2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00121

ScalaCert: Scalability-Oriented PKI with Redactable Consortium Blockchain Enabled "On-Cert" Certificate Revocation

Xinyi Luo*, Zhuo Xu*, Kaiping Xue*§, Qiantong Jiang*, Ruidong Li†, David Wei‡
* School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
† College of Science and Engineering, Kanazawa University, Kakuma-machi, Kanazawa 920-1192, Japan
‡ Department of Computer and Information Science, Fordham University, Bronx, NY 10458, USA
§Corresponding author, kpxue@ustc.edu.cn

***Abstract*—As the voucher for identity, digital certificates and the public key infrastructure (PKI) system have always played a vital role to provide the authentication services. In recent years, with the increase in attacks on traditional centralized PKIs and the extensive deployment of blockchains, researchers have tried to establish blockchain-based secure decentralized PKIs and have made significant progress. Although blockchain enhances security, it brings new problems in scalability due to the inherent limitations of blockchain's data structure and consensus mechanism, which become much severe for the massive access in the era of 5G and B5G. In this paper, we propose ScalaCert to mitigate the scalability problems of blockchain-based PKIs by utilizing redactable blockchain for "on-cert" revocation. Specifically, we utilize the redactable blockchain to record revocation information directly on the original certificate ("on-cert") and remove additional data structures such as CRL, significantly reducing storage overhead. Moreover, the combination of redactable and consortium blockchains brings a new kind of attack called deception of versions (DoV) attack. To defend against it, we design a random-block-node-check (RBNC) based freshness check mechanism. Security and performance analyses show that ScalaCert has sufficient security and effectively solves the scalability problem of the blockchain-based PKI system.**

***Index Terms*—PKI system, certificate revocation, redactable blockchain, consortium blockchain, scalability**

I. Introduction

Digital certificates provide an effective mean to voucher an entity's identity and thus play a vital role in all kinds of networks, including the Internet [1], Internet of Things (IoT) [2], Internet of Vehicles (IoV) [3], [4], and so on. Traditionally, there is a trusted third party called the certificate authority (CA) to verify entities' identities and manage their certificates, usually including issuance, update, and revocation [5]. CA plus some other auxiliary entities, such as registration authorities (RAs), form the entire certificate system, generally called the public key infrastructure (PKI) system [6]. For a very long time, the centralized PKI has been widely adopted and has become the cornerstone of the Internet, especially for the SSL/TLS protocols [7], [8]. However, with the continuous development of network technology, the centralized PKI has encountered two thorny challenges, i.e., security and trust. On the one hand, researchers have discovered numerous security vulnerabilities of the centralized PKI, such as single point of failure of CAs [9]. On the other hand, with the development of heterogeneous networks, there are usually multiple trust domains within one network, and how to establish trust relationships among these different domains is also a formidable challenge for traditional PKI systems [10].

In response to these challenges difficult for the centralized PKI to solve, researchers began to seek other more suitable architectures for today's certificate systems. An intuitive thought is to use multiple CAs to jointly manage certificates and conduct mutual audits to enhance security, such as ARPKI [11]. However, although the adoption of the multi-CA structure has alleviated some security problems caused by single-CA, there are still serious challenges, e.g., the data security and the trust gap. On account of this, researchers recently found that blockchain technology may provide suitable solutions due to its public and non-tampering storage and decentralized consensus and consequently proposed the blockchain-based certificate systems, such as CertChain [12] and CertLedger [13]. The main idea of these solutions is to use multiple CAs to construct a consortium blockchain [14] and store the operations on certificates in the blocks (just like transactions in cryptocurrencies). The usage of blockchain effectively guarantees the data security and establishes the solid mutual trust among CAs and users, properly satisfying today's security requirements on PKI system.

However, the introduction of blockchain brought new problems in terms of scalability for PKI system due to its inherent data structure and consensus requirements. Moreover, to guarantee correct certificate verification, the revocation information, such as certificate revocation lists (CRLs) [1], should also be stored on-chain, causing massive waste of the valuable on-chain space in blockchain. For example, CertChain needs about one-fifth of the block space to store the bloom-filter-based CRLs. If we only take the Internet SSL/TLS protocol requirements into consideration, as analyzed in CertLedger [13], this kind of waste will not cause essential effect. However, the rapid development of IoT and 5G and future B5G technologies promotes massive devices to access the network, leading to a dramatic increase in digital certificates and putting further pressure on PKIs' scalability [15]. According to the Ericsson

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00121

mobility report (June 2021) [16], mobile subscriptions are projected to reach 8.8 billion units by the end of 2026. Therefore, to make the blockchain-based PKI more practical for heterogeneous networks with massive devices, it is urgent to address the scalability problem.

In this paper, to enhance the scalability of the blockchain-based PKI, we propose an "on-cert" certificate revocation mechanism using redactable blockchain [17]. With the proposed mechanism, certificates can be revoked without additional data structures, and thus the problem on huge amount of storage waste caused by CRLs can be well addressed. Different from the general non-tampering blockchains, the redactable blockchain adopted in the "on-cert" revocation mechanism enables conditional rewriting by adopting the Chameleon hash [18]. In brief, a blockchain node with the Chameleon private key can modify the data of an existing block without affecting other blocks. Capitalizing on that, when a certificate is revoked, we can record the revocation operation directly on the original certificate stored in the block by modifying it using the Chameleon hash in the "on-cert" revocation mechanism. In such a situation, there are three significant challenges to be solved, i.e., permission to Chameleon hash, consensus on block modification, and version freshness check. First, which data can be modified by which nodes should be reasonably restricted. Then, how do the blockchain nodes reach a consensus on whether to accept the new block when a block is modified. And the last is caused by the combination of redactable and consortium blockchains. In consortium blockchain (which is adopted by blockchain-based PKIs), only consortium members are full nodes and participate in the consensus process. Other users, due to lack of computing power or access permission, need to access the blockchain by requesting blocks from the consortium members. It doesn't matter for the general blockchains, as users can determine whether a received block is valid by checking the block's hash value. However, in redactable blockchains, different versions of one block hold the same hash value. Therefore, users cannot independently determine whether a received block is the latest version.

To further solve the problems, we propose an "on-cert" certificate revocation mechanism and further ScalaCert using the redactable blockchain, where revocation information is recorded directly on the original certificate. In summary, this paper makes the following contributions:

1) We propose "on-cert" certificate revocation mechanism with redactable blockchains. Based on such mechanism, we further propose ScalaCert, which enables to record revocation operations without any additional data structure such as (compressed) CRL, thereby significantly reducing the storage overhead of blockchain-based PKIs. Furthermore, to solve the security vulnerabilities caused by the redaction on blockchain data, we set strict permission restrictions to Chameleon hash, and design a *redaction consensus* algorithm for the consensus on redactions and prove its safety in theory.
2) We analyze how the combination of redactable and consortium blockchains can affect the security and discover a new attack called *deception of version (DoV)* attack. To defend against it, we design a *random-block-node-check (RBNC)* based freshness check mechanism, and provide security analysis to prove that it can decrease the probability of successfully conducting DoV attacks to a statistically negligible level.
3) Since ScalaCert trades communication for storage, to evaluate how much the communication overhead is affected, we implement prototypes of both ScalaCert and CertChain and compare their performance in terms of storage, communication, computing, *et al*. Results show that, compared to CertChain, ScalaCert saves around one-fifth of the valuable on-chain storage space while doubling the communication overhead. Especially for scenarios with massive devices with a strong demand for scalability like IoT, such an exchange of overhead is reasonable.

