wide laws. In this chapter we use countries as examples of ISDs for several reasons: (1) international boundaries approximately map to DNS naming boundaries, which are also separated in SAINT, (2) national data privacy laws provide a reasonable example of security policies in top-level ISDs, and (3) the resulting set of domains represents an easily-understood choice among possible sets of trust roots, since users can more easily understand what it means to evaluate and trust a country (representing a set of trust roots) rather than doing so with individual trust roots.

5.2.2 Trust Root Configuration (TRC) Files

Trust root management in SAINT is handled by TRC files, which contain information about an ISD's trust roots, such as their public keys (see Section 5.4.2 for more details). TRC files are disseminated as network messages along the same channels as routing messages and DNS responses (see Section 5.4.3), providing update efficiency. Because routing messages are required to maintain connectivity, new TRC files can quickly propagate throughout the network in case a trust root is compromised. This mechanism allows Alice to quickly obtain up-to-date trust root information.

In addition to the above distribution mechanisms, TRC files can also be downloaded and chosen by the users as a new set of trust roots. Since a TRC file contains all of the necessary trust root information, a user like Alice can easily switch to a different set of trust roots by simply obtaining and selecting a different TRC file. In essence, SAINT provides trust agility by allowing users to easily modify their trust decisions at any time.

TRC files contain trust root information for a given ISD and thereby enable transparent authentication. Namely, when a trust root signs the information of another ISD's trust root (as explained below), it does so by signing the TRC file of the other ISD. Thus a chain of signatures clearly indicates domain boundaries by design. Alice can use this knowledge of ISD boundaries to evaluate the trustworthiness of this signature chain and determine whether or not to accept the authenticated information.

5.2.3 Cross-Signing Trust Roots

If we consider ISDs as countries, then we can clearly see that not all ISDs' trust roots will cross-certify one another, and it is unrealistic to expect countries to do so. Rather, we only require the trust roots of two ISDs to cross-certify one another if they share routing links, that is, if they are physically connected and route traffic through one another. This requirement ensures that the existence of a routing path implies the existence of a chain of signatures for a name or end-entity certificate, providing global verifiability by allowing Alice to verify this information for any entity she can reach (see Section 5.5 for more details).

This cross-signing requirement also helps to provide mobility: no matter where users are located, they can authenticate service information (names and end-entity certificates) starting from their own trust roots (named in their "home" TRC file) to the ISD of the entity whose information they are verifying. Thus as long as Alice can reach her

home ISD from the ISD in which she is located, she can use her existing trust decision for authentication anywhere in the world.

5.2.4 Separation of Authentication Types

SAINT separates routing authentication from service authentication (which certifies names and end-entity certificates). Because authentic routes are required to fetch necessary information during name lookups and end-entity certificate handshakes, we treat routing as a separate authentication mechanism. Moreover, we note that authentication of routes cannot rely on fetching external information, as this would itself require authentic routes and thus create a circular dependency.

The separation of routing and service authentication also helps provide trust mobility in SAINT. We observe that users' physical locations indeed influence their routing authentication; in particular, a route from Alice to Bob must be authenticated by trust roots of the ISDs in which Alice and Bob are located. However, this requirement does not hold for service authentication; thus Alice can use the trust roots of an ISD of her choosing to completely bypass the ISD in which she is located to authenticate names or end-entity certificates, providing trust mobility and greater resilience against MITM attacks.

5.3 Isolation Domains

We now discuss isolation domains in more detail. We begin by describing the physical layout of ISDs along with the structure of the namespace and the address space in SAINT. We then present the concept of trust anchor ISDs, which provide starting points for the authentication of routes, names, and end-entity certificates.

5.3.1 ISD Structure

An ISD is made up of multiple networks, or ASes, as shown in Figure 5.1, with ISDs connecting to one another to enable Internet-wide connectivity. Similarly to trust domains in SCION, SAINT arranges ASes hierarchically within an ISD by customer-provider relationships. The ASes in the top tier (those with no providers) are referred to as the *ISD core* and serve as the trust roots for routing (see Figure 5.3 and Section 5.4.1 for more details).

The connectivity of ASes in SAINT is similar to the current Border Gateway Protocol (BGP)-based relations: ISDs are primarily connected by links between ISD core ASes (similar to BGP transit links between tier-1 Internet service providers (ISPs)), but can also be connected by links between lower-tier ASes (similar to BGP provider-customer links and peering links). Paths between ISD cores are determined by a simple routing protocol among the core ASes of each ISD. Based on our analysis of the current Internet (Section 5.10), we expect that there will be on the order of

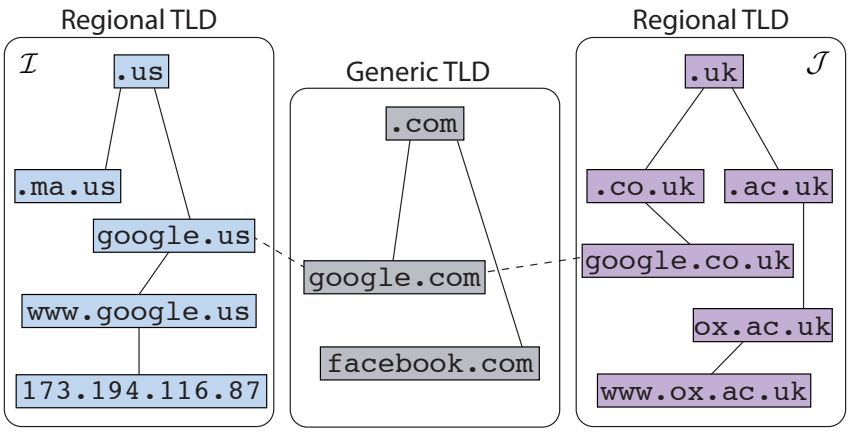


Figure 5.2: Namespace structure in SAINT. Solid lines indicate hierarchical relations, and dashed lines indicate redirections.

several hundred core ASes in total. Therefore, in running such a protocol we do not worry about significant overhead or scalability.

5.3.2 DNS Namespace

Each ISD has the autonomy to manage its own namespace. We structure SAINT's global namespace as a collection of top-level domains (TLDs) bound by an inter-domain web of trust [135] (as shown in Figure 5.2), rather than by a global root zone as is done in DNSSEC. However, SAINT's name resolution process is similar to that of DNSSEC.

Each SAINT DNS root server answers queries for hosts within its ISD. An ISD's namespace supports one of two top-level domain types:

- **Regional TLDs** in SAINT correspond to a specific ISD. In the example of Figure 5.2, the TLD `.us` represents the United States ISD, whereas `.uk` represents the United Kingdom ISD. In order to provide transparency, the DNS server responsible for a regional TLD guarantees that any address record (similar to A records in DNS) maps to an address within the corresponding ISD.
- **Generic TLDs** such as `.com` and `.org`, by contrast, are *not* bound to any specific ISD, and can thus name an entity located *anywhere* in the world. However, a name in a generic TLD can only map a redirection to another name, thereby ensuring that only names under regional TLDs map to addresses (and only within the TLD's corresponding ISD). This guarantee provides domain transparency during DNS lookups.

We expect that today's ccTLDs such as `.us` and `.uk` will continue to operate as regional TLDs under SAINT. Countries such as Tuvalu (whose ccTLD is the popular `.tv`) may choose to operate as a generic TLD and continue to sell names that map all over the world, but must do so through redirections to other names.

5.3.3 Address Space

We define an address in SAINT as a 3-tuple of the form $(\mathscr{I}, A, e)$, where $\mathscr{I}$ represents an ISD identifier, A represents an *AS identifier (ASID)*, and e represents an *endpoint identifier* endpoint identifier (EID). For example, if Alice wants to reach Bob, who has the EID 42ac6d in AS 567 and ISD $\mathscr{I}$, she would contact the address $(\mathscr{I}, 567, 42ac6d)$. In contrast to the current Internet, AS numbers and EIDs do not have significance outside of their respective ISDs and ASes, and thus can have any format. An EID, for example, can be an IPv4, IPv6, MAC, or self-certifying address.

Registry servers in ISDs (described in Section 5.4.1) assign ASIDs to ASes, and similar servers in each AS issue EIDs to endhosts. Due to the local significance of ASIDs and EIDs, $(\mathscr{I}, A, e)$ and $(\mathscr{J}, A, e)$ are distinct addresses even though both have the same ASID and EID. This addressing scheme allows full address to be globally unique while giving ISDs and ASes the autonomy to manage addressing as well as names within their own realm of control.

The above addressing system also allows for interoperability with the current IP addressing and AS numbering schemes while simultaneously enabling local deployments of other proposed addressing schemes. For example, some ISDs may choose to retain the current AS numbering and IP addressing schemes, while others may opt to provide ASes with human-readable identifiers and endpoints with IPv6 addresses.

Similarly to ASIDs, endpoint addresses (since only used locally in intra-AS routing) do not need to be globally allocated like the current IP address space. Since the routing authentication infrastructure only handles inter-AS routes, SAINT does not bind the endhost address space to public keys as RPKI does. Instead, forwarding from an edge router to an endpoint address is resolved and handled entirely within an AS.

5.3.4 Trust Anchor ISDs

Trust anchors in the current Internet, such as IANA for RPKI and BGPsec, ICANN for DNSSEC, and root CAs in TLS, represent starting points for authenticating information. Similarly, trust anchor ISDs are starting points for authenticating routes, names, and end-entity certification in SAINT. Due to the separation of authentication by type, Alice can anchor her trust for authenticating routing and service information in separate ISDs.

As discussed in Section 5.2.4, for routing purposes the trust roots of the ISD's in which Alice is currently located must certify all of her routes. However, Alice can select the trust roots of any ISD to authenticate service information (names and end-entity certificates). We call this ISD Alice's *trust anchor ISD*. Alice choice of this ISD can be easily changed at any time (as described in Section 5.4.3). An example of such trust bootstrapping is provided in Section 5.7.

5.4 Trust Roots

In this section, we cover what entities serve as trust roots and how trust roots are configured for an ISD. We also discuss how we update trust root information using network-level messages, and how this incorporation of trust management

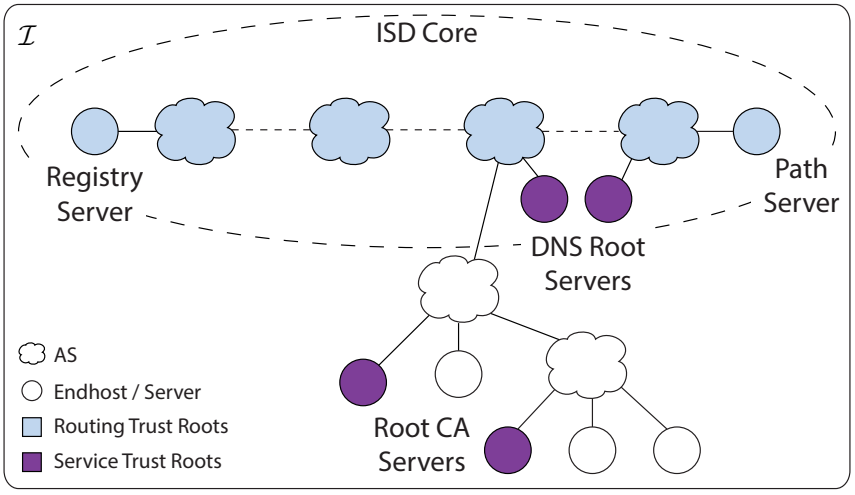


Figure 5.3: Logical and physical placements of trust roots in an ISD.

into the network allows for fast updates of trust root information. Finally, we describe our scheme of separating trust roots by ISD, and how separated categories of authentication enable trust agility.

