Checks and Balances: A Tripartite Public Key Infrastructure for Secure Web-based Connections

Jing Chen, Shixiong Yao, Quan Yuan, Ruiying Du, Guoliang Xue

***Abstract*—Recent real-world attacks against Certification Authorities (CAs) and fraudulently issued certificates arouse the public to rethink the security of public key infrastructure for web-based connections. To distribute the trust of CAs, notaries, as an independent party, are introduced to record certificates, and a client can request an audit proof of certificates from notaries directly. However, there are two challenges. On one hand, existing works consider the security of notaries insufficiently. Due to lack of systematic mutual verification, notaries might bring safety bottlenecks. On the other hand, the service of these works is not sustainable, when any party leaks its private key or fails. In this paper, we propose a Tripartite Public Key Infrastructure (TriPKI), using Certificates Authorities, Integrity Log Servers, and Domain Name Servers, to provide a basis for establishing secure SSL/TLS connections. Specifically, we apply checks-and balances among those three parties in the structure to make them verify mutually, which avoids any single party compromise. Furthermore, we design a collaborative certificate management scheme to provide sustainable services. The security analysis and experiment results demonstrate that our scheme is suitable for practical usage with moderate overhead.**

***Index Terms*—Public Key Infrastructure; DNS-based; Mutual Verification**

I. Introduction

Transport Layer Security (TLS) and its predecessor Secure Socket Layer (SSL) are two most popular cryptographic protocols that provide communication security over a computer network. Nowadays, secure web-based connections through SSL/TLS have been globally adopted in various online services, e.g. e-business, e-banking, and e-government. Due to its critical role in those services, SSL/TLS has captured great attention from both attackers and researchers [1].

In SSL/TLS, authentication and secure connection establishment are built based on Public Key Infrastructure (PKI), and the core of PKI is the ecosystem of Certification Authorities (CAs) which are responsible for issuing and maintaining SSL certificates [2]. However, recent compelling real-world attacks have demonstrated existing CAs' vulnerability. For example, some well-known CAs, TurkTrust [3], CNNIC [4], DSDtestProvider & eDellRoot [5] were reportedly *compromised*, which means that their private keys are leaked or they are out of control in some cases. Also, at least 500 SSL/TLS server certificates were fraudulently issued for many famous domains, e.g. `mail.google.com`, `www.google.com`, `login.yahoo.com`, `login.skype.com`, `login.live.com`, and `addons.mozilla.org`. Meanwhile, due to the lack of safety consciousness, many users tend to ignore warnings of self-signed certificates from browsers. As a result, these fraudulent and self-signed certificates can be used by adversaries to mount Man-in-the-Middle (MitM) attacks.

To solve such security issues of PKIs in SSL/TLS, researchers have designed a variety of proposals, which can be generally classified into two categories: *non-notary-based* and *notary-based*. Non-notary-based schemes mainly focus on defining certificate policies and further managing CAs to enhance system security, where the validity of certificates depends on the confidence level of CAs. In contrast, notary-based schemes distribute trust in CAs via employing notaries to validate certificates issued by CAs. Obviously, introducing notaries can further assist in detecting the compromised CAs. In 2013, Kim et al. [6] proposed an Accountable Key Infrastructure (AKI), which utilized a set of notaries to check certificates' authenticity and validity, as well as to provide certificate revocation mechanism. In 2014, based on AKI, Basin et al. [7] designed an Attack Resilient Public-Key Infrastructure (ARPKI) which improved security by using multiple CAs to sign and validate certificates in a serial mode.

However, there still exist two challenges. First, most of notary-based schemes heavily rely on notaries/validators for validating certificates and verifying CAs' behaviors. However, the employed notaries' security is not systematically discussed. More specifically, these schemes consider *mutual verification* insufficiently. Second, if any assigned CA is compromised or damaged physically, the serving PKI will be corrupted and the whole system needs to be reset. Even though ARPKI [7] adopts multiple CAs as an authentication chain to avoid single point of failure, the chain will be rebuild when a CA is compromised. Therefore, these CAs may become potential targets of Denial of Service (DoS) attacks. Thus, *sustainable service* is still missing. Besides, multi-signature schemes can be utilized to solve the single point failure. However, these schemes such as Batch Identification Game Model (BIGM) [8] can not provide adjustable trust.

To tackle these challenges, in this paper, we propose a Tripartite Public Key Infrastructure (TriPKI) for secure web-based connections. The main idea is to apply checks-and-

Chen and Yao are with State Key Laboratory of Software Engineering, Computer School, Wuhan University, Wuhan, China 430072. Email: {chenjing, derekysx}@whu.edu.cn. Yuan is with University of Texas-Permian Basin, TX, 79762. Email: dantes.yuan@gmail.com. Du is with Collaborative Innovation Center of Geospatial Technology, Wuhan, China 430072. Email: duraying@whu.edu.cn. Xue is with Arizona State University, Tempe, AZ 85287. Email: xue@asu.edu. This research was supported in part by NSF grants 1457262 and 1421685, the National Natural Science Foundation of China under Grant No. 61272451, 61173154, 61232002, 61373169, 61332019, and the Major State Basic Research Development Program of China under Grant No. 2014CB340600. The information reported here does not reflect the position or the policy of the funding agencies. The corresponding author is Shixiong Yao.

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

balances strategy to make different parties verify mutually. Specifically, inspired by DANE [9], besides CAs (issuing, updating, and revoking certificates) and ILSs (Integrity Log Severs, auditing certificates), we utilize Domain Name Systems (DNSs) as the third party to form this tripartite PKI for certificate management. In our scheme, DNSs assist CAs to manage certificates and verify other parties' behaviors. Instead of protecting DNSs' security separately as in DNSSEC [10], system security is guaranteed by using the checks-and-balances principle among CAs, ILSs, and DNSs. Based on this design, any single party compromise can be efficiently detected. Also, our infrastructure supports more flexible mechanisms on certificate managements. The main contributions are as follows.

- We first propose a novel TriPKI with DNSs, CAs, and ILSs for secure web-based connections. The checks-and-balances strategy is applied among those three parties to effectively detect any single party compromise, which enhances PKIs' attack-resilience.
- We design a distributed collaborative certificate management scheme where each party consists of multiple entities, and the trust of each party is distributed among its entities. Unlike ARPKI [7], every entity in our scheme is substitutable. Even if some entities are compromised or failed, our scheme can still provide sustainable services and tolerate service vulnerability.
- We prove the sustainable-service ability and security of TriPKI in different cases.
- We implement a proof-of-concept prototype and evaluate the performance of certificate operations in practice.

The reminder of the paper is organized as follows. In Section II, we review the related work. Section III describes system model, threat model, design goals, and preliminaries. We present the details of our TriPKI at system level and algorithm level in Section IV, and analyze the security of TriPKI in Section V. Section VI evaluates the performance of our scheme, and we conclude the paper in Section VII.