The rest of this paper is organized as follows. Section II introduces some important backgroup technologies. Section III describes the system model, security assumption and design goal, and Section IV expounds the proposed "on-cert" revocation mechanism and further ScalaCert. Section V and VI give the security and performance analysis. Section VII presents the related work. Section VIII concludes the paper.

II. Preliminaries

A. Blockchain and Consortium Blockchain

Blockchain has gained rapid development and wide application since Satoshi Nakamoto proposed Bitcoin [19] in 2008. It is regarded as a special decentralized data storage system that cannot be tampered with due to the well-designed data structure and consensus mechanism. On the one hand, in a blockchain, data is stored in linearly arranged blocks, and the correction of the arrangement order is guaranteed by recording the hash value of the previous block in each one. On the other hand, the consensus mechanism, e.g., proof of work (PoW) [20] and proof of stake (PoS) [21], ensures that all blockchain nodes have the same view of the blockchain. It also brings out a cost to generate blocks so that no blockchain node can generate an unlimited number of blocks. Except for the widely used public blockchains, such as Bitcoin or Ethereum [22], the consortium blockchains are also of great importance, especially in the scenes of commercial cooperation. In a consortium blockchain, only members in the consortium can access the blockchain directly and participate in the consensus process. Other entities can only access the blockchain by acquiring blocks from the consortium members.

B. Redactable Blockchain and Chameleon Hash

Most blockchains have the same important characteristic, i.e., the immutability of data. However, there is a special category of blockchain called redactable blockchain that supports limited change of data. By using the trapdoor hash function (TDH) [23], one who has the trapdoor can easily keep the hash value unchanged even when the message changes. With this, the redactable blockchain allows someone to modify one block

without affecting other blocks. In the proposed ScalaCert, we will use the Chameleon hash [18] which is an instantiation of TDH, and its details are as follows.

- $G(1^{\alpha})$: Given the security parameter α, generate the public key, pk, and private key, sk, for Chameleon hash.
- $Hash(pk, m, \lambda)$: Given the public key, pk, data, m, and a random number, λ, generate a hash value, hv, and the random number, ξ.
- $VerifyHash(pk, m, (hv, \xi))$: Given the public key, pk, data, m, hash value, hv, and the random number, ξ, check whether (hv, ξ) is a correct hash. If so, return 1; otherwise return 0.
- $Collision(sk, m')$: Given the private key, sk, and the new data, m', generate a new random number ξ', making $VerifyHash(pk, m', (hv, \xi')) = 1$.

III. System Model, Security Assumption and Design Goal

A. System Model

There are three kinds of entities in the proposed system, i.e., CAs, blockchain nodes, and users, as shown in Fig. 1.

- *CA*. Similar to traditional PKI systems, CA is the server responsible for issuing, updating, and revoking certificates for users. However, there is an important difference between traditional CAs and blockchain CAs: credibility. The traditional CAs act as trusted authorities, but the blockchain CAs are not. In our proposed system, multiple CAs form a CA alliance (CAA) to achieve credibility through consensus and mutual audits among them. Here we use $\mathbb{CAA} = \{CA_1, CA_2, ..., CA_n\}$ to denote the CAA and its CAs.
- *Blockchain Node*. Each CA in CAA is a blockchain node of the consortium blockchain, and other nodes do not have permission to join the consortium blockchain, so they cannot directly access the blockchain but can request blocks from the blockchain nodes. We use $\mathbb{BN} = \{BN_1, BN_2, ..., BN_n\}$ to denote the blockchain nodes.
- *User*. A user has two main demands for the PKI system, i.e., maintaining his/her own certificates and verifying others' certificates. As lightweight clients, users do not have sufficient capabilities to maintain the entire blockchain and are supposed to request blocks from BNs when needed and can verify the blocks' correctness according to the consensus proofs in the blocks.

B. Security Assumptions

We consider the underlying network communications reliable. That is, once a BN or user sends a message to another one, the receiver will definitely receive the correct message within the limited time. Moreover, according to the main consensus mechanisms of the consortium blockchain, such as practical Byzantine fault tolerance (PBFT) [24], [25] or delegated proof of stake (DPoS) [26], it is reasonably assumed that more than $\frac{2}{3}$ CAs in CAA are honest (implying that more than $\frac{2}{3}$ BNs are honest), and others may conduct malicious or negative operations, such as Sybil attacks or DoV attacks (see Section IV-E).

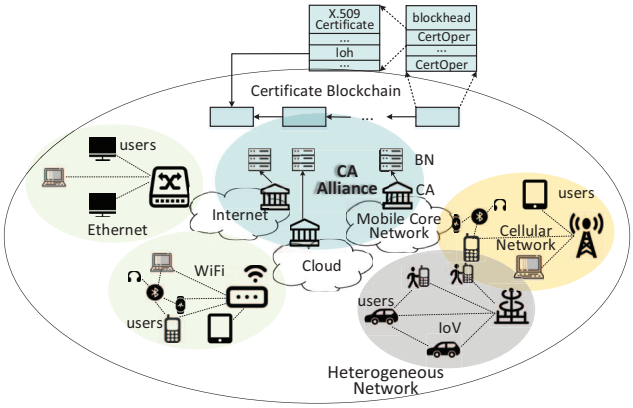


Fig. 1. System model of consortium-blockchain-based PKI.

C. Design Goal

1) Scalability: We transform this goal into no extra data structure for revocation because the existing blockchain-based PKI systems need a lot of storage for revocation and thus suffer from the scalability problem. To the best of our knowledge, we are the first to pursue this goal. To this end, we aim to remove all the extra data structures, such as the CRL or its digest, and only include the certificates in the blockchain.

2) Safety: All the valid certificates and the revocation operations should be appropriately recorded on the blockchain. The users who are not full nodes of the blockchain should be able to acquire correct data from blockchain nodes.

3) Liveness and Fairness: There may be some honest but negative BNs who do not actively participate in system maintenance to save his/her expenses or something else. Thus, liveness requires that there are always sufficient active BNs to participate in system maintenance. Besides, fairness requires that honest and active BNs are rewarded, and negative or malicious ones are punished. Both the two requirements can be guaranteed by adopting audit and incentive mechanisms in a blockchain system.