5.4.1 Trusted Parties

Trust roots sign authentication information for routes, names, or end-entity certificates, and set policies governing the ISD. These policies may include information such as preferences for certain encryption or signature algorithms or constraints on certificate validity. A trust root is an authority for either routing or service authentication (see Figure 5.3).

Because a trust root is only an authority in its own ISD, a compromised trust root cannot enable the impersonation of servers in other ISDs. Scoping trust root authority to the ISD level protects Alice and Bob's communication from many compromises in other ISDs and provides guarantees to Alice about authentication in her trust root ISD.

We classify trust roots into routing and service trust roots. The **routing trust roots** consist of the following parties:

- The *core ISPs* are responsible for sending out route announcements, which are propagated from providers to customers and establish cryptographically signed paths from the recipient to the core.
- The *path server* stores and provides a lookup service for mappings between an ASID and the AS's down-paths. These paths are registered by the ASes at the core. The path server is co-located with and operated by the core ISPs.
- The *registry server* issues and stores *AS certificates* binding an ASID to its public key (called its *AS key*), which are used to verify the signed paths provided by the path server. Like the path server, the registry server is co-located with and operated by the core ISPs.

The **service trust roots** consist of the trust roots for naming and for end-entity certification. The *DNS root* is

the starting point for verifying all names in the ISD's namespace. The DNS root also sets ISD-wide naming policies such as reserved or forbidden domain names and allowed signature algorithms to sign records. Because the failure of the DNS root can block user connectivity in an ISD, the DNS root should be highly robust and available, using mechanisms such as distributed anycast schemes and placing servers in the ISD core where they can be reached through highly available top-tier ASes.

The *root CAs* are the starting points for verifying end-entity certification information in an ISD. Root CAs in SAINT serve the same purpose as they do in today's PKIs by signing TLS public-key certificates. However, they are restricted to only signing end-entity certificates in ISDs in which they are root CAs. They can also sign intermediate CA certificates as in today's TLS PKI. If the ISD uses other public-key infrastructures such as CT or AKI (see Section 5.11), then the trust roots for end-entity certification also include public logs and auditors/validators.

5.4.2 Trust Root Configuration (TRC) Files

As mentioned in Section 5.2.2, a TRC file specifies the trust roots for an ISD, the public keys of those trust roots, and the authentication policies of the ISD. It also specifies the locations of the DNS root and TRC servers (described in Section 5.4.3) to allow users to reach these servers without performing DNS lookups. TRCs are created and managed by an ISD's trust roots and distributed through the routing mechanism. Specifically, a threshold of trust roots is required to sign a new or updated TRC file, and the core ISPs distribute the TRC file within the ISD through a broadcast mechanism that we describe below.

The quorum of trust roots required to update the TRC file is specified in the TRC file itself, providing the trust roots with the autonomy to set their own threshold for altering the TRC file. A higher threshold is more secure to a compromise of multiple trust roots, but also reduces the efficiency in updating TRC files. The TRC file also specifies a quorum of trust roots that must sign a cross-signing certificate to authenticate another ISD's trust roots. Cross-signing certificates are described in more detail in Section 5.5.

TRC format A TRC file is encoded as an XML file with the fields shown in Table **??**. The version number and timestamp ensure that users can verify information using recent policies and trust root information. The public keys of the ISD's trust roots provide starting points for verifying routes, names, and end-entity certificates. The TRC file may also contain the public keys of additional entities (e.g., public logs in CT [85] or In order to allow users to easily reach the DNS root of an ISD, the TRC also contains one or more addresses for the ISD's DNS root.

Policies A TRC file can also specify additional policies related to ISD management. For example, these policies might specify a minimum key length or required encryption algorithms for all end-entity certificates in the ISD.

Table 5.1: Fields in a TRC file.

Field	Description
`isd`	ISD identifier
`version`	Version of TRC file
`time`	Timestamp
`coreISPs`	List of core ISPs and their public keys
`registryKey`	Root registry server's public key
`pathKey`	Path server's public key
`rootCAs`	List of root CAs and their public keys
`rootDNSkey`	DNS root's public key
`rootDNSaddr`	DNS root's address
`trcServer`	TRC server's address
`quorum`	Number of trust roots that must sign new TRC
`trcQuorum`	Number of trust roots that must sign an ISD cross-signing cert
`policies`	Additional management policies for the ISD
`signatures`	Signatures by a quorum of trust roots

Systems such as PoliCert [126] have proposed similar policies on a per-domain basis; we leave a detailed design of additional ISD-wide policies to future work.

Updating the TRC file In the event that a TRC file needs to be updated, the trust roots confer out of band to determine what changes need to be made to the TRC file. Once they have decided to update a TRC file, the trust roots sign the new TRC file. Each of these signatures is appended to the `signatures` section of the new TRC file, and sent when a quorum of trust roots signs the TRC file. The trust roots can also use group signatures [41] or threshold signatures [120] to update the TRC file.

5.4.3 TRC Distribution and Management

Obtaining an initial TRC file. We envision that Alice will most commonly obtain a initial TRC file of her provider's ISD when forming a service agreement. If Alice wants to obtain a different ISD's TRC file, she can contact the *TRC server* of that ISD, a server that stores the TRC files and cross-signing certificates (see Section 5.5) of other ISDs. The TRC server's address is in the TRC file of the ISD, allowing Alice to directly query the server for other TRC files. In an extreme case where Alice does not trust the provider or ISD, she may download a TRC file from a publicly-accessible mirror site or obtain one in person from a trusted colleague or organization. Alice can also obtain a TRC file *a priori* if she plans to join such an ISD with a new device.

Obtaining updated TRC files ASes and users in an ISD are informed of the latest version of the TRC file with each routing announcement and DNS response. Thus, as long as Alice has an Internet connection and performs DNS lookups, she can quickly detect and obtain a new TRC. The version number is part of each routing announcement,

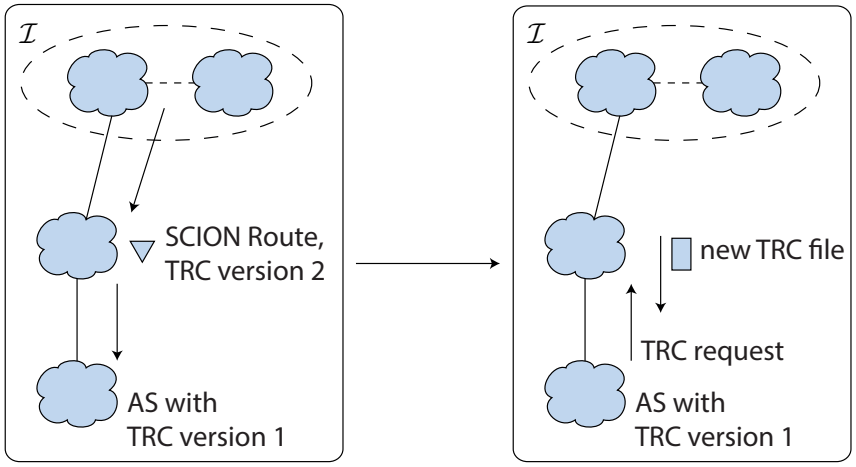


Figure 5.4: Distribution mechanism for updated TRCs. Arrows indicate sent network messages.

and a timestamped message signed by the trust roots accompanies each DNS answer (to avoid re-signing every DNS record in the ISD upon updating the TRC file). When Alice detects a new TRC file, she can fetch the new file from the provider or DNS root (see Figure 5.4).

Changing trust anchor ISDs Besides the ability to select trust roots as described in Section 5.3.4, trust agility also provides the ability to easily and quickly modify this selection. The above methods of obtaining TRC files provide this notion of trust agility, as Alice can change trust anchor ISDs by simply obtaining a new TRC file. Under normal circumstances Alice can simply download a new TRC file from her trust anchor ISD's TRC server, but if for example she discovers that her trust anchor ISD has been conducting state-level surveillance, she can instead obtain the TRC file manually or from an external server as described above.

5.5 Cross-Signing

In this section, we provide a detailed discussion of cross-signing in SAINT. We begin by describing cross-signing certificates, and then detail how these are used to enable inter-ISD authentication and global verifiability. We then discuss the tradeoff between global verifiability and the trustworthiness of authenticated information, and how the authentication policies expressed in an ISD's TRC file fits in this tradeoff.

5.5.1 Cross-Signing Certificates

In order to enable global verifiability, we require ISDs to issue *cross-signing certificates* for its neighboring ISDs, that is, ISDs with which they share routing links. The resulting web of cross-signing between ISDs ensures that by following a route from Alice's ISD to Bob's ISD, a corresponding chain of signatures from Alice's trust roots to

Bob's trust roots will exist, forming a chain of signatures from Alice's trust roots to Bob. ISDs without direct routing connections can also issue cross-signing certificates to one another, forming further chains of signatures to enable authentication between different ISDs directly.

A cross-signing certificate issued by $\mathscr{I}$ for $\mathscr{J}$'s trust roots contains (a) a timestamp, (b) $\mathscr{I}$, (c) the current version number of $T_{\mathscr{I}}$, (d) $\mathscr{J}$, (e) the current version number of $T_{\mathscr{J}}$, (f) a hash of $T_{\mathscr{J}}$, and (g) a signature by a quorum of $\mathscr{I}$'s trust roots (see Table 5.2 for an explanation of notation). The version numbers of the TRC files ensure that the trust roots' public keys can be checked against the appropriate versions of the TRC files.

The ISD stores these certificates in its TRC server for its users and also propagates the certificates along its inter-ISD routing links to provide each ISD with the necessary information to form a chain of signatures to a given destination ISD. Alice can then query her trust anchor ISD to obtain these chains of signatures and select one to authenticate information in Bob's ISD.

5.5.2 Inter-ISD Authentication

When Alice, whose trust anchor ISD is $\mathscr{K}$, wants to authenticates Bob, who is in another ISD $\mathscr{M}$, she needs to obtain cross-signing certificates to form a chain of signatures from $\mathscr{K}$ to $\mathscr{M}$. While verifying Bob's routes, name, and end-entity certificate, she obtains the appropriate cross-signing certificates from $\mathscr{K}$'s TRC server. If Alice is in an ISD $\mathscr{I}$, then every route from her to Bob will have a chain of signatures starting at $\mathscr{K}$, proceeding to the trust roots of $\mathscr{I}$, then to the trust roots of $\mathscr{M}$, and finally to Bob's AS.

If $\mathscr{K}$ and $\mathscr{M}$ do not share routing links but have issued cross-signing certificates to each other, Alice can verify Bob's name and end-entity certificate using $\mathscr{K}$'s cross-signing certificate for $\mathscr{M}$. These cross-signing "shortcuts" allow Alice to authenticate Bob's information with fewer ISDs authenticating information "in transit," providing fewer opportunities for a compromised trust root to disrupt authentication.

No matter where $\mathscr{M}$ is, Alice is guaranteed to find a chain of signatures to $\mathscr{M}$ and to Bob if she can find a route to Bob. Since she must be able to contact $\mathscr{K}$ from $\mathscr{I}$ to obtain the appropriate cross-signing certificates, she has a route between $\mathscr{K}$ and $\mathscr{I}$ and can thus obtain cross-signing certificates from $\mathscr{K}$ to $\mathscr{I}$, and similarly for $\mathscr{I}$ and $\mathscr{M}$. Though a chain of signatures may cross many ISDs, Alice is guaranteed to find at least one such chain.