II. Related Work

As the basis of SSL/TLS, the security issues of PKIs have been studied extensively. As illustrated in Fig. 1, existing schemes can be divided into two main categories: *non-notary-based* and *notary-based*.

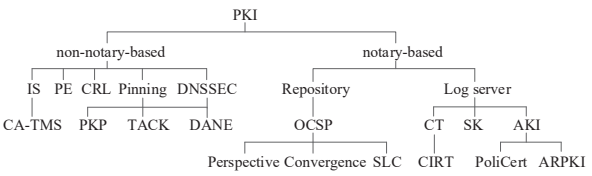


Fig. 1. Categories of PKI schemes

Non-notary-based approaches. Non-notary-based approaches enhance system security by improving certificate policies and CAs' management. Policy engine (PE) [11] allows clients to make their own trust decisions on certificates, rather than to rely on a global authority. This approach can be viewed as a complement of other PKI approaches. Individualized Set (IS) [12] permits a client to choose CAs from a fixed individual set. CA Trust Manage System (CA-TMS) [13] maintains a minimal set of trusted CAs by a reputation system. Even though it reduces the possibility of attacks, the solution for validating malicious certificates is not provided. To prevent clients from establishing a SSL/TLS connection with revoked certificates, Certificate Revocation List (CRL) is employed [14].

Google pioneered a technology known as public-key pinning which was first proposed in November 2011, and revised in October 2014. The basic idea is that, whenever a browser connects to a site using SSL/TLS, the site can claim that one or several public keys are pinned for that domain. Pinning-based schemes, such as Public Key Pinning (PKP) [15] has the advantages of simplicity and easy deployment, whereas they have a bootstrapping problem. For instance, an adversary can lock out a legitimate owner of a site by pinning a fraudulent public key when a client first visits the domain. To properly generalize public-key pinning to arbitrary domains, the proposal called DNS-based Authentication of Named Entities (DANE) [9] enables domain owners to assert some constraints, i.e. a list of acceptable CAs for issuing domains' certificates, specific acceptable certificates, and specific trusted anchors to validate certificates. Unfortunately, because of adhering to the DNS hierarchy, DANE falls short of the non-hierarchical trust, and its security depends on DNS SECurity (DNSSEC) [10] in which the root DNS is a security bottleneck.

Notary-based approaches. Notary-based approaches employ extra resources, such as notaries, to distribute trust in CAs for certificate validation. According to the function of notaries, notary-based approaches can be further divided into two subcategories: *repository-based* and *log-server-based*. In repository-based approaches, repositories provide clients the stored public keys or certificates to verify domains' authenticity. To resolve this online validation requirement, the Online Certificate Status Protocol (OCSP) [16] allows clients to check the domains' certificate status by querying CAs' OCSP servers. However, with insufficient consideration, OCSP has some flaws in the aspects of security, privacy, and efficiency, which are respectively resolved by Perspecitves [17], Convergence [18], and Short lived certificate (SLC) [19]. However, all these approaches cannot record CAs' behaviors which is an important clue for detecting the security breaches of PKIs.

In log-server-based approaches, domain owners not only record their certificates to public log servers, but also create accountability for CAs' actions [20]. For instance, in Sovereign Keys (SK) [21], each domain owner signs its SSL/TLS public key by a sovereign key pair, and this behavior is logged in a timeline server which only allows read and append operations. Whereas, it requires that the server must search the entire database in response to the queries of each client, which increases latency and sacrifices privacy. To improve the query efficiency, Certificate Transparency (CT) [22] introduces a

Merkle hash tree structure in log servers, and provides the audit proof to clients for a SSL/TLS connection establishment. However, CT does not consider how to handle the fraudulent certificates issued by compromised CAs. Certificate Issuance and Revocation Transparency (CIRT) [23] solves this problem by proposing an efficient revocation mechanism for CT, but it requires a client to change a new identity once its key is lost. To distribute trust among independent entities, besides CAs and notaries called Integrity Log Servers (ILSs), Accountable Key Infrastructure (AKI) [6] also setups a validator to verify ILS operations, and detect misbehaviors. However, without sufficient mutual verification, the security of the ILSs and the validator is vulnerable. PoliCert [24] extends the Accountable Key Infrastructure by allowing domains to restrict their own certificate properties of SSL/TLS connections. Nonetheless, it still does not specify the mechanisms for detecting and disseminating log misbehaviors. Attack Resilient Public-key Infrastructure (ARPKI) [7] implements mutual verification between CAs and ILSs by organizing these entities in a serial mode, which requires that each entity provides a legal signature. However, it does not sufficiently consider the security of validators. Moreover, the fixed sequential signature technique extends the attack interface of PKIs. In another word, if adversaries can compromise any entity, PKI service must be terminated and rearranged. In this paper, TriPKI focuses on designing a PKI with improved accountability, flexible mutual verification, and sustainable service.

III. Problem Statement

A. System model

In our scheme, there exist three parties: CAs, ILSs, and DNSs, to share the responsibility of managing certificates for domains. The descriptions of each entity in different parties of TriPKI are as follows.

- **Domain**. A named entity, with which clients desire to establish secure connections, usually refers to a website.

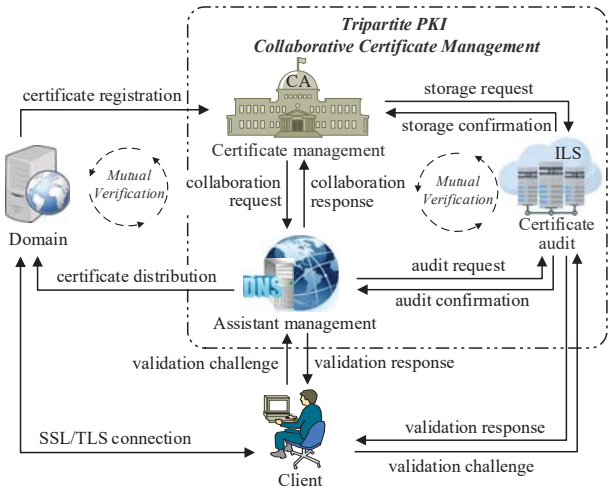


Fig. 2. The framework of TriPKI

- **Client**. An entity who intends to establish TLS connections with a domain.
- **Certification Authority (CA)**. An entity takes charge of authorizing, updating, and revoking $X.509$ certificates. Note that in TriPKI, multiple CAs authenticate domains in a collaborative way.
- **Integrity Log Server (ILS)**. Each ILS maintains an integrity tree which logs certificates, and updates the tree to keep consistent with each other. The integrity tree is implemented as a Merkle hash tree [25], whose leaves are the certificates of corresponding domains.
- **Domain Name System (DNS)**. A DNS server takes accountability for not only binding IP addresses and domain names, but also assisting CAs and verifying CAs' and ILSs' misbehaviors.

