IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 9, 2020, accepted May 12, 2020, date of publication May 25, 2020, date of current version June 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2996997

Attack-Resilient TLS Certificate Transparency

SALABAT KHAN[1], LIEHUANG ZHU[1], (Member, IEEE), ZIJIAN ZHANG[1,2], (Member, IEEE), MUSSADIQ ABDUL RAHIM[1], KHALID KHAN[3], (Graduate Student Member, IEEE), AND MENG LI[4,5]

[1]School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100018, China
[2]Department of Computer Science, The University of Auckland, Auckland 1010, New Zealand
[3]Kohat University of Science and Technology, Kohat 26000, Pakistan
[4]School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China
[5]Key Laboratory of Knowledge Engineering with Big Data, Ministry of Education, Hefei University of Technology, Hefei 230601, China

Corresponding authors: Liehuang Zhu (liehuangz@bit.edu.cn) and Zijian Zhang (zhangzijian@bit.edu.cn)

This work was supported in part by the China National Key Research and Development Program under Grant 2016YFB0800301, and in part by the National Natural Science Foundation of China (NSFC) under Grant 61872041.

ABSTRACT The security of Public-Key Infrastructure (PKI) for Internet-based communications has lately attracted researchers' attention because of Certification Authorities (CAs) crashes and consequent attacks. Google Certificate Transparency and subsequent log-based PKI proposals (e.g., AKI and ARPKI) have succeeded in making certificate-management processes more transparent, accountable, and verifiable. However, those proposals failed to solve the root CA generous delegation of trust to intermediate CAs, non-conformant certificate-issuance by them, and lack of rigorous authentication of domain ownership during certificate-issuance problems. This study presents Attack-Resilient TLS Certificate Transparency (ARCT) based on log servers to address these problems. ARCT enables root CA to enforce intermediate CAs to follow community standards through leveraging a log server at each root level. It also introduces a collaborative domain ownership verification method that deters false certificate-issuance and ensures that a set of CAs validates every certificate before any client will accept it. A certificate collectively approved by a set of CAs assures users that the certificate has been seen, and not instantly detected malicious, by a group of CAs. Finally, formal security and performance evaluations prove the reliability and effectiveness of ARCT.

INDEX TERMS TLS, PKI, log server, delegation of trust, collaborative identity verification.

I. INTRODUCTION

Transport Layer Security (TLS) is the backbone and tremendous success in securing Internet-based communications. Virtually all online purchase transactions are secured and protected by TLS. Although TLS was primarily designed to secure web traffic, today, most financial and non-financial communications depend on TLS for their security. TLS is highly reliant on the PKI for the secure connection establishment, in which CA-signed certificates are employed for authentication. CAs play the most critical role in the PKI, and PKI's security and defense depend solely on the trustworthiness, honesty, and security of CAs. Stealing secret-keys of CAs are enough to impersonate domain servers; this makes CAs' secret-keys the prime targets for hackers.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Khalil Afzal.

Unfortunately, we have observed notable crashes and exploits in the CA regime. Although the CA regime purports to shield and guard clients against Man-in-the-Middle (MitM) attacks, the current CA trust model is worryingly brittle. Recent CAs collapses have revealed its vulnerability in practice [1], [2]. The web browsers/operating systems blind trust in a substantial number of CAs' certificates is a big problem of the current TLS ecosystem. In the current CA model, CAs can issue TLS certificates for domains outside their scope, representing the idea of weakest-link security [3].

In response, several solutions [3]–[6] have been proposed. Certificate Transparency (CT) [4], [5] is a scheme proposed by Google to make the problem of false TLS certificates detection easier by registering all TLS certificates on a public log maintained in the form of a Merkle Hash Tree (MHT). However, it cannot guard clients against attacks when a corrupted CA issues false TLS certificates. References [3] and [6] are another line of proposals that attempts

to sign the TLS certificates from multiple CAs and frame certificate-management transparent by maintaining public Log Servers (LSes) of all TLS certificates. They also acquaint checks-and-balances to reduce the need to trust any single party and prevent MitM attacks.

However, the generous-delegation-of-trust by the root CA still lacks attention from the research community. Root CA generously delegates their signing authority to intermediate CAs,[1] and these unknown entities are blind as trusted as the root trust servers. This trust delegation results in a mass blind chain of trust. Further, intermediate CAs have abused the delegated power for false certificate-issuance [7]. Both before and after DigiNotar and Comodo incidents, a series of lucrative attacks on intermediate CAs came to light [7]. Most small intermediate CAs failed to follow root CAs and community standards and miss-issued certificates [8]. For example, in 2011, a Malaysian sub-CA was reported to have issued domain certificates with a key size smaller than required by the parent CA, and a Swiss intermediate CA shorter key was used to sign malware [7]. Most sub-CAs' miss-issuances have gone unmentioned in public forums like the Mozilla Developer Security Policy (MDSP), but deserve special attention [8].

Apart from trust delegation, CAs do not conduct strict validation of the domain identity before certificate-issuance [9], [10]. For example, in 2018, Birge-Lee *et al.* [9] and Brandt *et al.* [10] showed real-world attacks against top CAs such as Let's Encrypt, Comodo, GoDaddy, Symantec, InstantSSL, CertCom, NetworkSolutions, SSL.com, NetLock, and GlobalSign. They unveiled that the certificate-issuance process[2] is itself defenseless to MitM attacks. Again, Let's Encrypt was taken down by Borgolte *et al.* [11], and as a result, fake certificates were issued. Fake and maliciously acquired TLS certificates have been used to conduct cyber-attacks against users of various famous websites such as Google, Yahoo, and Facebook [12], [13]. A survey showed that 0.2% of all Facebook users' connections were attempted to tamper with malicious TLS certificates [14], and 3 million forged TLS certificates were identified for the top 10k Alexa sites [15].

To address these problems, we propose a new log-based PKI for managing domain certificates, called Attack-Resilient TLS Certificate Transparency. This new PKI scheme has verifiable distributed parties and makes four primary contributions.

- Root CA's generous-delegation-of-trust is an important problem for certificate-issuance in the current CA model, and this area has yet not received research attention. ARCT is the first architecture to resolve the problem by giving root CA active control over their delegated power and enabling root CA to restrict intermediate CAs from abusing and misusing the delegated authority.
- We explore attacks on CAs and certificate transparency. ARCT mitigates the attacks by introducing a collaborative certificate-issuance mechanism that enforces the rigorous identity verification of the domain to counter malicious certificate-issuance (see V). It introduces an improved revocation mechanism that keeps a record of all revoked certificates as well as efficient in terms of communication cost (see VII).
- A thorough security analysis of ARCT is conducted, which shows that ARCT can prevent impersonation attacks, even in the face of a strong adversary who is capable of compromising CA and LS simultaneously (see VI).
- A prototype of ARCT is built to evaluate the feasibility of the collaborative identity verification process. Lastly, ARCT is compared with some leading log-based PKI schemes using various performance metrics (see VII).

[1] In this article, intermediate CA and sub-CA are used interchangeably.

[2] In this article, we will focus on Domain Validation (DV) certificates because DV supporting CAs dominate certificate market shares and control over 99% of the shares [10].

II. RELATED WORK

Prior works can be broadly classified into traditional PKI schemes, blockchain-based PKI schemes, and log-based PKI schemes.

Traditional PKI Schemes: The X.509 standard PKI deployment includes CRL [16] that is periodically disseminated by CAs to prevent revoked certificates being used in TLS connections. This approach can only mitigate attacks if clients possess an updated CLR copy. Otherwise, attackers have an opportunity to launch cyber-attacks against clients using already revoked certificates [6]. On the other side, CLR is quite expensive in terms of communication costs. OCSP [17] servers are deployed to reduce CRL costs and enable real-time certificate revocation status. Short-lived Certificates [18] eliminates the need for checking the revocation of a certificate. However, the main issue of CRL, OCSP, and SLC is their heavy dependency on web browsers to identify and blacklist spurious certificates endorsed by dishonest CAs [3].

To empower clients, some proposals delegate the power of accepting or rejecting certificates to clients based on either their defined local policies [19] or comparison with the information stored in the repository [20]–[22]. Likewise, other proposals [23]–[25] empower domains to shield their keys despite CAs crashes by allowing them to announce their public-keys such that users can pin their keys. However, these approaches are vulnerable to MitM attacks on "first connection to a domain." DNS-based Authentication of Named Entities (DANE) [26] sanctions domains to assert their key-related information in their DNS security extension (DNSSEC). CA Authentication (CAA) [27] lets domains to explicitly define a list of qualified CAs that can endorse certificates for them. Reference [28] analyzed that a small set of CAs issue certificates of top-level domains (TLDs). Using the TLDs analysis of [28], CAge [29] restricts the set of TLDs, for which a specific CA is trusted to issue TLS certificates. While CA-TMS [30] and [31] collects information about CAs reputation from users, and they restrict CAs' scope

using the reputation data gathered from their clients. References [9], [10] successfully attacked several eminent domain validation-based commercially used CAs and showed that their certificate-issuance process itself is exposed to MitM attacks. References [9], [10] suggested several recommendations to mitigate attacks against the domain validation-based certificate-issuance process. The proposals are defenseless against attackers capable of compromising CAs' private-key.

Blockchain-based PKI Schemes: This class of schemes [32]–[36] tried to propose decentralized solutions based on blockchain for public-key management. SCPKI [33] integrated the web of trust with smart contracts to identify false certificates in the communication. Reference [34] managed .bit addresses (DNS) on a public ledger, which is forked of Bitcoin. In [34], self-signed certificates are inserted into DNS addresses as auxiliary data. Certcoin [37] extended [34] and further eliminated the need for using third-parties (CAs). Certcoin includes identity registration, verification, search, and revocation phases. PB-PKI [38] resolved the identity linking and privacy concerns of Certcoin [37]. However, the authors in Blockstack [39] identified that a single miner in [34] holds over 51% of computation power and clients are exposed to 51% attacks. Wang *et al.* [40] presented a PKI framework to manage TLS certificates and their revocation on a blockchain medium. In this blockchain-based PKI, each domain server has two key-pairs, namely, publishing key-pair and TLS key-pair. The publishing is endorsed by a group of web servers, which is used to publish transactions to the blockchain. While the TLS key-pair, which is approved by a CA and published on blockchain medium, is used in TLS communications. It is vulnerable to cyber-attacks where an attacker can communicate with a client through a revoked certificate whose transaction is still valid [41].