Note that cross-signing certificates do not necessarily indicate a trust relationship between ISDs; a cross-signing certificate instead only states: "These are the public keys of the trust roots for the following ISD." It is therefore up to Alice to determine the trustworthiness of a chain of signatures before accepting the information it certifies as authentic.

5.5.3 Authentication Policies

The above cross-signing requirement ensures that Alice can authenticate Bob's information regardless of which ISD he is in. While a compromised trust root on a chain of signatures from Alice to Bob can adversely affect authentication by certifying false information, Alice's trust anchor ISD $\mathscr{K}$ can mitigate this risk through the use of ISD-wide policies in the TRC file. These policies can also blacklist public keys, such as those contained in known unauthorized certificates or those of compromised trusted authorities. Using such policies, $\mathscr{K}$ can protect Alice from compromises in other ISDs. If others with $\mathscr{K}$ as their trust anchor ISD frequently contact Bob or other destinations in $\mathscr{M}$, then $\mathscr{K}$ may form a cross-signing relationship with $\mathscr{M}$ to minimize the risk of compromised trust roots in other ISDs.

ISDs face a tradeoff between enabling global verifiability and protecting their users from compromises in other domains. The default behavior in SAINT is to provide global verifiability. As illustrated above, an ISD must explicitly state any exceptions to this behavior in the policy field of its TRC file. The ability to restrict the authentication of known false information through policies provides a mechanism by which an ISD can protect not only its own users, but also users for whom a chain of signatures passes through the ISD.

5.6 Separated Authentication

In this section, we describe how SAINT separates routing and service authentication. We first describe our motivation for separating these two types of authentication, and then discuss how this separation provides Alice with trust mobility.

5.6.1 Routing and Service Authentication

Authentication in SAINT is classified and separated into routing and service authentication. We make this separation in part because we observe that the authentication of route information fundamentally differs from the authentication of service information. In particular, routing authentication cannot assume the existence of secure routes to obtain any external information, and therefore an entity must rely on pre-verified paths or be able to verify paths without fetching external information. By contrast, service authentication assumes the existence of authentic routes and thus allows contacting external entities to obtain authentication information.

Routing messages in SAINT propagate beginning from the ISD core and follow provider-customer AS links. Unlike in RPKI and BGPsec, all necessary information (e.g. AS certificates) are sent with the routing message, allowing an AS to verify routing messages upon arrival. Moreover, information such as AS certificates are short-lived, eliminating the need to propagate revocation information for AS keys.

By contrast, a DNS lookup, which falls under service authentication, must use a route to reach one or more nameservers and fetch the appropriate information for verifying a name-to-address mapping. TLS features such as OCSP also require contacting an external entity to determine the validity of an end-entity certificate. Due to this

dependence, Alice must verify routes to the ISD core of her current ISD $\mathscr{I}$, and form and verify routing paths from $\mathscr{I}$ to Bob's ISD $\mathscr{M}$ before she can authenticate Bob's service information.

5.6.2 Trust Mobility

Separating routing and service authentication also enables trust mobility. Suppose that Alice checks into a hotel in Oceania, a known surveillance state, and attempts to connect to her hotel's wireless Internet. If the Oceanian trust roots are compromised by the government, then it is inevitable that the government can see her packets themselves, as her physical location in Oceania enables the government to examine her packets. In other words, Oceanian trust roots must certify her routes out of the Oceanian ISD and thus these trust roots must be on the chain of signatures for routes from Alice to any destination in the Internet.

With SAINT, however, Alice can choose $\mathscr{K}$ as her trust anchor ISD for service authentication, since SAINT separates routing and service authentication. Moreover, this choice does not depend on Alice's current location and thus applies wherever Alice is in the Internet. In our example, this means that Alice does not have to rely on signatures from the Oceanian trust roots to verify Bob's name or end-entity certificate, even if she is connecting to the Internet from an Oceanian hotel.

5.7 Authentication Example

We now discuss the complete authentication process in SAINT. We first describe setup steps for a server, such as joining an ISD and registering domain names, routing paths, and end-entity certificates. We then describe how Alice (the client) checks the information that she receives about Bob (the server). We use a to denote Alice and b to denote Bob. As previously mentioned, Alice's trust anchor ISD is $\mathscr{K}$. Bob is part of the AS B in Mythuania $\mathscr{M}$, whose ccTLD is `.my`. Table 5.2 provides a list of the notation used.

5.7.1 AS Setup

Figure 5.5 depicts the steps of the AS setup process for AS B in the Mythuanian ISD $\mathscr{M}$:

1. $RS_{\mathscr{M}}$ assigns the ASID B to Bob's AS.
2. B creates an AS key pair (AK_B, AK_B^{-1}).
3. B sends $\{B, AK_B\}$ to $RS_{\mathscr{M}}$.
4. $RS_{\mathscr{M}}$ issues B an AS certificate AC_B.
5. B receives the TRC file $T_{\mathscr{M}}$ from its parent AS.
6. B receives SCION routing messages from its parent.
7. B selects a set of paths P_B (signed with AK_B^{-1}) and sends $\{P_B, AC_B\}$ to $PS_{\mathscr{M}}$.

Table 5.2: Notation.

Notation	Name	Use
Identifiers		
X	AS	AS with ASID X
y	Endhost	an end-entity such as a client or server
e_y	EID	locate endhost y within its AS and ISD
$\mathscr{Z}$	ISD	ISD with identifier $\mathscr{Z}$
Certificates		
AC_X	AS cert	bind X to AK_X (signed by RS of X's ISD)
EC_y	End-entity cert	store CA-signed public key information during connection setup
DC_y	CERT RR	store CA-signed DNS binding between y and DK_y
Keys		
AK_X	AS key	sign paths that can be used to reach X
DK_y	DNSKEY RR	sign DNS resource records in DNSSEC
EK_y	End-entity key	set up secure end-to-end connections, e.g., via TLS
K_y^{-1}	Private key	private key for public key K_y
Servers		
PS_Y	AS Path server	contact ISD path server for clients in Y
$PS_{\mathscr{Z}}$	ISD Path server	maintain database of signed paths for ASes in $\mathscr{Z}$
$RS_{\mathscr{Z}}$	Registry server	assign ASIDs and AS numbers in $\mathscr{Z}$
Messages		
P_X	Signed path set	sent to PS of X's ISD to register paths to reach X
$T_{\mathscr{Z}}$	TRC file	provide trust root information for $\mathscr{Z}$

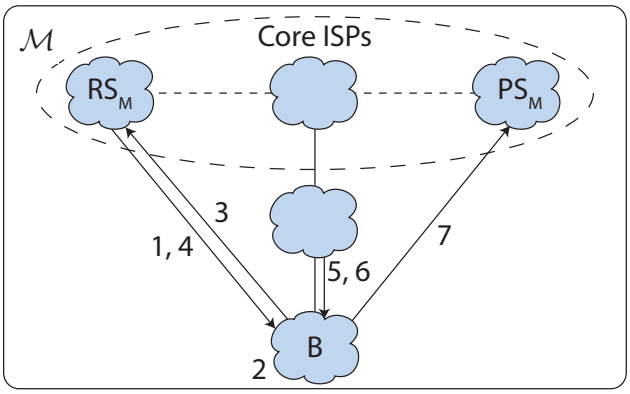


Figure 5.5: Diagram for AS setup steps (Section 5.7.1).

5.7.2 Server Setup

Figure 5.6 shows the steps of the server setup process for Bob:

1. AS B assigns Bob the EID e_b, making his address $(\mathscr{M}, B, e_b)$.
2. Bob chooses the name `b.my` and creates a domain-name key pair (DK_b, DK_b^{-1}).

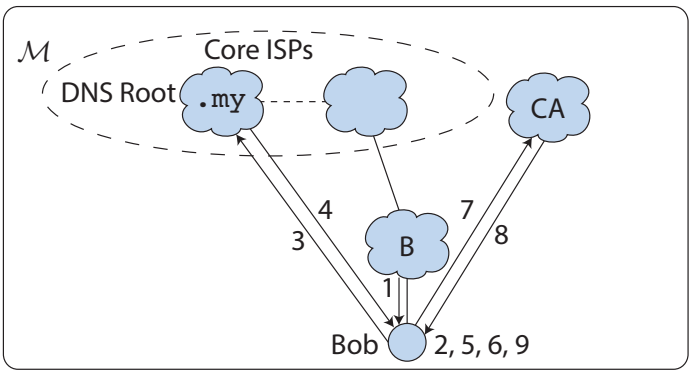


Figure 5.6: Diagram for server setup steps (Section 5.7.2).

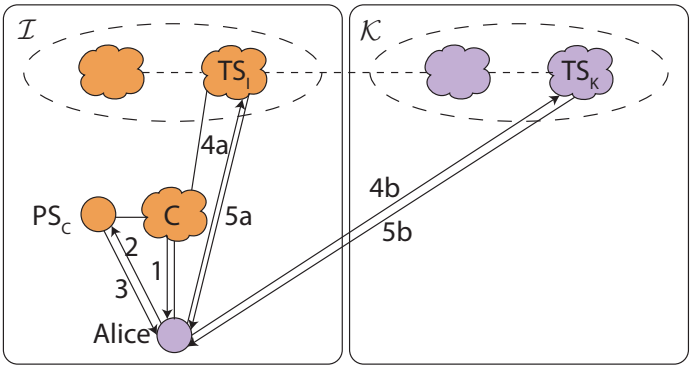


Figure 5.7: Diagram for obtaining a TRC file (Section 5.7.3).

3. Bob sends `b.my` and DK_b to the `.my` operator to register his name and key.[1]
4. The `.my` operator creates a delegation signer (DS) record to point to DK_b from the `.my` zone, as well as a record mapping `b.my` to Bob's nameserver.
5. Bob creates a mapping of `www.b.my` to $(\mathscr{M}, B, e_b)$, DK_b, and resource record signature (RRSIG) record of the mapping signed with DK_b^{-1}.
6. Bob creates an end-entity key pair (EK_b, EK_b^{-1}).
7. Bob sends $\{b, EK_b\}$ to a CA in $\mathscr{M}$.
8. The CA issues Bob an end-entity certificate $EC_b = \{b, EK_b\}_{K_{CA}^{-1}}$.
9. Bob creates a certificate $DC_b = \{b.my, DK_b\}_{EK_b^{-1}}$, and stores DC_b along with EC_b as a CERT record in his nameserver.

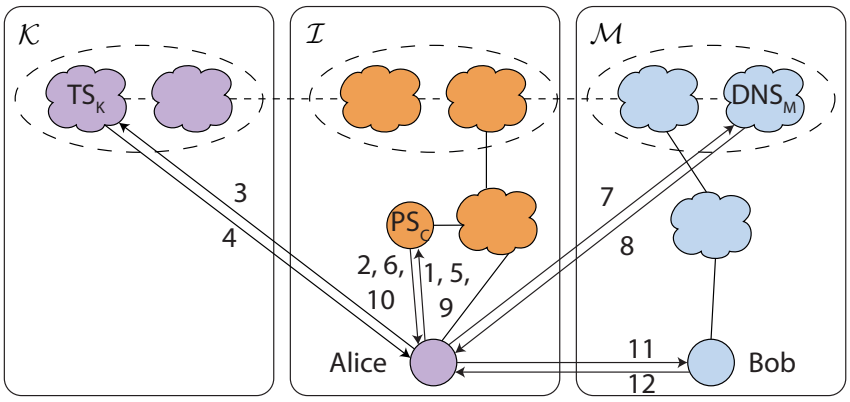


Figure 5.8: Client lookup and verification of server's name, route, and end-entity certificate (Section 5.7.4).