As shown in Fig. 2, three parties compose the TriPKI system and manage certificates collaboratively, while the main responsibilities of CAs, DNSs, and ILSs are certificate management, assistant management, and certificate audit, respectively. These three parties form a triangle structure and verify the behaviors mutually. In addition, the domain, CAs, and DNSs also constitute a verification loop. Clients can challenge CAs or ILSs for certificate validation determined by the three-way joint.

B. Threat model

From the practical perspective, our scheme should satisfy security requirements in the case of the following threat model.

- Many famous CAs, e.g. CNNIC and eDellRoot, have suffered attacks in the real world. Thus, we assume that adversaries can compromise any entity.
- Adversaries can compromise multiple entities simultaneously. However, for a PKI that satisfies any nontrivial security property, the number of compromised entities should have some constraints. Especially, for CAs and ILSs, the number of compromised entities should not exceed their predefined threshold values.
- We also assume that adversaries can eavesdrop, tamper, and forge messages, when entities communicate with each other in untrusted networks.

C. Design goals

Our design goal is to develop a tripartite public key infrastructure to establish secure SSL/TLS connections between domains and clients, which has the following properties.

- **Mutual verification**: The information flows among CAs, DNSs, and ILSs form a loop, which means that the misbehaviors from any party will be detected. This checks-and-balances framework always protects the system from attacks launched by any compromised party.
- **Sustainable-service**: Each entity in any party only has limited trust, and can be replaced by others. The whole system continues working properly even if parts of entities are compromised or broken down.
- **Random recruitment**: The involved CAs and ILSs in certificate management process are not fixed. Instead, they are randomly selected and can be replaced. This can

efficiently prevent adversaries from acquiring valuable information and launching DoS attacks.

- **Adjustable trust**: The system is allowed to change the number of entities in each party. Furthermore, the system manager can assign and adjust trust in CAs and DNSs at will, which determines the maximum quantity of compromised CAs and DNSs.

D. Preliminaries

***Definition** 1:* **Discrete-Logarithm Assumption.** Assume $\mathbb{G}$ is a cyclic group of order q, g is a generator of $\mathbb{G}$, and a uniform $b \in \mathbb{G}$. The discrete-logarithm problem is given g and b to compute $k = \log_g b$. We say that the discrete-logarithm assumption holds in the group $\mathbb{G}$, if for all probabilistic polynomial-time algorithms, the probability of solving this problem is negligible.

***Definition** 2:* (k, n)**-Threshold Signature Algorithm.** It consists of three parts: threshold key generation `Key_Gen`, threshold signature generation `TSign_Gen`, and threshold signature verification `TSign_Ver`.

- `Key_Gen`: this is a randomized interactive protocol which is run by each of n shareholders. With the input of the global information I, the protocol returns the group public key PK and a pair of (PK_i, SK_i) for each shareholder.
- `TSign_Gen`: this is a randomized interactive protocol divided into two parts: *partial signature generation* and *threshold signature generation*. In the former one, the process returns a parital signature σ_i, when taking I and a message M as input. The latter takes $I, \{\sigma_1, \ldots, \sigma_k\}$ as input, outputs the threshold signature (σ, M).
- `TSign_Ver`: this is a deterministic algorithm with input I, M, and σ. It returns 1 when (σ, M) is valid, or it returns 0.

IV. TriPKI: A Tripartite PKI

A. Main idea

With the above discussion, we see that one-way supervision or single sequential verification is vulnerable and inflexible. In our opinion, a tripartite structure is a more stable framework for mutual verification. Thus, besides CAs and ILSs, we introduce DNSs as the third party into our framework with the following considerations. First of all, since DNSs maintain mapping information of network addresses and domains, they are suitable to be used to assist certificate management. Secondly, DNS servers are distributed in different physical locations with different security configurations, which makes it challenging for adversaries to compromise all DNSs. Thus, our framework has good robustness. Thirdly, the mutual verification among three parties can resist attacks even if all entities in a single party are corrupted.

To further reduce trust dependency and enhance service reliability, in TriPKI, all entities in each party are substitutable. Our collaborative certificate management scheme not only considers the cooperation between CAs and DNSs based on threshold signature, but also achieves mutual verification among CAs, DNSs, and ILSs. The used notations in TriPKI is described in TABLE I.

TABLE I
THE NOTATIONS IN TRIPKI

Notation	Description
x_i	the ID of shareholder U_i
k, n	the threshold value and the number of shareholders
$f_i(x)$	a $k-1$ degree polynomial selected by U_i
λ_i	$\lambda_i = \sum_{j=1}^{n} f_j(x_i)$
h_i, d_i	random numbers selected by U_i in the process of `Key_Gen` and `PSign_Gen`
r_i	$r_i = g^{h_i}$ which is calculated by U_i and broadcasted to other $n-1$ shareholders.
R	$R = \prod_{i=1}^{n} r_i \bmod p$, all shareholders own it.
PK_i, SK_i	the public key and the private key of a shareholder U_i.
PK	the public key of the group.
T	a set of shareholders who participate in the process of threshold signature generation
C_i	Lagrangian interpolation coefficient of x_i
σ_i	the partial signature generated by U_i
M	the message needs to be signed
σ	the threshold signature composed by k partial signatures
l, m, w	the number of the CAs, DNSs, and ILSs.
$\{\cdot\}_{K_i^{-1}}$	a message signed by the private key of user i
$\{\cdot\}_{\sigma^{k-1}}, \{\cdot\}_{\sigma^k}$	$k-1$ and k threshold signature, respectively

B. Certificate format

In our scheme, certificates contain several extensions over standard X.509 certificates and feature the following additional fields:

- CA_List. The list of trusted CAs for certificate management;
- ILS_List. The list of trusted ILSs to log the certificate;
- DNS_List. The list of trusted DNSs for assisting the certificate management.

C. Collaborative certificate management scheme

For clarity, we present TriPKI in two levels: *System Level* and *Algorithm Level*. The former describes the implementation of TriPKI including *Initialization*, *authorization*, *update*, *revocation*, *validation*, and *dynamic trust management*. The latter focuses on the algorithms which support the operations at system level.