Yakubov *et al.* [42] proposed a decentralized PKI scheme to handle TLS certificates on a blockchain platform. In this scheme, a smart contract is created for each CA, which comprises the CA certificate, hash values of the CA-signed end-entities certificates, and certificate revocation information. Blockchain-related metadata are inserted into X.509 extension fields of each TLS certificate. In this framework, the hash values of domains' certificates are only stored, which makes it impracticable for domain owners to watch their certificates [41]. CertChain [43] proposed a novel data structure called CertOper for TLS certificates auditing. The proposed data structure comprises certificate-related data and related procedures such as registration, renewal, and revocation. CertChain adopted the Ouroboros [44] as a consensus algorithm, which relies on CA and bookkeeper's reputation for a leader selection. PBCert [45] identified shortcomings in CertChain and proposed enhancement to mitigate the issues. CertLegder [41] presented a PKI framework where all certificate-related operations and data are handled on a blockchain medium. It eliminates the key store maintained by clients and makes key revocation transparent. The major problem with blockchain-based PKI frameworks is scalability issues, and they substantially alter the underlying infrastructure of TLS PKI [46]. Second, synchronizing full-node takes considerable time (e.g., synchronizing full-node of Bitcoin with 145 GB ledger size takes three days [47]).

Log-based PKI Schemes: Unlike blockchain-based PKI approaches, log-based PKI proposals do not substantially alter the underlying architecture of certificate management. They enable anyone to monitor and audit CAs operations and detect their misconduct. Google Certificate Transparency (CT) gained widespread adoption, which will be reviewed in the next section. Peter Eckersley presented a new public-key framework called Sovereign Key (SK) [48] that is designed to securely verify the domains TLS certificates more robustly than the existing CAs model. In SK, a domain generates a sovereign key-pair to endorse a TLS certificate and register the public part of the key-pair on an append-only log called timeline servers. SK can guard against an impersonation attack even when CA gets compromised. However, in SK, domains and clients have to blindly trust mirrors of timeline servers, as mirrors are unable to generate a verifiable proof for the certificate included in the append-only log. Unfortunately, in SK, a client also requires querying to a timeline server, increasing latency, and sacrificing client privacy.

AKI [3] is a log-based proposal for transparent certificate management that distributes trust over multiple anchors. AKI has built-in support for revocation and enables domain owners to specify a list of trusted CAs and LSes. It uses the checks-and-balances mechanism to reduce trust in any party, resulting in the mitigation of MitM. ARPKI [6] is a refinement of AKI. In ARPKI, certificate registration requires a domain owner to designate a certain number of service providers (n) and contact one of them to monitor multi-signature certificate (ARCert) registration in LSes. It offers robust security against powerful adversaries and can protect clients from attacks, even when an adversary subverts n-1 service providers. However, ARPKI has no support for multi-domain certificates, and it has an extra delay because of the involvement of all designated parties in all processes [49]. TriPKI [50] utilized a threshold-signature scheme among a list of trusted CAs, trusted DNSes, and trusted LSes to distribute trust and assure secure communication. Reference [50] substitutes third-party validators with DNSes and adopts checks-and-balances among the trusted set of CAs, DNSes, and LSes to watch each other behavior.

DTKI [49] is another PKI framework that combines CT, AKI, and SK, where the sovereign key-pair (renamed as a master-key in DTKI) is used to restrict the CA's role while LS is maintained using a pair of chronological and lexical MHTs to have benefits of CT with revocation support. Unfortunately, in DTKI, everyone must trust a single log maintainer called Mapping Log Maintainer (MLM). DTKI has extra latency as clients must visit MLM before every connection, putting their privacy in danger. Policert [13] tries to give more control to domain owners over their TLS certificate and TLS connection by specifying detailed policy.

In Policert, a domain needs to register the Subject Certificate Policy (SCP) and multi-signature certificate (MSC) on LS. The domains present SCP, MSC, and their presence proof in LS to clients during the TLS connection. This scheme also supports TLS certificate revocation. However, in this proposal, the mechanism for disseminating and detecting LS misconduct is not specified [6]. ATCM [12] explores MitM attacks on Policert and mitigates the attacks by leveraging checks-and-balances and making the life cycle of TLS certificate transparent. ATCM provides stout security and can defend clients against an active adversary compromising most of the trusted servers. They also validated the security features that their scheme supports using the automated verification tool Tamarin prover [51].

A. CERTIFICATE TRANSPARENCY OVERVIEW

We review CT [4], [5] in detail for two main reasons: (1) ARCT is motivated by CT's architecture and utilizes some of its concepts; (2) ARCT resolves several problems that remain despite the CT model deployment. The main aim of CT is to mitigate the problem of fake and fraudulent TLS certificates by introducing an auditable public log of all issued certificates. Eventually, browsers would accept a certificate only if it had an inclusion proof in a public log. In CT, when a CA wants to issue a TLS certificate for a domain, it selects an LS and delivers the TLS certificate data to the LS. The LS generates signed-certificate-timestamp (SCT) for the certificate, which guarantees that the LS will insert the TLS certificate to its database within the maximum merge delay (MMD). The CA embeds the SCT as an X.509 extension in the signed TLS certificate or sends it with the certificate in OCSP extension after receiving SCT from the LS. The CT operates with the following main entities.

- **Log Server (LS).** Log servers are maintained using MHT in chronological form. All issued TLS certificates are stored in one of many LSes.
- **Monitor.** Monitors check the LSes for suspicious TLS certificates by examining all log entries.
- **Auditor.** Auditors verify that the LSes are behaving correctly, detect miss-issued TLS certificates based on partial information from the LSes, and check that SCTs they encounter are in the logs.

LSes are maintained in the form of MHT, where each leaf node of MHT is a hash of the TLS certificate, and internal nodes of MHT are a hash of two child nodes, enabling the LS to generate inclusion and extension proofs efficiently. The root of the MHT, which is signed and broadcasted by LS, is called a signed-tree-head (STH).

B. REMAINING CHALLENGES

We briefly highlight several issues that motivated this work and remained despite the prior works.

Root CA's generous-delegation-of-trust is a significant problem of the current CA model, where the parent CA delegates certificate-issuance authority to several sub-CAs that can issue TLS certificates to any domain without any restriction. After delegation, the root CA loses control over the delegated authority and cannot prevent sub-CAs from abusing and misusing the delegated power. Further, the root CA fails to enforce sub-CAs to comply with the community standards, whereas most erroneous certificates are issued by the intermediate CAs [8]. For example, Comodo CA has 29 sub-CAs, and most unsound and non-conformant certificates are signed by just one sub-CA (COMODO RSA DV Secure Server), which totals for 85% of the erroneous certificates [8].

CT and previous solutions (e.g., AKI, DTKI, and ARPKI) assume that CAs' certificate-issuance process is secure, and they perform rigorous verification of domain ownership. However, CAs use a vulnerable certificate-issuance process, and the process itself is exposed to MitM attacks [9].

Besides these issues, CT left revocation as an open problem and still relies on existing methods for revocation. Proposals like revocation transparency [52], CIRT [53], CT with Enhancements and Short Proofs [54], and PKI safety net (PKISN) [55] have been suggested to solve CT's revocation problem, but none has been incorporated in CT yet. While existing certificate revocation schemes CRL [16] and OCSP [17] are still unsatisfactory.

C. GENERAL NOTATIONS

Table 1 shows the important symbols and notations used in ARCT.

TABLE 1. Important Symbols and Notations.

Notation	Explanation
pk_P, sk_P	Public/private key-pair of party P
$Cert$	General notation of TLS certificate
$Cert_D^{ICA}$	A TLS certificate certified by intermediate CA for a domain D.com
$Cert_D^{'ICA}$	A fake TLS certificate certified by intermediate CA for a domain D.com
$sign(m, sk_P)$	Signing message m with private-key sk_P of party p
$verify(m, pk_P)$	Verification of signature of party p on message m
PoP	Proof of presence
STH	Signed tree head of log server (LS)
STH'	Fake signed tree head of log server (LS)
PoE	Proof of extension
PoN	Proof of non-revocation
PoN'	Fake proof of non-revocation
$certICA$	Intermediate CA certificate

III. PRELIMINARIES

A. MERKLE HASH TREE

We formally define MHT.

Definition 1: **(Merkle Hash Tree (MHT))** MHT is constructed according to the following rules: for any list L with length n, if length of L is 1 then $MHT(L) = H(0||L[0])$, otherwise $MHT(L) = H(1||MHT(L[0 : m])||MHT(L[m : n]))$. Here H(.) is collision-resistance hash function and || is concatenation operator.

The MHT has the following algorithm

- $MHT(L) \rightarrow TH$: Deterministic tree hashing algorithm that accepts as an argument a list of members (elements)

L and yields a tree head (root hash) TH that is a commitment to the list.

- $AuthenPath(elmnt, L) \rightarrow AP$: A deterministic algorithm that takes a list L and an element elmnt as inputs and outputs an authentication path AP representing the membership.
- $VerifyAuthenPath(elmnt, TH, AP) \rightarrow 0/1$: It accepts an element elmnt and authentication path AP as inputs and outputs either 1 or 0 (true or false).
- $ExtenProof(m, n, L) \rightarrow EP$: It takes as inputs an L, and two indices $0 \leq m \leq n \leq |L|$, and generates an extension proof consisting of a sequence of hashes. The proof provides evidence that the current tree consisting of entries $(0 : n)$ is extending the previous one, consisting of entries $(0 : m)$.
- $VerifyExtenProof(TH_i, TH_j, EP) \rightarrow 0/1$: It takes as inputs proof of extension EP, two root hashes TH_i, and TH_j' and verifies that the tree corresponding to TH_j is an extension of the MHT corresponding to TH_i. VerifyExtenProof(.) either returns true or false.

