# IKP: Turning a PKI Around with Blockchains

Stephanos Matsumoto[1,2] and Raphael M. Reischuk[2]

[1] Carnegie Mellon University
[2] ETH Zurich
smatsumoto@cmu.edu, reischuk@inf.ethz.ch

**Abstract.** Man-in-the-middle attacks in TLS due to compromised CAs have been mitigated by log-based PKI enhancements such as Certificate Transparency. However, these log-based schemes do not offer sufficient incentives to logs and monitors, and do not offer any actions that domains can take in response to CA misbehavior. We propose IKP, a blockchain-based PKI enhancement that offers automatic responses to CA misbehavior and incentives for those who help detect misbehavior. IKP's decentralized nature and smart contract system allows open participation, offers incentives for vigilance over CAs, and enables financial recourse against misbehavior. We demonstrate through a game theoretic model and through an Ethereum prototype implementation that the incentives and increased deterrence offered by IKP are technically and economically viable.

## 1 Introduction

Encrypted client-server communications in the Internet are secured using Transport Layer Security (TLS) [27] and have become increasingly common. This increase is in part due to the growth in sensitive information shared between users and services, the disclosure of global state-level surveillance [19, 35, 38], and the advent of services such as Let's Encrypt [31], which has issued over 5 million TLS certificates in the first 7 months since its launch in December 2015 [11]. The security of the TLS public-key infrastructure (PKI) largely relies on certificate authorities (CAs), who make a business out of certifying the authenticity of sites' public keys.

However, the CA ecosystem is fragile and prone to compromises and operational errors. These failures have occurred all around the world, including the US [24, 58], France [48], the Netherlands [41], Turkey [47], and China [49]. 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 [73, 77]. Thus while CAs play a critical role to the security of the TLS ecosystem, they have failed in this role time and again, issuing unauthorized certificates by mistake or even as a business [68]. 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 [59, 60].

To address the problem of misbehaving CAs, *log-based PKIs* have proposed to use highly-available public log servers that monitor and publicize the certificates issued by CAs. The certificates are not considered valid until they have been logged in this way, thus quickly exposing and thus deterring the issuance of unauthorized certificates. Google's Certificate Transparency (CT) [51] is the most widely deployed log-based PKI, currently available in both Chrome and Firefox. Other proposals add features like support for revocation [42, 71], error handling [78], and formally verified security guarantees [16] to log-based PKI. Unfortunately, despite these benefits, log-based PKIs suffer from several problems.

First, **log-based PKIs require a centralized, consistent source of information to operate securely**. Several log-based PKIs require each domain name to be associated with a single certificate or policy, and thus require all logs to periodically synchronize [42, 78], but do not describe a protocol for doing so securely. Without this synchronization, these PKIs could not determine whether or not a certificate was authorized for a particular domain. In CT, Google specifies a list of authorized logs [2], which must conform to certain operational standards such as availability and consistency [1]. Logs in violation of this policy have been removed for security and efficiency purposes [74–76]. However, without this list of logs and gossiping [23, 36], logs can remain unavailable, violate consistency, or equivocate to clients, undermining security.

Second, **log-based PKIs do not sufficiently incentivize recording or monitoring CA behavior**. The cost of operating a log seems to outweigh the benefits for the log operator, as indicated by a low willingness among CAs to deploy a log [37], the fact that several CA-operated logs were required to deploy logs in response to security incidents [49, 73], and the fact that one public log run by an individual ceased operation due to the costs of maintaining the server [57]. We note that logs are not financially compensated for contributing to the security of any log-based PKI despite their crucial role.

No compensation is given to those who discover unauthorized certificates, either. As a result, domains benefit from monitoring the logs for their own unauthorized certificates, but benefit little from reporting other unauthorized certificates. In particular, we observe that Google's incident reports in CT always include an unauthorized certificate for a Google product. To our knowledge, few other domains monitor the logs in this way, which diminishes the logs' usefulness [67].

Finally, **responding to CA misbehavior takes time and requires manual effort.** When a CA misbehaves, a domain has three options. First, the domain can contact the misbehaving CA to have the certificate revoked, until which MitM attacks against the domain are possible. However, while the domain suffers the risk of MitM attacks, the CA is the only one who can revoke the certificate, and if the CA is malicious, the certificate may never be revoked. Second, the domain can contact the browser vendors, who can update client browsers such that they do not accept the certificate [46] or in some cases, the CA itself [66]. However, this type of response only occurs if the offending certificate is for a

popular site. Third, the domain can pursue legal action against the CA. However, this process can be long, costly, and ultimately unsuccessful, due in part to the fact that CAs are located in approximately 52 countries [30].

Therefore, in this paper we ask: **how can we decentralize, incentivize, and automate the handling of CA misbehavior?** In particular, what does it mean for a CA to misbehave? How can we check if a certificate indicates CA misbehavior? In what ways can we automatically respond to CA misbehavior? How can we ensure that these responses actually execute?

As a first step towards answering these questions, we present Instant Karma PKI (IKP), an improvement to the TLS PKI. IKP relies on a decentralized entity that handles the definition and evaluation of CA misbehavior and automatically executes pre-defined reactions to this misbehavior. This entity provides financial incentives to CAs operating correctly, as well as to those who report unauthorized certificates. In this paper, we use Ethereum [81] to instantiate IKP, as the Ethereum blockchain is decentralized and provides natural financial incentives and a transaction framework to participants. Moreover, Ethereum's smart contracts execute autonomously, enabling automatic reaction to CA misbehavior. While other approaches may also be able to provide decentralization, automation, and financial incentives, we use Ethereum in the rest of the paper to provide a easy-to-follow and concrete description of IKP, as well as an evaluation of our realization.

In summary, we make the following contributions:

- We design and propose IKP, including frameworks for domain policies and reactions to CA misbehavior.
- We demonstrate through a game theoretic analysis that the system incentivizes good CA behavior and punishes misbehavior.
- We analyze the risk of IKP CAs using real-world data from existing CAs.
- We implement a prototype in Ethereum and discuss the present and future technical feasibility of IKP.

## 2 Background

In this section, we provide the background required to understand IKP. In particular, we discuss log-based PKIs and the CA ecosystem, as well as the Ethereum blockchain.

**Log-Based PKIs.** Log-based PKIs leverage high-availability servers called *public logs* that maintain append-only databases of certificates issued by CAs. These databases provide efficient proofs of a certificate's presence in a log [51] and of the log's temporal consistency [26]. These proofs are sent with a domain's certificate to prove 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 PKIs is that by ensuring that 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 [73]. Most log-based PKIs rely on the domain to take action against unauthorized certificates [51], since only the domains themselves know which certificates are authorized. Other approaches require the domain to publicize information about which of its certificates are authorized [42, 78].

**Blockchain-Based Cryptocurrencies.** The advent of decentralized cryptocurrencies such as Bitcoin [61] has brought about a number of *blockchain-based systems* that leverage a public ledger created and maintained through network consensus. While most cryptocurrencies use the ledger solely to track financial transactions, other proposals (e.g., Namecoin [62]) store name-value entries similar to those of DNS. Ledgers are most commonly implemented as a *blockchain*, a chain of *blocks* linked by hash pointers to the previous blocks.

The blockchain grows 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]. Using a cryptographically secure hash function requires a brute-force search to find $v$, making mining a proof-of-work scheme [29]. A node or *miner* is incentivized by the *block reward*, which transfers currency to whomever extends the blockchain by recording the new transactions and finding $v$, and then broadcasts the new block.

The security of blockchain-based cryptocurrencies relies on the fact that no entity controls a majority of the hashing power of the network (called a *51% attack*). Otherwise, that adversary can gain control of the cryptocurrency network, reversing its transactions to others and suppressing the transactions of others. Blockchains make the assumption that such an attack is impossible.

Rather than storing account balances and transactions in the blockchain, Ethereum [81] generalizes this notion to store *arbitrary state* in the blockchain. Ethereum's currency is used not only for ordinary transactions but also to activate executable code that manipulates the blockchain state. Code is stored in Ethereum *contracts*, autonomous accounts that run their code upon receiving a transaction. Miners perform the computation and mine new blocks with the resulting state. The language defined in Ethereum is Turing-complete, allowing arbitrary computation in the blockchain. To prevent malicious code from wasting computational resources, message senders must pay additional funds called *gas* that compensate miners for their computational and storage costs.

For further details on all issues related to blockchains, we refer readers to a more complete view of decentralized cryptocurrencies [18].