IV. Proposed ScalaCert

A. Overview

In this paper, to enhance the scalability of blockchain-based PKI systems, we propose an "on-cert" certificate revocation mechanism using the redactable blockchain with Chameleon hash. By such a mechanism, the revocation information is directly recorded on the original certificate, and therefore no extra data structure such as CRL is required. Our main idea is that multiple CAs together form the *CA alliance* (CAA) and maintain a consortium blockchain. All the certificates responsible by these CAs are recorded on the blockchain by consensus, and we set a special field (revocation field) in the certificate to denote whether it is a revoked one. When a certificate is required to be revoked, at first, CAA records

the revocation operation on the blockchain. Then the CA who issued this certificate updates the revocation field to denote its revocation. Finally, CAA conducts another specially designed consensus, called *redaction consensus*, to reach a consensus on the block's update. Besides, to avoid occupying too much space to describe the construction of blockchain-based certificate systems, which is not the main content of this work, we instantiate our proposed scheme based on an excellent job, the CertChain [12]. On the other hand, for a user who needs to verify a certificate's validity, he/she requests the block where the certificate is located to check whether it is valid or revoked according to the revocation field. During the verification process, a serious problem is that users cannot distinguish between different versions of the block because different versions have the same hash value. Thus, blockchain nodes can conduct the *deception of version (DoV) attack* to conceal the revocation records of certificates from users. To further solve this problem, we design a *freshness check* mechanism by acquiring random block sets from random node sets, thereby reducing the probability of successfully conducting DoV attacks to a statistically negligible level. Besides, TABLE I lists the main notations and functions used in ScalaCert.

TABLE I
IMPORTANT NOTATIONS AND FUNCTIONS

Notation or Function	Meaning
t	Time slot
h	Block height
h_t	Height of the block generated at slot t
H	Height of the latest block
$B[h]$	Block at height h
BN	Blockchain node
n	Total number of CAs/BNs
CertOper	Certificate operations
loh	Last operation height
$local$	In contrast to a received *modified* block or CertOper
$\mathbb{UDP}[t]$	Set of blocks modified at slot t
$\mathbb{ERR}[t]$	Set of blocks fail to be modified at slot t
$\mathbb{RN}$	Randomly selected BNs for $Freshcheck()$
$\mathbb{RH}$	Randomly selected block heights for $Freshcheck()$
$\mathbb{RB}[i]$	Received result from BN_i during $Freshcheck()$
$Revoke()$	Require to revoke a certificate
$Verify()$	Verify the validity of a certificate
$Extend()$	Common consensus for adding new blocks
$Redact()$	Redaction consensus for modified blocks
$Freshcheck()$	Check whether a block is the latest version
$Audit()$	Mutual audit among CAs
$Traceback()$	Trace back to the issuance CertOper
$Collision()$	Recompute Chameleon hash after changing loh

B. System Initialization

1) CertChain: CertChain is a representative construction of consortium blockchain-based PKI. It stores the certificates and the bloom-filter-based CRLs on the blockchain and adopts a PoS-like consensus called *dependability-rank based consensus* protocol. The most critical design of CertChain is CertOper. It is a data structure defined in CertChain to express certificate operations, including issuance, update, and revocation, and it is the content (i.e., transactions) of blocks. A CertOper is basically the same as an X.509 certificate, except for two special fields, i.e., the *Operation Type* and *Last Operation Height* (loh). *Operation Type* indicates which of the three operations is for this CertOper, and loh is used to link all operations of a certificate together such that it is easy to trace back to any previous operations. For clarity, the three kinds of operations and how to set the loh are as follows.

- CertOper^I: The issuance CertOper, generated when a certificate is firstly issued, and loh is set to 0.
- CertOper^U: The update CertOper, generated when a certificate is updated, and loh is set to the height of the block which contains the last CertOper of the same certificate.
- CertOper^R: The revocation CertOper is generated when a certificate is revoked, and the way to set loh is the same as update CertOper.

When a user requests to issue, update or revoke a certificate, the related CA then generates a CertOper for the operation. And at each time slot, a leader who is selected according to the dependability collects all the generated CertOpers to form a new block and adds the block to the blockchain by the consensus algorithm. Moreover, each block contains a bloom-filter-based CRL to record the revoked certificates. When a user needs to verify a certificate's validity, he/she first checks whether the related CertOper is contained in the blockchain and then refers to the CRL stored in the latest block to check whether the certificate is revoked or not.

Data Structure Modification: CertOper *with rewritable* loh. To enable "on-cert" revocation, we modify CertOper to enable the rewriting of loh, and use it as the revocation field. Specifically, when generating hash values, the loh field is hashed using the Chameleon Hash while others are as before. The CertOper mentioned below all refers to the CertOper with rewritable loh.

2) Key Pairs: CAs are responsible for two main steps, i.e., signing $CertOper$s for users and computing Chameleon hash. Thus, each CA owns a public-private key pair (pk^{CA}, sk^{CA}) for digital signature, and a key pair (pk^{CH}, sk^{CH}) for Chameleon hash. Besides, blockchain nodes are required to generate and sign blocks. Therefore, each blockchain node owns a key pair (pk^{BN}, sk^{BN}) for digital signature.

C. "On-Cert" Revocation and Verification

We herein design the "on-cert" revocation mechanism to realize certificate revocation and verification without any additional data structure such as CRL.

1) "On-Cert" Revocation: When a user discovers that the private key of its certificate is leaked, he/she will send a revocation request (RReq) with a private-key-signed signature to any CA in CAA to revoke it. When receiving a RReq, CA first checks whether it is valid, i.e., is signed by the certificate's owner. If valid, CA then generates the corresponding CertOper^R and waits for it to be included in the blockchain through *common consensus* (defined in Section IV-D). As Algorithm 1 shows, suppose that

$\{\mathsf{CertOper}_0^R, \mathsf{CertOper}_1^R, ..., \mathsf{CertOper}_k^R\}$ are the revocation operations that are added to blockchain during time slot t. Then at time slot $t+1$, for each $\mathsf{CertOper}^R$, its issuance CA updates the *last operation height* (loh) field from 0 to the block height h_t of the corresponding issuance $\mathsf{CertOper}^I$, and then broadcasts the updated $\mathsf{CertOper}^I$ for the *redaction consensus* to synchronize the block's modification among all the CAs in CAA (see Section IV-D).

Algorithm 1: Revocation of slot t

```
while slot t do
    CertOper^R ← Revoke(cert);
    B[h_t] ← Extend(CertOpers during slot t);
end
while slot t+1 do
    for each CertOper^R in B[h_t] do
        CertOper^I ← Traceback (CertOper^R);
        if CertOper^I.loh = 0 then
            CertOper^I.loh ← h_t;
            Collision(sk_CH, loh);
            Redact (CertOper^I);
        end
    end
end
```

2) Verification: When a user needs to verify whether a certificate cert is valid, he/she first generates a validation request $\langle \mathsf{cert}, h, pk_{CA} \rangle$ and sends it to any blockchain node (denoted by BN^0). Then, as Algorithm 2 shows, BN^0 refers to $B[h]$ to check whether cert is valid. If valid, BN^0 then traces back to cert's issuance CertOper through the *last operation height* field, denoted by $\mathsf{CertOper}^I$ with block height h^I. Then, BN^0 sends $B[h]$ together with $B[h^I]$ back to the user. The user first verifies the blocks' and CertOpers' validity by checking the hashes and signatures, and then checks the value of loh field in $\mathsf{CertOper}^I$. If the value is not 0, it is considered that cert has been revoked. However, it should be noted that $\mathsf{loh} = 0$ does not mean cert isn't revoked due to the probability of the DoV attack. To this end, the user should conduct the freshness check process when $\mathsf{CertOper}^I.\mathsf{loh} = 0$ (see Section IV-E).

