2017 IEEE Symposium on Security and Privacy

IKP: Turning a PKI Around with Decentralized Automated Incentives

Stephanos Matsumoto
Carnegie Mellon University/ETH Zurich

Raphael M. Reischuk
ETH Zurich

***Abstract*—Despite a great deal of work to improve the TLS PKI, CA misbehavior continues to occur, resulting in unauthorized certificates that can be used to mount man-in-the-middle attacks against HTTPS sites. CAs lack the incentives to invest in higher security, and the manual effort required to report a rogue certificate deters many from contributing to the security of the TLS PKI. In this paper, we present IKP, a platform that automates responses to unauthorized certificates and provides incentives for CAs to behave correctly and for others to report potentially unauthorized certificates. Domains in IKP specify criteria for their certificates, and CAs specify reactions such as financial penalties that execute in case of unauthorized certificate issuance. By leveraging smart contracts and blockchain-based consensus, we can decentralize IKP while still providing automated incentives. We describe a theoretical model for payment flows and implement IKP in Ethereum to show that decentralizing and automating PKIs with financial incentives is both economically sound and technically viable.**

I. Introduction

Transport Layer Security (TLS) [30] secures much of the encrypted client-server communication in the World Wide Web: HTTPS [77], which runs over TLS, is now estimated to be used in more than half of all page loads in web browsing [39]. The security of the TLS public-key infrastructure (PKI) heavily relies on *certificate authorities (CAs)*, who make a business out of certifying the authenticity of sites' public keys. Without the correct operation of CAs, the use of encryption provides no benefit, as clients may use a key that does not actually belong to the legitimate server.

Unfortunately, CAs have shown to be prone to compromises and operational errors. These failures have occurred all around the world, including the US [26, 64], France [51], the Netherlands [42], Turkey [50], and China [52]. Even Symantec, which has almost a quarter of the TLS-certificate market share [10], issued unauthorized certificates for Google and almost 2,500 unauthorized certificates for both real and unregistered domains as part of a test in 2015 [80, 81]. Thus, while CAs play a critical role to the security of the TLS ecosystem, they have failed in this role by issuing unauthorized certificates in error, maliciously issuing certificates to avoid changing browser requirements [56], or even selling CA authority as a service [74]. Some of these failures have led to man-in-the-middle (MitM) attacks, allowing the interception of communication with popular sites such as Google, Microsoft Live, Skype, and Yahoo [65, 66].

We observe that **despite the need for CAs to invest more in security, there are insufficient incentives for them to do so.** CAs that issue unauthorized certificates enable MitM attacks between clients and domains, who suffer the consequences of a rogue CA's misbehavior [59]. These MitM attacks are often only visible to those being attacked, since an adversary could present an unauthorized certificate for a domain only to some clients, so that the other clients do not even know that the unauthorized certificate exists. Moreover, while CAs may face the consequence of being distrusted by browsers [52, 72], some CAs are "too big to fail," meaning that their removal would block access to too many HTTPS sites and is thus unlikely. While proposed solutions to the above problems exist [54, 83], CAs generally gain little reward for a reputation of security and face few consequences for misbehaving [14].

We also observe that **due to a lack of automation, reporting unauthorized certificates is time and labor-intensive.** When a CA issues an unauthorized certificate for a domain, a *detector* (the entity who discovers the certificate) has several options. First, a detector can contact the misbehaving CA directly, because only the CA can revoke the certificate. However, if the CA is malicious, it may never revoke the certificate, leaving the domain open to potential MitM attacks. Even revocation may not help since some widely-used browsers do not check revocation information at all [55]. The detector could instead contact browser vendors, who can update client browsers to reject the certificate [49]. However, such a response is unlikely except for relatively popular sites. A detector could also pursue legal action against the CA, but this process may be long, costly and ultimately unfruitful, due in part to the fact that CAs are located in approximately 52 countries [33], each with its own legal system. Thus due to the unlikely recourse and the effort required, there are insufficient incentives for detectors to report unauthorized certificates.