Definition 2: **(MHT Collision)** MHT collision is a pair of two distinct MHTs corresponding to two lists L, and L' such that $MHT(L) = MHT(L')$ and $L \neq L'$.

Lemma 1: (Collision-resistance of MHT). If H is collision-resistant hash function, then the MHT is collision-resistant.

Proof: This follows from the work by [56], [57]. ■

Lemma 2: (Collision-resistance of Authentication path of MHT). If H is a collision-resistant, then an authentication path in MHT is collision-resistant.

Proof: This follows from the work by [56]–[58]. ■

B. AUTOMATIC CERTIFICATE MANAGEMENT ENVIRONMENT (ACME)

ACME [59] protocol is proposed by the Internet Security Research Group (ISRG) to automate interactions between CAs and domain servers, allowing automated certificate-issuance and deployment at a low cost. The ACME protocol supports domain validated (DV) certificate-issuance, where the ownership of a domain is validated through a challenge/response process. The ACME protocol supports a DNS- and HTTPS-based challenge/response process to verify control over a domain. The validation mechanism of the ACME protocol comprises three main steps: (1) When a CA receives a certificate signing request from a domain owner, the CA sends a set of challenges to the domain owner. (2) The domain owner replies with a set of responses to complete the set of challenges. (3) The CA validates the set of responses, and once the challenges are correctly completed, the CA issues a DV TLS certificate for the domain.

IV. SYSTEM AND THREAD MODEL

To build a PKI that is immune to the compromise of several parties and can recover from catastrophic casualties, we base ARCT on verifiable LSes. In the proposed PKI, we achieve strong security by introducing one more layer of transparency and validation at the root CA. At this layer, each root CA in collaboration with intermediate CAs inspects all newly issued certificates by performing the identity verification of the domain once again to prevent fake and false certificate-issuance. The parent CA also monitors that intermediate CAs follow best-practices, according to the root CA's and community requirements (e.g., Baseline Requirement and Root Program Owner) specifications. The root CA blacklists intermediate CAs for malpractice and erroneous issuance, such as issuing certificates with a key size smaller than the required. Next, we introduce the main entities and their responsibilities.

- Root CA: It is the main trust anchor whose certificate is shipped in clients' web browser's root stores. Each root store can contain several root CAs' certificates, and each root CA issues certificates only for intermediate CAs.
- Intermediate CA: This component endorses certificates to domains, and the root CA delegates this power to intermediate CA.
- Exclusive Log Server (ELS): This LS is a publicly auditable log of all certificates issued by the intermediate CAs of the concerned root CA. There is one ELS per root CA in the proposed PKI. The database of ELS is maintained as a pair of MHTs. The first tree is maintained as ChronTree [53]. To address the revocation problem, we additionally manage the revocation information as another MHT, maintained as in [60], which we call a revocation tree. The revocation tree is a hash tree with leaf nodes corresponding to a set of statements about the certificate's serial numbers. The set of statements is generated from revoked certificates and provides information about the certificate revocation status. The root of the revocation tree is inserted as the last node in the ELS ChronTree.
- Shared Log Server (SLS): This LS is a public log of all issued TLS certificates, as in CT. However, the database of SLS is extended and is maintained as a pair of MHTs. Both MHTs are kept in chronological form. The first tree maintains a database of certificates that we call certificate tree (CertTree), while the other tree maintains a database of ELSes' STHs, which is referred to as a witness tree (WitTree). Here the SLS acts as a witness for ELS and prevents ELS from split-world attacks.
- Monitor: Monitors regularly watch and scan the LSes (ELS and SLS) for malicious certificates by examining all their records as in CT.
- Auditor: Auditors verify that LSes (ELS and SLS) are behaving correctly as in CT, detect malicious certificates based on partial information from the LS, and check that the SCTs they encounter are in the logs. They are lightweight software and an integral part of client software.
- Domain: Each domain[3] has an ICA-signed certificate that is presented to its client during connection establishment.

[3]In this article, domain and domain server are used interchangeably

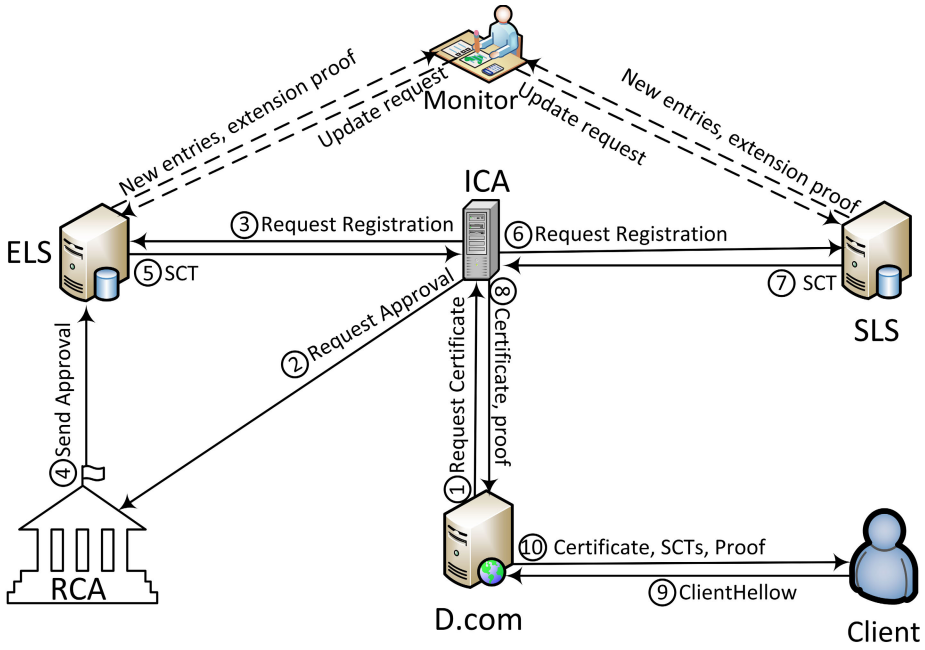

FIGURE 1. An overview of ARCT architecture. Here RCA and ICA stands for root and intermediate CA respectively.

- Client: A user agent (e.g., a web browser) that uses services offered by domain servers securely.

Figure 1 portrays the architecture of the ARCT scheme. Alice owns domain server D.com and wants to acquire a certificate from an intermediate CA to shield her clients against rogue and false certificates. Alice forwards certificate signing request to an intermediate CA, which upon successful D.com identity verification, requests parent CA to approve the certificate for logging in ELS. The ELS sends an SCT to the intermediate CA (or domain owner) if the certificate exists in the final list of certificates approved by the parent CA. The SLS only accepts a certificate for registration from the submitter (intermediate CA or domain) if it carries an SCT from the corresponding ELS. Alice now addendums her certificate with the SCTs from both LSes and non-revocation proof from ELS after registration with them. When clients open connections to D.com via HTTPS, the server sends the certificate along with the SCTs and proof to clients. For certificate validation and verification, the web browser uses the certificate chain ending at trusted root CA's certificates and pre-installed LSes' public-key on the browser to validate the LSes' information. The clients can occasionally confirm the validity of the LSes' STH with monitors and other clients.

A. ADVERSARY MODEL

We consider an adversary whose primary aim is to impersonate domain servers using HTTPS. To accomplish this aim, the adversary can steal CA (e.g., intermediate CA) private-key to acquire the malicious certificates and LS (e.g., ELS) private-key to log malicious certificates. We assume that the cryptographic building blocks that we use, such as the hash function and signature schemes, are secure. The hash function H is resistant to collision and pre-image attacks, and the signature scheme cannot be forged (EUF-CMA secure).

B. DESIRED PROPERTIES

- Complete audit: All entities taking part in certificate-issuance should be liable. They (e.g., sub-CAs) should be audited and compliant with standards at the same level as a root CA, none of their violation should go unnoticed.
- Low cost: The scheme should not tectonically expand the TLS handshake message size and should have an insignificant effect on processing time. Moreover, no extra delay should be added to the TLS connection establishment in particular round-trip time to external servers.
- Resilience against attacks: Building a complete secure infrastructure that can eliminate attacks in catastrophic failure is impossible. However, the system must have a brief system attack window.
- Privacy: Public-key infrastructure (PKI) must not reveal the clients' connection information.
- Checks and Balances: Root CAs should limit blind trust in sub-CAs, and limited trust should be given to participating parties (e.g., LSes) to mitigate the weakest-link security. Furthermore, participating parties should be able watch each other to detect misconduct.

V. ATTACK-RESILIENT TLS CERTIFICATE TRANSPARENCY

First, we discuss in detail the algorithms executed by ELS, SLS, monitors, and auditors to generate and verify proofs. The following are the main algorithms run by ELS and SLS.

$GenSCT(cert, t, sk_{ELS}) \rightarrow SCT$: It takes as inputs a certificate $cert$, current time t, and private-key to generate SCT.

$GenSTH(TH, t, sk) \rightarrow STH$: A randomized algorithm that takes TH, time t, its private-key sk as inputs and return sign-tree-head (STH).

$PresentLogEntries(st, STH) \rightarrow L/error$: A deterministic algorithm taking current st and STH as input arguments and returns an order list L or an error.

$GenPoP(st, cert, STH) \rightarrow PoP/error$: A deterministic algorithm that for certificate $cert$ executes $AuthenPath(cert, st)$ procedure to generate authentication path. Let the authentication path consist of $(h_1, h_i, \ldots, h_n)$ then $PoP = (h_1, h_i, \ldots, h_n)$ where h_i represent hash values.