5.7.3 Client Setup

In order for Alice to verify information, she must first possess a TRC file to configure her set of trust roots. Even if she does not have a TRC file, we assume that she can verify a TRC file from her trust anchor ISD $\mathscr{K}$. In fact, she can verify this TRC file in any ISD $\mathscr{I}$ — even if she does not trust $\mathscr{I}$. As illustrated in Figure 5.7, after connecting to the Internet in $\mathscr{I}$, Alice does the following to obtain and verify a TRC file:

1. Alice's ISP (AS C) assigns her the EID e_a, making her address $(\mathscr{I}, C, e_a)$. C also sends her the latest TRC file $T_{\mathscr{I}}$ for the ISD $\mathscr{I}$.
2. Alice requests from PS_C a path to the TRC server $TS_{\mathscr{I}}$ of $\mathscr{I}$ or $TS_{\mathscr{K}}$ of $\mathscr{K}$ (if she knows the address).
3. PS_C returns to Alice the path she requested.
4. Alice now contacts either $TS_{\mathscr{I}}$ (4a in Figure 5.7) or $TS_{\mathscr{K}}$ (4b) and requests $T_{\mathscr{K}}$. In the case of 4b, Alice also requests a cross-signing certificate for $\mathscr{I}$ from $TS_{\mathscr{K}}$ to ensure that all authentication (even for routes) begins from $T_{\mathscr{K}}$.
5. If Alice contacted $TS_{\mathscr{I}}$, she receives $T_{\mathscr{K}}$ (5a). Otherwise, she receives $T_{\mathscr{K}}$ and the cross-signing certificate for $\mathscr{I}$ from $TS_{\mathscr{K}}$ (5b).

We assume that Alice verifies the authenticity of $T_{\mathscr{K}}$ through an out-of-band mechanism, e.g., if she makes plans to travel to $\mathscr{I}$ and considers it a "hostile" ISD, then Alice can obtain a hash of the public keys of $\mathscr{K}$'s trust roots *a priori*, or she can obtain this information in an embassy of $\mathscr{K}$ within $\mathscr{I}$.

5.7.4 Client Verification

Figure 5.8 illustrates the complete process that Alice executes to authenticate Bob. We assume that Alice has completed the client setup process and thus has $T_{\mathscr{K}}$ to use as the starting point for authenticating Bob. The authentication process is as follows:

1. Alice begins by looking up `www.b.my`, and thus first obtains $T_{\mathscr{M}}$. She contacts PS_C to obtain a path to $TS_{\mathscr{K}}$.
2. PS_C returns to Alice a set of paths that she can use to reach $TS_{\mathscr{K}}$.
3. Alice contacts $TS_{\mathscr{K}}$ to request $T_{\mathscr{M}}$.
4. $TS_{\mathscr{K}}$ returns $T_{\mathscr{M}}$ and a cross-signing certificate for $\mathscr{M}$.
5. Alice contacts PS_C to request a path to the DNS root of $\mathscr{M}$, whose address she has from $T_{\mathscr{M}}$.
6. PS_C returns to Alice a set of paths to $\mathscr{M}$'s DNS root.
7. Alice contacts $\mathscr{M}$'s DNS root to query `www.b.my`.
8. Alice performs DNSSEC resolution to obtain $(\mathscr{M}, B, e_b)$ as well as Bob's domain and end-entity certs DC_b and EC_b.
9. Alice requests a path to Bob's address from PS_C.
10. PS_C returns to Alice B's AS certificate AC_B and a set of paths P_B to reach B.
11. Alice contacts Bob to initiate the TLS handshake.
12. Bob sends Alice his end-entity certificate EC_b.

Alice verifies that Bob's end-entity public key EK_b contained in the end-entity certificate she obtained from $\mathscr{M}$'s DNS root matches the end-entity public key she receives during the TLS handshake. If the keys match, she proceeds with the TLS handshake to establish a secure end-to-end connection with Bob.

Throughout this process, Alice verifies that valid authentication paths exist for each entity she contacts: PS_C, $TS_{\mathscr{K}}$, $\mathscr{M}$'s DNS root, B, and Bob. When she receives information signed by the trust roots of an ISD other than $\mathscr{K}$, Alice uses the appropriate cross-signing certificate to verify the public keys of the ISD's trust roots, thus ensuring that all authentication ultimately begins with trust roots listed in the TRC of her trust anchor ISD $\mathscr{K}$.

Error handling A verification failure at any stage in the authentication process will prevent Alice from authenticating and establishing a connection to Bob. In the event that the verification of a routing path fails, Alice will not be able to reach Bob or entities such as DNS roots and TRC servers. However, Alice likely cannot detect this failure from her browser. In the event that the verification of Bob's name-to-address mapping fails, Alice will not know the address at which she can reach Bob. While most modern browsers indicate such a failure, Alice cannot proceed with verification after such a failure. From the perspective of Alice's browser, a failure to verify Bob's end-entity certificate is the most

[1] In practice, Bob will create multiple key pairs and use one of the private keys to sign the others, but for simplicity we assume here that Bob uses DK_b both to sign his DNS zone information and to self-sign DK_b.

informative, as most modern browsers display the type of error that occurred and in some cases provide the option to continue with the connection anyway.

5.8 Analysis

In this section, we present a formal model of SAINT and analyze how our model achieves the desired properties presented in Section 5.1.

5.8.1 Model

In our model, we associate an ISD $\mathscr{I}$ with a set of trust roots $\mathsf{TR}(\mathscr{I})$, a set of neighbors $\mathsf{N}(\mathscr{I})$, and a set of *cross-signing neighbors* $\mathsf{CN}(\mathscr{I})$. The set of neighbors of $\mathscr{I}$ is the set of all ISDs to which $\mathscr{I}$ has a direct routing connection exists, and the set of cross-signing neighbors of $\mathscr{I}$ is the set of all ISDs $\mathscr{J}$ such that a cross-signing certificate $\mathsf{CSC}(\mathscr{I},\mathscr{J})$ exists from $\mathscr{I}$ to $\mathscr{J}$.

Assumptions We assume that routing connections are symmetric, so that $\mathscr{I} \in \mathsf{N}(\mathscr{J})$ implies that $\mathscr{J} \in \mathsf{N}(\mathscr{I})$. We also assume from Section 5.5 that cross-signing certificates exist wherever routing connections exists, and hence $\mathsf{N}(\mathscr{I}) \subseteq \mathsf{CN}(\mathscr{I})$ for all $\mathscr{I}$. We further assume that cross-signing certificates are symmetric, that is, $\mathscr{I} \in \mathsf{CN}(\mathscr{J})$ implies that $\mathscr{J} \in \mathsf{CN}(\mathscr{I})$. Because the existence of a cross-signing certificate $\mathsf{CSC}(\mathscr{I},\mathscr{J})$ is equivalent to $\mathscr{J} \in \mathsf{CN}(\mathscr{I})$ by definition, we can also say that the existence of $\mathsf{CSC}(\mathscr{I},\mathscr{J})$ implies the existence of $\mathsf{CSC}(\mathscr{J},\mathscr{I})$.

For the purposes of obtaining routes, names, and certificates, we use two types of sequences of ISDs. We first define a *path* as follows:

Definition 1. A *path* from $\mathscr{X}_1$ to $\mathscr{X}_n$ is a sequence of ISDs denoted $(\mathscr{X}_1,\ldots,\mathscr{X}_n)$ where for all $1 \leq i < n$, there exists a direct routing link between $\mathscr{X}_i$ and $\mathscr{X}_{i+1}$. We call $\mathscr{X}_1$ and $\mathscr{X}_n$ the *source* and *destination* of the path, respectively.

We can also define a similar sequence of ISDs called a *certificate chain* as follows:

Definition 2. A *certificate chain* from $\mathscr{X}_1$ to $\mathscr{X}_n$ is a sequence of ISDs denoted $\langle \mathscr{X}_1,\ldots,\mathscr{X}_n \rangle$ where for all i such that $1 \leq i < n$, $\mathsf{CSC}(\mathscr{X}_i,\mathscr{X}_{i+1})$ exists and is valid in the sense that the signature $\mathsf{CSC}(\mathscr{X}_i,\mathscr{X}_{i+1})$ can be verified using the public keys of $\mathsf{TR}(\mathscr{X}_i)$.

As with paths, certificate chains also have the concept of a source and destination identical to the one presented in Definition 1.

Clients have two notable relations to ISDs. The first is the concept of being physically located in an ISD:

Definition 3. We say that a client A is *located in* an ISD $\mathscr{I}$ if for any destination ISD $\mathscr{D}$, all routing paths along which A can send packets to $\mathscr{D}$ have the source $\mathscr{I}$. We call $\mathscr{I}$ the *routing source ISD* of A.

In order to formalize the second relation, we first define the concept of a client evaluating a certificate chain:

Definition 4. Let $\mathscr{C}$ be a certificate chain. We define the *evaluation function* of A as a function $\mathsf{eval}_A : \{\mathscr{C}\} \to \{0,1\}$ where in any protocol step where A receives a certificate chain $\mathscr{C}$, $\mathsf{eval}_A(\mathscr{C}) = 0$ implies that A immediately aborts the protocol. When calculating $\mathsf{eval}_A(\mathscr{C})$, we say that A *evaluates* $\mathscr{C}$. If $\mathsf{eval}_A(\mathscr{C}) = 1$ then we say that A *accepts* $\mathscr{C}$, and if $\mathsf{eval}_A(\mathscr{C}) = 0$ then we say that A *rejects* $\mathscr{C}$.

Suppose that $\mathsf{eval}_A(\mathscr{C}) = 1$ if and only if $\mathscr{C}$ has the ISD $\mathscr{T}$ as its source. Then we call eval_A *sound*. Furthermore, we say that A *anchors its trust* in $\mathscr{T}$, and we call $\mathscr{T}$ the *trust anchor ISD* of A.

Intuitively, to accept a certificate chain $\mathscr{C}$ with destination $\mathscr{D}$ means behaving as if any route, name, or EE certificate signed by $\mathsf{TR}(\mathscr{D})$ is correct. Thus a client A who accepts a certificate chain with destination $\mathscr{D}$ will, for example, negotiate a session key with a destination B located in $\mathscr{D}$ to communicate confidentially with B.

We now prove two lemmas that will be used in our analysis of SAINT's desired properties.