Therefore, in this paper we ask two fundamental questions: **how can we better incentivize correct CA behavior and the reporting of misbehavior**, and **how can we automate the processing of an unauthorized certificate report**? In particular, 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, we propose **Instant Karma PKI (IKP)**, an automated platform for defining and reporting CA misbehavior that incentivizes CAs to correctly issue certificates and detectors to quickly

DOI 10.1109/SP.2017.57

report unauthorized certificates. IKP allows domains to specify policies that define CA misbehavior, and CAs to sell insurance against misbehavior. We also propose a formal model for incentive analysis to show that IKP provides incentives for CAs and detectors and punishes misbehaving CAs. We further show that with our incentive structure, even CAs that collude with other domains or detectors cannot profit financially.

More 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.

We have implemented a prototype of IKP in Ethereum [87], a blockchain-based smart contract platform that provides important properties for achieving incentivization and automation. Ether, the cryptocurrency underlying Ethereum, is a natural basis for implementing financial transactions and incentives. The smart contract ecosystem provides a public, automated mechanism for handling detector reports and executing financial transactions, ensuring quick responses to CA misbehavior. Furthermore, Ethereum provides *decentralization* so that no trusted third party is needed to register DCPs, RPs, and financial assets. While incentivization and automation are possible with a centralized third party, we protect IKP itself against compromise by building it on top of Ethereum.

To provide realistic incentive amounts, we also analyze certificate offers from the most widely-used CAs, quantifying and bound the risks of CA misbehavior. These insights allow us to predict realistic payment amounts for RP.

In summary, we make the following contributions:

- We present the design of IKP, including a framework for domain policies and reactions to CA misbehavior.
- We demonstrate through an economic analysis that IKP incentivizes good CA behavior and punishes misbehavior.
- We implement an IKP prototype in Ethereum and discuss the present and future technical feasibility of IKP.
- We analyze real-world data from existing CAs to determine realistic values for RP offerings.

II. Problem Definition and Adversary Model

In a nutshell, the goal of this paper 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.

A. 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).

B. 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.

III. IKP 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. An extended version [60] of this paper provides additional material.

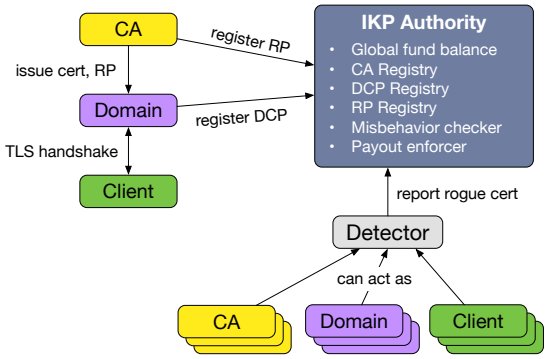


Fig. 1. Overview of the entities and functions in IKP.

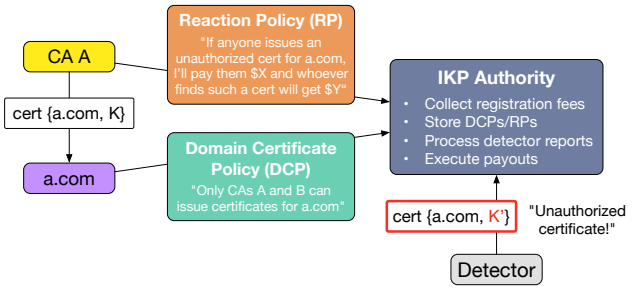


Fig. 2. Sample interactions between entities in IKP. As in Figure 1, yellow denotes a CA and purple denotes a domain.

A. 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 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)*, 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 2 shows intuitive examples of a DCP and an 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.

B. Operation

We now summarize the actions that occur in IKP, some of which are shown in Figure 2.

TABLE I
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)

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 I. To update its registration, the CA must provide signatures on the update with a threshold number of its update private keys.

Domain registration. A domain registers a DCP with the IKP authority. Specifically, the domain registers its Domain Name System (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 an 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 IV and V, 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.