$GenPoE(st, STH, STH') \rightarrow PoE/error$: To prove that the prover (e.g., ELS and SLS) is an extension of its previous version, an LS sends one node per level to the prover by calling $ExtenProof(m, n, L) \rightarrow EP$, which generates extension proof. If the current tree extends the previous one, then the previous tree is a sub-tree of the current tree.

The $GenPoN(.)$ given below is executed by ELS only.

$GenPoN(st, cert, s(i, j), STH) \rightarrow PoN/error$: To prove that a certificate $cert$ is not revoked, the prover searches for the range that bracket the serial number such that $x_i < x < x_{i+1}$, where x_i is the serial number of certificate. It executes $AuthenPath(cert, st)$ that generates the proof and forwards to the verifier.

To validate that both classes of LSes are working correctly, monitors and auditors execute the following algorithms.

$CheckSCT(SCT, pk) \rightarrow 0/1$: It validates that the SCT is issued for the corresponding certificate and endorsed by a trusted LS.

$CheckSTH(STH, pk) \rightarrow 0/1$: It takes SHT and public-key pk of a concerned LS and returns either 1 or 0.

$CheckPoP(cert, STH, PoP, pk) \rightarrow 0/1$: A deterministic algorithm which executes $CheckSTH(STH, pk)$ and $VerifyAuthenPath(cert, TH, PoP)$ to validate that the certificate $cert$ is logged or not in the LS.

$CheckPoN(cert, s(i, j), STH, PoN, pk) \rightarrow 0/1$: It first checks that $cert$ serial number is greater than i but less than j. After verifying the serial, it then computes root hash from the authentication path. If the computed hash is equal to the published root hash, then it is accepted, otherwise rejected.

$CheckLogEntries(L, STH, pk) \rightarrow 0/1$: Let $e_1, e_2, \ldots, e_n$ be the list of elements of L, then the algorithm checks wether $H(H(H(e_1)||H(e_2)) \ldots . h(H(e_{(n-1)}||H(e_n)) \stackrel{?}{=} TH$.

$CheckPoE(STH_i, STH_j, PoE, pk) \rightarrow 0/1$: To validate that the current version is extended from the previous one, the verifier executes $CheckPoE(STH_i, STH_j, PoE, pk)$ by giving two signed tree head, proof of extension, and public-key of the LS as input, then CheckPoE(.) returns either 1 or 0.

In the rest of the section, we describe the proposed scheme in detail based on four main phases, namely certificate-issuance, certificate registration, certificate revocation, and connection establishment.

A. CERTIFICATE-ISSUANCE

To start using ARCT, a domain must possess a valid certificate. To acquire a certificate, the domain owner of a domain (e.g., D.com) generates a key-pair and forwards Certificate Signing Request (CSR) to intermediate CA. Upon receiving a CSR, the intermediate CA must validate that the submitter is the actual owner of the D.com. To enforce rigorous control verification, we introduce a new collaborative domain identity verification method that complements the traditional certificate-issuance process and ensures verification by multiple parties (CAs). As shown in Figure 2, the collaborative process works as follows.

1 A domain owner sends a CSR for her domain, e.g., ''D.com,'' to an intermediate CA, e.g., ''ICA(1)''.
2 The ICA1 issues a challenge to the owner of the domain, through which ICA(1) requires to accomplish to verify ownership of the domain.
3 Once the domain owner receives the challenge, she hosts the challenge1 signed by its private-key at the URL, e.g., ''https://D.com'' to serve as the domain validation resource.
4 The ICA1 validates the domain by accessing the challenge1 placed at the URL as well as verify the signed challenge using the public-key of the domain.
5 After initial verification of a domain's identity, the ICA, e.g., ''ICA(1),'' forwards approval request to its parent CA.
6 The root CA selects m numbers of sub-CAs and multi-casts collaborative identity verification requests to them, where m is a system parameter set by the root CA.
7 Each m number of identity validation includes carrying out step 7 to step 9, which are the repetition of step 2 to step 4 but with different challenge types and from different vantage points.

After collaborative verification, the root CA prepares a list of certificates that passed the scrutiny and sends the final approved list to the ELS. Note, each sub-CA can implement different challenge types (e.g., DNS challenge, see Sec. VII-C for different challenge types).

B. CERTIFICATE REGISTRATION

Before a certificate can be used in secure communications, it must be registered at the ELS and at least at one or more

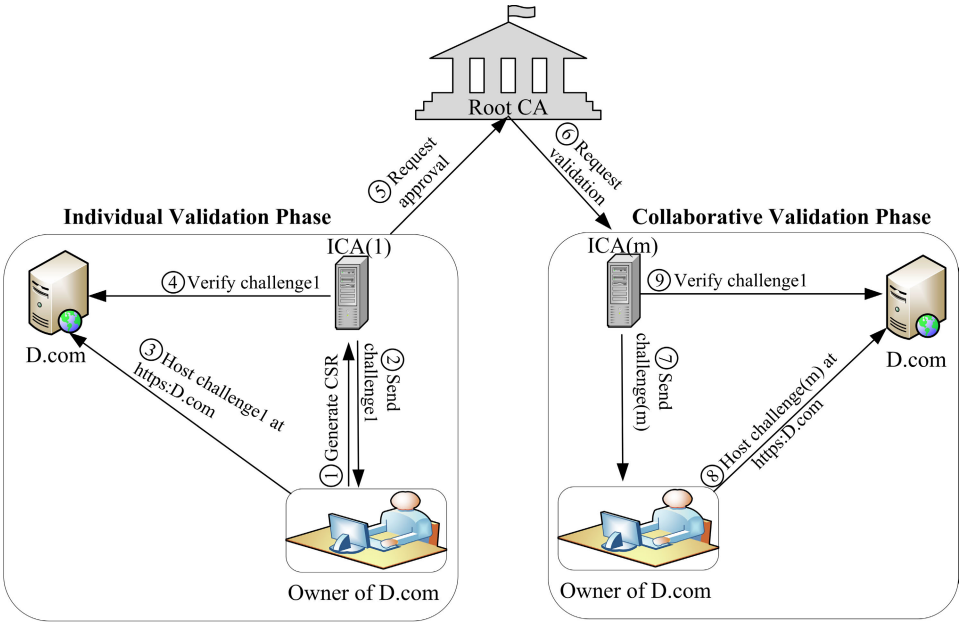

FIGURE 2. Collaborative domain verification process, where a group of m CAs validate each domain before certificate-issuance.

than one SLS(es). First, we discuss registration at ELS and then describe the logging process at SLS.

1) EXCLUSIVE LOG SERVER

After the domain identity validation, each intermediate CA sends the certificate(s) to the ELS for registration. The certificate(s) must be present in the final approved list, say L_{app} sent by the root CA to the ELS. The ELS generates an SCT for each certificate after checking the soundness and correctness of each TLS certificate (see Algorithm 1), where the SCT acts as an insurance that the ELS will attach the TLS certificate to its database in the next update. All erroneous certificates are filtered out at this stage by the ELS.

2) SHARED LOG SERVER

After receiving an SCT from the ELS, the submitter (e.g., intermediate CA or domain owner) forwards the certificate to SLSes for logging the certificate in SLSes. Each SLS validates the SCT of the ELS, generates an SCT after performing the necessary checks as given in Algorithm 2, and sends the SCT back to the submitter.

C. CERTIFICATE REVOCATION

When a domain's private-key leaks, it generates a signed Certificate Revocation Request (SCR) and forwards the SCR to an ELS. The ELS verifies the SCR and checks whether the certificate is included in its database. If it exists in its database, then the ELS generates a signed-revocation-timestamp (SRT) given in Algorithm 3, which acts as a promise that the ELS will revoke the certificate in the next update. If an intermediate CA's private-key is compromised or an intermediate CA misbehaves, the ELS adds the intermediate CA certificate in a blacklist and does not log any TLS certificate signed by the intermediate CA. This action prevents the intermediate CA from issuing a valid certificate (without causing collateral damage) for any domain. All previously signed certificates by the intermediate CA will still be valid until revoked by the domains.

Algorithm 1 ELS Certificate Registration

Input: $cert$, L_{app}
Output: SCT, "unsound certificate"
begin
  if $((verify(cert, certICA[pk]) = true) \wedge cert \in L_{app})$ **then**
    if $((cert[validity.notBefore] < t_{now} < cert[validity.notAfter]) \wedge (cert[pk.lengh] = "Root - CA - defined - length") \wedge (cert[issuer] = certICA[subject]) \wedge (domainName = cert[subject]) \wedge (cert[basicConstriant] = false) \wedge (cert[signatureAlgorithm] = "Root - CA - defined - algorithm") \wedge (cert[keyUsage] = "digitalsignature") \wedge (cert[ExtendKeyUsage] = "serverAuth"))$ **then**
      $GenSCT(cert, t, sk_{ELS})$
      add($cert$) to pending registration list.
    else
      "unsound certificate"
    end
  end

Algorithm 2 SLS Certificate Registration

Input: $cert$, SCT
Output: SCT, "unsuccessful registration"
begin
  if $((verify(cert,\ certICA[pk]) = true) \wedge CheckSCT(SCT, pk_{ELS}))$ **then**
    if $((cert[validity.notBefore] < t_{now} < cert[validity.notAfter]) \wedge (cert[issuer] = certICA[subject]) \wedge (domainName = cert[subject]) \wedge (cert[keyUsage] = "digitalsignature") \wedge (cert[ExtendKeyUsage] = "serverAuth"))$ **then**
      $GenSCT(cert, t, sk_{SLS})$
      add($cert$) to pending registration list.
    else
      "unsuccessful registration"
    end
  end

Algorithm 3 Certificate Revocation

Input: $cert$, SCR
Output: SRT, "invalid revocation"

begin
  if $((verify(SCR,\ cert[pk]) = true) \wedge cert \in ELS)$ **then**
    if $((t_{now} < cert[validity.notAfter]) \wedge (domainName = cert[subject]))$ **then**
      $SCR \leftarrow sign(\{cert, SCR, t\}, sk_{ELS})$
      add($cert$) to pending revocation list.
    else
      "invalid revocation"
    end
  end

D. CONNECTION ESTABLISHMENT IN ARCT

Every domain periodically fetches proof of non-revocation from the corresponding ELS after each update. The client opens an SSL/TLS connection with the D.com server using Algorithm 4, and the D.com server sends the certificate along with evidence during the TLS handshake. The certificate is authenticated using the pre-installed CA's certificate in browsers, and SCTs and proof are validated using LSes' public-keys that are delivered to the browser. The leaf node of revocation tree s(i,j) represents the range that brackets the certificate $Cert_D^{ICA}$ serial number sn such that $i < sn < j$ whereas pk_{RCA} represents root CA's public-key.