## 3 Problem Definition and Adversary Model

In a nutshell, our goal is to automate responses to CA misbehavior in a way that incentivizes correct CA behavior (*i.e.*, due diligence when issuing certificates) and does not require a centralized trusted entity. In particular, we aim to design a system that can 1. *define* CA misbehavior, 2. *evaluate* whether a given certificate constitutes misbehavior, 3. *specify* reactions that will occur in response

to CA misbehavior, and 4. *execute* a reaction after a CA has misbehaved. The automation of this process thus deters CA misbehavior. In addition, being able to define and evaluate CA misbehavior provides a framework to protect clients from accepting unauthorized certificates, as in existing log-based PKIs.

We additionally require our system to 1. *record* CA behavior (though not necessarily on the blockchain), 2. *report* misbehavior using information from entities monitoring the recorded information, and 3. *reward* entities that record or report CA misbehavior. Rewarding entities for recording CA operations and reporting misbehavior provides incentives for such actions, increasing the number of entities monitoring CAs and thus the probability that an unauthorized certificate is quickly detected. Decentralizing and automating this process guarantees that these entities will easily obtain their reward.

By providing the above functions, a decentralized, automated, and incentive-driven system can provide comparable security to that of log-based PKIs and further offer incentives for CAs to behave correctly, as well as for other entities to monitor CAs to ensure correct behavior.

### 3.1 Desired Properties

A system addressing the above problems should have at least the following properties:

- **Auditability**: all information required to detect an unauthorized certificate is publicly accessible.
- **Efficiency**: once CA misbehavior has been detected, reactions should automatically proceed without additional action.
- **Incentivization**: entities that behave correctly are financially rewarded.
- **Deterrence**: any misbehaving entity or set of entities faces financial penalties.

Achieving auditability ensures that any entity who wants to review CA operations and detect CA misbehavior can access the necessary information to do so. By opening detection of misbehavior to the public, then, we can provide open deployment as well as make misbehavior detection more likely. Efficiency ensures that reactions to misbehavior can be automatic, proceeding without any additional necessary actions. Incentivization, particularly those offered by design in the system, ensures that entities will deploy without influence from a centralized initiator. Finally, deterrence ensures that the rate of unauthorized certificates decreases, thus leading to a more secure TLS PKI.

### 3.2 Adversary Model

Our adversary's goal is to issue a rogue certificate while maintaining a positive expected return on investment (ROI). The adversary controls one or more CAs (and can thus issue arbitrary certificates from these CAs), as well as colluding domains and logs. The adversary may take whatever actions it wants 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

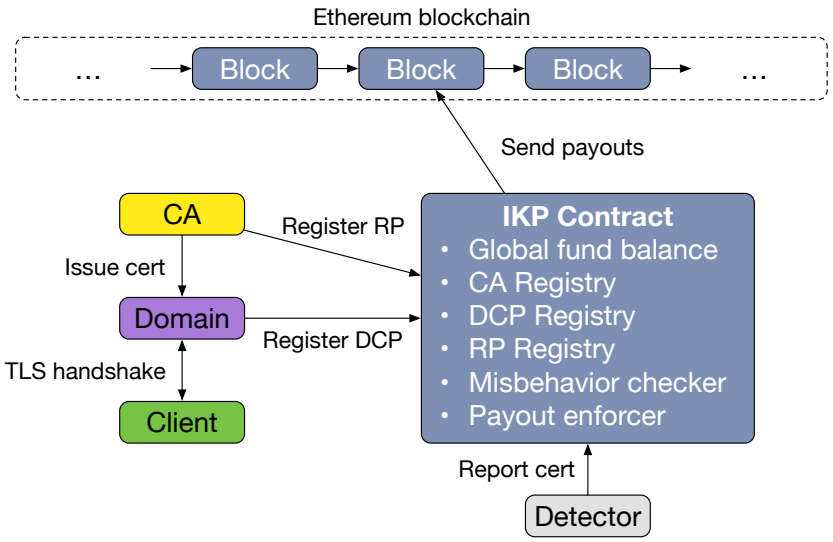


**Fig. 1.** Overview of the entities and functions in IKP.

or forging digital signatures. The adversary also cannot compromise the private keys of arbitrary domains. Given that our solution leverages a blockchain, we also assume that the adversary cannot control a majority of hashing power in the blockchain network.

## 4 IKP Overview

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

### 4.1 Architecture

As shown in Figure 1, IKP extends the standard TLS architecture with the Ethereum blockchain. As in TLS, CAs issue certificates to domains, who carry out TLS handshakes with clients. However, CAs can also issue *reaction policies (RPs)*, which take effect if an unauthorized certificate for a domain is issued, acting as a sort of "insurance policy" against CA misbehavior. Domains can publicize criteria for their certificates through *domain certificate policies (DCPs)*, which provide a way to computationally determine whether a given certificate is authorized or not for a domain.

To summarize, IKP has five main entities:

1. **CAs**, which issue certificates and RPs to domains. To join IKP, they register their information in the Ethereum blockchain.

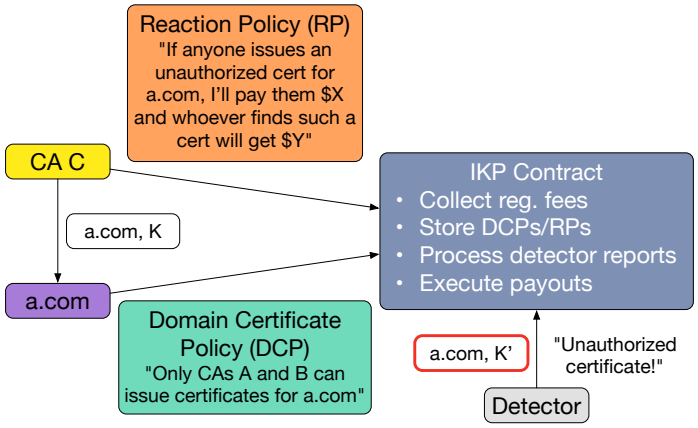


**Fig. 2.** Sample interactions between entities in IKP. As in Figure 1, yellow denotes a CA and purple denotes a domain.

2. **Domains**, which purchase certificates and RPs from CAs. To join IKP, they register DCPs in the blockchain.
3. **Clients**, who perform TLS handshakes with domains, accepting or rejecting the presented public-key certificates based on information publicized by browser vendors or in the blockchain.
4. The **IKP contract**, which contains the logic to carry out the functions described in Section 4.2, including registering CAs and domains. The IKP contract also maintains its own balance used to escrow funds and to provide rewards. The contract exists exactly once in IKP and is responsible for all CAs, domains, and clients.
5. **Detectors**, who monitor CA operations for suspicious certificates (much like log monitors in CT) and report any certificates they deem to be unauthorized.

IKP also has additional entities (such as browser vendors and miners), but the roles of these entities are identical to those of their counterparts in TLS or Ethereum respectively and thus we do not discuss them here.

### 4.2 Operation

To enable the operations described above for each entity in IKP, the IKP contract handles the various functions critical to the operation of IKP (some of which are shown in Figure 2):

- **CA registration**: a CA registers its information, such as its Ethereum address and its public key used to verify its digital signatures, with the IKP contract.
- **Domain registration**: a domain registers its DCP, which includes its domain name, its Ethereum address, and the address of the contract whose

code is run to determine whether or not a certificate is authorized, with the IKP contract.
- **RP negotiation**: a domain agrees to purchase a certain RP from a CA, with the IKP contract acting as a facilitator and escrow mechanism.
- **Certificate purchase**: a domain obtains a certificate from a CA. The CA may be the same one that issued the domain's RP, or it may be a different CA.
- **Misbehavior report**: a detector sends an unauthorized certificate (which indicates CA misbehavior) to the IKP contract, which checks the certificate against the domain's DCP and triggers the appropriate RP if the certificate is indeed unauthorized.
- **Reaction**: if an unauthorized certificate is detected for a domain, the IKP contract triggers the reaction specified in the domain's RP.

DCPs and RPs are simply contract accounts in Ethereum, allowing them to be both expressive and extensible. As we describe in Section 5 and Section 6, DCPs can provide features such as CA whitelisting, public key pinning, and short-lived certificate enforcement, while RPs can provide reactions such as financial payouts and CA revocation.

## 5 Domain Certificate Policies (DCPs)

In this section, we take an in-depth look at DCPs. In particular, we describe the features and format of DCPs, and present several examples of DCPs that enable various useful defenses against CA misbehavior.

### 5.1 Design Principles

We begin by describing the fundamental principles on which we base our design for DCPs. In particular, we identify three main design principles: 1) domain-specified information, 2) blockchain-based storage, and 3) policy expressiveness. 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 information.** The information used to determine certificate authorization is specified by that domain itself. We observe that only domains know for certainty which certificates they have and have not authorized. Therefore, domains must play the crucial role of publicizing the information required for determining certificate authorization. Whereas most log-based PKIs publicize information to allow domains to detect unauthorized certificates, IKP publicizes information from the domains themselves to allow any entity to detect unauthorized certificates with certainty.