1) System level:

a) ***Initialization**:* The major construct of TriPKI consists of l CAs, m DNSs, and w ILSs. The domain A chooses its own CA_List, DNS_List, and ILS_List from this group. In our scheme, since CAs and DNSs are the shareholders in the threshold signature algorithm, the total amount of shareholders in the algorithm is $n = l + m$ and the threshold value is k ($k \leq n$). These shareholders select three public parameters: big primes p, q (q is a prime factor of $p-1$), and a generator g (the order of g is q in the $\mathbb{Z}_p$). Then, they run `Key_Gen` (described in Section IV-C2a) to generate some initial information, such as their own public and private key pairs, such as (PK_{CA_i}, SK_{CA_i}) and (PK_{DNS_i}, SK_{DNS_i}), and the group public key PK. To resist MitM attacks, we also utilize the RSA-2048 algorithm in some steps. Besides their own RSA private keys, e.g. $K_{CA_i}^{-1}$, $K_{DNS_i}^{-1}$, and $K_{ILS_i}^{-1}$, CAs,

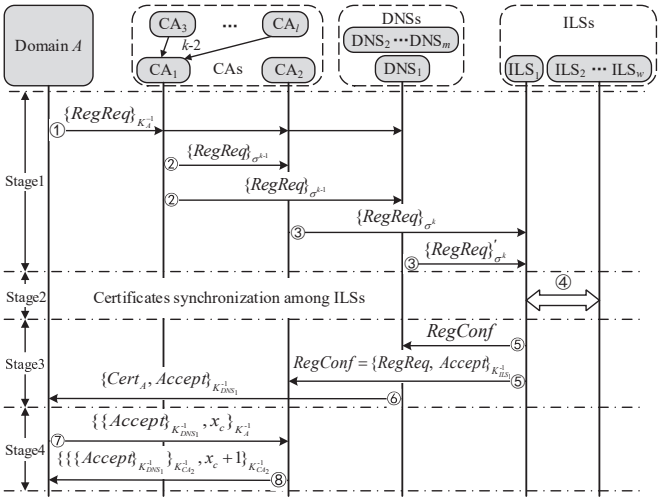


Fig. 3. The information flows in TriPKI

DNSs, and ILSs are equipped with RSA public keys of all CAs, DNSs, ILSs, e.g. K_{CA_i}, K_{DNS_i}, and K_{ILS_i}. In addition, domain A owns its public and private key pair (K_A, K_A^{-1}), and all of the public keys mentioned above.

b) ***Certificate authorization****:* In TriPKI, the mutual verification occurs among two CAs, one DNS, and one ILS, represented as CA_1, CA_2, DNS_1, and ILS_1. The information flow is exhibited in Fig. 3. According to the certificate operations, this process can be divided into four stages.

Stage 1: *Signature generation* (step:1-3)

From Fig. 4, we observe that this stage includes three steps. In step 1, domain A generates its certificate and assigns a trusted CA denoted as CA_1. Then, it sends a registration request $RegReq$ to all CAs and DNS_1. After that, all of the shareholders which have received $RegReq$ ensure that they are in CA_list or DNS_list and then authenticate the identity of domain A. Furthermore, CA_1 randomly selects CA_2 and other $k-2$ CAs from CA_List, and DNS_1 from DNS_list. These CAs and DNS_1 form a threshold signature set T which contains $k+1$ shareholders. By applying the algorithm `PSign_Gen` (described in Section IV-C2b), all of these $k+1$ shareholders generate their own partial signatures. Then the $k-2$ CAs send their own partial signatures to CA_1. If these partial signatures are legal and correct, CA_1 combines them with its own partial signature to generate a $(k-1,n)$-threshold signature, which is $\{RegReq\}_{\sigma^{k-1}}$ via `TSign_Gen` (described in Section IV-C2d). Note that each partial signature is verified by `PSign_Ver` (described in Section IV-C2c) and the step will be terminated if there exists any verification failure. In step 2, CA_1 transmits the $\{RegReq\}_{\sigma^{k-1}}$ to CA_2 and DNS_1. In step 3, if $\{RegReq\}_{\sigma^{k-1}}$ is legal, CA_2 and DNS_1 generate the full threshold signatures $\{RegReq\}_{\sigma^k}$ and $\{RegReq\}'_{\sigma^k}$, and send them to ILS_1.

Stage 2: *Certificate synchronization* (step:4)

The main task in this stage is that ILS_1 synchronizes the new certificate among all ILSs. To provide the audit proof, we store certificates in ILSs with Merkle hash trees as in ARPKI [7]. The process of certificate synchronization is shown in Fig. 5. After ILS_1 receives $\{RegReq\}_{\sigma^k}$ and $\{RegReq\}'_{\sigma^k}$, it ensures itself in ILS_List. Then, it estimates the legality of the request by `TSign_Ver` (described in Section IV-C2e) and checks the consistency of them. If the verification succeeds, it broadcasts a synchronization request $SynReq$ to all ILSs. Until ILS_1 receives ξ synchronization responses $SynResp$, which means that they agree to store the certificate, it broadcasts a synchronization commitment $SynCommit$ to inform all ILSs to store the certificate. The synchronization process is completed when ILS_1 obtains synchronization acknowledgments $SynAck$ from ξ ILSs, and then it generates a notification to all ILSs.

Stage 3: *Certificate distribution* (step:5-6) This stage includes two steps which are showed in Fig. 6. In step 5, both DNS_1 and CA_2 receive the registration confirmation $RegConf$ from ILS_1. To prove the consistency of ILSs, the $List(SynAck)$ is attached in $RegConf$. In step 6, if the

Step 1:		
1	A:	Generate $Cert_A$
2	$A \rightarrow$ all CAs:	$RegReq = \{Cert_A, CA_1, CA_2, DNS_1, ILS_1\}_{K_A^{-1}}$
	$A \rightarrow DNS_1$:	$RegReq$
3	CA_1, DNS_1:	Ensure CA_1 in CA_List, verify the signature in $RegReq$, and randomly select CA_i in CA_List
4	$CA_i \rightarrow CA_1$:	$\{RegReq\}_{SK_{CA_i}}$ (single partial signature)
5	CA_1:	$\{RegReq\}_{\sigma^{k-1}}$ ($k-1$ threshold signature)
Step 2:		
6	CA_1:	Ensure CA_2 in CA_List and DNS_1 in DNS_List
7	$CA_1 \rightarrow CA_2$:	$\{RegReq\}_{\sigma^{k-1}}$
	$CA_1 \rightarrow DNS_1$:	$\{RegReq\}_{\sigma^{k-1}}$
Step 3:		
8	CA_2, DNS_1:	get $\{RegReq\}_{\sigma^k}$, $\{RegReq\}'_{\sigma^k}$ from $\{RegReq\}_{\sigma^{k-1}}$, then verify them
	CA_2, DNS_1:	Ensure ILS_1 in ILS_List
9	$CA_2 \rightarrow ILS_1$:	$\{RegReq\}_{\sigma^k}$
	$DNS_1 \rightarrow ILS_1$:	$\{RegReq\}'_{\sigma^k}$