E. CROSS LOGGING ELS SIGNED TREE HEAD (STH)

Periodically, at a well-known interval, each ELS updates its database by inserting all new certificates and publishes a fresh root for the current version of its database, which is signed by ELS as well as the root CA. The fresh STH is

Algorithm 4 TLS Secure Connection Establishment

Input: $Cert_D^{ICA}$,STH, SCT, $proof\,(PoN)$
Output: "Connected", "Unsuccessful attempt"
begin
  if $((pre\text{-}Validate(Cert_D^{ICA},\ name) = 0))$ **then**
    "Unsuccessful attempt"
  end
  if $((CheckSCT(SCT, pk_{SLS}) = 0) \wedge (CheckSTHT(STHT, pk_{SLS}) = 0) \wedge (CheckSCT(SCT, pk_{ELS}) = 0) \wedge (CheckSTH(STH, pk_{ELS}) = 0) \wedge (CheckSTH(STH, pk_{RCA}) = 0) \wedge (CheckPoN(Cert_D^{ICA}, s(i,j), STH, PoN, pk_{ELS}) = 0))$ **then**
    "Unsuccessful attempt"
  else if $(verify(Cert_D^{ICA}, pk_{ICA}) = 0)$ **then**
    "Unsuccessful attempt"
  else
    "Connected"
  end

forwarded to SLSes for witnessing, where each SLS generates signed receipt after performing necessary checks given in Algorithm 5, which is a cryptographic assurance of inserting to WitTree. This witnessing strengthens security against forking attack, in which ELS shows different signed versions of its log (one version for one set of users and another version another set of users).

Algorithm 5 Witnessing STH

Input: STH, PoE
Output: $Rcpt$, "invalid STH"

begin
  if $(verify(STH_j, pk_{ELS}) = true) \wedge CheckPoE(STH_i, STH_j, PoE, pk_{ELS}) = 1 \wedge (STH_j.t < t_{now}) \wedge (STH_j \notin L_{wit})$ **then**
    $Rcpt \leftarrow sign((STH_j, t_{now}), pk_{SLS})$
    add STH_j to pending witness list L_{wit}
  end
  else
    "Invalid STH'"
  end

VI. SECURITY ANALYSIS

In this section, we conduct an informal and formal security analysis and verify the main security feature that ARCT guarantees. Further, the proposed scheme security relies on the two lemmas for security. First, Table 2 compares security of ARCT with the following six log-based proposals:

TABLE 2. Comparison of ARCT with log-based PKI proposals based on security metric.

	SK	*CT*	*AKI*	*ARPKI*	*DTKI*	*Policert*	*ARCT*
Security							
MitM attack mitigation	√	×	√	√	√	√	√
Active control over delegated authority	×	×	×	×	×	×	√
Non-conformant certificate prevention	×	×	×	×	×	×	√
Client connection privacy	×	√	√	√	×	√	√
Certificate revocation	×	×	√	√	√	√	√

SK [48], CT [4], [5], AKI [3], ARPKI [6], DTKI [49], and Policert [13]. Second, in Theorem 1, we prove that ARCT thwarts impersonation when at most, one party is honest.

For the "MitM attack mitigation" metric, CT cannot guard clients against impersonation when a subverted CA issues a bogus TLS certificate for a domain. In SK and DTKI, each TLS certificate is cross-signed with a key-pair owned by the domain, whereas, in AKI, ARPKI, and Policert, a subject needs to get certified from more than one CA. The proposed scheme can prevent such attacks as a set of CAs verifies each domain identity (see Sec. V-A). Root CAs do not endorse end-entity certificates directly but cryptographically transfer that power to intermediate CAs. The intermediate CA can issue certificates (maybe a fake certificate) even for domains that are outside its scope, and most successful attacks are also launched against sub-CAs [61]. Compared with the log-based PKI frameworks, ARCT counters such failures of intermediate CAs by giving root CA the ability to prevent the exploitation of signing authority by them.

Log-based PKI proposals are still in infancy, and they failed to address the problem of non-conformant certificate-issuance by intermediate CAs. Reference [8] showed that the majority of small intermediate CAs have a high miss-issuance rate. However, the proposed scheme refrain intermediate CAs from issuing an erroneous certificate, since every certificate passes through a filtering process where ELS checks each certificate for soundness and correctness.

Theorem 1: If MHT and its authentication path are collision-resistant and the signature scheme is unforgeable, then an adversary $\mathcal{A}$ subverting log servers and intermediate CA cannot impersonate a domain (e.g., D.com) by logging a certificate $cert_D^{'ICA}$ for the domain.

Proof: Let intermediate CA issues a fake certificate $cert_D^{'ICA}$ for domain D.com. Since $cert_D^{'ICA}$ must be logged in the ELS (to be more specialized) to be accepted by a client, ELS must win the experiments defined in the below cases to impersonate D.com.

Case 1: If $\mathcal{A}$ wins $Exp_{ELS}^{Fake-Entry}(\mathcal{A})$ given in Figure 3, then $ELS^{\mathcal{A}()}$, which runs $\mathcal{A}$, finds a collision in MHT.

We demonstrate that a successful adversary $\mathcal{A}$ outputs two lists under the same root of the MHT used in ELS in the proposed scheme, which, by lemma 1, results in a hash collision.

$Exp_{ELS}^{FakeEntry}(\mathcal{A})$
1: $(L, L', STH, pk_{ELS}) \leftarrow \mathcal{A}()$
2: return 1
3: $if((CheckLogEntries(L, STH, pk_{ELS}) = 1) \wedge$
4: $(CheckLogEntries(L', STH, pk_{ELS}) = 1) \wedge (L \neq L'))$

FIGURE 3. Security experiment for generating a fake entry.

Suppose $\mathcal{A}$ wins $Exp_{ELS}^{Fake-Entry}(\mathcal{A})$ by forging MHT. Then $\mathcal{A}$ outputs (L, L', STH, pk) such that CheckLogEntries(L, STH, pk_{ELS}) = CheckLogEntries(L', STH, pk_{ELS}), but $L \neq L'$. Based on the definition of CheckLogEntries, this implies MHT collision $MHT(L) = MHT(L')$ but $L \neq L'$. By definition 2, this is a collision in MHT, which by Lemma 1 results in a collision in the hash function H.

Case 2: If $\mathcal{A}()$ wins $Exp_{ELS}^{Fake-proof}(\mathcal{A})$ given in Figure 4 by outputting $(cert_D^{ICA}, L, STH, PoN', pk_{ELS})$, then $ELS^{\mathcal{A}()}$, which runs $\mathcal{A}()$ and outputs $(cert_D^{ICA}, L, STH, PoN', pk_{ELS})$ breaks collision-resistance of authentication path of MHT.

$Exp_{ELS}^{Fake-PoN'}(\mathcal{A})$
1: $(cert_D^{'ICA}, L, STH, PoN', pk_{ELS}) \leftarrow \mathcal{A}()$
2: return 1
3: $if((CheckLogEntries(L, STH, pk_{ELS}) = 1)$
4: $\wedge(CheckPoN(cert_D^{'ICA}, s(i,j), STH, PoN', pk_{ELS}) = 1)$
5: $\wedge(s(i,j) \notin L))$

FIGURE 4. Security experiment for generating a fake PoN.

Suppose $\mathcal{A}$ wins $Exp_{ELS}^{Fake-PoN'}(\mathcal{A})$ by producing a fake proof of non-revocation and a tuple $(cert_D^{'ICA}, L, STH, PoN', pk_{ELS})$ such that CheckLogEntries(L, STH, pk_{ELS}) = 1 and CheckPoN(cert, s(i,j), STH, PoN', pk_{ELS}) = 1, but $s(i,j) \notin L$. As the proof PoN' contains hash values of nodes on the authentication path from range s(i,j) at some position say, m to the root, given a valid proof PoN' and certificate serial number range s(i,j), we can also calculate the hash values for the nodes on the authentication path. Generate an MHT for a list of series of ranges L, starting from the root node, and consider the first node at which node hash value in the authentication path varies from the node in the MHT for the list of series of ranges L. This breaks collision resistance of authentication path. By Lemma 2, a break of collision-resistance of authentication path of MHT results in a collision in H.

$Exp_{ELS}^{Fake-STH}(\mathcal{A})$
1:$(L', STH', PoN', pk_{RCA}) \leftarrow \mathcal{A}$
2: return 1
3: $if((CheckLogEntries(L', STH', pk_{ELS}) = 1)$
4: $\wedge(CheckPoN(cert_D'^{ICA}, STH', PoN', pk_{ELS}) = 1)$
5: $\wedge(CheckSTH(STH', pk_{RCA}) = 1)$
6: $\wedge(STH.TH \neq STH'.TH))$

FIGURE 5. Security experiment for generating a fake STH.

Case 3: In this case, an ELS acts as an adversary $\mathcal{A}$, and spawns a new MHT to launch a forking (split-view) attack, as the clients accept STH if it is signed by the root CA (see Algorithm 4). If $\mathcal{A}$ wins $Exp_{ELS}^{Fake-STH}(\mathcal{A})$ given in Figure 5 by outputting (L', STH, PoN', pk_{RCA}), then $ELS^{\mathcal{A}()}$, which runs $\mathcal{A}()$ and outputs (L', STH', PoN', pk_{RCA}) can forge signature scheme (EUF-CMA secure). For this, we create another signature forger adversary $\mathcal{B}$, and the attack game.