**2) Blockchain-based storage.** The information used to determine certificate authorization lives in the blockchain, or can be accessed by information (e.g., pointers) found in the blockchain. This approach contrasts with previous proposals, which store such information in public logs, in several ways. First, the

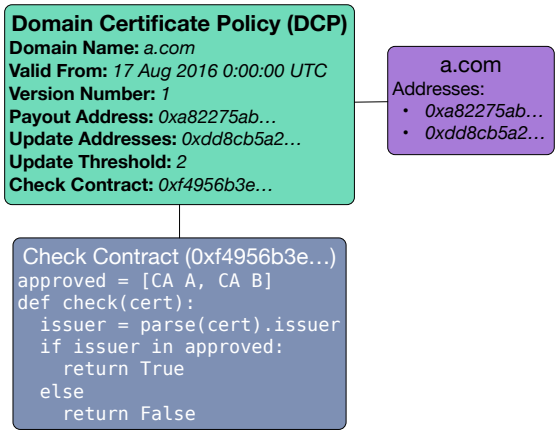


**Fig. 3.** A sample DCP with a check contract written in pseudocode.

blockchain provides consistency through Nakamoto consensus, avoiding the problem of globally synchronizing logs and resolving conflicts among them (to ensure that a domain has one policy active at a given time). Additionally, the mining process on the blockchain (which occurs regardless of whether IKP is deployed) provides natural financial incentives for miners to keep up-to-date, correct information. Finally, the peer-to-peer nature of a blockchain network can maintain better availability than a centralized public log.

**3) Policy expressiveness.** The information used to determine certificate authorization is able to represent a wide range of policies. While some log-based PKIs allow domains to specify policies 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 (for major changes) existing domain policies. IKP provides a general format for DCPs that simply specify a contract address to determine certificate authorization. This generality allows domains to reuse existing contracts for their DCPs or define their own; the range of policies is only restricted by the underlying blockchain language.

### 5.2 DCP Contents

We now describe the contents and format of DCPs. Figure 3 shows the format of a sample DCP, and Table 1 describes the fields of a DCP. In short, a DCP contains the basic information necessary to identify the domain, including its DNS name and Ethereum addresses, and the specific DCP by its `Valid From` and `Version Number` fields. A DCP also contains policy information, namely the number of signatures required to authorize changes to the DCP and how a certificate is deemed to be authorized or not.

The `Valid From` and `Version Number` fields of a DCP are used in part to determine whether or not a certificate constitutes CA misbehavior. In particular,

**Table 1.** 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 |
| Payout address | authenticate and receive payments for domain |
| Check Contract | address of the DCP's check contract |
| Update addresses | (default empty) authorize DCP updates |
| Update threshold | (default 1) threshold of payout/update addrs. for DCP updates |

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 began after the DCP's `Valid from` time, (2) the RP's `Version number` field matches that of the DCP, and (3) the check contract deems the certificate to be unauthorized (as described below).

The *check contract* of a DCP is simply an Ethereum contract account that provides a method `check` that takes in the serialized bytes of a public-key certificate (assumed in this paper to be X.509) and returns a boolean value representing whether or not the certificate is authorized. Using contract accounts to perform a certificate authorization check provides the generality necessary for domains to specify a wide range of certificate policies.

Finally, the update addresses and update threshold protect a domain that loses control of its payout address due to loss or compromise of its corresponding private key. Because blockchains do not offer any mechanisms to recover a lost or compromised private key, such an event can result in permanent loss of control over a domain's DCP. To prevent this lockout, we allow a domain to specify a set of addresses (called *update addresses*) from which the domain can authorize updates to its DCP. Moreover, a domain can also specify an *update threshold*, a minimum number of addresses required to authorize a DCP update. The addresses updating the DCP may be the payout address or any of the update addresses of the DCP.

We note that our recovery system is not foolproof, as the domain must maintain control over a threshold number of its addresses, but our approach provides a way for a domain to recover from the loss or compromise of one or more of its private keys, and the domain can tune this level of security to its liking. The domain can even store some of its private keys offline or with trusted peers in order prevent mass loss of compromise of its private keys.

### 5.3 Expressiveness of Check Contracts

We now present a range of possible check contracts in IKP. In particular, we present five simple modules representing criteria that domains may use in their check contracts and explain how each would work. We then present a modular approach for combining multiple modules into a single check contract.

In order to specify modules based on fields of a certificate such as the issuer or public key, we can write contract code that parses a certificate into its constituent fields. We note that this approach allows each check contract to handle its own certificate format or even multiple formats. Thus many certificate formats can coexist in IKP DCPs. In this paper, we use the X.509 v3 format [25] in our check contracts.

**CA whitelisting.** A check contract 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 a specific set of CA signing keys, the check contract can instead extract the `Authority Key Identifier` extension for X.509 and check the identifier against a whitelist. While the whitelist itself must be included in the check contract, we can take a modular approach for checking the certificate issuer or authority key identifier against the whitelist by encoding this logic in a contract.

**Public key pinning.** A check contract 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 checking logic itself can be outsourced to an existing contract account, leaving the check contract to simply specify the whitelist.

**Short-lived certificates.** A check contract can enforce the use of short-lived certificates [79] 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 check contract can prevent the use of wildcard certificates by extracting the `Subject Name` field and checking that the wildcard character `*` does not appear.

**Certificate Transparency.** A check contract can implement criteria similar to those of Certificate Transparency by requiring proof that the certificate has been logged. To check that a certificate has been logged, the check contract must be able to process both SCTs (which prove that a certificate was received by the log) and log proofs (which prove that a certificate is in the log). The contract must also list one or more logs from which it accepts proofs, or rely on an authenticated list of approved logs, such as that of Google.

**Combining check contracts.** A domain can use a combination of existing check contracts by creating a check contract which in turn calls the `check` method of each of the existing contracts. A domain can specify that all criteria in the called check contracts must be fulfilled by using the AND Boolean relation, or specify that some subset of the criteria must hold by using the OR relation. A domain can thus create a check contract whose `check` method relies on other contracts.

## 6 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 reaction policies. We then describe the contents of reaction policies and present the two major types of reaction contracts: client-facing contracts and payout contracts.

### 6.1 Design Principles

We begin by describing the design principles upon which we base our design of RPs. In particular, we identify three main design principles for reaction policies: 1. certificate-independence (i.e., reactions should be defined independently from certificates), 2. policy-adherence (i.e., reactions should be based on specific domain policies), and 3. single-use (i.e., reactions should be executed at most once). These principles help ensure that reactions to misbehavior do not cause unintended incentives or consequences.

**1) Certificate-independence.** A reaction policy should be *decoupled from public-key certificates*. Like certificates, reaction policies are negotiated between CAs and domains. However, certificates and reaction policies are independent: CAs issue reaction policies in addition to certificates. Reaction policies 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.

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

**3) Single-use.** A reaction policy should be *limited to a single instance of CA misbehavior*. Because reaction policies may execute financial payments for which funds must be available, enforcing single-use reaction policies helps ensure the availability of such one-time resources for each instance of misbehavior. Thus each time a CA issues a certificate that violates a domain's DCP, one of the domain's reaction policies is triggered and then terminated. We note that domains can have multiple reaction policies at a given time to protect against repeated CA misbehavior.

### 6.2 RP Contents

We now describe the contents and format of reaction policies. Figure 4 shows the format of a sample reaction policy, and Table 2 describes each field of a reaction policy. Like a DCP, a reaction policy contains identifying information for the domain as well as for the issuing CA. A reaction policy also specifies a validity period and identifies the version of the domain's DCP for which it is

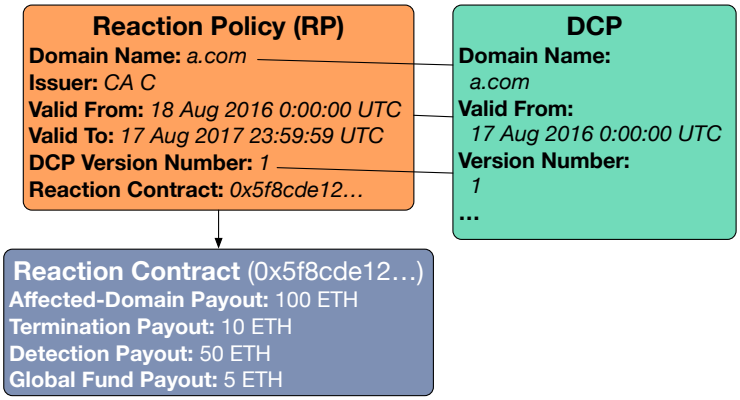


**Fig. 4.** A sample RP with a payout reaction contract. The domain name and version number in the RP must match those of the DCP, and the start of the RP's validity must be after that of the DCP.