Lemma 1. *The existence of a path $(\mathscr{X}_1, \ldots, \mathscr{X}_n)$ implies the existence of a certificate chain $\langle \mathscr{X}_1, \ldots, \mathscr{X}_n \rangle$.*

Proof. By definition of a path, for all i such that $1 \leq i < n$, $\mathscr{X}_{i+1} \in \mathsf{N}(\mathscr{X}_i)$. Since $\mathsf{N}(\mathscr{X}_i) \subseteq \mathsf{CN}(\mathscr{X}_i)$, $\mathscr{X}_{i+1} \in \mathsf{CN}(\mathscr{X}_i)$. This implies that for all i such that $1 \leq i < n$, $\mathsf{CSC}(\mathscr{X}_i, \mathscr{X}_{i+1})$ exists. Thus the certificate chain $\langle \mathscr{X}_1, \ldots, \mathscr{X}_n \rangle$ exists. □

Lemma 2. *Paths are closed under the substring operation, symmetric, and transitive. That is, for all paths $(\mathscr{X}_1, \ldots, \mathscr{X}_m)$ and $(\mathscr{Y}_1, \ldots, \mathscr{Y}_m)$, the following statements hold:*

1. *for all i and j where $1 \leq i \leq j \leq n$, $(\mathscr{X}_i, \ldots, \mathscr{X}_j)$ is a path.*
2. *$(\mathscr{X}_m, \ldots, \mathscr{X}_1)$ is a path.*
3. *if $\mathscr{X}_m = \mathscr{Y}_1$, then $(\mathscr{X}_1, \ldots, \mathscr{X}_m, \mathscr{Y}_2, \ldots, \mathscr{Y}_n)$ is a path.*

Proof. We prove this by examining each statement separately.

1. Consider k such that $i \leq k < j$. Then $1 \leq i \leq k$, and thus $1 \leq k$. Also, $k < j \leq n$, and thus $k < n$. Since for all values of k we have $1 \leq k < n$, $\mathscr{X}_{k+1} \in \mathsf{N}(\mathscr{X}_k)$. Since this holds for all k where $i \leq k < j$, $(\mathscr{X}_i, \ldots, \mathscr{X}_j)$ is also a path.
2. Consider i such that $1 \leq i < m$. We know that $\mathscr{X}_{i+1} \in \mathsf{N}(\mathscr{X}_i)$, and this implies that $\mathscr{X}_i \in \mathsf{N}(\mathscr{X}_{i+1})$. We can thus say that for all $m \geq j > 1$, $\mathscr{X}_{j-1} \in \mathsf{N}(\mathscr{X}_j)$. Therefore, $(\mathscr{X}_m, \ldots, \mathscr{X}_1)$ is a path.
3. Due to the existence of the route $(\mathscr{Y}_1, \ldots, \mathscr{Y}_n)$ we can say that for all i such that $1 \leq i < n$, $\mathscr{Y}_{i+1} \in \mathsf{N}(\mathscr{Y}_1)$. Since $\mathscr{X}_m = \mathscr{Y}_1$, we know that $\mathscr{Y}_2 \in \mathsf{N}(\mathscr{X}_m)$. Then, combining this with the fact that for all i such that $1 \leq i < m$, $\mathscr{X}_{i+1} \in \mathsf{N}(\mathscr{X}_i)$ and for all j such that $2 \leq j < n$, $\mathscr{Y}_{j+1} \in \mathsf{N}(\mathscr{Y}_j)$, we can conclude that $(\mathscr{X}_1, \ldots, \mathscr{X}_m, \mathscr{Y}_2, \ldots, \mathscr{Y}_n)$ is a path.

We thus conclude that paths are closed under the substring operation, symmetric, and transitive. □

We can combine Lemmas 1 and 2 to show that certificate chains are also closed under the substring operation, symmetric, and transitive.

5.8.2 Analysis of Desired Properties

We now analyze our model in terms of the desired properties presented in Section 5.1. We present precise definitions for all properties except transparent authentication, and proofs of global discoverability and isolated authentication. For the remaining properties, we argue that our design of SAINT provides these properties.

Global discoverability Intuitively, the global discoverability property states that for a given client A with routing source ISD $\mathscr{S}$ and trust anchor ISD $\mathscr{T}$, A can obtain a certificate chain from $\mathscr{T}$ to any destination ISD $\mathscr{D}$ that has a route from $\mathscr{S}$.

Definition 5. Let A be a client with routing source ISD $\mathscr{S}$ and trust anchor ISD $\mathscr{T}$. We say that *global discoverability* holds if for any destination ISD $\mathscr{D}$ where there exists a route from $\mathscr{S}$ to $\mathscr{D}$, A can obtain a certificate chain with source $\mathscr{T}$ and destination $\mathscr{D}$.

We can show that if a client A knows a path to its trust anchor ISD $\mathscr{T}$, the A knows a certificate chain from $\mathscr{T}$ to $\mathscr{D}$, i.e., A can authenticate $\mathscr{D}$.

Theorem 1. *Let A be a client with a routing source ISD $\mathscr{S}$ and trust anchor ISD $\mathscr{T}$. Then for all domains $\mathscr{D}$ that have a route from $\mathscr{S}$, if A knows a route from $\mathscr{S}$ to $\mathscr{T}$, A can construct a certificate chain from $\mathscr{T}$ to $\mathscr{D}$.*

Proof. We observe that there exist routes from $\mathscr{S}$ to $\mathscr{T}$ and from $\mathscr{S}$ to $\mathscr{D}$. By symmetry of routes (Lemma 2), there exists a route from $\mathscr{T}$ to $\mathscr{S}$. Then by transitivity of routes (Lemma 2), we conclude that there exists a route from $\mathscr{T}$ to $\mathscr{D}$. Then by Lemma 1, a certificate chain from $\mathscr{T}$ to $\mathscr{D}$ exists. □

We note that A does not need to communicate with $\mathscr{T}$ at all in this process—with paths to $\mathscr{T}$ and $\mathscr{D}$ from $\mathscr{S}$, A has all it needs to construct the certificate chain. If A can reach $\mathscr{T}$ from $\mathscr{S}$, then it may be able to obtain a shorter certificate chain than one going through $\mathscr{S}$. Finally, even if A cannot reach $\mathscr{T}$ from $\mathscr{S}$, it may be able to obtain or construct a certificate chain from $\mathscr{T}$ to $\mathscr{D}$, using an out-of-band method or by using cached cross-signing certificates. Without making assumptions about the graph of ISDs, we cannot prove global discoverability, but we argue that given network connectivity today, A can likely reach $\mathscr{T}$ and thus obtain a certificate chain to $\mathscr{D}$.

Isolated authentication Intuitively, the isolated authentication property states that given a client A and a certificate chain $\mathscr{C}$, an ISD $\mathscr{M}$ can produce a signature that would cause A to reject $\mathscr{C}$ if and only if $\mathscr{M} \in \mathscr{C}$. In other words, any $\mathscr{M} \notin \mathscr{C}$ cannot cause A to reject $\mathscr{C}$. We can thus formalize isolated authentication as follows.

Definition 6. Let A be a client and $\mathscr{C} = \langle \mathscr{X}_1, \ldots, \mathscr{X}_n \rangle$ be a certificate chain. We say that *isolated authentication* holds if for an ISD $\mathscr{M}$, the trust roots of $\mathscr{M}$ can produce a signed message that causes A to reject the chain if and only if $\mathscr{M} = \mathscr{X}_i$ for some i where $1 \leq i < n$.

In SAINT, a client simply accepts a certificate chain if eval_A is sound. We show that this implies isolated authentication in SAINT.

Theorem 2. *Let A be a client where eval_A is sound with respect to an ISD $\mathscr{T}$. Then isolated authentication holds for all ISDs in the SAINT network.*

Proof. Suppose that an ISD $\mathscr{M}$ is not on a certificate chain $\mathscr{C} = \langle \mathscr{X}_1, \ldots, \mathscr{X}_n \rangle$, that is, for all i in $1 \leq i \leq n$, $\mathscr{M} \neq \mathscr{X}_i$. Then no signature or certificate by the trust roots of $\mathscr{M}$ will appear in $\mathscr{C}$. Since A only looks at the certificates on $\mathscr{C}$ to determine whether or not the chain is valid, $\mathscr{M}$ cannot produce a message that will cause A to reject the chain.

Suppose on the other hand that $\mathscr{M}$ is on the chain, that is, $\mathscr{M} = \mathscr{X}_i$ for some i where $1 \leq i < n$. Then we know that $\mathsf{CSC}(\mathscr{M}, \mathscr{X}_{i+1})$ exists. A will validate the chain $\mathscr{C}$, and $\mathsf{eval}_A(\mathscr{C}) = 0$ if $\mathscr{M}$ signs an invalid cross-signing certificate, such as one containing the wrong keys of $\mathsf{TR}(\mathscr{X}_{i+1})$. We note that A rejects $\mathscr{C}$ in this case, and thus $\mathscr{M}$ can cause A to reject $\mathscr{C}$. □

We thus conclude that a very simple method for computing $\mathsf{eval}_A(\mathscr{C})$ can provide isolated authentication. While we expect that in practice other methods could be used (such as those that rely on multiple certificate chains), we consider these methods out of the scope of our analysis, and instead propose challenges for future work to investigate these methods.

Trust agility Intuitively, the trust agility property states that A has a choice of trust root ISDs to verify a destination ISD and that A can easily modify this choice at any time. More precisely, A should be able to download the TRC file for any ISD $\mathscr{T}$ and begin using it as a trust anchor ISD.

Definition 7. Let $\mathscr{D}$ be the set of ISDs in the SAINT network, and let A be a client. We say that *trust agility* holds if for all $\mathscr{T} \in \mathscr{D}$, A can obtain and verify the TRC file of $\mathscr{T}$.

When analyzing trust agility in SAINT, we consider whether or not A has an existing trust anchor ISD $\mathscr{T}_0$, and whether or not A has a path to $\mathscr{T}$. We thus analyze four cases:

1. If A anchors its trust in $\mathscr{T}_0$ and has a path to $\mathscr{T}$, then by Theorem 1 A can verify a certificate chain from $\mathscr{T}_0$ to $\mathscr{T}$.
2. If A anchors its trust in $\mathscr{T}_0$ but does not have a path to $\mathscr{T}$, then A can obtain the TRC file of $\mathscr{T}$ from $\mathscr{T}_0$'s TRC server, or contact other ISDs that A can reach to obtain the TRC file.

3. A has no existing trust anchor but has a path to $\mathscr{T}$, then A can trust its routing source ISD by default to verify its route to $\mathscr{T}$, and in the process obtain a certificate chain to $\mathscr{T}$ that can be verified.
4. If A has no existing trust anchor and no path to $\mathscr{T}$, then A must obtain $\mathscr{T}$'s TRC file out of band.

As with global discoverability, we note that A may rely on cached cross-signing certificates or out-of-band communication to obtain and verify $\mathscr{T}$'s TRC file. As stated in Section 5.4.3, A can obtain an initial TRC file in a variety of ways. In fact, how A obtains the TRC file is irrelevant, and in the absence of any certificate chains to verify the TRC file, A can rely on out-of-band information for the verification as well.

Update efficiency Intuitively, update efficiency means that a client automatically obtains the latest trust root information. The purpose of providing update efficiency is so that relying parties such as clients, servers, and ISDs can have the latest trust root information when verifying a route, name, or EE certificate. We observe, however, that a relying party does not need to have the latest trust root information until it must use the information for verification. We can thus concretely define update efficiency in the following manner:

Definition 8. Let A be a relying party. We say that *update efficiency* holds if A processes at most one route, name record, or EE certificate each with outdated trust root information.

We can argue that this definition holds in SAINT if A performs a single request for a route, name, or EE certificate at a time (i.e., A does not request multiple routes, names, or certificates at once). Upon receiving a route, name, or certificate with a newer TRC version, A simply performs an additional round trip to its parent or to its trust anchor ISD to obtain the latest TRC file.

Transparent authentication Since the idea of transparent authentication is simply that a client can determine what ISD's trust roots are responsible for a part of the certificate chain, we present a simple argument to show that SAINT provides transparent authentication. We can observe that a client can simply start with its trust root ISD and traverse the certificate chain. For each cross-signing certificate observed, the client can conclude that the ISD whose trust roots are authenticated in the cross-signing certificate is now responsible for all subsequent certificates until the next cross-signing certificate is observed.