$Exp_{SIG}^{euf-cma}(\mathcal{B})$
1: $(pk, sk) \leftarrow KeyGen()$
2: $(m, \sigma) \leftarrow \mathcal{B}^{Sign}(pk_{(RCA)})$
3: return 1
4: $if(Verify(m, \sigma) = 1) \wedge (m \notin M))$

FIGURE 6. Security experiment for generating a forge signature.

First, we create a signature forgery attacker $\mathcal{B}$ given in Figure 6 against the signature algorithm, which works as follows. First, $\mathcal{B}$ creates a state with an empty set of certificate ranges. It then simulates the $Exp_{ELS}^{Fake-STH}$ for $\mathcal{A}$, providing the root CA public key as input in the game. It furthermore forms the signatures using its signing oracle when needed in the simulation of the game and stores all the values in a list queried to the signing oracle. If $\mathcal{B}$ wins by outputting (L', STH', PoN', pk_{RCA}), at least tree head TH was not outputted through the $\mathcal{B}$ simulation as we excluded the MHT collision case. Since TH was not contained in the list, $\mathcal{B}$ outputs this as a valid signature forgery. ■

VII. EVALUATION

To assess the effectiveness of ARCT, we first compare the revocation method of the proposed approach with state-of-the-art standardized schemes like CRL, OCSP, and log-based techniques. We then examine the storage cost of each LS (ELS and SLS) and also evaluate the performance of ARCT by building a prototype. At last, we compare the proposed framework with existing log-based proposals using various performance metrics.

A. NUMERICAL ANALYSIS

The communication cost of used data for revocation is optimal, and the proof supplied by the ELS is logarithmic in size, a domain can hold a short proof of non-revocation. The parameters we considered in evaluation are:

- There are $3.31*10^8$ domains [62], though only a fraction has TLS certificates. Estimated total number of domain TLS certificates ($n = 10^8$)
- Estimated average number of domain certificates managed by a root CA ($k = 10^6$)
- Estimated certificates revocation rate is 10% (p = 0.1)
- The TLS certificate status querying rate is 10^8 times per day (q = 10^8)
- Revocation update per day (T = 1)
- The hash function is SHA-256 ($l_{hash} = 256$ bits)
- Length of certificate serial number ($l_{sn} = 20$ bits)
- Number of bits required to represent the certificate revocation status ($l_{stat} = 100$ bits)
- In our scheme signature length ($l_{sign} = 512$ bits).

Values for p, T, l_{sn}, and l_{stat} are taken from [63], l_{hash}, and l_{sign}, are specific to ARCT as SHA-256 and ECDSA (SECP256k1) are used as hash function and signature algorithm. While k and q are based on the value of n, which is taken from [62]. Table 3 summarizes the bandwidth, update, and query cost of the CRL, OCSP, log-based PKIs, and ARCT. Log-based PKIs include AKI, ARPKI, DTKI, and Policert, whereas SK and CT are excluded as they do not support certificate revocation. The above mentioned log-based PKIs use the identical kind of data structure to prove that the certificate is revoked (proof of the absence of a certificate is provided to prove that the certificate is revoked). Without loss of generality, we presume that all parties use machines with a Core i7-8550U CPU @ 1.8, 8GB RAM, Window 10, using which, we calculate and get the following computational costs:

- Hash operation takes 4 μs.
- ECDSA (SECP256k1) signature generation takes 0.02 s.
- ECDSA (SECP256k1) verification takes 0.07 s.

We apply these values to Table 3 and get numerical cost, which is given in Table 4.

Analysis. We can observe from Table 4 that OCSP has less cost as compared to the other schemes. However, OCSP has a higher system risk because of heavy reliance on an online trusted third party and extra additional connection, which induces significant time overhead and violates user connection privacy [3]. From Table 4, we can observe that ARCT has less cost as compared to CRL and log-based PKI, and is comparable to OCSP in terms of cost. Additionally, ARCT introduces no extra latency, preserves client connection privacy, and reduces the need for trust in any trusted third party. We can conclude that ARCT revocation has a reasonable cost among certificate revocation schemes.

B. STORAGE COST OF ARCT

1) ELS

Space. The approximate size of a TLS certificate that uses 256 bits public key and signed with ECDSA is 2^9 [41]. We consider Let's Encrypt CA that had signed around 70 million certificates till July 2018.[4] Let's Encrypt needs storing

[4]https://letsencrypt.org/stats/

TABLE 3. Formulaic presentation of bandwidth, update, and query cost on CA, revocation prover, and verifier.

	CRL	OCSP	Log-based PKIs	ARCT
Update bandwidth cost	$T.n.p.l_{sn}$	0	0	0
Query bandwidth cost	$q.(p.k.l_{stat} + l_{sign})$	$q.l_{sig}$	$2.q.l_{hash}.log_2(p.k)$	$q.(log_2(p.k).l_{hash} + l_{sign})$
Update operational cost on CA	$T.C_{sign}$	0	0	0
Update operational cost on revocation prover	$T.C_{verify}$	0	$T.(2.q.l_{hash}.log_2(p.k).C_{hash} + C_{verify})$	$T.(q.log_2(p.k).C_{hash} + l_{sign})$
Query operational cost on revocation prover	0	$q.C_{sign}$	$T.(2.q.l_{hash}.log_2(p.k).C_{hash} + C_{verify})$	0
Query operational cost on verifiers	$q.C_{verify}$	$q.C_{verify}$	$T.(2.q.l_{hash}.log_2(p.k).C_{hash} + C_{verify})$	$T.(q.log_2(p.k).C_{hash} + l_{sign})$

TABLE 4. Numerical presentation of bandwidth, update, and query cost on CA, revocation prover, and verifier.

	CRL	OCSP	Log-based PKIs	ARCT
Update bandwidth cost	2×10^8	0	0	0
Query bandwidth cost	1.0×10^{15}	5.12×10^{10}	2.38×10^{12}	4.25×10^{11}
Update operational cost on CA	0.02	0	0	0
Update operational cost on revocation prover	0.07	0	1.32×10^4	6.64×10^3
Query operational cost on revocation prover	0	2×10^6	1.32×10^4	0
Query operational cost on verifiers	7×10^6	7×10^6	1.32×10^4	6.64×10^3

all 70×10^6 TLS certificates which is approximately in the order of $70 \times 10^6 \times 2^9 \approx 35.84$ *GB*, while revocation information requires storing 70×10^5 serial numbers which is approximately in the order of $2 \times 20 \times 70 \times 10^5 \approx 35$ *MB*. The price of a 1 GB storage space is around 0.054$,[5] the cost of storing an ELS is about 2$, which is a negligible amount. Again, the number of stored certificates in the log tree and the hash function used in computing the MHT determines the proof size. In ARCT SHA256 is used and the number of certificates is 70×10^6, the proof contains approximately 27 hashes, together with signed root hash; while revocation tree compromises 70×10^5 leaves, the proof contains around 23 hashes, together with signed root hash and leaf node. This is less than 1 KB in both cases.

Computation. Lookup, insertion, and update on ELS each involve $70 \times 10^6 \approx 27$ operations (in the worst case). Similarly, verifying proof requires $70 \times 10^6 \approx 27$ hash operations, which takes a negligible time because SHA256 takes around 4μs.

2) SLS

Space. We consider the cost of storing the witness tree (WitTree) as storing TLS certificates by SLS requires space depending on the number of certificates, as already examined in the above section. Currently, there are around 122 root CAs [10] supported by various web browsers/operating systems. Hence, WitTree requires storing 122 ELS STH, which is in the order of $122 \times 160 \times 365 \approx 7$ *MB*, per year, and when ELSes update their database once per day, where STH size is around 160 bytes. The cost is insignificant in terms of storage costs. Like in ELS, the number of stored certificates in the SLS and the hash function employed in yielding MHT determines the proof size as well as verification cost.

C. EXPERIMENTAL SETUP

To demonstrate the realization of ARCT in practice, we built a prototype with bare functionality. The prototype consists of 1) a web server (an ACME client) that requests a certificate from an intermediate CA, 2) a set of four CAs that perform collaborative verification. The CAs are implemented by modifying Pebble Challenge Test Server[6] to our requirements. On intermediate CA1,[7] we have enabled HTTP-01 and disabled DNS-01 as well as TLS-ALPN-01 challenge types. Likewise, intermediate CA2 and intermediate CA3 can only perform DNS-01 and TLS-ALPN-01 for domain verification, respectively. This experiment is conducted on mini-network with five computers where root CA (fourth CA) can select any challenge type randomly, and setup is connected to the Internet via 10 Mbit/s WiFi. Similarly, an instance of a log server is implemented by modifying an already existing Merkle tree[8] implementation in Python.

[5] https://diskprices.com.

[6] https://github.com/letsencrypt/pebble

[7] To ensure that CAs verify domain identity using different challenge type.

[8] https://github.com/jvsteiner/ merkletree

TABLE 5. Comparison of ARCT with log-based PKI proposals based on security, deployability, efficiency, and monopoly metrics.

	SK	*CT*	*AKI*	*ARPKI*	*DTKI*	*Policert*	*ARCT*
Deployability							
CA side changes required	×	√	×	√	×	×	√
Domain side changes required	√	×	√	√	√	√	√
Efficiency							
Extra latency for TLS connection setup	RTT	×	×	×	RTT	×	×
End-user additional action required	√	×	×	×	√	×	×
Monopoly							
Log key needed to be built into browser	√	√	√	√	MLM	√	√
Monitor key needed to be built into browser	√	×	√	√	×	√	×

Cost and Overhead. In the experiment, the domain server sends a certificate signing request (CSR) to intermediate CA1. The CA1 sends an HTTPS challenge to verify the identity of the domain. Similarly, intermediate CA2 and CA3 send their challenge to verify the domain identity. The whole process takes around 2 minutes, averaged over 100 runs. Going forward, the challenge verification takes 20 seconds per CA, averaged over 100 repetitions. From this, we can estimate that even for Symantec CA having 22 intermediate CAs, the certificate issuance will take about 7 minutes (if all intermediate CAs perform verification), which currently issues a certificate in about 6 minutes [9]. This shows that the collaborative certificate-issuance process is feasible in practice, and even [9], [10] suggested inducing delay to Lets Encrypt CA issuance process to avoid path poisoning attacks against it.