**Table 2.** 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 Contract | address of the RP's reaction contract |

active. Finally, the specific reactions that take place are specified as an address to a contract.

Similarly to check contracts, a *reaction contract* is a contract account. The contract provides a method `trigger` that, when called from the IKP contract, performs a set of reactions. Reaction contracts can be financial, as described in Section 6.3 or specify other actions to be taken against CA misbehavior. At the end of the method, the contract destroys itself. The use of a reaction contract allows domains and CAs to negotiate a wide range of reactions, providing expressiveness.

We note that a reaction policy has a start and end time for its validity, rather than only a start time as a DCP does. A reaction policy, like a certificate, has a limited validity period, but can be prematurely terminated if the issuing CA misbehaves. If a reaction policy is terminated, we split a specified amount of funds between the domain and the issuing CA based on the validity period of the reaction policy.

The reaction policy itself also provides two additional methods through the IKP contract, namely `terminate` and `expire`. The `terminate` method can be called by a domain if the CA issuing its reaction policy has misbehaved. The

`expire` method can be called by the issuer once the validity of the reaction policy has ended.

### 6.3 Payout Reaction Contracts

We now provide a framework for payout reaction contracts, which specify a series of financial payments that execute in response to CA misbehavior. Financial payments are an important foundation to our study of automatic reactions, as the means for exchanging them are built into blockchain-based cryptocurrencies, they represent natural incentives, and can be concretely analyzed in contrast to other responses, such as revoking a certificate. Our goal in designing a framework for this class of reaction contracts is to provide a general model for who should receive payments under different circumstances of misbehavior.

We identify four parties who should receive payments in case of CA misbehavior:

1. the domain, which we denote with $D$,
2. the issuer, which we denote with $C$,
3. the detector, which we denote with $d$, and
4. the global fund, which we denote with $F$.

As Figure 4 shows, a payout reaction contract specifies four payouts: 1. *affected-domain payouts*, 2. *termination payouts*, 3. *detection payouts*, and 4. *global fund payouts*. To ensure the availability of these funds, a fraction $\epsilon$ (where $0 < \epsilon < 1$) of the total amount is escrowed in the contract itself. Though a reaction policy specifies certain payout amounts, these values may differ if instead of a registered CA an unregistered CA misbehaves. We call these scenarios *internal misbehavior* and *external misbehavior*, respectively.

**Affected-domain payouts.** The affected-domain payout is paid to $D$ in the event that *any* 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 reaction policy specifies the affected-domain payout for internal misbehavior, which we denote as $a_i$. In the event of external misbehavior, the payout is a systemwide parameter $a_e$.

**Termination payouts.** The termination payout is split between $D$ and $C$ if $D$ terminates the reaction policy. The termination payout compensates $D$ for lost trust in $C$ after its misbehavior and contributes towards 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 reaction policy's validity. To ensure that $D$ receives some minimum amount of funds, we set a systemwide parameter $\tau$ that $D$ is guaranteed to receive. If we denote the termination payout with $t$, the amount that the domain receives with $t_D$, the amount that the issuer receives with $t_C$ (thus $t = t_D + t_C$), and the fraction of the reaction policy's remaining validity, then

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

$$t_C = (1 - \alpha)(t - \tau) \tag{2}$$

Because $0 \leq \alpha \leq 1$, we can see that

$$\tau \leq t_D \leq t \tag{3}$$
$$0 \leq t_C \leq t - \tau \tag{4}$$

We note that although $C$ does receive funds from the termination payout in spite of its misbehavior, we show in Section 8 that $C$ loses funds compared to if it had behaved correctly.

**Detection payouts.** The detection payout is the amount paid to whomever reports an unauthorized certificate for the domain 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 reaction policy specifies the detection payout for internal misbehavior, which we denote as $\delta$. In the event of external misbehavior, the payout is the reporting fee $m$, which we describe in Section 7.3.

**Global fund payouts.** The global fund payout is the amount paid to the global fund when the reaction policy is triggered. The payout replenishes the global fund to ensure that it has enough funds to continue its operation. As we will see in Section 8, the payout amount is designed to compensate for any losses that the global fund may incur, with high probability. We denote the global fund payout by $f$.

### 6.4 Client-Facing Contracts

Client-facing contracts place specially-formatted events in the blockchain. Applications can monitor the blockchain for these events and use them to improve client security. For example, a browser extension can look for certificates that violate a domain's DCP and if so, then not connect to the site based on these events. The browser extension can also use these events to display custom error messages to users, prompting them to check a certain public log for corroborating evidence of the site's certificate.

Client-facing contracts allow domains and CAs to negotiate a far wider range of reactions that can include customized warning messages for specific users or tailored browser behavior (stepping back in history, closing a browser tab, etc). However, we leave a more detailed design and analysis of client-facing contracts to future work.

## 7 Operational Considerations

In this section, we discuss the operation of IKP. Specifically, we describe the processes for (1) registering a CA or DCP in the blockchain, (2) purchasing certificates and reaction policies, and (3) reporting CA misbehavior.

**Table 3.** Explanation of fields in a CA registration.

| Field | Use |
|---|---|
| CA name | identify CA |
| Valid from | specify start period of information validity |
| Payout address | authenticate and receive payments to CA |
| Public keys | list of CA's public keys |
| Update addresses | (default empty) authorize updates to this information |
| Update threshold | (default 1) threshold of payout/update addrs. for updates |

### 7.1 CA and DCP Registration

CAs and domains follow similar processes to register in the blockchain. For both CAs and domains, we want to ensure that the process of registering in the blockchain is secure in the sense that for a given identifier such as a DNS name, only the entity that owns the identifier in the existing Internet can claim and maintain control of the identifier in IKP. We achieve this by requiring a proof of control over an identifier whenever information associated with the identifier is registered or updated.

**Initial registration.** A CA or a domain requests to initially register in the blockchain by sending a transaction to the IKP contract containing the identifier it wants to claim, the information to associate with the identifier, and a bootstrap proof (described below). Domains simply associate a DCP with their name, where the format of the DCP is given as described in Section 5.2. CAs associate a *registration* with their name, where the fields of the registration are shown in Table 3. In contrast to a DCP, a CA registration has no version number and specifies a list of public keys used to issue certificates within IKP in place of a check contract address.

**Bootstrap proofs.** In a nutshell, a *bootstrap proof* provides a way for a domain or CA to leverage another PKI to show control over a desired identifier when initially registering with the IKP contract. Because IKP is tied to TLS and hence to CA and DNS names, we do not try to prevent name squatting within a namespace unique to IKP; rather, we try to protect claims on names that already exist today. We use a different type of bootstrap proof for CAs and for domains.

A bootstrap proof for a CA is a chain of certificates to one of several seed CAs. We can initialize the IKP contract with the keys of several existing root CAs and require that any CA who wants to register a certain identifier present a chain of certificates (up to some maximum length) rooted in a specified CA. Among the top 5 CAs, who root the certificate chains for 92.2% of all HTTPS sites [10], we found 33 certificates in major root certificate stores, each of whose keys could be included in the IKP contract. Alternately, we could use the public keys of a set of 28 root certificates which are present in a majority of popular desktop and mobile operating systems and web browsers [69].

A bootstrap proof for a domain can be based on DNSSEC [14] or on TLS. In the DNSSEC-based proof, a domain sends a chain of signatures within DNSSEC leading back to ICANN's root DNSSEC key, which can be verified in the IKP contract. This approach allows domains to demonstrate control over their DNS name within a PKI that has had far fewer compromises than that of TLS, and also only requires a single root key to be stored in the IKP contract. However, in a measurement we conducted using Censys [28] and Amazon's Alexa Web Information Service, we found that only 649 of the top 100,000 most popular domains use both DNSSEC and HTTPS, comprising only 1.03% of all HTTPS pageviews. We therefore also allow domains to register by sending a bootstrap proof of at least three certificates from CAs that have registered in IKP, as well as signatures with each of the private keys corresponding to these certificates.

**Updates.** A domain or CA can update its information by sending a transaction to the IKP contract with its new DCP or registration. The domain or CA sends this transaction from at least a threshold number of its payout and update addresses. The IKP contract verifies that the information in each of these transactions is the same, and then updates the entry in its storage. In order to prevent minor updates of a domain's DCPs from invalidating all of its existing reaction policies, the IKP contract only increments the version number if the check contract address is updated. A domain is responsible for obtaining new reaction policies if it updates its check contract.