IV. 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.

A. 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 [44, 84] allow domains to specify only a small set of parameters (e.g., governing how their certificates should be 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.

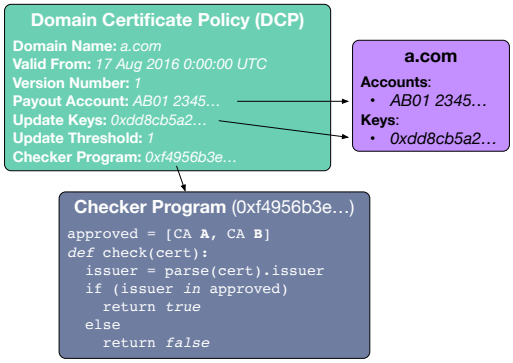


Fig. 3. A sample DCP with a checker program written in pseudocode.

TABLE II
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

B. DCP Contents

We now describe the contents of a DCP. Figure 3 shows a sample DCP, and Table II 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 an 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 an 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 [82].

C. 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 [27], 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 [85] 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 $\star$ does not appear.

Certificate Transparency. A checker program can implement criteria similar to those of Certificate Transparency [54] 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.

D. 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 [13] 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 [31], we found that only 649 of the top 100,000 most popular domains use both DNSSEC and HTTPS. Therefore, few domains will be able to reap the benefits of using DNSSEC-based bootstrap proofs.

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 [75]. 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.

V. 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 an RP.

A. 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. An 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, an RP protects a domain against any unauthorized certificate issuances during the lifetime of the RP.

2) Policy-adherence. An RP should be bound to a specific policy for a domain. In particular, since a DCP may change over time, an RP should represent a reaction to violations of a specific *version* of a domain's DCP. Binding an 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 an RP.

3) Single-use. An 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 an 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.

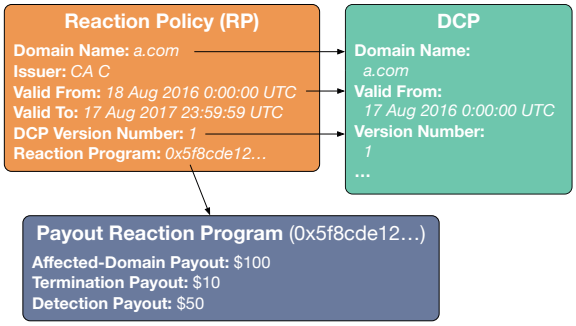


Fig. 4. A sample RP with a payout reaction program. The domain name and version number in the RP must match those of the DCP, and the start of the RP's validity must be after that of the DCP.

TABLE III
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

B. RP Contents

We now describe the contents and format of RPs. Figure 4 shows the format of a sample RP, and Table III describes each field of an RP. Like a DCP, an RP contains identifying information for the domain as well as for the issuing CA. An 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 V-C, 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 an RP has a start and end time for its validity, rather than only a start time as a DCP does. An RP, like a certificate, has a limited validity period, but can be prematurely terminated if the issuing CA misbehaves. If an 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.

C. 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 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

$$t_D = \alpha \cdot (t - \tau) + \tau \tag{1}$$

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

$$\tau \ \leq \ t_D \ \leq \ t \tag{2}$$

We note that although C does receive funds from the termination payout in spite of its misbehavior, we show in Section VI 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 VI.

D. RP Operations

We now describe relevant operations for an 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 an 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 an 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:

$$t < \rho < a + \tau \tag{3}$$

$$m < \delta \tag{4}$$

These constraints are justified in Section VI.

Once C and D have agreed on the terms of the RP, we must ensure that a domain who purchases an 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 VIII).

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 an 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 an 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.

VI. 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.

A. Model

We begin by analyzing the payments that occur within the model described by the constraints in the previous sections. Table IV 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

TABLE IV
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	$E - t_D$

- 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:

- *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.