ARCT increases certificate size roughly by 300 bytes because of sending two or more than two SCTs with each certificate (embedding in X.509 certificate extension or sending as OCSP extension). Fortunately, it does not induce extra delay to the TLS handshake. Nevertheless, ARCT requires about 1KB of additional bandwidth for the TLS connection set up due to non-revocation proof stapling with each certificate. We measured the MHT generation, proof generation, and proof verification in python on a machine with Core i7-8550U CPU @ 1.8, 8GB RAM, Window 10, experimented with a million of revoked certificates logged in ELS. The MHT generation time averaged 12 s, proof generation time averaged 15 μs, while verification time-averaged to 240 μs.

D. COMPARISON

Based on deployability, efficiency, and monopoly metrics, Table 5 compares our scheme with six log-based proposals: SK [48], CT [4], [5], AKI [3], ARPKI [6], DTKI [49], and Policert [13].

Deployability. SK, AKI, DTKI, and Policert do not change the CA business model, as the CA can sign TLS certificates only, while in ARPKI CAs monitor LSes and other CA's behavior on behalf of their clients. In CT and our proposed scheme, the preferred process for CAs is to embed SCTs (Other mechanism includes sending them via TLS extension) received from LSes in the TLS certificates. For the "Domain side changes required" metric, SK and DTKI require each domain to generate an SK pair and a master-key pair to countersign the TLS certificate. In AKI, ARPKI, and Policert a domain contact multiple trusted CAs and staple their certificates. In the proposed framework, a domain server needs to download PoN and staple it with its certificate after each update of ELSes.

Efficiency. For the "Extra latency TLS connection setup/ End-user additional action required" metrics, in SK and DTKI, each client must visit timeline and MLM servers, respectively, before every connection to a domain (webserver). This extra connection induces additional latency that is equal to round-trip-time (RTT) as well as violates user privacy and exposes users to blocking attacks.

Monopoly. The last point we test is the monopoly. DTKI has more flexibility to add LSes since browsers have to store only MLM public-key, which lets new LSes to be flexibly added. However, as mentioned earlier, connections with MLM risks users' privacy, induces extra latency, and opens clients to blocking attacks.

VIII. DISCUSSION

Proactive defense. The private keys of CAs are lucrative targets for adversaries (e.g., criminals, hackers, and spy agencies) to use their keys secretly to compromise domains and their clients. We envision that in an ARCT in which not just a single CA but many of them examine and collaboratively issue a certificate, stolen CA keys (e.g., Comodo and DigiNotar) would not by themselves be useable to sign a fake but valid certificate for domains that clients would accept.

Deployment aspects. Many CAs, websites, and browsers have widely adopted CT. Google has also recently begun mandating CT logging for all TLS certificates. ARCT ELS could be deployed at each root CA level, and SLS could be deployed directly on the CT's server with the least modifications, given their resemblance.

While CT's LSes are optimized for deployability, ARCT is optimized both for security and deployability. The ARCT framework is planned to be interoperable with the existing CT model. CAs also have incentives to deploy ARCT due to security and transparency reasons. Therefore, we believe that with the heightened interest in a secure network framework, CAs and Internet society can deploy ARCT on the top of the CT with minor changes to the current CT model.

Transitioning, and Tradeoff between security and delay. One of the most significant issues when introducing a new method is the transitioning stage. However, for the adoption of our model, this is not a problem. The collaborative certificate-issuance focuses on ACME protocol, but it can be generalized and can be applied to any certificate-issuance because it involves only repeating the identity verification from multiple vantage points. Nevertheless, the proposed method induces extra latency of a few minutes, as discussed in VII-C.

The delay can be decreased at the cost of security (proactive approach) tradeoff in two ways. First, root CA and ELS allow logging certificates with one or two authorities verification, which will strictly decrease security level to the current certificate-issuance that is vulnerable to MitM attacks. Another option is to generate two SCTs in parallel: the first SCT merely attests that CAs and ELS have seen the certificate, and the second SCT attests that CAs and ELS have validated the certificate. ELS then provides the former SCT but withholds the second SCT if the CAs cannot accomplish their domain's identity validation in the specified time interval.

Third-party validator. Root CA determines when to start a validation round, and multicasts to all CAs the certificate(s) to be validated. In our collaborative domain's identity validation process, root CAs can also take the help of third-party validators (e.g., Internet Service Providers (ISPs), famous domains, or any interested party) in a domain's identity verification. This type of validation with third-party validators can decrease the burden over CAs and assure high security. Similarly, CAs that either issue free certificates (e.g., Let's Encrypt) or have less number of intermediate CAs can benefit from third-party validators directly.

Economic incentive. Root CAs are running businesses and sell TLS certificates to their clients. Security breaches, erroneous certificates, and lack of control over their delegated trust have undermined their reputations, which were challenging to mitigate and control. ELS and collaborative verification help root CAs to counter these challenges. Therefore, root CAs understand the lucrative value of ELS that make controlling their delegated trust and observing their intermediate CAs operations more accessible and transparent. Further, ELS would make it easier for domain owners to decide to select which root CA as their root of trust based on its previous history. Thus, a root CA with high security, quality certificate-issuance, and internal transparency (making intermediate CAs and their operation open to public scrutiny) would gain a competitive advantage, which would cause an increase in its market share.

IX. CONCLUSION

This article presents a new log-based PKI framework that leverages a log server (ELS) per root CA to make intermediate CA management transparent. The ELS counters the lack of root CAs control over their delegated trust as well as prevents malformed certificate-issuance. The new collaborative identity verification of domain offers strong resilience against adversaries, who can compromise a CA's private-keys. The method ensures that adversaries cannot maliciously issue a fake certificate that clients would accept, without exposing that certificate to the set of CAs for public scrutiny. The validation of each certificate by a group of CAs causes a high probability of immediate detection of malicious certificates. Security analysis shows that ARCT can successfully defend against impersonation and forking attacks. Evaluation results and comparison with log-based PKI schemes prove that ARCT is cost-effective and practical.

REFERENCES

[1] T. Fadai, S. Schrittwieser, P. Kieseberg, and M. Mulazzani, "Trust me, i'm a root CA! Analyzing SSL root CAs in modern browsers and operating systems," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 174–179.

[2] M. Li, L. Zhu, Z. Zhang, X. Du, and M. Guizani, "PROS: A privacy-preserving route-sharing service via vehicular fog computing," *IEEE Access*, vol. 6, pp. 66188–66197, 2018.

[3] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor, "Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure," in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, New York, NY, USA, 2013, pp. 679–690, doi: 10.1145/2488388.2488448.

[4] B. Laurie, A. Langley, and E. Kasper, *Certificate Transparency*, document RFC 6962, Internet Requests for Comments, RFC Editor, Jun. 2013.

[5] B. Laurie, "Certificate transparency," *Commun. ACM*, vol. 57, no. 10, pp. 40–46, Sep. 2014.

[6] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "Design, analysis, and implementation of ARPKI: An attack-resilient public-key infrastructure," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 393–408, May 2018.

[7] S. B. Roosa and S. Schultze, "Trust darknet: Control and compromise in the Internet's certificate authority model," *IEEE Internet Comput.*, vol. 17, no. 3, pp. 18–25, May 2013.

[8] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "Tracking certificate misissuance in the wild," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 785–798.

[9] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, "Bamboozling certificate authorities with BGP," in *Proc. 27th USENIX Conf. Secur. Symp. (SEC)*, New York, NY, USA, 2018, pp. 833–849.

[10] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient PKI," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2018, pp. 2060–2076, doi: 10.1145/3243734.3243790.

[11] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna, "Cloud strife: Mitigating the security risks of domain-validated certificates," in *Proc. 25th Netw. Distrib. Syst. Secur. Symp. (NDSS)*, P. Traynor and A. Oprea, Eds., 2018, p. 15, doi: 10.14722/ndss.2018.23327.

[12] S. Khan, Z. Zhang, L. Zhu, M. Li, Q. G. K. Safi, and X. Chen, "Accountable and transparent TLS certificate management: An alternate public-key infrastructure with verifiable trusted parties," *Secur. Commun. Netw.*, vol. 2018, pp. 1–16, Jul. 2018.