**Funding accounts.** The IKP contract maintains a balance for each CA. The CA can deposit into and withdraw from this balance. The purpose of the funding account is to allow CAs to escrow funds that they may need to pay out. While escrowing some funds for payout reaction contracts is required as described below, depositing into the balance allows a CA to provide potential customers with greater confidence in its ability to pay out in case of misbehavior.

### 7.2 Purchasing Certificates and Reaction Policies

When a domain $D$ wants to purchase a reaction policy or a certificate within IKP, it first agrees upon the terms of the reaction policy or certificate, such as the price and validity period. We want to ensure that a domain who purchases a reaction policy or certificate in IKP obtains what it agreed on with the CA, and conversely, we also want to ensure that the CA receives the appropriate payment for the reaction policy or certificate that it has issued. In other words, we want a fair exchange mechanism.

We can achieve such a fair exchange by having the IKP contract act as a third-party escrow. Because the IKP contract's operation is controlled entirely by its code and is executed in a decentralized manner, we can use the IKP contract like a third-party escrow service. In particular, a domain sends the payment for the reaction policy or certificate to the IKP contract, along with the hash of the reaction policy or certificate and the issuer. The issuer creates and sends the reaction policy or certificate, along with any necessary funds, to the IKP contract. In particular, for a payout reaction contract, let $I = a_i + t + \delta + f$.