D. Consensus and Audit

There are two kinds of consensus in the proposed system, i.e., the common consensus and the redaction consensus. The common consensus is the common blockchain consensus, with the purpose of generating new blocks and adding them to the blockchain. And the redaction consensus aims at reaching a consensus on the block's redactions.

1) Common Consensus: Common consensus is the common blockchain consensus used to add new blocks to the blockchain. As the proposed revocation mechanism is instantiated on CertChain, we reserve the *dependability-rank based consensus protocol* of CertChain and further extend it by adding the redaction consensus and new audit mechanisms.

Algorithm 2: Verify(cert, h, pk_{CA})

```
BN^0 do
if B[h] not include CertOper then
    cert is invalid;
end
B[h^I] ← Traceback(CertOper);
BN^0 sends (B[h], B[h^I]) to user;
The user do
if B[h^I].CertOper^I.loh ≠ 0 then
    cert is revoked;
end
else
    Freshcheck(B[h^I]);
end
```

The dependability-rank-based consensus protocol is similar to the proof-of-stake (PoS) consensus protocol, where each blockchain node has an attribute called *dependability* (like *stake* in PoS), and the probability of generating blocks is relative to the value of dependability. At each time slot, a leader is selected according to the dependability value, and it then generates a block consisting of CertOpers generated during this slot.

2) Redaction Consensus: At each time slot t, all the blockchain nodes run the redaction consensus for the redacted blocks generated by the certificate revocation mechanism during the last slot $t-1$. When a certificate is revoked, the corresponding $\mathsf{CertOper}^R$ is generated and included in the latest block. Then, in the next time slot, the issuance CA should update the related $\mathsf{CertOper}^I$ to record the revocation operations back to the certificates. As such, the modification of blocks should be synchronized among all the blockchain nodes, and we call this process the *redaction consensus*.

Algorithm 3: Redact(CertOper) at slot t

```
for each BN_i ∈ 𝔹𝕟 do
    if CertOper.loh ≠ h_{t-1} then
        Terminate;
    end
    if B[h_{t-1}] not include CertOper^R then
        Terminate;
    end
    if CertOper_local.loh ≠ 0 then
        Terminate;
    end
    Update CertOper_local to CertOper;
    𝕌ℙ𝔻[t − 1].append(CertOper);
end
```

As Algorithm 3 shows, at time slot t, the CA modifies $\mathsf{CertOper}^I$ for the revocation operation $\mathsf{CertOper}^R$ generated at time slot $t-1$ and broadcasts the modification to other blockchain nodes. Then each blockchain node should make

the following judgments to determine whether to accept a modified block:

- $\mathsf{CertOper}^I.\mathsf{loh} = h_{t-1}$. To check whether the revocation operation is generated in the last slot. As required, all $\mathsf{CertOper}^R$s generated at time slot t should be processed at time slot $t+1$. Otherwise, the revocation is failed, and the CA should generate a new $\mathsf{CertOper}^R$ to request for revocation again.
- $B[h_{t-1}]$ includes $\mathsf{CertOper}^R$. To check whether the last time slot includes the related revocation operation.
- $\mathsf{CertOper}^R_{local}.\mathsf{loh} = 0$. To check whether the certificate is revoked for the first time. Since each certificate can only be revoked once, for security, the blockchain node only accepts the modified block when loh in the local block is 0.

3) Audit and Incentive: To prevent some negative BNs from being reluctant to record revocation information in time for saving expenses or something else, we design an audit mechanism to punish the negative BNs. In each time slot, all the blockchain nodes together conduct an internal audit to measure the credibility of each node's behaviors in this slot and adjust the dependability value accordingly. Specifically, at the end of each time slot t, each blockchain node checks whether each revocation operation in block $B[h_{t-1}]$ has been correctly recorded in the original certificate. If a revocation operation has not been appropriately recorded, this implies that the CA who issued the certificate fails to complete the due obligations in time, and it should therefore be punished by decreasing its dependability to decrease its probability to generate blocks.

Algorithm 4: *Audit($B[h_{t-1}]$)* at slot t

```
for each CertOper^R ∈ B[h_{t-1}] do
    if CertOper^R ∉ 𝕌ℙ𝔻[t − 1] then
        𝔼ℝℝ[t].append(CertOper^R);
    end
end
for each CertOper^R ∈ 𝔼ℝℝ[t] do
    Decrease the dependability of the corresponding
     CA;
end
```

E. Deception of Version (DoV) Attack and Freshness Check

Definition 1: (*Deception of Version (DoV) Attack*) A full node of the blockchain sends an old version of the requested block to the user, and the user cannot discover this because the hash values of a block's different versions are the same. BNs cannot send a "new" version of the requested block because there is no revocation CertOper in the related block.

In redactable consortium blockchains, the full nodes are certain to keep all the blocks synchronized to the latest state. However, the lightweight users who have no capability to keep the entire blockchain have to request blocks from the full nodes, which leads to a risk that the full nodes may conduct DoV attacks. In the proposed PKI system, DoV attacks will interfere with a user's judgment on whether a certificate has been revoked because even if the certificate has been revoked and the revocation operation has been recorded to the issuance CertOper, the blockchain node can still provide an old version of the block that has not recorded the revocation. To defend against DoV attacks, we design a random-block-node-check (RBNC) mechanism for freshness check and prove that the probability of successfully conducting DoV attacks can be reduced to a statistically negligible level by adopting RBNC even with very few blocks and nodes.