B. 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 V shows the results for this scenario, that is, how much is paid out to the involved entities, according to Table IV, 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 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

$$\rho < a + \tau \tag{5}$$

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

$$m < \delta \tag{6}$$

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 5, which sets a tighter bound on ρ.

TABLE V
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

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 2, we simply let

$$\rho > t \tag{7}$$

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.

C. 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 an 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 V shows the results for the external scenario, that is, how much is paid out to the involved entities, according to Table IV, aggregated into three cases as above.

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 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 an 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 IV), so that the detector's expected reward is zero (see Table V).

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 5, Equation 6, and Equation 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 IX-B.

VII. Blockchain 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 [21].

A. Blockchain Principles and Bitcoin

At a high level, decentralized cryptocurrencies such as Bitcoin [67] 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 [1]. 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 [68] 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 [15]. 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 [32]. 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 [37, 40, 70, 79]. 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.

B. Ethereum

Ethereum [87] 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.

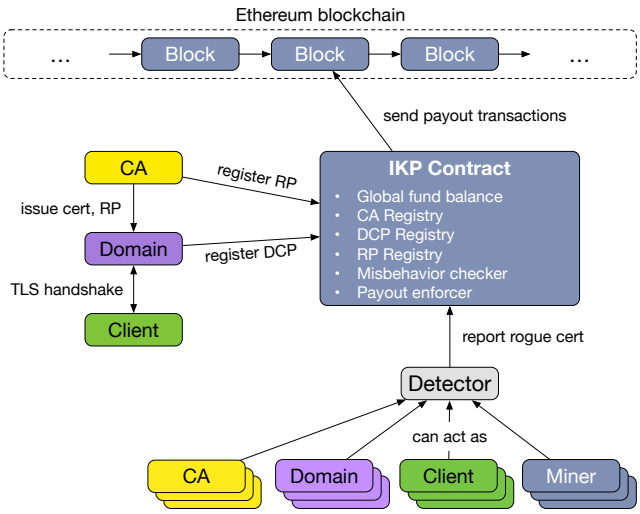


Fig. 5. IKP architecture in our Ethereum instantiation.

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 [43]. Ethereum thus offers the possibility of decentralized trusted entities, a feature not possible in Bitcoin.

VIII. IKP in Ethereum

We now describe our instantiation of IKP in Ethereum, whose architecture is shown in Figure 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:

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.

A. 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 I 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 an RP. Because Ethereum requires each entity to pay the gas costs of the computation that results from the transaction, the incentives discussed in Section VI 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.

B. 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 an RP or add more than is necessary

Algorithm 1 IKP contract handling a misbehavior report.

```
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
```

when issuing an 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 an 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 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.

C. 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 [69] to report misbehavior: a detector d first sends a "pre-report" containing the reporting fee and a commitment hash $H(C\|s)$ to the IKP contract, where 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 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 C is until d opens the commitment. Because reporting misbehavior requires waiting for a set number of blocks, frontrunning is not possible.

TABLE VI
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

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 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.

IX. 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 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.

A. 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 [2] 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 [18]. 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 [19] and is currently planned for future versions of Ethereum [23].

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

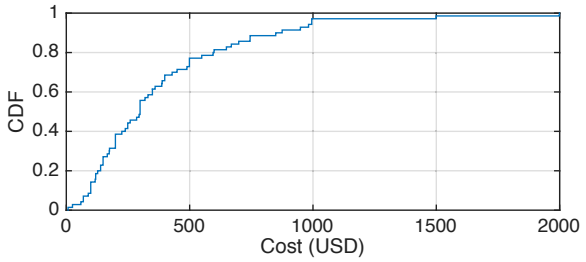
Table VI shows the costs of various operations in gas and USD. We observe that by far the highest cost in the system is for checking a bootstrap proof. Much of this cost comes from simply handling data that is the size of a standard 2048-bit certificate, since we can also see that the cost of verifying 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.