**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:     DCPD ← lookup D in DCP map
4:     CC ← get check contract address from DCPD
5:     if CC.check(C) then
6:         RPL_D ← lookup reaction policy 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
```

Then the CA sends $\epsilon I$ where $0 < \epsilon < 1$. This amount is escrowed in the payout reaction contract. The IKP contract verifies that the reaction policy or certificate hashes to the value provided by the client, and that the issuing CA is correct. The funds are then transferred to the appropriate parties. In particular, the fee $\rho$ for the reaction policy is transferred to the issuer, and the appropriate fraction of funds $\epsilon I$ is transferred to the reaction contract.

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

When choosing a CA from whom to purchase a reaction policy 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.

### 7.3 Reporting Misbehavior

To report misbehavior, a detector needs to send an unauthorized certificate to the IKP contract. To ensure that detectors are disincentivized from spamming the IKP contract with spurious reports, we require each detector to pay a small fee for reporting misbehavior, and set the constraint that all payout reaction contracts must provide a detection payout greater than this fee. Thus any failed report results in a net loss for a detector, discouraging detectors from, for example, reporting every certificate to the IKP contract.

We also need to 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 [63] 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 required waiting for a set number of blocks, frontrunning is not possible.

Upon receiving the detector's report, the IKP contract checks that the certificate and secret sent by $d$ matches the committed value sent earlier. The contract then carries out the check shown in Algorithm 1. If the check contract returns `true` for the certificate $C$, the IKP contract triggers the reaction contract for the oldest of the domain's reaction policies. 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.

In some cases, a misbehaving CA may not have sufficient funds to pay out all entities. In this case, the IKP contract divides the funds that the CA does have and distributes them proportionally to the appropriate parties as they are in the reaction policy. It also bans the CA from continuing to issue reaction policies in IKP and raises an event in the blockchain that the CAdid not have sufficient funds to pay out. The record of this event can then be used as a basis to pursue other legal action against the misbehaving CA. We have acknowledged the problems with manually pursuing actions against misbehaving CAs; however, IKP improves upon the existing ecosystem by providing some automatic reactions, and only requiring manual intervention in extreme cases.

### 7.4 Client Lookups

We note that despite the benefits that IKP offers to domains whose certificates are compromised, clients must also be protected from the use of unauthorized certificates in MitM attacks. We achieve this through a lightweight browser extension that checks all certificates it receives in a TLS handshake using the domain's DCP. Because this check is done locally, the extension does not need to carry out the check by sending a transaction in the blockchain; rather, it can simply carry out the check locally, as long as it has the blockchain state. An extension can instead use a feature such as Ethereum's event logging to check for specific unauthorized certificates, providing efficient proofs of misbehavior suitable for light clients, which confirm such events without downloading the entire blockchain at the cost of trusting other full nodes to do so. This approach allows clients to defend against unauthorized certificates that have already been reported. Because of the incentives offered to detectives, we anticipate that unauthorized certificates will be reported very soon after discovery.

**Table 4.** List of payments sent for each event. Note that external misbehavior can only occur in the non-issuer scenario.

| **Event** | **From** | **To** | **Amount** |
|---|---|---|---|
| Register CA | $C$ | $F$ | $r_C$ |
| Register domain | $D$ | $F$ | $r_D$ |
| Issue reaction policy | $D$ | $F$ | $\rho + g_D$ |
| | $C$ | $F$ | $\epsilon I + g_C$ |
| | $F$ | $C$ | $\rho$ |
| Expire reaction policy | $F$ | $C$ | $\epsilon I$ |
| Terminate reaction policy | $F$ | $D$ | $t_D$ |
| | $F$ | $C$ | $\epsilon I - t_D$ |
| Report false misbehavior | $d$ | $F$ | $m$ |
| Report internal misbehavior | $d$ | $F$ | $m$ |
| (issuer) | $F$ | $D$ | $a_i + t_D$ |
| | $F$ | $d$ | $\delta$ |
| | $F$ | $C$ | $\epsilon I - t_D - \delta - f$ |
| Report internal misbehavior | $d$ | $F$ | $m$ |
| (non-issuer) | $F$ | $D$ | $a_i + t_D$ |
| | $F$ | $d$ | $\delta$ |
| | $F$ | $R$ | $\epsilon I - t_D$ |
| | $C$ | $F$ | $a_i + \delta + f$ |
| Report external misbehavior | $d$ | $F$ | $m$ |
| (non-issuer) | $F$ | $D$ | $a_e + t_D$ |
| | $F$ | $d$ | $m$ |
| | $F$ | $R$ | $\epsilon I - t_D$ |

## 8 Analysis

In this section, we analyze the design of IKP. In particular, we analyze the flow of payments among the different entities to determine what constraints must hold in order to guarantee that entities benefit from behaving correctly and are punished for misbehaving.

### 8.1 Model

In our analysis, we consider the payments in Table 4, which shows the series of payments that occur for each action in IKP. We then consider two major scenarios:

1. The *issuer scenario*: A CA $C$ that has issued a reaction policy to $D$ issues a certificate to $D$. In doing so, $C$ can issue a certificate that complies with $D$'s DCP or one that does not. A detector $d$ can then choose whether to report this certificate or not.

**Table 5.** Rewards for each entity in the three possible cases of the issuer scenario.

| Entity | Unreported | Behave | Misbehave |
|---|---|---|---|
| $D$ | $-\rho$ | $-\rho$ | $-\rho + a_i + t_D$ |
| $d$ | 0 | $-m$ | $-m + \delta$ |
| $C$ | $\rho$ | $\rho$ | $\rho - a_i - t_D - \delta - f$ |
| $F$ | 0 | $m$ | $m + f$ |

2. The *non-issuer scenario*: A CA $C$ that is not registered in the blockchain issues a certificate for $D$, who has a reaction policy issued by a different CA $R$. In doing so, $C$ can choose whether to register in IKP or not, and whether to issue an authorized or an unauthorized certificate. After this, a detector $d$ can choose whether to report the certificate or not.

For each case, we consider the payments made in the series of events that must have occurred and can determine the reward of each entity by summing the payments it received and subtracting the sum of the payments it made. We note that in our analysis we do not consider payments made outside of IKP, as we cannot enforce any guarantees on these amounts.

For each scenario, we want to derive constraints for which the following is true:

- a domain with a DCP profits from internal misbehavior,
- a CA who internally misbehaves loses money,
- a CA who externally misbehaves cannot profit, and
- a detector who reports misbehavior makes money.

In order to ensure that a misbehaving CA cannot profit from misbehavior, we need to consider *collusion attacks* in IKP. In particular, we must verify that a misbehaving $C$ cannot collude with other entities, thus summing their rewards, to gain a profit. We observe that $C$ will only collude with entities that receive a positive reward on their own, but can purposely misbehave in order to trigger reaction policy 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 both scenarios, we assume that the CA $C$ has registered in the blockchain, and that the domain $D$ has registered a DCP in the blockchain. We do not consider these in our analysis due to the fact that they occur once and thus should not factor into the analysis of a single reaction policy's lifetime, which may occur (with its costs) many times.

### 8.2 Issuer Scenario

For the issuer scenario, we consider whether or not $C$ misbehaves, and whether or not $d$ reports the misbehavior. We assume that the issuance has taken place and the appropriate payments made. We observe that if $d$ does not report any misbehavior, then the reaction policy will eventually expire, regardless of whether $C$ misbehaves. Thus we consider three cases: 1. $d$ does not report misbehavior,

**Table 6.** Rewards for each entity in the six possible cases of the non-issuer scenario.

| Entity | Unrep. | Behave | Misbehave |
|---|---|---|---|
| **Registered** | | | |
| $D$ | $-\rho$ | $-\rho$ | $-\rho + a_i + t_D$ |
| $d$ | 0 | $-m$ | $-m + \delta$ |
| $R$ | $\rho$ | $\rho$ | $\rho - t_D$ |
| $C$ | $-r_C$ | $-r_C$ | $-r_C - a_i - \delta - f$ |
| $F$ | $r_C$ | $r_C + m$ | $r_C + m + f$ |
| **Unregistered** | | | |
| $D$ | $-\rho$ | $-\rho$ | $-\rho + a_e + t_D$ |
| $d$ | 0 | $-m$ | 0 |
| $R$ | $\rho$ | $\rho$ | $\rho - t_D$ |
| $C$ | 0 | 0 | 0 |
| $F$ | 0 | $m$ | $-a_e$ |

2. $C$ behaves, but $d$ reports the certificate, and 3. $C$ misbehaves and $d$ reports the certificate.

Table 5 shows our results. We observe that in the case of reported misbehavior, $D$ receives an additional $a_i + t_D$ than it would otherwise. In order for $D$ to profit, $a_i + t_D > \rho$. Since by Equation 3 we know that $\tau \leq t_D$, we set the constraint $\rho < a_i + \tau$. We also observe that if $d$ reports misbehavior correctly, it receives $-m + \delta$. Thus for $d$ to make a profit, $\delta > m$ must hold. Finally, we observe that for $C$ to lose money due to its misbehavior, $\rho < a_i + t_D + \delta + f$. Again, since $\tau \leq t_D$, we have the constraint $\rho < a_i + \tau + \delta + f$. However, this constraint is subsumed by the first, which sets a tighter bound on $\rho$.

To avoid collusion attacks in the issuer scenario, we consider the entities besides $C$ receiving a positive reward. In the issuer scenario we observe that though both $D$ and $d$ profit in the case of misbehavior, if we sum the rewards of $D$, $d$, and $C$, the result is $-m - f$, and thus $C$ does not profit.

### 8.3 Non-Issuer Scenario

In the non-issuer scenario, we consider whether or not $C$ registers in the blockchain, whether or not $C$ misbehaves, and whether or not $d$ reports misbehavior. We again assume that $D$ has purchased a reaction policy; however, since $C$ is not the issuer of the reaction policy, we denote the issuer as $R$. We again observe that if $d$ does not report misbehavior, then the reaction policy expires and $C$'s misbehavior status does not matter. However, we need to consider $C$'s registration status, and thus we analyze six cases instead of three.

Table 6 shows our results. We observe that if $C$ has registered in the blockchain, then $D$ and $d$ have the same rewards as in the issuer scenario. However, the other rewards provide us with more constraints. We observe that $R$ should still profit since it has not misbehaved, and thus $\rho > t_D$. Because $t_D \leq t$, we have the

constraint $\rho > t$. For $C$, there is the additional penalty of the registration cost $r_C$. However, $C$ does not get $\rho$ in this case, since it did not issue $D$'s reaction policy. Thus while $C$ faces the same penalty as for the issuer scenario, it receives $\rho + r_C$ less overall.