Algorithm 5: Freshcheck($B[h], \mathbb{BN}$)

```
if | 𝔹ℕ |≤ (2/3)n then
    Certificate is revoked; Terminate;
end
ℝℕ ← Random(𝔹ℕ, s);
ℝℍ ← Random(⟨1, H⟩\{h}, k − 1) ∪{h};
for each BN_i in ℝℕ do
    Request blocks in ℝ𝔹;
    ℝ𝔹[i] ← result from BN_i;
end
if exist ℝ𝔹[i] where CertOper.loh ≠ 0 then
    Certificate is revoked; Terminate;
end
if ℝ𝔹[i](1 ≤ i ≤ s) are identical then
    Certificate isn't revoked; Terminate;
end
for each CertOper^I with loh ≠0 in all ℝ𝔹[i] do
    𝔹ℕ.delete(BN_i if CertOper^I.loh =0 in ℝ𝔹[i]);
end
Freshcheck(B[h], 𝔹ℕ);
```

Freshcheck(). When a user receives a CertOper with loh $= 0$, he/she should then conduct the *Freshcheck($B[h], \mathbb{BN}$)*, where $B[h]$ is the block that CertOper is located, and $\mathbb{BN}$ is the set of blockchain nodes. As Algorithm 5 illustrates, the user first randomly selects s blockchain nodes from $\mathbb{BN}$ and k block heights (must include h) from all generated blocks (i.e., $\langle 1, H\rangle$), and forms the random-node-set $\mathbb{RN}$ and random-block-set $\mathbb{RH}$, respectively. Then, the user requests all the blocks in $\mathbb{RH}$ from each BN in $\mathbb{RN}$. The result (i.e., the set of requested blocks) from BN_i is denoted as RB[i]. Next, there are three cases according to RB[i]:

- *Exist* CertOper.loh $\neq 0$: If there exists a $RB[i]$ where the loh of the target CertOper is not 0, the user then considers the certificate revoked and terminates the process. It should be noted that a malicious BN is unable to generate a valid block that contains CertOper.loh $\neq 0$ when the certificate is not revoked because one BN is unable to generate a valid consensus proof for the block.
- *All* CertOper.loh $= 0$ *and all* RB[i]*s are identical*: If no result shows that loh $\neq 0$ of the target CertOper and all

the results are totally identical, then the user considers the certificate effective (not revoked) and terminates the process.

- *All* $\mathsf{CertOper.loh} = 0$ *and* $\mathsf{RB}[i]s$ *have differences*: In such case, those BNs who provided results that are different from the majority were conducting DoV attacks. The user should delete them from $\mathbb{BN}$ and perform the *Freshcheck()* process again. It should be noted that the initial $\mathbb{BN}$ is the set of all the n BNs, and when the number of BNs is less than $\frac{2}{3}n$, the user then considers the target certificate revoked.

The security and efficiency of the above process is provided in Section V-B.

V. Security Analysis

A. Security of the Redaction Consensus

Theorem 1: Safety. All the valid certificates and revocation operations will be correctly recorded on the blockchain, and users who are not full-nodes of the blockchain will acquire correct data from blockchain nodes.

Proof: The safety of valid certificates is guaranteed by the common consensus which is just the common PoS consensus, and its safety has been widely proved; we therefore omit this part here. To revoke a certificate, there are three steps in ScalaCert: $\mathsf{CertOper}^R$ generation, $\mathsf{CertOper}^I$ modification, and redaction consensus. $\mathsf{CertOper}^R$ generation is completed by the blockchain extension process and its safety is also guaranteed by the common consensus. For $\mathsf{CertOper}^I$ modification, since that one can modify the message while keeping the Chameleon hash value unchanged only if he/she owns the private key, only the CA who issued $\mathsf{CertOper}^I$ can modify the loh, and this is in line with practical demands. What's more, ScalaCert only allows each $\mathsf{CertOper}^I$'s loh to be modified once, thereby reducing the impact on the system when a CA is compromised. For redaction consensus, the reliable underlying network communication ensures that the block modification information broadcasted by each BN will definitely be received by all other BNs in the CAA. As such, all the honest BNs will always have the same view on the blockchain because they all use the same judgment rules to decide whether to accept a modified block, i.e., *Redact(*$\mathsf{CertOper}$*)* shown in Algorithm 3.

B. Security of Certificate Verification

For certificate verification, the most important thing is to make sure that once a certificate is revoked, any user who verifies the certificate from the blockchain will acquire its revocation information. As we prove in the last subsection, any revocation information will be timely recorded to the issuance $\mathsf{CertOper}^I$. Therefore, the verification is correct as long as the user acquires the correct $\mathsf{CertOper}^I$. However, as introduced in Section IV-E, the combination of redactable and consortium blockchains brings the DoV attack, and the following proof shows that the proposed freshness check mechanism can well defend against DoV attacks.

Theorem 2: Defense Against DoV Attacks. By freshness check, the probability of successfully conducting DoV attacks can be reduced to a statistically negligible level.

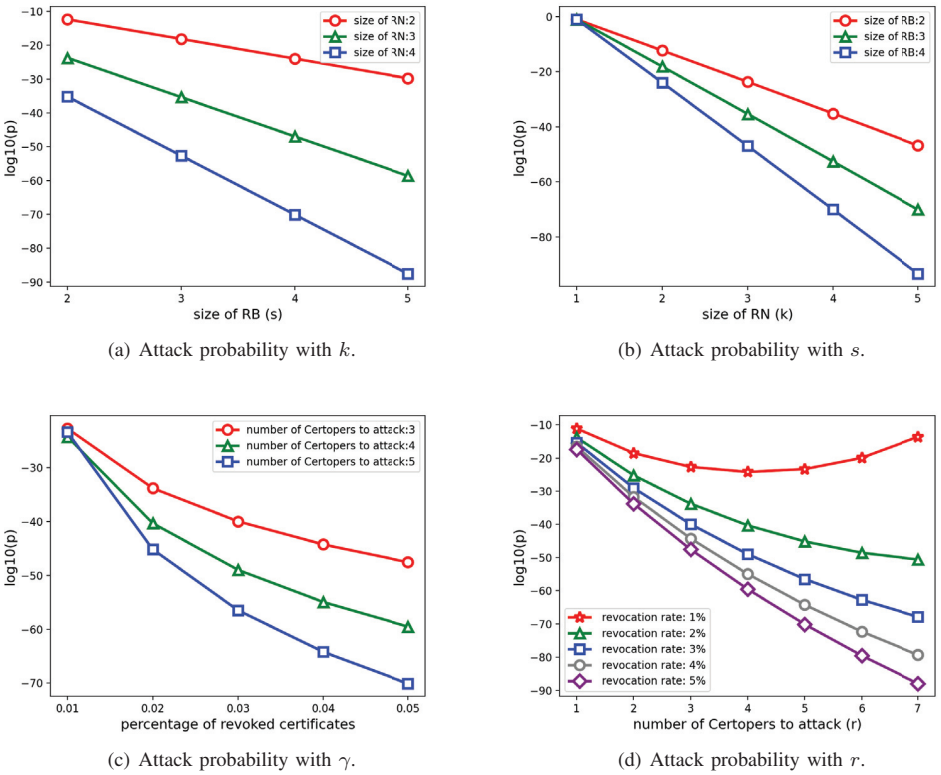
Proof: When a certificate is revoked, the loh field of its issuance operation $\mathsf{CertOper}^I$ is then modified from 0 to a positive integer. Thus, if a malicious BN sends the user an old-version $\mathsf{CertOper}^I$ with $\mathsf{loh} = 0$, the user will then consider the certificate effective. This is how DoV attacks work in the proposed scheme. However, it should be noted that once the user acquires a $\mathsf{CertOper}^I$ with non-zero loh, he/she can therefore confirm that the certificate has been revoked. Hence, to defend against DoV attacks, we design the RBNC-based freshness check mechanism (Algorithm 5) where a user requests several blocks from several BNs. In this way, BNs cannot know which certificate the user needs, and therefore a malicious BN cannot conduct the DoV attack against a specific $\mathsf{CertOper}^I$, but can only randomly select one or several $\mathsf{CertOper}^I$s from all the revoked certificates from all the blocks in $\mathbb{RB}$ to do the attack. Then, successfully conducting a DoV attack means that all the BNs in $\mathbb{RN}$ are malicious, and they successfully choose the same $\mathsf{CertOper}^R$ to deceive the user.