5.9 Implementation and Evaluation

In this section, we describe our prototype implementation of SAINT. We used this implementation to evaluate the performance of authentication and trust root management functions; we also discuss our evaluation results here.

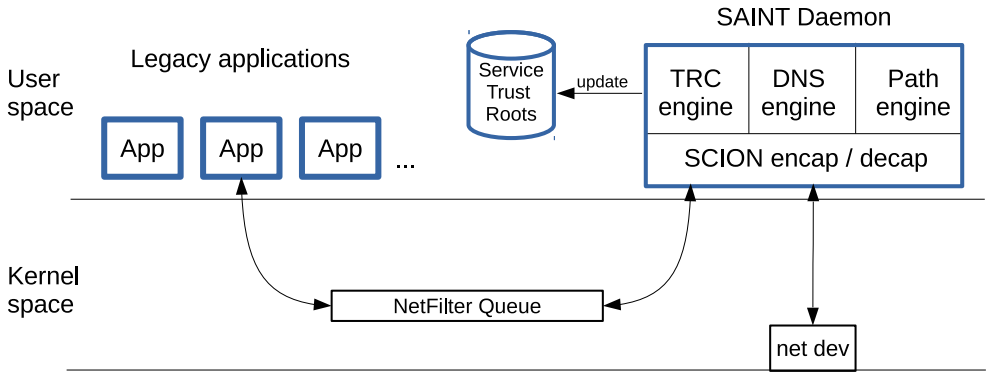


Figure 5.9: Architecture for endhost implementation.

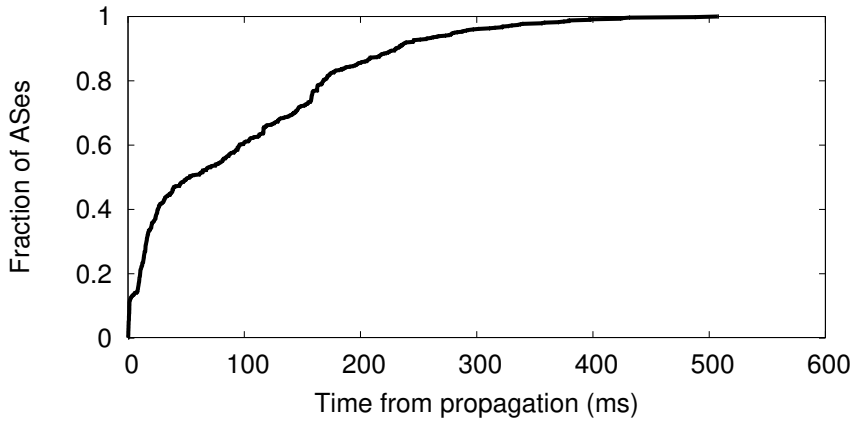


Figure 5.10: CDF of the percentage of vantage point ASes using a new TRC after propagation from the core ISPs.

5.9.1 Implementation

We implemented the endhost side of SAINT Figure 5.9). The main component of our implementation is the *SAINT daemon*, which acts as a gateway between applications and the network. The SAINT daemon includes SCION layer support for packet encapsulation and decapsulation, a path engine for route management and verification, a name lookup engine for SAINT name queries, and a TRC engine, which allows users to obtain and verify TRC files.

Traffic generated by the applications is delivered to the SAINT daemon by the NetFilter queue, allowing legacy applications to deploy SAINT without requiring any changes. We ran our simulation on an Intel Core i5-3380M CPU at 2.90 GHz, 16 GB of RAM, Python 3.4, and gcc 4.8.2. We used ed25519 [33] as our signature scheme for name and path verification, and RSA-2048 for TLS certificates.

5.9.2 TRC Updates

We measured the efficiency of TRC distribution and updates by simulating the propagation through the current AS topology. We used the CAIDA inferred AS relationships dataset from October 2014 [40] as our model of the current

Measurement	Min	Max	Med	Avg
DNS resolution	199	967	557	500
Path verification	348	1 691	691	652
Certificate validation (TLS)	210	1 123	222	233
Certificate validation (TLS+CRL)	401	1 775	440	460

Table 5.3: Evaluation results (in microseconds).

topology, and used traceroutes from iPlane datasets[2] to estimate inter-AS latency. From each of iPlane's vantage point ASes (distributed throughout the world), we identified the latency (half of the RTT) to each of the top-tier ASes identified using the CAIDA-based topology.

Our evaluation demonstrates that more than half of our vantage point ASes receive a new TRC file within 100 ms of the file being sent from the core ISPs (see Figure 5.10). Moreover, all vantage point ASes receive the new TRC file within 600 ms. Our vantage point ASes included stub ASes (that is, ASes with no customer ASes), demonstrating that end users around the world can quickly receive updated TRC files. These results show that our TRC propagation mechanism is significantly more efficient than the current trust root update mechanisms in browsers (which typically occur on the order of days).

5.9.3 Authentication Overhead and Performance

To test the actual latency of authentication in SAINT, we measured 300 end-to-end secure connection establishments on a sample SCION topology of virtual machines. Table 5.3 shows the timing results, which only reflect the cryptographic verifications and do not take into account network delay, since a full network deployment of SCION is not available at this time. However, our results give us an insight into the overhead of SAINT. In particular, all functions take less than 1 ms on average, which is significantly less than the end-to-end round trip time in an actual connection establishment.

As a baseline, we measured end-to-end connection establishments to 100 HTTPS sites on the current Internet randomly selected from `httpsnow.org`. We observed 418 separate TLS connection establishments. Using DNSSEC, BGPsec, and TLS, we measured the latency of the total page loading time, which included blocking of the connection request, the DNS lookup, the connect, send, and wait, and receive times, and the TLS handshake. The observed latencies ranged from 15 ms to 7 969 ms, with an average of 534 ms and a median of 1 811 ms. For SAINT, we assumed that a path lookup has an equivalent latency to a DNS query (one round trip), fast cryptography is used in verification (ed25519), and an average of 8 keys are authenticated during the DNS and routing lookups. In this setting, we observed latencies ranging from 19 ms to 7 974 ms, with an average of 586 ms and a median of 1 863 ms. These

[2] http://iplane.cs.washington.edu/data/data.html

latencies are not based on a full deployment of SAINT, but indicate a reasonable 10% increase in connection latency on average as compared to the current Internet.

5.10 Deploying SAINT

We now discuss several deployment challenges for SAINT and propose several solutions to facilitate the deployment of SAINT in the current Internet. Specifically, we discuss SAINT's interoperability with the current Internet, and describe how ISDs can be initially deployed. We then propose a method for the initial distribution of TRC files.

The Legacy ISD To facilitate the incremental deployment of SAINT, we propose a special ISD called the *legacy ISD* $\mathscr{L}$, which represents the set of all domains that have not yet joined a SAINT ISD. For example, suppose that `a.com` maps to the IP address 1.2.3.4 in the current Internet, which is located in AS 567, which has not yet deployed SAINT. The domain `a.com` would then correspond to the address $(\mathscr{L}, 567, 1.2.3.4)$. The security guarantees for names, routes, and end-entity certificates in the legacy ISD are only as strong as they are in today's Internet.

One challenge we face in practice is that names in SAINT's generic TLDs may already exist in the legacy ISD. Not only do such collisions cause a problem for name resolution, but they also create vulnerabilities to downgrade attacks in SAINT's name lookup mechanism, since an adversary could simply return an unsecured DNS response in the legacy ISD for a query with a name collision. We therefore require SAINT name resolvers to query the legacy ISD as a last resort, and only with a proof of the name's absence in the SAINT namespace (such as an NSEC3 record).

Interoperability SAINT is designed with a focus on incremental deployability. ISDs can be deployed among individual networks, since the remainder of the Internet that has not yet deployed SAINT joins the the legacy ISD. The naming infrastructure is interoperable with that of the current Internet in that it can query the legacy DNS nameservers as is done today. Routing between two physically separated domains in SCION can utilize IP tunneling to communicate over the legacy ISD.

In order to maintain connectivity to the current Internet, servers and clients must support legacy authentication. In particular, clients and servers must continue to support DNSSEC, BGPsec, and TLS. Additionally, when servers receive incoming connections from the legacy ISD, they should not respond with SAINT-specific messages such as signed path sets or cross-signing certificates. However, TRC files will be made available through the legacy ISD in order to support the initial bootstrapping of trust roots.

ISD Deployment SAINT offers benefits even for early deployers of ISDs. A single-ISD deployment of SAINT provides isolation of compromises within that ISD. Additionally, the legacy ISD will be protected from compromised trust roots in every ISD deploying SAINT. Moreover, ISDs deploying SAINT enjoy greater flexibility in their choice

of alternative PKIs by enabling the benefits of the PKIs without requiring their global deployment. In the policy field of the TRC file, an ISD would specify its choice of the underlying PKI, which would prevent protocol downgrade attacks.

If using countries as ISDs, then a newly-deploying ISD can simply attach to the existing namespace at its corresponding ccTLD. This construction allows the DNS to provide a scaffolding during the deployment of SAINT and also allows the DNS in SAINT to distribute TRC files.

However, we recognize the challenges that come with using countries as ISDs. In particular, the deployment of such a scheme would require the core ISPs, root CAs, and Internet registries in each country to create a federation of trust roots. In practice, we may see corporations rather than countries form ISDs. In this case, ISDs would have to form IP tunnels in order to form inter-ISD routing relationships. Additionally, since there are far more corporations than countries, cross-signing relationships may not scale to this number of ISDs. However, because most corporations do not form many business relationships relative to the number of corporations that exist, we do not expect that the number of cross-signing relationships will grow to an unsustainable scale.

TRC Distribution The initial distribution of TRCs must occur securely since TRCs are the starting point for all authentication in SAINT. Many trust roots in the current Internet may continue to serve as trust roots in SAINT, and thus may be able to "inherit" user trust in SAINT that they already have in the current Internet. However, SAINT will likely result in the creation of new trust roots, and thus must have a mechanism for bootstrapping trust in the initial public keys of these roots.

To address this challenge, we suggest to perform the initial distribution of SAINT TRC files through DNSSEC. Since ISDs can deploy by attaching to specific ccTLDs in the current DNS namespace, an ISD can create a reserved domain name such as `trc.us`, whose DNS record contains the TRC (e.g., in a TXT record). Clients can then fetch the TRC by looking up the appropriate domain name. Additional work has been done in distributing authentication information through out-of-band means such as over public radio [118], but these strategies are beyond the scope of this project.

5.11 Discussion

Feasibility of country-based ISDs In order to determine the feasibility of having countries as ISDs in SAINT as described in Section 5.2.1, we mapped AS numbers to countries and examined the resulting inter-ISD relationships. We used the AS relationships database from CAIDA [40] and Team Cymru's IP to AS number mapping tool,[3] to map AS numbers to countries. We identified 228 "countries" in total, including the EU and ZZ (indicating that the AS's country was unknown). We identified 2 636 unique country pairs between which an inter-AS link existed. These links

[3] http://www.team-cymru.org/Services/ip-to-asn.html

signify direct routing connections, and thus we expect cross-signing for each ISD pair. The most prolific cross-signing ISDs were the US (196), the EU (135), and the UK (124), but half of the ISDs cross-sign on the order of tens of other ISDs.