B. CA Certificate Offerings

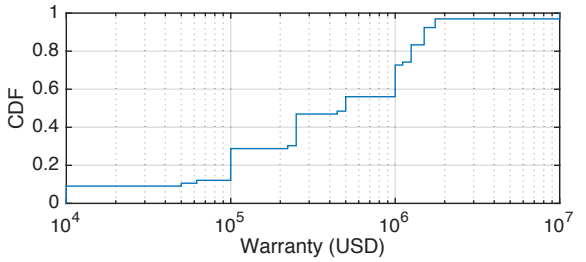
To get an estimate of sample RP payout values, we collected data from the most popular CAs. In particular, we examined each of the standard TLS certificate offerings of the 20 CAs with a market share of at least 0.1%, representing 99.9% of all TLS certificates on the Web [10]. For each certificate, we 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 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 VII, the risk for each certificate varied widely, ranging from around 0.001% up to almost 8.5%.

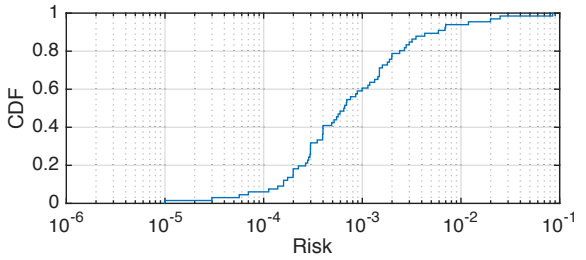
To set sample RP values, we can conservatively estimate the risk of a CA to be 10%; thus the affected domain payout could be 10 times the RP cost. Using the median cost of a certificate as a reference, we can estimate that a standard RP will cost $\rho = \$299$, and thus $a = \$2990$. Similarly, we can 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 VI are easily satisfiable, for



(a) Distribution of 1-year certificate costs.



(b) Distribution of certificate warranty amounts.



(c) Distribution of calculated risk amounts.

Fig. 6. Empirical CDFs of certificate costs, warranties, and assessed risks from the most popular CAs [10].

TABLE VII
RISK UPPER-BOUNDS INFERRED FROM CA CERTIFICATE AND WARRANTY AMOUNTS (IN US DOLLARS) FROM CA WEBSITES.

CA	Certificate	Cost	Warranty	Risk
Highest-Risk				
GlobalSign [5]	Wildcard	$849	$10,000	8.49e−2
GlobalSign	DomainSSL	$249	$10,000	2.49e−2
StartCom [9, 71]	Ext. Validation	$199	$10,000	1.99e−2
StartCom	Org. Validation	$119	$10,000	1.19e−2
Entrust [7]	Wildcard	$699	$100,000	6.99e−3
...	...	...	...	...
Certum [3]	Commercial SSL	$25	$222,000	1.13e−4
Starfield [8]	Standard SSL	$7	$100,000	7.00e−5
Comodo [4]	EV SSL	$99	$1,750,000	5.66e−5
IdenTrust [6]	Multi Domain SSL	$299	$10,000,000	2.99e−5
IdenTrust	Standard SSL	$99	$10,000,000	9.90e−6
Lowest-Risk				

example, $\rho = \$299$, $a = \$2990$, $t = \$150$, and $\delta = \$100$.

X. 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 [22], allowing double-spending attacks and suppression of selected transactions. Section XI describes attacks that can be mounted with less than half of the network's hashrate. Moreover, there may be bugs in the IKP contract which could result in exploits such as the one that plagued the DAO in Ethereum [24], and check and reaction contracts may have bugs as well. Learning to write secure contracts is difficult [29], but we can build on existing work such as smart contract formalization [45] 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 [34]. Moreover, CAs can use IKP to prove a history of good behavior, attracting more business.

Protecting Clients. In this paper, 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 [87]. 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 [88] 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 [16, 73] to formally verify the incentive structure of IKP.

XI. Related Work

In this section, we discuss work related to IKP. In particular, we cover four main areas: log-based PKIs enhancements, alternatives to CA-based PKIs, incentives on blockchains, and insurance schemes.