Fig. 4. Stage 1 - Signature generation

Step 4:		
1	ILS_1:	Ensure ILS_1 in ILS_List, Verify signatures in $\{RegReq\}_{\sigma^k}$ and $\{RegReq\}'_{\sigma^k}$
2	$ILS_1 \rightarrow ILS_w$:	$SynReq = \{RegReq\}_{K_{ILS_1}^{-1}}$
3	ILS_w:	verify signatures in $SynReq$, Ensure no $Cert_A$ is registered for domain A
4	$ILS_w \rightarrow ILS_1$:	$SynResp = \{H(RegReq)\}_{K_{ILS_w}^{-1}}$
5	ILS_1:	collect $SynResp$ from more than ξ ILSs
6	$ILS_1 \rightarrow ILS_w$:	$SynCommit = \{H(RegReq)\}_{K_{ILS_1}^{-1}}$
7	$ILS_w \rightarrow ILS_1$:	$SynAck = \{H(RegReq)\}_{K_{ILS_w}^{-1}}$
8	ILS_1:	collect $SynAck$ from more than ξ ILSs $Accep = \{H(Cert_A)\}_{K_{ILS_1}^{-1}}$

Fig. 5. Stage 2 - Certificate synchronization

Step 5:		
1	$ILS_1 \rightarrow DNS_1$:	$RegConf = \{RegReq, Accept, List(SynAck)\}_{K_{ILS_1}^{-1}}$
	$ILS_1 \rightarrow CA_2$:	$RegConf$
Step 6:		
2	DNS_1, CA_2:	both of them verify $RegConf$, and record it, respectively.
3	$DNS_1 \rightarrow A$:	$\{Cert_A, Accept\}_{K_{DNS_1}^{-1}}$

Fig. 6. Stage 3 - Certificate distribution

Step 7:
1 $A \rightarrow CA_2$: $RegChall = \{\{Accept\}_{K^{-1}_{DNS_1}}, x_c\}_{K^{-1}_A}$

Step 8:
2 $CA_2 \rightarrow A$: $RegResp = \{\{\{Accept\}_{K^{-1}_{DNS_1}}\}_{K^{-1}_{CA_2}}, x_c+1\}_{K^{-1}_{CA_2}}$

Fig. 7. Stage 4 - Challenge and response

$RegConf$ is legal, DNS_1 and CA_2 record it. Then, DNS_1 signs and sends it to domain A.

Stage 4: *Challenge and response* (step:7-8)

In this stage, as shown in Fig. 7, domain A sends a registration challenge $RegChall$ signed by itself to CA_2, where $RegChall$ consists of $\{Accept\}_{K^{-1}_{DNS_1}}$ and a random number x_c. Then, CA_2 returns a signed registration response $RegResp$ composed of $\{\{Accept\}_{K^{-1}_{DNS_1}}\}_{K^{-1}_{CA_2}}$ and $x_c + 1$.

After the above four stages, domain A can show a certificate $Cert_A$ with a legal signature to a client to establish a TLS connection. In terms of this framework, the mutual verification occurs in two triangle structures. One is among two CAs (e.g. CA_1 and CA_2), DNS_1, and ILS_1, and the other is among DNS_1, domain A, and CA_2.

c) ***Certificate update****:* This process is similar to the process of certificate authorization. Before a certificate expires, domain A generates a new key pair and a new certificate $Cert_u$. It signs the new certificate with the old private key, and starts an update request $\{Cert_A, \{\{Accept\}_{K^{-1}_{DNS_1}}\}_{K^{-1}_{CA_2}}, Cert_u, CA_i, CA_j, DNS_k, ILS_u\}_{K^{-1}_A}$, where CA_i, CA_j, DNS_k, ILS_u are the trusted entities designated by domain A in CA_List, DNS_List, and ILS_List. CA_i, CA_j, ILS_u can be different from CA_1, CA_2, ILS_1, but DNS_k must be the same as DNS_1 to maintain the consistency of certificates between the DNS and ILSs. Note that $\{\{Accept\}_{K^{-1}_{DNS_1}}\}_{K^{-1}_{CA_2}}$ is signed by CA_2, DNS_1, and ILS_1 in the process of certificate authorization, and it is the legality proof of the old certificate $Cert_A$. After the mutual verification, if all operations are correct, domain A can receive a new signature for the new certificate $Cert_u$ while all ILSs and DNS_1 update the certificate uniformly.

d) ***Certificate revocation****:* When a domain's private key is compromised or lost, its certificate should be removed, and a notification of certificate revocation needs to be sent to TriPKI system. Generally, in this case, if a malicious certificate is claimed and the authenticity of the notification is confirmed by k random CAs, these CAs will launch the mutual verification as the process of certification authorization, while the goal is to abolish the malicious certificate. Specially, the $RegReq$ turns into a revocation request $RevReq$, although the content is the same, e.g. $RevReq = \{Cert_A, CA_1, CA_2, DNS_1, ILS_1\}_{K^{-1}_A}$. As a result, all ILSs and the corresponding DNS delete the record of the malicious certificate in their Merkle hash trees, respectively.

e) ***Certificate validation****:* If a client tries to establish a TLS connection with a domain A for the first time, it should validate $Cert_A$ in three aspects: (1) $Cert_A$ is signed by the entities in CA_List and DNS_List, (2) $Cert_A$ has not expired ($Cert_A$ is stored in merkle hash tree), (3) the threshold signature of $RegReq_{\sigma^k}$ is correct. The client can send a certificate validation challenge to DNS_1 and an ILS. On one hand, if DNS_1 has authorized and distributed this certificate, it returns a response about the certifcate threshold signature to the client. On the other hand, the ILS will return the root hash and auxiliary hashes of the merkle hash tree if the certificate has been stored. If both of the validation responses are correct, the client can establish a security connection with domain A. Thus, TriPKI can prevent single compromised entity from deceiving the client with a forged validation response. To maintain the consistency of DNSs' certificate records, DNSs should execute the synchronize process as ILSs at intervals.

f) ***Dynamic trust management****:* Besides fraudulent certificates, entities in TriPKI also may fail or become a target of attacks. Thus, the dynamic trust management is necessary, which allows to distribute trust to corresponding entities on demand in terms of shareholders' secret information.

We mainly consider two cases. One case is that TriPKI may adjust the security level with the actual demand. For example, if we intend to achieve the maximum security level, we can request all shareholders to participate in certificate management. The minimum requirement of the mutual verification needs two CAs and one DNS. To implement this function, each shareholder needs to recalculate its secret information by `Threshold_Up` (described in Section IV-C2f). The other case is that some entities join or exit TriPKI due to system expansion or entities' compromise or failure. If the transformation only occurs in ILSs, it is easy to add or delete some ILS servers directly. However, if it happens in CAs and DNSs parties, the secret information of each shareholder should be updated by `Membership_Up` (described in Section IV-C2g).