Political Concerns Given concerns over governmental nation-wide surveillance, readers may worry about centralizing trust roots in a large ISD. While we acknowledge that states may compromise these entities on a large scale, SAINT's trust agility allows users to protect themselves from the interception of sensitive connections. Additionally, our efficient method of updating trust root information allows ISDs to quickly recover from a compromised CA.

Another concern we anticipate is that SAINT encourages fragmentation in the Internet. While this is true, SAINT is designed to preserve global reachability while simultaneously protecting users through the use of ISDs and trust agility. We thus structure inter-ISD authentication to provide users with the best of both worlds.

Sub-ISDs To add further scalability to SAINT, we propose the use of *sub-ISDs*, i.e., ISDs being completely contained in other ISDs. A sub-ISD can be used to provide additional policies and finer-grained control of trust roots in an ISD, and can additionally enable isolated authentication within an ISD. For example, governments or medical organizations may use their own network within a country ISD to ensure data privacy and further scope the authority of their trust roots.

A sub-ISD structurally resembles a top-level ISD, but authenticates to other sub-ISDs in the same parent ISD using the core of the parent ISD, and has its trust roots of a certified by its parent ISD via a cross-signing certificate. Connections within an ISD could then be negotiated using the lowest-level common ISD. For example, two hospitals in an ISD would share data about a patient using the medical sub-ISD rather than the general ISD.

Some details regarding authentication for sub-ISDs still remain, however. For example, compromised trust roots could also affect authentication entirely within a sub-ISD due to the requirement that parent ISDs certify the trust roots of their sub-ISDs. Furthermore, users cannot select sub-ISDs as their trust anchor ISDs without also trusting the corresponding parent ISDs. We hope to investigate these challenges in future work.

Optimizations As described in Section 5.7, the authentication process involves six round-trip connections from Alice. Each communication with the path server requires verification of the path server's signature, the signed set of routing paths returned by the path server, the destination AS certificate, and possibly cross-signing certificates for the path server and destination AS's ISDs. To contact a destination outside her trust anchor ISD, Alice must also obtain and verify a cross-signing certificate. Contacting the DNS server requires verification of at least the DNS root key, server DNS key, and signed DNS record, and contacting the server (Bob) requires verification of at least the server certificate.

However, in practice many of these verifications may not be necessary. Caching cross-signing certificates, for example, eliminates the need to reach a TRC server and to verify a cross-signing certificate with each end-to-end connection establishment. Additionally, some of these verifications can be handled by entities other than the client; for example, paths can be verified by the client's AS, and DNS records by a trusted DNS stub resolver. Since ASes and stub resolvers serve multiple clients, caching verification results can further reduce the connection latency, especially for popular names, routes, and end-entity certificates.

In order to further decrease the size of messages sent in the network, we can also split the TRC file into routing and a service TRC files. The routing TRC can then be propagated along AS links, and the service TRC can be obtained from the TRC server. This scheme ensures that users only receive the portion of the TRC that they need for a particular type of authentication, thus reducing the size of TRC files sent in the network.

Chapter 6

Conclusion

My work on IKP, CAPS, and SAINT shows that minimal changes to existing or emerging technologies can be used to build deployable solutions that reduce MITM attacks in the Web PKI. Each of these systems has helped advance the state of effective and practical improvements to the Web PKI and opened new avenues of exploration to further improve the resilience of PKIs against compromised trusted parties while being easy to use and deploy.

With IKP, I used incentives to encourage good CA behavior and the reporting of CA misbehavior while preventing large-scale abuse of these incentives. By designing a decentralized, incentivized platform to handle this reporting and to handle responses to CA misbehavior, I ensured that misbehaving CAs could be swiftly detected and punished without having to trust a centralized entity. My model and analysis of the incentives show that IKP is effective in deterring MITM attacks, and my implementation in Ethereum shows that deployment, even with a rich set of possible policies, is simple and inexpensive. My work on IKP highlights that there is still work to be done in exploring the incentives behind CA actions and in how the risk of different trusted parties can be more rigorously calculated. IKP also illustrates both the advantages and the challenges of building an incentivizing system on a platform such as Ethereum. However, IKP provides a concrete first step toward the goal of securing the Web PKI through incentives.

With CAPS, I used automated, global certificate monitoring to quickly and easily protect against MITM attacks in the current Web PKI. By using public logs to track HTTPS deployment and certificates for each domain, I ensured that domains could protect against the concerted misbehavior of multiple CAs while avoiding downgrade attacks and much of the perils of misconfiguration or loss of a private key. My approach to representing domains' HTTPS deployment status and certificates is efficient from both theoretical and practical standpoints. Furthermore, though certificate policies in CAPS are simple, they are easy to update in the current PKI in case of catastrophic misbehavior at either the domain or at CAs. My work on CAPS highlights that our view on the global certificate system is still incomplete and can be improved, but that the current view of certificates is sufficient to stop many of the MITM attacks in the Web PKI. CAPS also highlights that deploying complex certificate policies in the current Web PKI is difficult, but offers a

stepping stone to stronger improvements to the Web PKI.

With SAINT, I used the inherent business relationships underlying routing connections to explore desriable properties in a federated PKI system. The network of routing channels provides a basis for isolated domains for authentication, providing flexibility and resilience in the public-key authentication process. By distributing trust root information as network messages, I enable users to quickly obtain up-to-date information about compromised or updated trust roots. By mandating cross-signing relationships based on routing connections, I ensure that users can authenticate information throughout the Internet. By separating routing and service authentication, I allow users' trust root decisions to apply anywhere in the world. While there is still much exploration to be done in the design space of federated PKIs, my work on SAINT highlights fundamental and desirable properties that any global PKI should strive to achieve.

Through the work presented in this thesis and through continued research in this space, I hope that one day, misbehaving CAs will cease to be catastrophic to the Web PKI. Through carefully chosen incentives, practical deployment strategies, and sound design principles, we can move towards PKIs in which MITM attacks are rare, and in which users can have a rigorous basis for confidence in a domain's public key. Though much work remains to be done to this end, my work has advanced the Web PKI towards this goal. I hope that I can continue to improve the Web PKI and other PKIs—one certificate at a time.

Bibliography

[1] "Let's Encrypt," https://letsencrypt.org/.

[2] "Namecoin," https://namecoin.info/.

[3] "Comodo fraud incident 2011-03-23," https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, March 2011.

[4] "VASCO announces bankruptcy filing by DigiNotar B.V." https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_vasco_announces_bankruptcy_filing_by_diginotar_bv.aspx, September 2011.

[5] "Information technology – Open Systems Interconnection – the directory: Public-key and attribute certificate frameworks," ITU-T X.509, October 2012.

[6] "Script," httsp://en.bitcoin.it/wiki/Script, October 2014.

[7] "Information technology – ASN.1 encoding rules: Specification of Basic encoding rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)," ITU-T X.690, August 2015.

[8] "Lenovo statement on Superfish," https://news.lenovo.com/article_display.cfm?article_id=1929, February 2015.

[9] "Register and configure .bit domains," https://wiki.namecoin.info/index.php?title=Register_and_Configure_.bit_domains, May 2015.

[10] "Certum by Asseco," https://en.sklep.certum.pl/data-safety/ssl-certificates.html, July 2016.

[11] "Comodo," https://ssl.comodo.com/ssl-certificate.php, July 2016.

[12] "GlobalSign SSL," https://www.globalsign.com/en/ssl/, July 2016.

[13] "IdenTrust SSL," https://www.identrustssl.com/buy.html, July 2016.

[14] "SSL certificate comparison," https://www.entrust.com/ssl-certificate-comparison/, July 2016.

[15] "Starfield technologies," https://www.starfieldtech.com/, July 2016.

[16] "StartCom," https://www.startssl.com/, July 2016.

[17] "Usage of SSL certificate authorities for websites," http://w3techs.com/technologies/overview/ssl_certificate/all, July 2016.

[18] "The Verisign domain name industry brief," https://www.verisign.com/assets/domain-name-report-Q42016.pdf, February 2017.

[19] "Baseline requirements for the issuance and management of publicly-trusted certificates, version 1.6.2," CA/Browser Forum, December 2018.

[20] "Guidelines for the issuance and management of extended validation certificates, version 1.6.8," CA/Browser Forum, March 2018.

[21] A. A. Akplogan, J. Curran, P. Wilson, R. Housley, F. Chehadé, J. Arkko, L. S. Amour, R. Echeberría, A. Pawlik, and J. Jaffe, "Montevideo statement on the future of Internet cooperation," https://www.icann.org/news/announcement-2013-10-07-en, October 2013.

[22] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *USENIX Annual Technical Conference (ATC)*, June 2016.

[23] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," RFC 4033, March 2005.

[24] ——, "Protocol modifications for the DNS Security Extensions," RFC 4035, March 2005.

[25] ——, "Resource records for the DNS Security Extensions," RFC 4034, March 2005.

[26] H. Asghari, M. J. G. van Eeten, A. M. Arnbak, and N. A. N. M. van Eijk, "Security economics in the HTTPS value chain," in *Workshop on the Economics of Information Security (WEIS)*, November 2013.

[27] A. Back, "Hashcash: A denial of service counter-measure," http://www.cypherspace.org/adam/hashcash/, August 2002.

[28] D. Barrera, R. M. Reischuk, P. Szalachowski, and A. Perrig, "SCION five years later: Revisiting scalability, control, and isolation on next-generation networks," arXiv:1508.01651, August 2015.

[29] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub, "Computer-aided verification in mechanism design," arXiv:1502.04052v4 [cs.GT], October 2016.

[30] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 382–393.

[31] A. Beregszaszi, "RSA signature verification in Ethereum," https://github.com/axic/ethereum-rsa, April 2016.

[32] ——, "Support RSA signature verification," https://github.com/ethereum/EIPs/issues/74, March 2016.

[33] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[34] A. D. Birrell, B. W. Lampson, R. M. Needham, and M. D. Schroeder, "A global authentication service without global trust," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 1986, pp. 223–223.

[35] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[36] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies," in *IEEE Symposium on Security and Privacy (S&P)*, May 2015.

[37] V. Buterin, "On mining," Ethereum Blog, June 2014.

[38] ——, "Understanding Serenity, part I: Abstraction," https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/, December 2015.

[39] ——, "Critical update re: DAO vulnerability," Ethereum Blog, June 2016.

[40] CAIDA, "The CAIDA AS relationships dataset," http://www.caida.org/data/as-relationships/, October 2014.

[41] D. Chaum and E. Van Heyst, "Group signatures," in *Advances in Cryptology—EUROCRYPT'91*. Springer, April 1991, pp. 257–265.

[42] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A longitudinal, end-to-end view of the DNSSEC ecosystem," in *USENIX Security Symposium*, August 2017, pp. 1307–1322.

[43] T. Chung, R. van Rijswijk-Deij, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Understanding the role of registrars in DNSSEC deployment," in *Internet Measurement Conference (IMC)*, November 2017.

[44] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," in *IEEE Symposium on Security and Privacy (S&P)*, May 2013, pp. 511–525.

[45] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg, "On the risk of misbehaving RPKI authorities," in *Twelfth ACM Workshop on Hot Topics in Networks (HotNets)*. ACM, 2013, p. 16.

[46] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and T. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," RFC 5280, May 2008.

[47] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *USENIX Security Symposium*, August 2009, pp. 317–334.

[48] J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson, "Incremental construction of minimal acyclic finite-state automata," *Computational Linguistics*, vol. 26, no. 1, pp. 3–16, March 2000.