[13] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible TLS certificate management," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2014, pp. 406–417, doi:

[14] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 83–97.
[15] M. Cui, Z. Cao, and G. Xiong, "How is the forged certificates in the wild: Practice on large-scale SSL usage measurement and analysis," in *Computational Science—ICCS*, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, Eds. Cham, Switzerland: Springer, 2018, pp. 654–667.
[16] R. Housley, W. Polk, W. Ford, and D. Solo, *Internet x.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, document RFC 3280, Internet Requests for Comments, RFC Editor, USA, Apr. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3280.txt
[17] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP*, document RFC 2560, Internet Requests for Comments, RFC Editor, USA, Jun. 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2560.txt
[18] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh, "Towards short-lived certificates," in *Web 2.0 Security and Privacy*. May 2012, pp. 1–9.
[19] M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, "Global authentication in an untrustworthy world," in *Proc. 14th USENIX Conf. Hot Topics Operating Syst. (HotOS)*, New York, NY, USA, 2013, p. 19.
[20] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2008, pp. 321–334.
[21] M. Marlinspike. (2011). *Convergence*. Accessed: May 10, 2019. [Online]. Available: http://convergence.io/
[22] P. Eckersley and J. Burns. (2010). *The EFF SSL Observatory*. Accessed: May 11, 2019. [Online]. Available: https://www.eff.org/observatory
[23] A. Langley. (2011). *Public Key Pinning. Imperial Violet: Adam Langley's Weblog*. Accessed: Jul. 1, 2019. [Online]. Available: https://www.imperialviolet.org/2011/05/04/pinning.html
[24] C. Evans and C. Palmer. *Public Key Pinning Extension For*, document IETF Secretariat, Internet-Draft draft-ietf-websec-key-pinning-01, Working Draft, Dec. 2011. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-websec-key-pinning-01.txt
[25] M. Marlinspike, *Trust Assertions for Certificate Keys*, document, Working Draft, IETF Secretariat, Internet-Draft draft-perrin-tls-tack-02, Jan. 2013. [Online]. Available: http://www.ietf.org/internet-drafts/draft-perrin-tls-tack-02.txt
[26] P. Hoffman and J. Schlyter, *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*, document RFC 6698, Internet Requests for Comments, RFC Editor, Aug. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6698.txt
[27] P. Hallam-Baker, R. Stradling, and B. Laurie, *DNS Certification Authority Authorization (CAA) Resource Record*, document RFC 6844, Internet Engineering Task Force, 2013.
[28] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: Detection of widespread weak keys in network devices," in *Proc. 21st USENIX Conf. Secur. Symp.*, 2012, p. 35.
[29] J. Kasten, E. Wustrow, and J. A. Halderman, "Cage: Taming certificate authorities by inferring restricted scopes," in *Financial Cryptography and Data Security*, A.-R. Sadeghi, Ed. Berlin, Germany: Springer, 2013, pp. 329–337.
[30] J. Braun, F. Volk, J. Classen, J. Buchmann, and M. Mühlhäuser, "CA trust management for the Web PKI," *J. Comput. Secur.*, vol. 22, no. 6, pp. 913–959, Dec. 2014.
[31] J. Classen, J. Braun, F. Volk, M. Hollick, J. Buchmann, and M. Mühlhäuser, "A distributed reputation system for certification authority trust management," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 1349–1356, doi: 10.1109/Trustcom.2015.529.
[32] K. Lewison and F. Corella, "Backing rich credentials with a blockchain PKI," Pomcor, Sacramento, CA, USA, Tech. Rep., 2016. [Online]. Available: https://www.pomcor.com/techreports/BlockchainPKI.pdf
[33] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts (BCC)*, New York, NY, USA, 2017, p. 35, doi: 10.1145/3055518.3055530.
[34] A. Loibl and J. Naab. (2014). *Namecoin*. Accessed: Jun. 5, 2019. [Online]. Available: https://www.namecoin.org/
[35] L. Dykcik, L. Chuat, P. Szalachowski, and A. Perrig, "BlockPKI: An automated, resilient, and transparent public-key infrastructure," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2018, pp. 105–114.
[36] Z. Guan, A. Garba, A. Li, Z. Chen, and N. Kaaniche, "AuthLedger: A novel blockchain-based domain name authentication scheme," in *Proc. 5th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, vol. 1, 2019, pp. 345–352, doi: 10.5220/0007366803450352.
[37] C. Fromknecht, D. Velicanu, and S. Yakoubov, "Certcoin: A namecoin based decentralized authentication system," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2014, vol. 6, pp. 46–56. [Online]. Available: https://courses.csail.mit.edu/6.857/2014/files/19-fromknecht-velicann-yakoubov-certcoin.pdf
[38] L. Axon and M. Goldsmith, "PB-PKI: A privacy-aware blockchain-based PKI," in *Proc. 14th Int. Joint Conf. e-Bus. Telecommun. (ICETE)*, 2017, pp. 311–318, doi: 10.5220/0006419203110318.
[39] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proc. 2016 USENIX Conf. Usenix Annu. Tech. Conf. (ATC)*, 2016, pp. 181–194.
[40] Z. Wang, J. Lin, Q. Cai, Q. Wang, J. Jing, and D. Zha, "Blockchain-based certificate transparency and revocation transparency," in *Financial Cryptography and Data Security*, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds. Berlin, Germany: Springer, 2019, pp. 144–162.
[41] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "Certledger: A new PKI model with certificate transparency based on blockchain," *Comput. Secur.*, vol. 85, pp. 333–352, Aug. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404818313014
[42] A. Yakubov, W. M. Shbair, A. Wallbom, D. Sanda, and R. State, "A blockchain-based PKI management framework," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–6.
[43] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "CertChain: Public and efficient certificate audit based on blockchain for TLS connections," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2060–2068.
[44] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology—CRYPTO*, J. Katz and H. Shacham, Eds. Cham, Switzerland: Springer, 2017, pp. 357–388.
[45] S. Yao, J. Chen, K. He, R. Du, T. Zhu, and X. Chen, "PBCert: Privacy-preserving blockchain-based certificate status validation toward mass storage management," *IEEE Access*, vol. 7, pp. 6117–6128, 2019, doi: 10.1109/ACCESS.2018.2889898.
[46] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2016, pp. 168–179, doi: 10.1145/2976749.2978404.
[47] M. Al-Bassam and S. Meiklejohn, "Contour: A practical system for binary transparency," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, J. Garcia-Alfaro, J. Herrera-Joancomartí, G. Livraga, and R. Rios, Eds. Cham, Switzerland: Springer, 2018, pp. 94–110.
[48] P. Eckersley. (2012). *Sovereign Key Cryptography for Internet Domains*. Accessed: Jun. 10, 2019. [Online]. Available: https://www.eff.org/sovereign-keys
[49] J. Yu, V. Cheval, and M. Ryan, "DTKI: A new formalized PKI with verifiable trusted parties," *Comput. J.*, vol. 59, no. 11, pp. 1695–1713, Nov. 2016.
[50] J. Chen, S. Yao, Q. Yuan, R. Du, and G. Xue, "Checks and balances: A tripartite public key infrastructure for secure Web-based connections," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9, doi: 10.1109/INFOCOM.2017.8057201.
[51] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Proc. 25th Int. Conf. Comput. Aided Verification (CAV)*, vol. 8044. Berlin, Germany: Springer-Verlag, 2013, pp. 696–701.
[52] B. Laurie and E. Kasper, "Revocation transparency," Google Res., Tech. Rep., Sep. 2012.
[53] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS). Internet Soc. (NDSS)*, Jan. 2014, p. 14.
[54] A. Singh, B. Sengupta, and S. Ruj, "Certificate transparency with enhancements and short proofs," in *Information Security and Privacy*, J. Pieprzyk and S. Suriadi, Eds. Cham, Switzerland: Springer, 2017, pp. 381–389.
[55] P. Szalachowski, L. Chuat, and A. Perrig, "PKI safety net (PKISN): Addressing the too-big-to-be-revoked problem of the TLS ecosystem," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 407–422.

[56] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology—(CRYPTO)*. Berlin, Germany: Springer-Verlag, 1989, pp. 218–238.
[57] B. Dowling, F. Günther, U. Herath, and D. Stebila, "Secure logging schemes and certificate transparency," in *Computer Security—ESORICS*, I. Askoxylakis, S. Ioannidis, and C. Katsikas, S. Meadows, Eds. Cham, Switzerland: Springer, 2016, pp. 140–158.
[58] T. Pulls and R. Peeters, "Balloon: A forward-secure append-only persistent authenticated data structure," in *Computer Security—ESORICS*, G. Pernul, P. Y. A. Ryan, and E. Weippl, Eds. Cham, Switzerland: Springer, 2015, pp. 622–641.
[59] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, *Automatic Certificate Management Environment (ACME)*, document RFC 8555, RFC Editor, Internet Requests for Comments, Mar. 2019.
[60] P. C. Kocher, "On certificate revocation and validation," in *Financial Cryptography*, R. Hirchfeld, Ed. Berlin, Germany: Springer, 1998, pp. 172–177.
[61] A. Langley. (2015). *Maintaining Digital Certificate Security*. Accessed: Jul. 7, 2019. [Online]. Available: https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html
[62] *The Verisign Domain Name Industry Brief 2018*. Accessed: Aug. 10, 2019. [Online]. Available: http://www.verisign.com/en_US/innovation/dnib/index.html
[63] W. Zhang, M. Kandemir, A. Sivasubramaniam, and M. J. Irwin, "Performance, energy, and reliability tradeoffs in replicating hot cache lines," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst. (CASES)*, New York, NY, USA, 2003, pp. 309–317, doi: 10.1145/951710.951750.

SALABAT KHAN is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. His current research interests include secure and transparent public-key infrastructure (PKI), transparent and secure key management in VANETs, distributed ledger technology, and blockchain.

LIEHUANG ZHU (Member, IEEE) is currently a Professor with the Department of Computer Science, Beijing Institute of Technology. He is selected into the Program for New Century Excellent Talents in University from the Ministry of Education, China. His research interests include the Internet of Things, cloud computing security, and the Internet and mobile security.

ZIJIAN ZHANG (Member, IEEE) is currently with the Beijing Institute of Technology and The University of Auckland. His research interests include entity authentication and key exchange, entity identification, and privacy protection.

MUSSADIQ ABDUL RAHIM is currently pursuing the Ph.D. degree with the School of Computer Science, Beijing Institute of Technology. His research interests include artificial intelligence and privacy mining and its applications in information security.

KHALID KHAN (Graduate Student Member, IEEE) is currently a Ph.D. Scholar with the Kohat University of Science and Technology, Kohat, Pakistan. His research interests include secure data aggregation in wireless sensor networks, secure authentication in the Internet of Things, and intrusion detection system in *ad hoc* networks and the Internet of Things.

MENG LI received the Ph.D. degree from the School of Computer Science and Technology, Beijing Institute of Technology. He is currently an Associate Researcher with the College of Computer Science and Information Engineering, Hefei University of Technology. He was sponsored by the China Scholarship Council to study as a Visiting Ph.D. Student with Wilfrid Laurier University, in 2017. His research interests include applied cryptography, security and privacy, vehicular networks, fog computing, and blockchain.