2) Algorithm level: Algorithm level is implemented based on [26], which presents a generic construction of threshold signature schemes. Due to the compromising risk of each entity, our implementation has no trusted dealer. The operations include seven algorithms: `Key_Gen`, `PSign_Gen`, `PSign_Ver`, `TSign_Gen`, `TSign_Ver`, `Threshold_Up`, `Membership_Up`.

a) ***Key_Gen****:* Firstly, every shareholder selects an public ID x_i and a $k-1$ degree polynomial $f_i(x) \bmod q$, then calculates $\lambda_{i,j} = f_i(x_j) \bmod q$ ($j \in \{1, \ldots, n\}$) for other $n-1$ shareholders and broadcasts them and keeps $\lambda_{i,i}$ for itself. After U_i receives all $\lambda_{j,i}$, it can get

$$\lambda_i = \sum_{j=1}^{n} \lambda_{j,i} \bmod q = \sum_{j=1}^{n} f_j(x_i) \bmod q, \qquad (1)$$

Now, we define a new function $F(x) = \sum_{i=1}^{n} f_i(x) \bmod q$, and each U_i can get $\lambda_i = F(x_i)$. U_i selects a random number h_i from $\{1, \ldots, q-1\}$, and computes $SK_i = \lambda_i h_i \bmod q$ as its *private key*, $PK_i = g^{SK_i} \bmod p$ as its *public key*. U_i gets $r_i = g^{h_i} \bmod p$, and broadcasts it to other shareholders. Every shareholder can get $R = \prod_{i=1}^{n} r_i \bmod p$. k shareholders can recover $F(0) = [\sum_{i=1}^{k} F(x_i) \cdot C_i] \bmod q$, $C_i = \prod_{j=1, j\neq i}^{k} \frac{(-x_j)}{(x_i - x_j)}$. $PK = g^{F(0)} \times R \bmod p$ acts as the *group public key*.

*b) **PSign_Gen**:* The shareholders, who participate in the threshold signature generation process, compose a set T. Each of them selects a random number d_i from $\{1, \ldots, q-1\}$. U_i can calculate $D_i = g^{d_i} \bmod p$, $v_i = g^{d_i \times h_i^{-1}} \bmod p$, where h_i^{-1} is an inverse element of h_i in a prime field $\mathbb{Z}_p$. Each U_i broadcasts D_i and v_i to other shareholders. Then, U_i can get $V = \prod_{U_j \in T} v_j \bmod p$.

Through the congruence $h_i s_i = (h_i \lambda_i C_i) \times h(M) - V \times d_i \bmod q$, s_i could be deduced as follows:

$$s_i = (\lambda_i C_i) \times h(M) - V \times d_i \times h_i^{-1} \bmod q, \tag{2}$$

where C_i is the Lagrangian interpolation coefficient, and $h(\cdot)$ is a one-way hash function. U_i sends the partial signature $\sigma_i = (M, r_i, s_i, PK_i, D_i, i)$ to the designated combiner. Note that i is used to deduce C_i and to determine the signer's identity.

*c) **PSign_Ver**:* The combiner can verify the σ_i by the following equation.

$$r_i^{s_i} (D_i)^V \stackrel{?}{=} (PK_i)^{h(M) C_i} \bmod p. \tag{3}$$

*d) **TSign_Gen**:* After receiving all σ_i, the combiner generates the combination of them by equation (4).

$$S = \sum_{i=1}^{k} s_i = \sum_{U_i \in T} (\lambda_i C_i) h(M) - V \sum_{U_i \in T} (d_i h_i^{-1}) \bmod q. \tag{4}$$

The threshold signature is $\sigma = (M, S, V, R)$.

*e) **TSign_Ver**:* Everyone can verify the threshold signature by the following equation:

$$g^S V^V \stackrel{?}{=} (PK \times R^{-1})^{h(M)} \bmod p. \tag{5}$$

where R^{-1} is an inverse element of R in $\mathbb{Z}_p$.

*f) **Threshold_Up**:* This algorithm can adjust the threshold value k as follows.

- The group shareholder U_i selects a $\varphi - 1$ degree polynomial $f_i'(x) \bmod q$, and then calculates $\lambda'_{i,j} = f_i'(x_j) \bmod q$ ($j \in \{1, \ldots, n\}$) for other shareholders and broadcasts them. Every shareholder calculates λ'_i as equation (1).
- Every shareholder updates $F'(x)$ and gets its new public and private keys by $PK'_i = g^{SK'_i} \bmod p$ and $SK'_i = \lambda'_i h_i \bmod q$.
- φ shareholders can recover $F'(0)$. Then the group public and private keys can be replaced by $PK' = g^{F'(0)} \times R \bmod p$.

After these steps, the threshold value k is replaced by φ. In addition, the parameters $v_i^{'} = g^{d'_i \times (h_i)^{-1}} \bmod p$, $V^{'} = \prod_{U_j \in S} v_j^{'} \bmod p$ also need to be updated in partially signature generation stage.

*g) **Membership_Up**:* To simplify description, we assume that the joined shareholder is U_u and its ID is x_u.

- U_u exchanges $\lambda_{u,j}$ with each U_j. Then all of them update their $\lambda'_j = \lambda_j + \lambda_{u,j}$, U_u gets $\lambda_u = \sum_{j=1}^{n^{'}} \lambda_{j,u}$.
- U_u generates its private key: $SK_u = \lambda_u \cdot h_u \bmod q$ and the public key: $PK_u = g^{SK_u} \bmod p$. Every shareholder updates its private and public key pair by $SK'_i = \lambda'_i h_i \bmod q$ and $PK'_i = g^{SK'_i} \bmod p$. U_u broadcasts r_u with its signature to other shareholders.
- Every shareholder verifies r_u and updates R by $R^{'} = R \cdot r_u$. U_u randomly selects a shareholder to request the current R for calculating its own $R^{'}$.
- $F^{'}(x)$ also needs to be updated by $F^{'}(x) = F(x) + f_u(x) \bmod q$. The current group public key is replaced by $PK = g^{F^{'}(0)} \times R^{'} \bmod p$, where $F^{'}(0)$ also should be updated.

V. Security Analysis

***Lemma** 1:* In TriPKI, if discrete-logarithm problem is hard, a probabilistic polynomial-time (PPT) adversary cannot attack our threshold signature scheme. In other words, it cannot forge a partial or a group signature.

Proof: We assume that a PPT adversary tries to forge a partial signature $\tilde{\sigma_i}$ to pass the validation in algorithm `PSign_Ver`. From the algorithm `Key_Gen`, we can deduce the relationship between PK_i and r_i ($PK_i = r_i^{\lambda_i} \bmod p$). Furthermore, the other parameters (D_i, PK_i, i) are in public, and therefore the adversary can only fabricate $\tilde{s_i}$ for a forged $\tilde{\sigma_i}$. This problem is reduced to the discrete-logarithm problem according to the equation (3). Based on Definition 1, therefore, the adversary can only forge a partial signature with a negligible probability. Similarly, if the adversary aims to forge a threshold signature $\tilde{\sigma}$ directly, it should generate $\tilde{S}$. However, the difficulty of this task is also equivalent to solving discrete logarithm problem. In conclusion, a PPT adversary cannot forge a partial or a threshold signature. ■

***Theorem** 1:* Since TriPKI achieves mutual verification, as a result, it can resist all entities' compromises in any single party under the threat model in section III-B.