Since $\mathbb{RN}$ and $\mathbb{RB}$ are all randomly selected, we can therefore assume that the malicious BNs can only randomly choose $\mathsf{CertOper}^I$s to conduct DoV attacks, as they have no relevant information to infer which CertOper the user needs. Consequently, successfully conducting a DoV attack means that for *Verify()* the user chooses a malicious BN, and for *Freshcheck()* all the BNs selected by the user are malicious and the CertOpers they randomly select are just the same and include the target certificate. Suppose that the revocation rate is γ, each block contains m CertOpers, and each malicious BN selects $r(r \geq 1)$ revoked $\mathsf{CertOper}^I$ for each block to attack. Then, the probability of successfully conducting a DoV attack is

$$\mathcal{P} = \frac{1}{3} \cdot \frac{\binom{s}{n/3}}{\binom{s}{n}} \cdot \left[\frac{1}{\binom{r}{\gamma m} - \binom{r-1}{\gamma m-1}} \right]^{s-1} \cdot \left[\frac{1}{\binom{r}{\gamma m}} \right]^{(s-1)(k-1)},$$

where n is the total number of the BNs, $\frac{1}{3}$ is the highest proportion of dishonest BNs, s, k are the size of $\mathbb{RN}$ and $\mathbb{RB}$ respectively.

We simulate the probabilities of successfully conducting a DoV attack under different values of the parameters and with each block containing 800 CertOpers ($m = 800$) and $\frac{1}{3}$ malicious nodes. Fig. 2(a) and 2(b) show how the sizes of $\mathbb{RB}$ and RN impact the probability of successfully conducting a DoV attack when the revocation rate is 5% and each BN selects 5 CertOpers. The results show that with the sizes of $\mathbb{RB}$ and RN increase, the probability decreases drastically. Even when $s, k = 2$, the probability is still as low as 10^{-10}. Fig. 2(c) and 2(d) show the results under the condition of $s, k, r = 5$ and $m = 800$. Fig. 2(c) gives the probability with different percentages of revoked certificates, i.e., the revocation rate γ. Fig. 2(d) shows that the probability decreases when r increases. Thus, attackers may always set $r = 1$ to make



(a) Attack probability with k.

(b) Attack probability with s.

(c) Attack probability with γ.

(d) Attack probability with r.

Fig. 2. Probabilities of successfully conducting a DoV attack under different values of the parameters.

the probability the highest. However, even with $r = 1$, under the condition of $s, k = 5$ and $m = 800$, the probability of successfully conducting a DoV still does not exceed 10^{-10}.

C. Liveness and Fairness

If a BN does not update the related $\mathsf{CertOper}^I$ in time, when a revocation operation $\mathsf{CertOper}^R$ is generated and added to the blockchain, this negative behavior will be detected by all other BNs during *Audit()* process shown in Algorithm 4. As such, the BN's dependability will decrease, and its probability of generating blocks and issuing certificates will be reduced, which finally leads to a reduction in its actual income. In other words, honest and active BNs will acquire more dependability and rewards than malicious ones, guaranteeing system fairness. Driven by this incentive mechanism, BNs seeking income will keep honest and active and, therefore, maintain the entire system's liveness.

VI. Performance Analysis

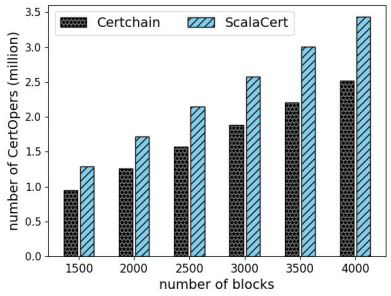
A. Implementation

We evaluate the performance of ScalaCert from three aspects, i.e., the storage, the "on-cert" revocation and redaction consensus, and the verification and freshness check. To this end, we make a prototypical implementation on ScalaCert and conduct the experiments on macOS Catalina (v10.15.7) with Intel Core i5 CPU @2 GHz and 16 GB RAM. Moreover, since ScalaCert sacrifices communication for storage, we also prototype CertChain for comparison to evaluate how much ScalaCert affects the performance in terms of communication. The blockchain-related parts, including the data structure and consensus, are implemented based on Ethereum, and the operations of CAs, BNs, and users are programmed in Python 3. The cryptographic parts are implemented based on PyCryptodome (v3.10.1) and gmpy2 (v2.0.8), and the parameters of the Chameleon hash are generated by SageMath (v9.2).

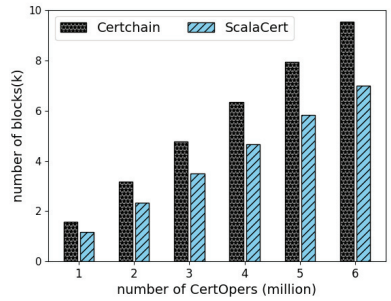
B. Storage

Referring to CertChain, the size of a block is limited to 2MB, an empty block is about 2.6KB, a single CertOper is about 1.8KB, and a DCBF (which is the bloom filter based CRL used in CertChain) is about 412KB. Since each block is supposed to contain a DCBF, one block maximally contains about 900 CertOpers. In ScalaCert, however, with the help of redactable blockchain, the revocation information is directly recorded on the original $\mathsf{CertOper}^I$. Hence, no additional data structure like CRL is needed. Thus, the whole block except for the blockhead can be used for the storage of CertOpers. Therefore, the number of CertOpers contained in one block increases to around 1100, consequently saving about one-fifth of the storage space. With the scale of the system increases, the scale of certificates that the saved space contains will

be considerable and be of enormous influence. For example, as Fig. 3(a) shows, when there are 1500 blocks, ScalaCert contains 0.3 million more CertOpers than those that CertChain contains. However, when the number of blocks increases to 4000, the gap then sharply increases to 1 million. On the other hand, as Fig. 3(b) shows, to support 1 million CertOpers, CertChain needs only around 200 more blocks than ScalaCert. And when there are 6 million CertOpers, CertChain then needs around 3000 more blocks than ScalaCert. What's more, when the number of certificates increases, the performance of DCBF used in CertChain will deteriorate due to the contradiction of the bloom filter between size and correctness. But there is no such problem in ScalaCert.



(a) Blockchain's capacity for certificates as blocks increases.