[49] J. Daciuk and D. Weiss, "Smaller representation of finite state automata," *Theoretical Computer Science*, vol. 450, no. 7, pp. 10–21, September 2012.

[50] K. Delmolino, A. Mitchell, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *Financial Cryptography and Data Security (FC)*, February 2016.

[51] C. Dillow, "An order of seven global cyber-guardians now hold keys to the Internet," http://www.popsci.com/technology/article/2010-07/order-seven-cyber-guardians-around-world-now-hold-keys-internet, July 2010.

[52] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[53] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *USENIX Security Symposium*, vol. 8, August 2013, pp. 47–53.

[54] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology (CRYPTO)*, August 1992, pp. 139–147.

[55] D. Eastlake, "Transport Layer Security (TLS) extensions: Extension definitions," RFC 6066, January 2011.

[56] 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.

[57] ——, "Sovereign Key cryptography for Internet domains," https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=master, June 2012.

[58] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," RFC 7469, April 2015.

[59] I. Eyal and E. G. Sirer, "Majority mining is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security (FC)*, March 2014.

[60] V. D. Gligor, S.-W. Luan, and J. N. Pato, "On inter-realm authentication in large distributed systems," in *IEEE Symposium on Security and Privacy*. IEEE, 1992, pp. 2–17.

[61] D. Goodin, "Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections," https://arstechnica.com/information-technology/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/, February 2015.

[62] M. Goodwin, "Revoking intermediate certificates: Introducing OneCRL," https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/, March 2015.

[63] P. Gutmann, "PKI: it's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[64] P. Hallam-Baker and R. Stradling, "DNS certificate authority authorization (CAA) resource record," RFC 6844, January 2013.

[65] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security)*, 2015, pp. 129–144.

[66] J. Hodges, C. Jackson, and A. Barth, "HTTP strict transport security (HSTS)," RFC 6797, November 2012.

[67] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA," RFC 6698, August 2012.

[68] H. Hoogstraaten, R. Prins, D. Niggebrugge, D. Heppener, F. Groenewegen, J. Wettink, K. Strooy, P. Arends, P. Pols, R. Kouprie, S. Moorrees, X. van Pelt, and Y. Z. Hu, "Black Tulip: Report of the investigation into the DigiNotar certificate authority breach," www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf, August 2012.

[69] R. Housley, W. Ford, T. Polk, and D. Solo, "Internet X.509 public key infrastructure certificate and CRL profile," RFC 2459, January 1999.

[70] G. Huston, G. Michaelson, and R. Loomans, "A profile for X.509 PKIX resource certificates," RFC 6487, February 2012.

[71] C. Jentzsch, "Decentralized autonomous organization to automate governance," White paper, November 2016.

[72] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *Workshop on the Economics of Information Security (WEIS)*. Citeseer, 2015.

[73] J. Kasten, E. Wustrow, and J. A. Halderman, "CAge: Taming certificate authorities by inferring restricted scopes," in *Financial Cryptography and Data Security (FC)*. Springer, April 2013, pp. 329–337.

[74] D. Kim, B. J. Kwon, and T. Dumitras, "Certified malware: Measuring breaches of trust in the Windows code-signing PKI," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, November 2017, pp. 1435–1448.

[75] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor, "Accountable Key Infrastructure (AKI): A proposal for a public-key validation infrastructure," in *International World Wide Web Conference (WWW)*, May 2013, pp. 679–690.

[76] J. C. Klensin, "Internationalized Domain Names in Applications (IDNA): Protocol," RFC 5891, August 2010.

[77] H.-J. Knobloch and A. Domnick, "Certificate management vulnerability in Sennheiser HeadSetup," CVE-2018-17612, October 2018.

[78] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy (S&P)*, May 2016.

[79] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 265–310, 1992.

[80] A. Langley, "Revocation checking and Chrome's CRL," https://www.imperialviolet.org/2012/02/05/crlsets.html, February 2012.

[81] ——, "Enhancing digital certificate security," http://googleonlinesecurity.blogspot.com/2013/01/enhancing-digital-certificate-security.html, January 2013.

[82] ——, "Further improving digital certificate security," http://googleonlinesecurity.blogspot.com/2013/12/further-improving-digital-certificate.html, December 2013.

[83] ——, "Maintaining digital certificate security," http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html, March 2015.

[84] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *IEEE Symposium on Security and Privacy (S&P)*, May 2017.

[85] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," RFC 6962, June 2013.

[86] M. Lepinski and S. Kent, "An infrastructure to support secure internet routing," RFC 6480, February 2012.

[87] M. Lepinski and K. Sriram, "BGPsec protocol specification," RFC 8205, September 2017.

[88] C. Lynn, S. Kent, and K. Seo, "X.509 extensions for IP addresses and AS identifiers," RFC 3779, June 2004.

[89] G. Markham, "WoSign and StartCom," https://docs.google.com/document/d/1C6BlmbeQfn4a9zydVi2UvjBGv6szuSB4sMYUcVrR8vQ, September 2016.

[90] M. Marlinspike, "New tricks for defeating SSL in practice," http://www.thoughtcrime.org/software/sslstrip/, 2009.

[91] ——, "SSL and the future of authenticity," http://www.youtube.com/watch?v=Z7Wl2FW2TcA, BlackHat 2011, August 2011.

[92] M. Marlinspike and T. Perrin, "Trust assertions for certificate keys," https://tools.ietf.org/html/draft-perrin-tls-tack-02, (work in progress)., January 2013.

[93] S. Matsumoto and R. M. Reischuk, "Certificates-as-an-Insurance: Incentivizing accountability in SSL/TLS," in *NDSS Workshop on Security of Emerging Network Technologies (SENT)*, February 2015.

[94] ——, "IKP: Turning a PKI around with decentralized automated incentives," in *IEEE Symposium on Security and Privacy (S&P)*, May 2017.

[95] S. Matsumoto, R. M. Reischuk, P. Szalachowski, T. H.-J. Kim, and A. Perrig, "Authentication challenges in a global environment," *ACM Transactions on Privacy and Security (TOPS)*, vol. 20, no. 1, pp. 1–34, January 2017.

[96] S. Matsumoto, P. Szalachowski, and A. Perrig, "Deployment challenges in log-based PKI enhancements," in *European Workshop on System Security (EuroSec)*, April 2015.

[97] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Coniks: Bringing key transparency to end users," in *24th USENIX Security Symposium (USENIX Security 15)*, August 2015, pp. 383–398.

[98] R. C. Merkle, "A digital signature based on a conventional encryption function," *Advances in Cryptology (CRYPTO)*, pp. 369–378, 1988.

[99] Microsoft, "Erroneous VeriSign-issued digital certificates pose spoofing hazard," https://technet.microsoft.com/library/security/ms01-017, Mar. 2001.

[100] P. Mockapetris, "Domain names – implementation and specification," RFC 1035, November 1987.

[101] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Whitepaper*, October 2008.

[102] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, March 2016.

[103] E. Nigg, "StartCom certificate policy and practice statements," https://www.startssl.com/policy.pdf, May 2016.

[104] J. Nightingale, "DigiNotar removal follow up," https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/, September 2011.

[105] N. Nisan and A. Ronen, "Algorithmic mechanism design," in *ACM Symposium on Theory of Computing*, 1999, pp. 129–140.

[106] M. Nystrom and B. Kaliski, "PKCS #10: Certification request syntax specification," RFC 2986, November 2000.

[107] N. Percoco, "Clarifying the Trustwave CA policy update," http://blog.spiderlabs.com/2012/02/clarifying-the-trustwave-ca-policy-update.html, February 2012.

[108] H. Perl, S. Fahl, and M. Smith, "You won't be needing these any more: On removing unused certificates from trust stores," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 307–315.

[109] R. Perlman, "An overview of PKI trust models," *IEEE Network*, vol. 13, no. 6, pp. 38–43, 1999.

[110] V. Ramasubramanian and E. G. Sirer, "Perils of transitive trust in the Domain Name System," in *5th ACM Internet Measurement Conference (IMC)*, 2005, pp. 379–384.

[111] M. K. Reiter and S. G. Stubblebine, "Authentication metric analysis and design," *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 138–158, May 1999.

[112] E. Rescorla, "HTTP over TLS," RFC 2818, May 2000.

[113] ——, "The Transport Layer Security (TLS) protocol version 1.3," RFC 8446, August 2018.

[114] R. L. Rivest, "Can we eliminate certificate revocation lists?" in *Financial Cryptography*. Springer, 1998, pp. 178–183.

[115] K. Salikhov, G. Sacomoto, and G. Kucherov, "Using cascading Bloom filters to improve the memory usage for de Brujin graphs," *Algorithms for Molecular Biology*, vol. 9, no. 2, February 2014.

[116] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet public key infrastructure online certificate status protocol - OCSP," RFC 6960, June 2013.

[117] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in Bitcoin," arXiv:1507.06183v2 [cs.CR], July 2015.

[118] A. Schulman, D. Levin, and N. Spring, "RevCast: Fast, private certificate revocation over FM radio," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2014, pp. 799–810.

[119] C. Sharp and D. York, "State of DNSSEC deployment 2016," Internet Society Report, December 2016.

[120] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology — EUROCRYPT 2000*. Springer, May 2000, pp. 207–220.

[121] R. Sleevi, "Sustaining digital certificate security," https://googleonlinesecurity.blogspot.com/2015/10/sustaining-digital-certificate-security.html, October 2015.

[122] ——, "Intent to deprecate and remove: Trust in existing Symantec-issued certificates," https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/eUAKwjihhBs/rpxMXjZHCQAJ, March 2017.

[123] S. Somogyi and A. Eijdenberg, "Improved digital certificate security," https://googleonlinesecurity.blogspot.com/2015/09/improved-digital-certificate-security.html, September 2015.

[124] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, L. Gasser, N. Gailly, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE Symposium on Security and Privacy (SP)*, May 2016.

[125] P. Szalachowski, L. Chuat, and A. Perrig, "PKI safety net (PKISN): Addressing the too-big-to-be-revoked problem of the TLS ecosystem," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, Mar. 2016.

[126] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible TLS certificate management," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, November 2014, pp. 406–417.

[127] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh, "Towards short-lived certificates," *Web 2.0 Security and Privacy*, 2012.

[128] TURKTRUST, "Technical details," http://turktrust.com.tr/en/kamuoyu-aciklamasi-en.2.html, January 2013.

[129] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman, "Towards a complete view of the certificate ecosystem," in *ACM Internet Measurement Conference (IMC)*, November 2016, pp. 543–549.

[130] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing," in *USENIX Annual Technical Conference (ATC)*, June 2008, pp. 321–334.

[131] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," White Paper, 2015.

[132] J. Yu and M. Ryan, "Evaluating Web PKIs," in *Software Architecture for Big Data and the Cloud*, 1st ed., I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, and B. Maxim, Eds. Elsevier, June 2017, ch. 7, pp. 105–126.

[133] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An authenticated data feed for smart contracts," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2016, pp. 270–282.

[134] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, control, and isolation on next-generation networks," in *IEEE Symposium on Security and Privacy (S&P)*, May 2011, pp. 212–227.

[135] P. Zimmerman, "PGP user's guide," http://www.pa.msu.edu/reference/pgpdoc1.html, October 1994.

[136] M. Zusman and A. Sotirov, "Sub-prime PKI: Attacking extended validation SSL," *Black Hat Security Briefings, Las Vegas, USA*, 2009.