Proof: We analyze the attack-resilience ability of the three parties in TriPKI, respectively.

a) CAs party compromised

Let us consider the worst situation, in which all CAs are compromised to generate a certificate registration request $RegReq = \{Cert_{\mathcal{A}}, CA_1, CA_2, DNS_1, ILS_1\}$ for a malicious domain $\mathcal{A}$. Since the number of compromised CAs is equal to l ($l \geq k$), obviously, CA_1 can generate a malicious $\{RegReq\}_{\sigma^{k-1}}$ and send it to DNS_1. As shown in Fig. 3, as well as CA_2, DNS_1 can verify the signature in $RegReq$ and check the identity of the domain $\mathcal{A}$. Then, it will detect this malicious domain and terminate this certificate registration request. Furthermore, ILS_1 receives two $RegReq$ from CA_2 and DNS_1 respectively. By comparing these two certificate registration requests and verifying the signatures, ILS_1 can find the error information. By tracing back, the corrupted CA_2 will be detected. Thus, TriPKI can resist all entities' compromises in the CAs party.

b) DNSs party compromised

In TriPKI, only one DNS is involved in certificate management, but m DNSs participate in the initialization process as substitutions. As shown in Fig. 2, there are two mutual verification loops involved with DNS. In other words, the misbehaviors of DNS_1 will be detected. For instance, ILS_1

can verify DNS_1's behaviors by comparing $\{RegReq\}_{\sigma^k}$ with $\{RegReq\}'_{\sigma^k}$ and validating the signatures after step 3 in stage 1. Domain A checks the DNS_1's operations by challenging CA_2. Hence, TriPKI can resist all entities' compromises in the DNSs party.

c) ILSs party compromised

ILSs' duty is to synchronize certificates and provide certificate audit proofs to the client. In the first case, ILS_1 needs to send a synchronization feedback to DNS_1 and CA_2. DNS_1 and CA_2 verify the feedbacks by comparing them with the locally stored certifications. In other words, the misbehaviors from ILS_1 will be detected. In the second case, the client can verify ILS_1's behaviors by requesting an audit proof from DNS_1. As a result, TriPKI can resist all entities' compromises in the ILSs party.

In conclusion, mutual verification provides a good improvement in security for TriPKI. ■

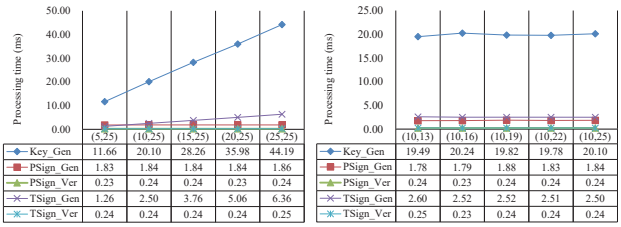
***Theorem** 2:* TriPKI can provide sustainable service when θ CAs ($\theta < l$), ϕ DNSs ($\phi < m$), and $\xi - 1$ ILSs ($\xi < w$) are compromised or failed at most, where ξ is the threshold value of ILSs, $\theta + \phi < \min(n - k, k)$.

Proof: Based on Lemma 1, we learn that a PPT adversary cannot forge a legal threshold signature. However, if it compromises a CA, it can generate a legal partial signature for a malicious domain. This behavior will be detected by the combiner when it verifies the threshold signature. As long as the number of the compromised CAs is $\theta < k - 1$ (there is only one DNS participating in certificate registration), and not all the DNSs are compromised, e.g. $\phi < m$, the adversary cannot generate a legal threshold signature for a certificate from the malicious domain. Furthermore, in order to guarantee that a legal threshold signature can be generated, the number of normal shareholders should be not less than k. Since the synchronization process requires ξ ILSs to reach an agreement at least, malicious ILSs cannot achieve fraudulent synchronization to break the system, if the number of them is less than or equal to $\xi - 1$. As a result, TriPKI can provide sustainable service when the numbers of malicious entities in different parties are limited in these ranges. ■

VI. Performance Evaluation

A. Implementation

We implement TriPKI by the X.509 extension with additional fields described in Section IV-B. The main processes of TriPKI are written in C++ and Bourne Again SHell (BASH). The threshold signature algorithm is realized by the GMP library and RSA-2048 operations invoke OpenSSL (version 1.0.1p) APIs. We implement the domain by extending an Apache HTTP server (version 2.4.7), create CAs with OpenSSL, and establish DNSs by BIND9 (Berkeley Internet Name Domain). ILS servers are carried out in python, where they maintain Merkle hash trees by SHA-512 to record certificates, and provide audit proofs. We implement the client by extending the chromium web browser, and establish connections using the TLS protocol. During client-server connections, the server's certificates are sent to the client in the handshake process while confirmations are provided to the browser by the existing Online Certificate State Protocol (OCSP). For simplicity, entities in the same party are simulated in one PC. Hence, the prototype is composed by five PCs with Intel Celeron E4300 (2.6GHz) CPU, 4G RAM, and Ubuntu 14.04 64bit operation system.



(a) $(k, 25)$ (b) $(10, n)$

Fig. 8. The processing time of the main algorithms. (a) threshold value k grows when $n = 25$. (b) n increases when $k = 10$.

TABLE II
THE PROCESSING TIME OF THE ENTITIES IN EACH STAGE (*ms*)

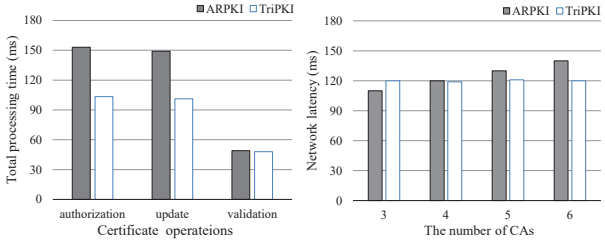
	Init	Stage 1		Stage 2	Stage 3	Stage 4
		PSign	TSign	Synch	Distri	Chall-Resp
CA_1	3.54	0.92	0.13	–	–	–
CA_2		0.90	0.23	–	6.15	16.98
CA_3		0.91	–	–	–	–
DNS_1		0.92	0.22	–	17.02	–
ILS_1	137	–	–	34.09	–	–
ILS_2		–	–	33.92	–	–

B. Experiment Analysis

To evaluate the practicability of TriPKI, we analyze the performance of our prototype in a real-world scenario.