To avoid collusion attacks in the non-issuer case, we again consider the entities besides $C$ making a positive reward. In the case that $C$ has registered in the blockchain, we observe that $D$, $d$, and $R$ all have a positive reward. However, if we sum the rewards of $D$, $d$, $R$, and $C$, we still end up with a total reward of $-r_C - m - f$, resulting in a negative reward for $C$. In the case that $C$ does not register in the blockchain, however, we observe that $C$ does not need to pay anything, and thus 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 $D$ if the reward $-\rho + a_e + t_D$ is positive. To avoid this, we must set a constraint so that this reward is nonpositive, that is, $\rho \geq a_e + t_D$. Observing that $t_D \leq t$, we set the constraint $\rho \geq a_e + t$, which prevents collusions attacks by $C$. For the same reason, we also set $d$'s reward to $m$ instead of $\delta$ in this case. While it may seem counterintuitive to penalize $D$ for such an incident, we observe that $D$ still makes a higher reward than it would without reporting the misbehavior; thus for $D$ reporting the misbehavior is the better strategy, even at a loss.

## 9 Evaluation

In this section, we investigate the technical feasibility of IKP in today's blockchains. In particular, we detail our prototype implementation in Ethereum and describe the changes necessary for full deployment. We then analyze the risk of CA misbehavior today in an attempt to estimate the fraction $\epsilon$ of funds in a payout reaction contract that should be escrowed.

### 9.1 Prototype Implementation

We implemented IKP in Solidity, a high-level Ethereum language that resembles JavaScript. We wrote the entire IKP contract in 166 lines of Solidity code. For comparison, we note that a simple Solidity implementation of Namecoin in Ethereum requires fewer than 10 lines of code. Using a Solidity compiler, we estimated 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. For the purposes of estimation, we assumed that all strings representing CA identifiers and domain names were constrained to a maximum of 32 bytes, and that all arrays were of fixed length. To convert the cost in gas to USD, we used the current standard price of 20 Gwei $\approx 2.03 \times 10^{-7}$ USD per unit of gas.

Table 7 shows the costs of various operations in gas and USD. Looking at the relative costs, we make several observations. As expected, DCP registration

**Table 7.** Cost of various IKP operations.

| | Approximate Cost | | | Approximate Cost | |
|---|---|---|---|---|---|
| **Operation** | **Gas** | **USD** | **Operation** | **Gas** | **USD** |
| Register CA | 91 400 | $0.0186 | Register DCP | 212 581 | $0.0432 |
| Update CA | 34 656 | $0.0070 | Update DCP | 181 226 | $0.0368 |
| Order RP | 40 599 | $0.0082 | Pre-report cert | 40 639 | $0.0083 |
| Create RP | 226 892 | $0.0461 | Report cert | 149 284 | $0.0303 |
| Terminate RP | 99 461 | $0.0202 | Send payouts | 107 962 | $0.0219 |
| Expire RP | 39 823 | $0.0081 | CA Balance | 39 716 | $0.0081 |
| **IKP Contract Creation** | | | | 628 640 | $0.1277 |

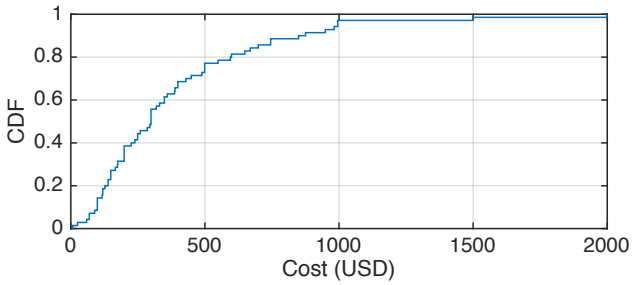


**Fig. 5.** Distribution of 1-year certificate costs.

is more expensive than CA registration due to the size of the DCP that must be stored in the blockchain by the IKP contract. We also note that reporting misbehavior is comparatively expensive, since when misbehavior is detected, the function must determine which case of misbehavior has occurred and send the payments accordingly. We note that all costs, including creating the IKP contract, are well below $1. However, several features proved to be a challenge. The cost of parsing certificates and verifying signatures other than ECDSA signatures in Ethereum is currently prohibitively expensive. In future work, we plan to make changes to the underlying Ethereum virtual machine to build in these types of functions, thus allowing us to deploy a production version of IKP. Besides these features, we note that the cost of the other operations in Ethereum is relatively low, making IKP a feasible solution once we have certificate parsing and alternative signature verification algorithms.

### 9.2 Risk Analysis

In order to evaluate the real-world risk of CA misbehavior, we collected data measuring a CA's risk assessment. We examined each of the standard TLS certificate offerings of all CAs with a market share of at least 0.1%, of which there are 20 CAs [10]. For each certificate, we noted the cost of a 1-year certificate

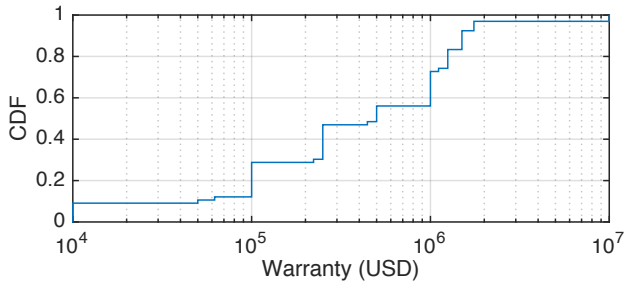


**Fig. 6.** Distribution of certificate warranty amounts.

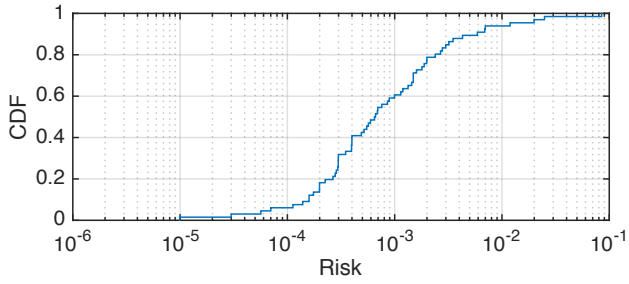


**Fig. 7.** Distribution of calculated risk amounts.

(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 5, Figure 6, and Figure 7 show the CDF for the cost, warranty, and calculated risk of each of these certificates, respectively. 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 8, the risk assessments for each certificate varied widely, ranging from around 0.001% up to almost 8.5%. To be conservative, we can estimate that any CA likely to be used in practice will have a risk assessment of at most 10%, and we can thus reasonably assume that of all certificates issued by a CA, at most 10% will be unauthorized (if the CA is not malicious). We thus propose using a value of 0.1 for $\epsilon$.

**Table 8.** 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, 65] | 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 | Multi Domain SSL | $299 | $10,000,000 | 2.99e−5 |
| IdenTrust [6] | Standard SSL | $99 | $10,000,000 | 9.90e−6 |
| **Lowest-Risk** | | | | |

## 10 Discussion

In this section, we discuss various limitations and future work of IKP. We also clarify several points that the attentive reader may have raised.

**Blockchain Weaknesses.** Blockchains have several weaknesses which have been demonstrated in practice. For example, large pools of miners have at times controlled a majority of hashing power in the network [20], allowing double-spending attacks, suppression of selected transactions, and other subversion of blockchain consensus. As discussed in Section 11, these attacks can also take place with miners that control less than half of the network's hashrate. Another weakness is that the IKP contract has not been verified, which may result in code vulnerabilities such as the one that plagued the Decentralized Autonomous Organization (DAO) in Ethereum [21]. These vulnerabilities may result in hard forks that perform otherwise invalid transactions based on community consensus, as the DAO exploit did [22]. These weaknesses is particularly relevant due to the large amount of funds that may be stored in the IKP contract. In this paper, we considered both of these weaknesses out-of-scope.

**Compelled certificates.** In this work, we did not explicitly attempt to defend against nation-states who can compel CAs to issue unauthorized certificates. This type of adversary will likely not worry about cost, and in any case the CA will bear the brunt of responsibility. However, this adversary will still be unlikely to mount such an attack due to the browser extension that protects IKP clients from MitM attacks, irrespective of the payments among other entities.

**Deployment Incentives.** In addition to detectors and miners, all domains, CAs, and logs have incentives to adopt IKP. A domain receives compensation or any misbehavior that affects its own security, and due to the blockchain's high

availability is quickly alerted of any such misbehavior. The deterrence provided by IKP's automatic handling of misbehavior reduces the risk of attacks and provides the domain with greater confidence in its CA, particularly IKP CAs with good reputations.

CAs, particularly the majority of CAs who have never misbehaved, can prove their good reputation using the public nature of the blockchain. CAs receive the added benefit of the profits from RPs in case of good behavior. Moreover, RPs provide a value-added service for CAs to compete with free certificate efforts like Let's Encrypt [31]. Although Let's Encrypt does not offer EV certificates or extra services as other CAs do, RPs provide an additional extra service that unlike EV verification or customer support can be completely automated.

Logs, like any entity, can can act as detectors or miners, submitting or processing information that proves CA misbehavior. However, public logs have the advantage that the necessary information is locally available, though at the cost of maintaining high availability and large databases for the security of the system. In IKP, however, they can receive financial rewards for doing so. The advantage of logs for detection also incentivizes the creation of more logs, thus mitigating the effect of a single misbehaving log.

Clients can use the browser extension to protect themselves against malicious certificates. Because DCPs can handle a general class of policies, we can provide protection as good as existing log-based systems with the additional advantage of financial incentives. Additionally, clients can develop additional extensions that for example will automatically report any detected unauthorized certificates.

**Complementing Log-based PKIs.** Using IKP in conjunction with log-based PKIs such as CT [51], ARPKI [16] and PoliCert [78] offers a multi-layer PKIs defense: log-based PKIs provide provable security guarantees and a foothold in current deployment, while IKP offsets the cost of the visibility of CA behavior by incentivizing log operation and usage. Components of IKP can also improve existing log-based PKIs: the blockchain can be used to provide globally unique and consistent domain policies (needed in PoliCert) or to confirm the latest state of a log's database (needed in ARPKI).

**Future Work.** We next plan to explore the following improvements to IKP. First, we plan to develop further reaction policies to explore what other reactions are effective against CA misbehavior. In conjunction with this effort, we also plan to conduct usability testing with our browser extension to see which reactions are most helpful to users. Finally, we believe that incentivizing logs in systems like Certificate Transparency provide a nice complement to our work and hope to bring that to fruition.

## 11 Related Work

In this section, we discuss work related to IKP. In particular, we cover four main areas: log-based PKIs, alternatives to CA-based PKIs, incentives on blockchains, and insurance schemes. We do not describe the workings of log-based PKIs here; instead, we refer the reader to Section 2 for an overview.

**Log-based PKIs.** CT [51] was the first to propose the use of public logs in their current form, though earlier proposals such as Sovereign Keys [30] used similar entities. CT, however, provides no support for revocation, nor does it provide any information as to whether the logged certificates are authentic. Revocation Transparency [50] and CIRT [71] both provide mechanisms to enable revocation checking in public logs. AKI [42] 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 [16] presents a formally-verified extension of AKI that provides stronger security guarantees. PoliCert [78] 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 offers incentives for correct behavior or enables automatic response even if misbehavior is detected.

**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 [40] allows domains to place public keys or certificates in DNSSEC [14], 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 [45], HPKP [32], or TACK [53] store information about a domain's public key at the client browser. Perspectives [80] and Convergence [52] leverage the public keys observed by notary servers throughout the Internet to detect targets 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 [12] (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 [34] leverages Namecoin [62] to implement a blockchain-based PKI, storing identity information in a Merkle hash tree and using the Kademlia DHT [55] 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 [17] does not implement a PKI on its own, but rather uses the Ethereum blockchain to audit a centralized key servers for CONIKS [56]. However, neither EthIKS nor CONIKS provides any means for responding to equivocation or other misbehavior by key servers.

**Blockchain-based incentives.** Most previous studies of incentives in blockchains have been concerned with the incentives of mining. In particular, the selfish min-

ing attack [33] shows that mining in the Bitcoin network is not incentive compatible. Subsequent work shows further improvements on the strategy [72] and how selfish mining can be composed with network attacks such as the eclipse attack [39] to increase the revenue of selfish mining [64]. 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 [13]. 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 [43]. 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.