A. Log-Based PKI Enhancements

Log-based PKI enhancements provide an alternate approach to deterring CA misbehavior. They leverage high-availability servers called *public logs* that maintain append-only databases of certificates issued by CAs. Logs maintain Merkle hash trees [63], which allow a log to provide efficient proofs that a certificate is present in the log and that no previously recorded certificates have been tampered with or deleted [28, 54]. These proofs are sent to a client along with a domain's certificate to show that a log has recorded the certificate, ensuring that an adversary attempting to use an unauthorized certificate has exposed it to the public. *Monitors* can then watch logs for suspicious certificates and report any instances of suspected misbehavior.

The core idea of log-based PKI enhancements is that by ensuring certificates are publicized, misbehavior can be quickly detected, thus deterring CAs from issuing unauthorized certificates. Such exposure can also help detect unauthorized certificates issued by accident [80]. Most log-based PKI enhancements rely on the domain to take action against unauthorized certificates [54], since only the domains themselves know which certificates are authorized. Other approaches require the domain to publicize policies used to determine which of its certificates are authorized [44, 84].

Certificate Transparency (CT) [54] was the first to propose the use of public logs in their current form, though earlier proposals such as Sovereign Keys [33] used similar entities. However, CT provides no support for revocation, nor does it

provide any information as to whether the logged certificates are authorized. Revocation Transparency [53] and CIRT [78] both provide mechanisms to enable revocation checking in public logs. AKI [44] embeds policies into certificates that enable recovery from private key loss or compromise, and uses a checks-and-balances system among clients, domains, CAs, logs, and validators (who monitor logs) to detect and report misbehavior. ARPKI [17] presents a formally-verified extension of AKI that provides stronger security guarantees. PoliCert [84] separates policies from certificates and supports multiple certificates per domain, hierarchical policies that apply to all subdomain certificates, and domain-specified error handling. While the idea of policies inspired DCPs in IKP, no log-based PKI enhancement offers incentives for correct behavior or automatic responses to misbehavior.

B. Alternatives to CA-based PKIs

Some previous approaches have also sought to diminish or eliminate the role of CAs by providing authenticity through other sources. For example, DANE [41] allows domains to place public keys or certificates in DNSSEC [13], but does not preclude CAs. Additionally, the security of DNSSEC inherently relies on a PKI of its own roots at ICANN, which is a single point of failure for the system and has not been widely deployed. Public key pinning schemes such as Chrome's HTTPS pin [48], HPKP [35], or TACK [58] store information about a domain's public key at the client browser. Perspectives [86] and Convergence [57] leverage the public keys observed by notary servers throughout the Internet to detect MitM attacks. However, in both of these approaches, it is difficult to determine whether a domain has legitimately changed its key or if a MitM attack is taking place, since the domain does not provide any other information such as a DCP to characterize its certificates.

Other work has sought to move PKI functionality onto the blockchain. For example, Blockstack [11] (formerly OneName) leverages the Bitcoin blockchain to provide a name registration service that also allows entities to bind public keys to their names. However, Blockstack uses its own namespace and a pricing rule based on the name length and the presence of nonalphabetic characters, and does not attempt to secure names that exist in today's DNS. Certcoin [38] leverages Namecoin [68] to implement a blockchain-based PKI, storing identity information in a Merkle hash tree and using the Kademlia DHT [61] for fast lookup. However, Certcoin does not protect existing names, and does not provide any recoverability for identities that are falsely claimed on the blockchain. EthIKS [20] does not implement a PKI on its own, but rather uses the Ethereum blockchain to audit a centralized key server for CONIKS [62]. However, neither EthIKS nor CONIKS provides any means for responding to equivocation or other misbehavior by key servers.

C. Blockchain-based Incentives

Most previous studies of incentives in blockchains have been concerned with the incentives of mining. The miner's dilemma [36], for example, analyzes the mining pools' game-theoretic incentives to infiltrate and attack one another. The selfish mining attack [37] shows that mining in the Bitcoin network is not incentive compatible. Subsequent work further improves on the strategy [79] and demonstrates that composing with network attacks such as the eclipse attack [40] can increase the revenue of selfish mining with less than half of the network hashing power [70]. These works are orthogonal to IKP, focusing on the incentives of blockchain consensus rather than of applications built on the blockchain.