(b) The demand for blocks as certificates increasing.

Fig. 3. Comparison of storage performance of ScalaCert and CertChain.

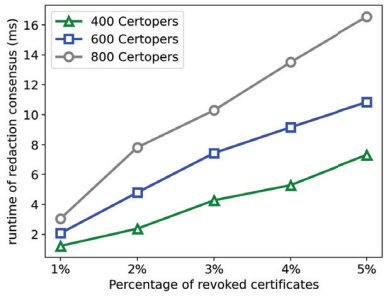
C. Runtime of "On-Cert" Revocation

We evaluate the runtime of the three kinds of operations in ScalaCert, i.e., certificate issuance, update, and revocation, and the results shown in TABLE II are the average of 1000 tests. TABLE II shows the runtime of generating a CertOper of different operations. The issuance operations consume about 9 ms and the update and revocation operations consume about 7.5ms. This is because for "on-cert" revocation, the loh field is hashed by the Chameleon hash for the issuance CertOpers while is hashed by SHA 256 for the update and revocation CertOpers. Hence, the difference between the Chameleon hash and the SHA256 hash, i.e., Chameleon hash takes about 1.5ms more time than SHA 256 on average, leading to the different runtimes. Besides, for recording the revocation information, after generating and adding the revocation $\mathsf{CertOper}^R$, the issuance CA of the certificate should also modify the loh field in the issuance $\mathsf{CertOper}^I$ and compute for keeping the hash value unchanged.

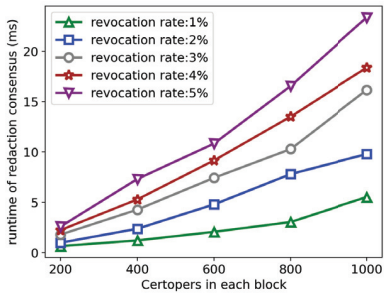
TABLE II
RUNTIME OF GENERATING DIFFERENT KINDS OF CERTIFICATE OPERATIONS

Operation	Runtime of CA (ms)
Issuance	9.07
Update	7.46
Revocation	7.46
Revocation Record	2.14

We also evaluate the time consumption of the *redaction consensus* process under different revocation rates and block sizes (i.e., how many CertOpers included in one block). Fig. 4(a) and Fig. 4(b) show the runtime of redaction consensus under different revocation rates and block sizes, respectively. It is clear that the time consumption increases roughly linearly with the increase of both the revocation rate and the block size. This is consistent with the theoretical expectation, because the computing and communication overhead of the redaction consensus is basically linear with the number of revoked certificates. Specifically, as Fig. 4(b) shows, when the revocation rate is 5% and each block contains 1000 CertOpers, the redaction consensus takes about 25ms. Compared with the block generation speed, for example, 15s per block in



(a) Runtime of redaction consensus under different revocation rates.



(b) Runtime of redaction consensus under different block sizes.

Fig. 4. Time consumption of redaction consensus under different settings

Ethereum, the time consumption of the redaction consensus is definitely within an acceptable level.

D. Certificate Verification and Freshness Check

To evaluate the performance of certificate verification, we compare ScalaCert and CertChain under the condition of $s, k = 5$ for freshness check. Besides, to make the result closer to the practical situation, we test the response time of requesting the Ethereum node to simulate the communication overhead between a user and a BN. Fig. 5 shows that the time consumption of certificate verification is around 600 ms in ScalaCert and 300 ms in CertChain. In general, ScalaCert requires twice the verification time as CertChain. This is because if $\mathsf{CertOper}^I.\mathsf{loh} = 0$, the user should conduct the freshness check process and wait for one more block request time. It should be noted that although a user needs to request for more than one block, the multiple requests are conducted at the same time, and thus only one more request time is needed. Moreover, when the revocation rate increases, the verification time in ScalaCert will slowly decrease. This is because the more certificates are revoked, the lower the probability of conducting *Freshcheck()*. Therefore, the average verification time decreases.

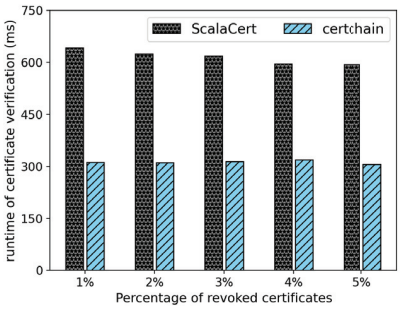
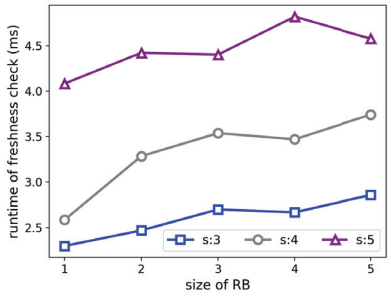


Fig. 5. Time consumption of certificate verification.

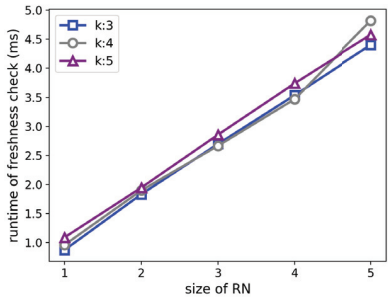
In addition to the runtime of the entire verification process, we also evaluate the separate runtime of users' local computing consumption of *Freshcheck()* with different values of s and k. The results are shown in Fig. 6. The size of $\mathbb{RB}$ (Fig. 6(a)) has little effect on the runtime, but the size of $\mathbb{RN}$ (Fig. 6(b)) has a more obvious influence. This is because a user is supposed to initialize the connection with each single BN in $\mathbb{RN}$ (and we simulate this step by web3.eth interface), therefore leading to an increase in the overall runtime. Thus, users can prefer to increase the size of $\mathbb{RB}$ instead of $\mathbb{RN}$ to improve security. By comprehensively weighing the time cost (Fig. 6) and the attack probability (Fig. 2), users can decide the most suitable parameter settings for themselves.

VII. Related Work

In terms of blockchain-based certificate systems, researchers have proposed many related solutions. According to PGP Web of Trust [27] and CA-based PKI [1], blockchain-based PKIs



(a) Runtime of *Freshcheck()* with size of RB (k).



(b) Runtime of *Freshcheck()* with size of RN (s).