**Insurance schemes.** Even before cryptocurrencies, the idea of electronic insurance policies were used to evaluate services in distributed systems [44]. The idea of insurance was also proposed as an example of an authentication metric that followed good design principles [70]. 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 [54]. Our work on IKP adds a cryptocurrency-based instantiation of their model as well as proofs of incentivization compared to log-based PKIs.

## 12 Conclusions

In this paper, we proposed IKP: smart contracts for detecting, publicizing, and automatically responding to CA misbehavior. 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 help guide such a realization of IKP. 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.

## References


1. Certificate Transparency log policy. `https://www.chromium.org/Home/chromium-security/certificate-transparency/log-policy`
2. Certificate Transparency known logs. `http://www.certificate-transparency.org/known-logs` (April 2016)
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. Aas, J.: Progress towards 100% HTTPS. https://letsencrypt.org/2016/06/22/https-progress-june-2016.html (June 2016)
12. Ali, M., Nelson, J., Shea, R., Freedman, M.J.: Blockstack: A global naming and storage system secured by blockchains. In: USENIX Annual Technical Conference (ATC) (June 2016)
13. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. In: Proceedings of the 35th IEEE Symposium on Security and Privacy. pp. 443–458 (May 2014)
14. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS security introduction and requirements. RFC 4033 (March 2005)
15. Back, A.: Hashcash: A denial of service counter-measure. http://www.cypherspace.org/adam/hashcash/ (August 2002)
16. Basin, D., Cremers, C., Kim, T.H.J., Perrig, A., Sasse, R., Szalachowski, P.: ARPKI: Attack resilient public-key infrastructure. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 382–393. ACM (2014)
17. Bonneau, J.: EthIKS: Using Ethereum to audit a CONIKS key transparency log. In: 3rd Workshop on Bitcoin and Blockchain Research (BITCOIN) (February 2016)
18. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P) (May 2015)
19. Borger, J.: GCHQ and European spy agencies worked together on mass surveillance. http://www.theguardian.com/uk-news/2013/nov/01/gchq-europe-spy-agencies-mass-surveillance-snowden (November 2013)
20. Buterin, V.: On mining. Ethereum Blog (June 2014)
21. Buterin, V.: Critical update re: DAO vulnerability. Ethereum Blog (June 2016)
22. Buterin, V.: Hard fork completed. Ethereum Blog (July 2016)
23. Chuat, L., Szalachowski, P., Perrig, A., Laurie, B., Messeri, E.: Efficient gossip protocols for verifying the consistency of certificate logs. In: IEEE Conference on Communications and Network Security (CNS). pp. 415–423 (2015)
24. Comodo: Comodo fraud incident 2011-03-23. https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html (March 2011)
25. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, T.: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280 (May 2008)
26. Crosby, S.A., Wallach, D.S.: Efficient data structures for tamper-evident logging. In: Proceedings of the 19th USENIX Security Symposium. pp. 317–334 (August 2009)
27. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.2. RFC 5246 (August 2008)

28. Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., Halderman, J.A.: A search engine backed by Internet-wide scanning. In: 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 542–553. ACM (2015)
29. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Proceedings of Advances in Cryptology (CRYPTO). pp. 139–147 (August 1992)
30. Eckersley, P.: 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)
31. Encrypt, L.: Let's encrypt. https://letsencrypt.org
32. Evans, C., Palmer, C., Sleevi, R.: Public key pinning extension for HTTP. RFC 7469 (April 2015)
33. Eyal, I., Sirer, E.G.: Majority mining is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security (FC) (2014)
34. Fromknecht, C., Velicanu, D., Yakoubov, S.: A decentralized public key infrastructure with identity retention. Cryptology ePrint Archive, Report 2014/803 (November 2014)
35. Gellman, B., Poitras, L.: U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program. http://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html (June 2013)
36. Gillmor, D.K., Nordberg, L.: Certificate Transparency gossip. IETF-92 (March 2015)
37. Google: Certificate Transparency: Feb 2014 survey responses. http://www.certificate-transparency.org/feb-2014-survey-responses (February 2014)
38. Greenwald, G., Ackerman, S.: How the NSA is still harvesting your online data. http://www.theguardian.com/world/2013/jun/27/nsa-online-metadata-collection (June 2013)
39. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on Bitcoin's peer-to-peer network. In: 24th USENIX Security Symposium (USENIX Security). pp. 129–144 (2015)
40. Hoffman, P., Schlyter, J.: The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC 6698 (August 2012)
41. Hoogstraaten, H., Prins, R., Niggebrugge, D., Heppener, D., Groenewegen, F., Wettink, J., Strooy, K., Arends, P., Pols, P., Kouprie, R., Moorrees, S., van Pelt, X., Hu, Y.Z.: 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)
42. Kim, T.H.J., Huang, L.S., Perrig, A., Jackson, C., Gligor, V.: Accountable Key Infrastructure (AKI): A proposal for a public-key validation infrastructure. In: Proceedings of the 22nd international conference on World Wide Web. pp. 679–690 (May 2013)
43. Kumaresan, R., Bentov, I.: How to use bitcoin to incentivize correct computations. In: Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM (2014)
44. Lai, C., Medvinsky, G., Neuman, B.C.: Endorsements, licensing, and insurance for distributed system services. In: Proc. of the 2nd ACM Conference on Computer and Communications Security (1994)
45. Langley, A.: Public key pinning. https://www.imperialviolet.org/2011/05/04/pinning.html (May 2011)

46. Langley, A.: Revocation checking and Chrome's CRL. https://www.imperialviolet.org/2012/02/05/crlsets.html (February 2012)
47. Langley, A.: Enhancing digital certificate security. http://googleonlinesecurity.blogspot.com/2013/01/enhancing-digital-certificate-security.html (January 2013)
48. Langley, A.: Further improving digital certificate security. http://googleonlinesecurity.blogspot.com/2013/12/further-improving-digital-certificate.html (December 2013)
49. Langley, A.: Maintaining digital certificate security. http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html (March 2015)
50. Laurie, B., Kasper, E.: Revocation transparency. http://www.links.org/?p=1272 (September 2012)
51. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. RFC 6962 (June 2013)
52. Marlinspike, M.: SSL and the future of authenticity. http://www.youtube.com/watch?v=Z7Wl2FW2TcA, BlackHat 2011 (August 2011)
53. Marlinspike, M., Perrin, T.: Trust assertions for certificate keys. https://tools.ietf.org/html/draft-perrin-tls-tack-02 (January 2013)
54. Matsumoto, S., Reischuk, R.M.: Certificates-as-an-Insurance: Incentivizing accountability in SSL/TLS. Proceedings of the NDSS Workshop on Security of Emerging Network Technologies (SENT '15) (2015)
55. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
56. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: Coniks: Bringing key transparency to end users. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 383–398 (August 2015)
57. Messeri, E.: Certificate transparency, or: How hard can it be to change the public key infrastructure? (2015)
58. Microsoft: Erroneous VeriSign-issued digital certificates pose spoofing hazard. https://technet.microsoft.com/library/security/ms01-017 (Mar 2001)
59. Microsoft: Improperly issued digital certificates could allow spoofing. Microsoft Security Advisory 3046310 (March 2015)
60. Mills, E., McCullagh, D.: 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)
61. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Whitepaper (October 2008)
62. Namecoin: Namecoin. http://namecoin.info
63. Namecoin: Register and configure .bit domains. https://wiki.namecoin.info/index.php?title=Register_and_Configure_.bit_Domains (May 2015)
64. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: IEEE European Symposium on Security and Privacy (EuroS&P) (2016)
65. Nigg, E.: StartCom certificate policy and practice statements. https://www.startssl.com/policy.pdf (May 2016)
66. Nightingale, J.: DigiNotar removal follow up. https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/ (September 2011)

67. Palmer, M.: It's 10pm, do you know where your SSL certificates are? http://www.hezmatt.org/~mpalmer/blog/2015/07/07/its_10pm_do_you_know_where_your_ssl_certs_are.html (2015 July)
68. Percoco, N.: Clarifying the Trustwave CA policy update. http://blog.spiderlabs.com/2012/02/clarifying-the-trustwave-ca-policy-update.html (February 2012)
69. Perl, H., Fahl, S., Smith, M.: You won't be needing these any more: On removing unused certificates from trust stores. In: Financial Cryptography and Data Security, pp. 307–315. Springer (2014)
70. Reiter, M.K., Stubblebine, S.G.: Authentication metric analysis and design. ACM Transactions on Information and System Security 2(2), 138–158 (May 1999)
71. Ryan, M.D.: Enhanced certificate transparency and end-to-end encrypted mail. In: Proceedings of the Network and Distributed Security Symposium (NDSS). NDSS (February 2014)
72. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in Bitcoin. arXiv:1507.06183v2 [cs.CR] (July 2015)
73. Sleevi, R.: Sustaining digital certificate security. https://googleonlinesecurity.blogspot.com/2015/10/sustaining-digital-certificate-security.html (October 2015)
74. Sleevi, R., Gautier, F., Hurst, R.M., Salz, R., Messeri, E.: Upcoming CT log removal: Certly.IO. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/nyRtXSWSeaM (April 2016)
75. Sleevi, R., Liang, D., Bowen, P.: Pending CT log disqualification: WoSign. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/LEOAlqMHISQ (April 2016)
76. Sleevi, R., Ronca, R., Gautier, F.: Upcoming CT log removal: Izenpe. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA (May 2016)
77. Somogyi, S., Eijdenberg, A.: Improved digital certificate security. https://googleonlinesecurity.blogspot.com/2015/09/improved-digital-certificate-security.html (September 2015)
78. Szalachowski, P., Matsumoto, S., Perrig, A.: PoliCert: Secure and flexible TLS certificate management. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (2014)
79. Topalovic, E., Saeta, B., Huang, L.S., Jackson, C., Boneh, D.: Towards short-lived certificates. Web 2.0 Security and Privacy (2012)
80. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: Improving SSH-style host authentication with multi-path probing. In: USENIX Annual Technical Conference. pp. 321–334 (June 2008)
81. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. White Paper (2015)