Other work has examined incentives that can be built on top of the blockchain. For example, Andrychowicz et al. examined incentives to ensure the security of multi-party computation in the Bitcoin blockchain [12]. They showcase the feasibility of timed commitments in Bitcoin as well as a lottery protocol. Kumaresan and Bentov examined incentivization for verifiable computation and proposed a mechanism to non-interactively reward bounties for solving hard problems [46]. The presented approach, however, is impractical as it suffers from the limitations of Bitcoin's language *script* and from the hybrid model that relies on ideal functionalities, which are implemented through costly garbled circuits and zero-knowledge proofs. While support for zero-knowledge proofs is planned for Ethereum [25], we instead focus on the incentives of a TLS-like PKI built on the blockchain.

D. Insurance Schemes

Even before cryptocurrencies, the idea of electronic insurance policies were used to evaluate services in distributed systems [47]. The idea of insurance was also proposed as an example of an authentication metric that followed good design principles [76]. However, both of these proposals offer little accountability and cannot be effectively realized without cryptocurrencies. Certificates-as-an-Insurance (CaaI) was the first to propose the idea of integrating insurance into TLS certificates as a means of balancing CA control and liability, but only presented challenges and principles for desgning such a system [59]. Our work on IKP adds a cryptocurrency-based instantiation of the CaaI model as well as proofs of incentivization compared to log-based PKIs.

XII. Conclusions

In this paper, we proposed IKP, a platform for reporting unauthorized certificates and responding to CA misbehavior in an automated and incentivized fashion. We described the full process from registering a CA to claiming reaction payouts. We developed a model describing reaction payouts, which helped us discover the constraints to guide the negotiation of reasonable reaction policies. Finally, we discussed the deployability incentives in today's Internet and created an decentralized instantiation of IKP based on Ethereum. Our work does not stop all misbehaving CAs, nor does it always enforce accountability on CAs that are misbehaving. We observe, however, an urgent need to incentivize good CA behavior in this way in order to make TLS more secure, and we argue that IKP is a first concrete step towards that goal.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers, Virgil Gligor, Maverick Woo, and Bryan Parno for their comments on drafts of this paper. This work was supported in part by NSF Grant DGS1252522.