Fig. 6. Users' local runtime of *Freshcheck()* with different settings.

are roughly classified into two categories: fully decentralized PKI based on public blockchains and semi-decentralized PKI based on consortium blockchains. Earlier solutions are usually based on Bitcoin, such as NameCoin and CertCoin. NameCoin [28] is the first fork of Bitcoin and serves as a decentralized domain name system (DNS). Inspired by NameCoin, Fromknecht *et al.* proposed CertCoin [29] which is a Bitcoin-based fully decentralized PKI system. There is no trusted third party such as CA in fully decentralized PKIs in CertCoin, and all the users are peer entities. However, these fully decentralized PKIs cannot provide sufficient scalability, and the no-CA structure cannot meet the needs of lots of practical applications. Thus, aiming at CA-based PKIs, Chen *et al.* proposed CertChain [12] based on X.509 standard in 2018. In CertChain, multiple CAs establish a consortium; the PKI system is constructed on the consortium blockchain, and a CRL compressed by the bloom filter is used for revocation verification. Cheng *et al.* [30] proposed a digital certificate management system based on the smart contracts. In 2019, Kubilay *et al.* proposed CertLedger [13] in which they used the Merkel Hash Three for the storage of revocation information. Additionally, aiming at privacy preserving, Axon *et al.* proposed PB-PKI [31] and adapted CertCoin to be privacy-aware. Jia *et al.* proposed PROCESS [32] on the basis of CertChain and designed a data structure called BORL to further compress the space used for storing revocation status (reduced about half-space compared with CertChain). However, it still requires additional space for revocation verification. What's more, it causes extra computing

overhead due to the introduction of the Chameleon hash. In summary, most existing blockchain-based PKIs use additional data structures for revocation verification, thereby bringing severe scalability problems.

VIII. Conclusion

In this paper, we proposed ScalaCert, which is a scalability-oriented and blockchain-based PKI system. By introducing the redactable blockchain and Chameleon hash technology, ScalaCert realizes to record the revocation information on the blockchain without any additional data structure such as CRL, thus significantly reducing the storage overhead and enhancing the scalability of blockchain-based PKIs. To solve the security problems brought by the redaction of data, we designed a permission restriction on the Chameleon hash and the redaction consensus for reaching a consensus on block versions. Furthermore, aiming to defend against the DoV attack caused by the combination of redactable and consortium blockchains, ScalaCert provides a freshness check mechanism to help users determine whether a received block is the latest version. Through security analysis and the experimental implementations, we proved that ScalaCert has sufficient security and enhances the scalability though it brings reasonable communication overhead.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61972371 and No. U19B2023, Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093, and JSPS KAKENHI under Grant No. JP19H04105.

References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. T. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," 2008. RFC 5280, Accessed on 2021-07-27.

[2] D. Díaz-Sánchez, A. Marín-Lopez, F. A. Mendoza, P. A. Cabarcos, and R. S. Sherratt, "TLS/PKI challenges and certificate pinning techniques for IoT and M2M secure communications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3502–3531, 2019.

[3] M. Khodaei and P. Papadimitratos, "Efficient, scalable, and resilient vehicle-centric certificate revocation list distribution in VANETs," in *Proceedings of the 11th ACM conference on security & privacy in wireless and mobile networks (WiSec)*, pp. 172–183, ACM, 2018.

[4] I. García-Magariño, S. Sendra, R. Lacuesta, and J. Lloret, "Security in vehicles with IoT by prioritization rules, vehicle certificates, and trust management," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 5927–5934, 2018.

[5] S. Yi and R. Kravets, "Moca: Mobile certificate authority for wireless ad hoc networks," 2004.

[6] R. Hunt, "PKI and digital certification infrastructure," in *Proceedings of the Ninth IEEE International Conference on Networks (ICON)*, pp. 234–239, IEEE, 2001.

[7] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (SSL) protocol version 3.0," 2011. RFC 6101, Accessed on 2021-07-27.

[8] E. Rescorla and T. Dierks, "The transport layer security (TLS) protocol version 1.3," 2018. RFC 8446, Accessed on 2021-07-27.

[9] Microsoft, "Erroneous VeriSign-issued digital certificates pose spoofing hazard," 2001. Microsoft Security Bulletin MS01-017,accessed: Jul., 2021.

[10] L. Lei, K. Zheng, and J. Chen, "Challenges on wireless heterogeneous networks for mobile cloud computing," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 34–44, 2013.

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

[12] 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 *Proceedings of the 2018 IEEE Conference on Computer Communications (INFOCOM)*, pp. 2060–2068, IEEE, 2018.

[13] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with certificate transparency based on blockchain," *Computers & Security*, vol. 85, pp. 333–352, 2019.

[14] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE transactions on industrial informatics*, vol. 14, no. 8, pp. 3690–3700, 2017.

[15] X. Chen, D. W. K. Ng, W. Yu, E. G. Larsson, N. Al-Dhahir, and R. Schober, "Massive access for 5G and beyond," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 3, pp. 615–637, 2020.

[16] Ericsson, "Ericsson mobility report June 2021," 2021. accessed: Jul., 2021.

[17] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*, pp. 124–138, IEEE, 2019.

[18] G. Ateniese and B. d. Medeiros, "Identity-based chameleon hash and applications," in *Proceedings of the 2004 International Conference on Financial Cryptography (FC)*, pp. 164–180, Springer, 2004.

[19] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/en/bitcoin-paper, 2008. Bitcoin white paper, accessed: Jul., 2021.

[20] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Čapkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (CCS)*, pp. 3–16, ACM, 2016.

[21] A. Kiayia, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proceedings of the 2017 Annual International Cryptology Conference (CRYPTO)*, pp. 357–388, Springer, 2017.

[22] V. Buterin, "Ethereum: A secure decentralised generalised transaction ledger." https://ethereum.org/en/whitepaper/, 2013. Ethereum white paper, accessed: Jul., 2021.

[23] N. Döttling, S. Garg, Y. Lshai, G. Malavota, T. Mour, and R. Ostrovsky, "Trapdoor hash functions and their applications," in *Proceedings of the 2019 Annual International Cryptology Conference (CRYPTO)*, pp. 3–32, Springer, 2019.

[24] H. Sukhwani, J. M. Martínez, X. Chang, and K. S. Trivedi, "Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric)," in *Proceedings of the 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 253–255, IEEE, 2017.

[25] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain," 2018.

[26] G. Xu, Y. Liu, and P. W. Khan, "Improvement of the dpos consensus mechanism in blockchain based on vague sets," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4252–4259, 2019.

[27] "OpenPGP." https://www.openpgp.org/. accessed: Jul., 2021.

[28] "Namecoin." https://www.namecoin.org. accessed: Jul., 2021.

[29] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," *IACR Cryptologt ePrint Archive*, no. 803, pp. 1–16, 2014.

[30] J.-C. Cheng, N.-Y. Lee, C. Chi, and Y.-H. Chen, "Blockchain and smart contract for digital certificate," in *Proceedings of the 2018 IEEE international conference on applied system invention (ICASI)*, pp. 1046–1051, IEEE, 2018.

[31] L. Axon and M. Goldsmith, "PB-PKI: A privacy-aware blockchain-based PKI," in *InProceedings of the 14th International Joint Conference on e-Business (ICETE)*, pp. 311–318, SECRYPT, 2017.

[32] M. Jia, K. He, J. Chen, R. Du, W. Chen, Z. Tian, and S. Ji, "PROCESS: Privacy-preserving on-chain certificate status service," in *Proceedings of the 2021 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–10, IEEE, 2021.