First of all, to evaluate the influence of system's trust level and group size, we investigate the processing time of the five main algorithms (`Key_Gen`, `PSign_Gen`, `PSign_Ver`, `TSign_Gen`, and `TSign_Ver`) with different parameters. In, Fig. 8(a), the threshold value k varies and the group size n is fixed to 25. The processing time of `Key_Gen` and `TSign_Gen` increases when k grows, and that of `PSign_Gen`, `PSign_Ver`, and `TSign_Ver` keep stable at a low level. The reason is that `Key_Gen` algorithm is responsible for calculating k-degree polynomials, and `TSign_Gen` algorithm combines k partial signatures. The other algorithms have no additional computation overhead as k grows. As the description in Section IV-C, the collaboration certificate management scheme depends on those algorithms and k determines the system's trust level, thus, we can conclude that the higher trust is required, the more processing time is needed in the CAs party and DNSs party. In Fig. 8(b), the threshold value k is fixed to 10 and the group size n changes. All curves are stable, which means that the processing time of those algorithms does not grow with the increasing group size. This is because the degree of polynomials and the number of partial signatures are constants when n grows.

Secondly, we measure the processing time of each entity in every stage referring to Fig 3. In this experiment, we setup 1 domain, 3 CAs, 1 DNSs, and 2 ILSs, while a $(3, 4)$-threshold



(a) Certificate management (b) Network latency

Fig. 9. The comparison of processing time

signature scheme is used. The results are the average values in 20 tests as shown in TABLE II. From this table, we can find that the processing time in stage 1 is much shorter than that in other stages. In fact, the computation cost in stage 1 stems from our threshold signature algorithm, and those in other stages are mainly induced by RSA. Thus, we can conclude that the threshold signature algorithm has much lower computation cost than RSA. In addition, from above data, we can estimate that our system can generate 91 certificates per second, while, in practice, the Electronic frontier foundation (EFF) only adds 2.05 certificates per minute on average into its database [11]. Obviously, DNSs have enough time to deal with their own businesses. Thus, the practical effect of TriPKI is acceptable.

Thirdly, we compare the processing time between TriPKI and ARPKI, which are the most relevant schemes, in the aspects of certificate managements and network operations. For ease of comparison, in Fig. 9(a), we compare the processing time of three main operations (authorization, update, validation) rather than different stages in various operations. We can find that TriPKI has less processing time than ARPKI in the first two certificate operations. The main reason is that the two operations in TriPKI are conducted in a parallel mode, while those in ARPKI are handled in a serial mode. Fig. 9(b) shows the network latency with different number of CAs in certificate authentication operation, while other two operations have similar network interactions. In TriPKI, the number of interactions among entities is constant when the number of CAs rises, but that in ARPKI increases linearly. We can find that when the amount of CAs meets some requirements (more than 4 in our experiment), TriPKI has lower network latency than ARPKI.

VII. Conclusion

To establish secure SSL/TLS connections, we proposed a novel Tripartite Public Key Infrastructure (TriPKI) in this paper. It applies a checks-and-balances strategy among CAs, DNSs, and ILSs, to resist the compromise of any single party which induces the weakest-link security problem in current PKIs. We also design a distributed collaborative certificate management scheme to distribute the trust of entities in the three parties and enhance the attack-resilience and fault-tolerance. The security proof and experiment results show that TriPKI can stand against compromises and failures with acceptable time consumption.

References

[1] J. Clark and P. C. V. Oorschot, "Sok: SSL and Https: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2013.

[2] L. Chen, H. W. Lim, and G. Yang, "Cross-Domain Password-Based Authenticated Key Exchange Revisited," in *Proceedings IEEE International Conference on Computer Communications (INFOCOM)*, 2013.

[3] A. Langley, "Enhancing digital certificate security," http://googleonlinesecurity.blogspot.ch/2013/01/enhancing-digitalcertificate\\-security.html, 2013.

[4] "Maintaining Digital Certificate Security," http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html, 2015.

[5] "DSDTestProvider, eDellRoot," http://www.kb.cert.org/vuls/id/925497; http://www.kb.cert.org/vuls/id/870761, November 2015.

[6] T. H. Kim, L. Huang, A. Perrig, C. Jackson, and V. Gligor, "Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure," in *Proceedings of the International World Wide Web Conference (WWW)*, 2013.

[7] D. Basin, C. Cremers, T. H. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack Resilient Public-key Infrastructure," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[8] J. Chen, K. He, Q. Yuan, G. Xue, and R. Du, "Batch identification game model for invalid signatures in wireless mobile networks," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[9] P. Hoffman and J. Schlyter, "The DNS-based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," http://tools.ietf.org/html/rfc6698, 2012, IETF RFC 6698.

[10] R. Arends, "DNS Security Introduction and Requirements," http://tools.ietf.org/html/rfc4033, 2005, IETF RFC 4033.

[11] M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, "Global authentication in an untrustworthy world," in *Proceedings of ACM USENIX Association*, 2013.

[12] J. Braun and G. Rynkowski, "The Potential of an Individualized Set of trusted CAs: Defending against CA Failures in theWeb PKI," in *Proceedings of the IEEE International Conference on Social Computing and Networking (SocialCom)*, 2013.

[13] N. Buchmann and H. Baier, "Towards a more secure and scalable verifying pki of emrtd," in *Journal of Computer Security*, 2014.

[14] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile," Tech. Rep., 2002.

[15] "Public Key Pinning," http://www.imperialviolet.org/2011/05/04/pinning.html, 2011.

[16] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," in *Internet Request for Comments 2560*, 1999.

[17] A. Bates, J. Pletcher, T. Nichols, and B. Hollembaek, "Forced Perspectives, Evaluating an SSL Trust Enhancement at Scale," in *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2014.

[18] "Convergence," 2011, http://convergence.io/.

[19] E. Topalovic, B. Saeta, L. Huang, C. Jackson, and D. Boneh, "Towards Short-Lived Certificates," in *Proceedings of Web 2.0 Security and Privacy*, 2012.

[20] S. Matsumoto and R. M. Reischuk, "Certificates-as-an-Insurance: Incentivizing Accountability in SSL/TLS," in *Proceeding of 2015 Network and Distributed System Security Symposium (NDSS)*, 2015.

[21] J. Wierzbicki, "Sovereign Key Cryptography for Internet Domains," https://git.eff.org/?p=sovereign-keys.git;a=summary.

[22] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," http://tools.ietf.org/pdf/rfc6962.pdf, 2013, IETF RFC 6962.

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

[24] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and Flexible TLS Certificate Management," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[25] G. Becker, "Merkle signature schemes, merkle trees and their cryptanalysis," *Online im Internet: http://imperia.rz.rub.de*, vol. 9085, 2008.

[26] S. Kim, J. Kim, J. H. Cheon, and S. Ju, "Threshold Signature Schemes for ElGamal Variants," in *Computer Standards & Interfaces*, 2011.