REFERENCES

[1] "Script," Bitcoin Wiki, https://en.bitcoin.it/wiki/Script, October 2014.
[2] "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.
[3] "Certum by Asseco," https://en.sklep.certum.pl/data-safety/ssl-certificates.html, July 2016.
[4] "Comodo," https://ssl.comodo.com/ssl-certificate.php, July 2016.
[5] "GlobalSign SSL," https://www.globalsign.com/en/ssl/, July 2016.
[6] "IdenTrust SSL," https://www.identrustssl.com/buy.html, July 2016.
[7] "SSL certificate comparison," https://www.entrust.com/ssl-certificate-comparison/, July 2016.
[8] "Starfield technologies," https://www.starfieldtech.com/, July 2016.
[9] "StartCom," https://www.startssl.com/, July 2016.
[10] "Usage of SSL certificate authorities for websites," https://w3techs.com/technologies/overview/ssl_certificate/all, July 2016.
[11] 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.
[12] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on Bitcoin," in *IEEE Symposium on Security and Privacy (S&P)*, May 2014, pp. 443–458.
[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," RFC 4033, March 2005.
[14] 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.
[15] A. Back, "Hashcash: A denial of service counter-measure," http://www.cypherspace.org/adam/hashcash/, August 2002.
[16] 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.
[17] 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*. ACM, 2014, pp. 382–393.
[18] A. Beregszaszi, "RSA signature verification in Ethereum," https://github.com/axic/ethereum-rsa, April 2016.
[19] ——, "Support RSA signature verification," https://github.com/ethereum/EIPs/issues/74, March 2016.
[20] J. Bonneau, "EthIKS: Using Ethereum to audit a CONIKS key transparency log," in *3rd Workshop on Bitcoin and Blockchain Research (BITCOIN)*, February 2016.
[21] 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.
[22] V. Buterin, "On mining," Ethereum Blog, June 2014.
[23] ——, "Understanding Serenity, part I: Abstraction," https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/, December 2015.
[24] ——, "Critical update re: DAO vulnerability," Ethereum Blog, June 2016.
[25] ——, "Privacy on the blockchain," https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/, January 2016.
[26] Comodo, "Comodo fraud incident 2011-03-23," https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, March 2011.
[27] 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.
[28] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *USENIX Security Symposium*, August 2009, pp. 317–334.
[29] 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.
[30] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.2," RFC 5246, August 2008.
[31] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 542–553.
[32] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology (CRYPTO '91)*, August 1992, pp. 139–147.
[33] P. Eckersley, "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.
[34] L. Encrypt, "Let's encrypt," https://letsencrypt.org.
[35] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," RFC 7469, April 2015.
[36] I. Eyal, "The miner's dilemma," in *IEEE Symposium on Security and Privacy (S&P)*, May 2015, pp. 89–103.
[37] I. Eyal and E. G. Sirer, "Majority mining is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security (FC)*, 2014.
[38] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," Cryptology ePrint Archive, Report 2014/803, November 2014.
[39] Google, "HTTPS usage," https://www.google.com/transparencyreport/https/metrics/, 2016.
[40] 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.
[41] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA," RFC 6698, August 2012.
[42] 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.
[43] C. Jentzsch, "Decentralized autonomous organization to automate governance," White paper, November 2016.
[44] 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.
[45] 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 2015.
[46] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2014.
[47] C. Lai, G. Medvinsky, and B. C. Neuman, "Endorsements, licensing, and insurance for distributed system services," in *Proc. of the 2nd ACM Conference on Computer and Communications Security*, 1994.
[48] A. Langley, "Public key pinning," https://www.imperialviolet.org/2011/05/04/pinning.html, May 2011.
[49] ——, "Revocation checking and Chrome's CRL," https://www.imperialviolet.org/2012/02/05/crlsets.html, February 2012.
[50] ——, "Enhancing digital certificate security," http://googleonlinesecurity.blogspot.com/2013/01/enhancing-digital-certificate-security.html, January 2013.
[51] ——, "Further improving digital certificate security," http://googleonlinesecurity.blogspot.com/2013/12/further-improving-digital-certificate.html, December 2013.
[52] ——, "Maintaining digital certificate security," http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html, March 2015.
[53] B. Laurie and E. Kasper, "Revocation transparency," http://www.links.org/?p=1272, September 2012.
[54] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," RFC 6962, June 2013.
[55] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's PKI," in *ACM Internet Measurement Conference (IMC)*. ACM, 2015, pp. 183–196.

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

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

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

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

[60] ——, "IKP: Turning a PKI Around with Blockchains," Cryptology ePrint Archive, http://eprint.iacr.org/2016/1018, 2017.

[61] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[62] 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.

[63] R. C. Merkle, "A digital signature based on a conventional encryption function," *Advances in Cryptology (CRYPTO '87)*, pp. 369–378, 1988.

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

[65] ——, "Improperly issued digital certificates could allow spoofing," Microsoft Security Advisory 3046310, March 2015.

[66] E. Mills and D. McCullagh, "Google, Yahoo, Skype targeted in attack linked to Iran," http://www.cnet.com/news/google-yahoo-skype-targeted-in-attack-linked-to-iran/, Mar. 2011.

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

[68] Namecoin, "Namecoin," http://namecoin.info.

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

[70] 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)*, 2016.

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

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

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

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

[75] 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.

[76] 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.

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

[78] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail," in *Network and Distributed Security Symposium (NDSS)*. NDSS, February 2014.

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

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

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

[82] 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.

[83] 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.

[84] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible TLS certificate management," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.

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

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

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

[88] 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.