CertChain: Public and Efficient Certificate Audit Based on Blockchain for TLS Connections

Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, Ruiying Du

Abstract—**In recent years, real-world attacks against PKI take place frequently. For example, malicious domains' certificates issued by compromised CAs are widespread, and revoked certificates are still trusted by clients. In spite of a lot of research to improve the security of SSL/TLS connections, there are still some problems unsolved. On one hand, although log-based schemes provided certificate audit service to quickly detect CAs' misbehavior, the security and data consistency of log servers are ignored. On the other hand, revoked certificates checking is neglected due to the incomplete, insecure and inefficient certificate revocation mechanisms. Further, existing revoked certificates checking schemes are centralized which would bring safety bottlenecks. In this paper, we propose a blockchain-based public and efficient audit scheme for TLS connections, which is called Certchain. Specially, we propose a dependability-rank based consensus protocol in our blockchain system and a new data structure to support certificate forward traceability. Furthermore, we present a method that utilizes dual counting bloom filter (DCBF) with eliminating false positives to achieve economic space and efficient query for certificate revocation checking. The security analysis and experimental results demonstrate that CertChain is suitable in practice with moderate overhead.**

I. Introduction

As HTTPS has been globally adopted in various online services, e.g., e-business, e-banking, and e-government, Transport Layer Security (TLS) protocol, the cornerstone of HTTPS, plays a critical role in secure web-based connections over a computer network. In TLS, authentication and secure connection establishment are built based on Public Key Infrastructure (PKI) whose core component is *certificate authorities* (CAs). By signing and issuing certificates, CAs provide the trust foundation to guarantee integrity, confidentiality, and undeniability for web traffic.

However, recent compelling real-world attacks have demonstrated existing CAs' vulnerability. For example, some well-known CAs, e.g., TurkTrust [1], CNNIC [2], DSDtestProvider & eDellRoot [3], were compromised to issue unauthorized certificates for malicious domains. Such CAs' failures can further be exploited by adversaries to mount Man-in-the-Middle (MitM) attacks. To tackle this issue, researchers have presented a variety of proposals, which can be generally classified into two categories: *CA-based trust disperse* schemes and *log-based misbehavior monitor* schemes. *CA-based trust disperse* schemes mainly focus on diminishing the trust of CAs by introducing multiple CAs or other entities to assist *certificate operations* including (registration, update, and revocation), which then prevents an individual CA from generating unauthorized certificates. For instance, in TriPKI [4], Jing et al. propose a tripartite PKI which utilizes threshold signature among CAs and DNSs to avoid single-point-failure of CAs. In Cosigning [5], Syta et al. design a multi-signature scheme that adapts a scale of thousands of witnesses to participate in decentralized cosigning. In ARPKI [6], Basin et al. enhance the system security by using multiple CAs to sign and validate certificates in a serial mode. On the other hand, *Log-based misbehavior monitor* schemes bring in log servers that maintain a merkle Hash Tree to record certificates issued by CAs, such as Certificate Transparent (CT) [7], Sovereign Keys (SK) [8], AKI [9], and ARPKI [6]. The main idea of these schemes is that by publicizing certificates, CAs' misbehavior can be detected in time. Generally, compared to the former, the latter category utilizes log servers to share CAs' responsibility in terms of operation storage and certificate validation, and it then results in better security and users' web-browsing experience.

We observe that there are still several critical issues in existing log-based misbehavior monitor schemes. First of all, in most of existing schemes, even though multiple log servers are employed to record certificates, the data security still depends on an individual log server that is chosen to synchronize certificates. If the chosen log server is compromised, the data security cannot be guaranteed. In addition, due to the expensive bandwidth cost, low query efficiency, and high latency, many browser vendors and client applications decline to check whether the target domains' certificates have been revoked. Attackers could use these revoked certificates, which unfortunately are still considered as valid by clients, to perform effective MitM and phishing attacks against clients. Regarding this issue, in Certificate Issuance and Revocation Transparency (CIRT) [10], Ryan proposes an efficient revocation mechanism for CT, but it requires a domain to change a new identity once his key is lost. In CRLite [11] and CCSP [12], the authors present an efficient revoked certificate query scheme. Unfortunately, these schemes are centralized systems which

Chen is with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Computer School, Wuhan University, Wuhan, 430072 and Science and Technology on Communication Security Laboratory, Chengdu, China. Email: chenjing@whu.edu.cn. Yao and He are with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Computer School, Wuhan University, Wuhan, 430072. Email: derekysx@whu.edu.cn, milloglobe@gmail.com. Yuan is with University of Texas-Permian Basin, TX, 79762. Email: dantes.yuan@gmail.com. Ji is with Zhejiang University, Zhejiang, China and Georgia Institute of Technology, Atlanta, USA. Email:sji@gatech.edu. Du is with Collaborative Innovation Center of Geospatial Technology, Wuhan, China, 430072. Email: duraying@whu.edu.cn. This research was supported in part by the National Natural Science Foundation of China under Grant No. 61572380, 61772383, 61702379, U1536204, and the Major State Basic Research Development Program of China under Grant No. 2014CB340600. The information reported here does not reflect the position or the policy of the funding agencies. The corresponding author is Shixiong Yao.

are vulnerable to single-point-failure attacks.

Inspired by blockchain, a distributed database that is used to maintain a continuously growing list of records in bitcoin [13], we put forward a comprehensive certificate management system to address the above issues in log-based misbehavior monitor schemes. Considering its decentralization and tamper-proof features, we intend to utilize blockchain to record certificates and their associated certificate operations for *public audit*, where anyone is allowed to verify the correctness of the certificates operations by querying blockchain records. Note that introducing blockchain in certificate management is not trivial since it brings three important challenges as follows: 1) ***Centralization in practice***. The most popular and widely used consensus protocols in blockchain, such as PoW (Proof of Work), PoS (Proof of stake) or DPoS (Delegated Proof of Stake), still have privileged nodes which possess the stronger computing ability or more stake in the system. These nodes usually generate most of the blocks, and control the blockchain to some extent. Such phenomenon deviates the intention of decentralization in certificate management system. 2) **Mandatory traversal**. When directly applying the blockchain technique in a certificate management system, if we need to learn a domain's history certificate operations in blockchain, we have to traverse the whole blockchain which is tedious and time-consuming. 3) **Block size limitation**. The currently size of one block in dominant blockchain system is limited [14], while the size certificate revocation list (CRL) reaches up to 76MB [15] in some case, which obviously exceeds the capacity of one block. As the certificate revocation information keeps increasing, more blocks are generated in order to store this revocation information. In this way, checking whether a certificate has been revoked via traversing blockchain becomes inefficient. To improve the users' experience, the revocation information needs to be treated specially for efficient query response.

In this paper, to solve the challenges discussed above, we propose a blockchain-based public and efficient certificate audit scheme for TLS connections, called **CertChain**, by introducing new entities called bookkeepers to record certificate operations into blockchain for public audit. Specifically, our certificate management system is developed based on a four-layer blockchain architecture which includes data layer, network layer, extension layer, and application layer. In summary, we make the following contributions:

1) To the best of our knowledge, this is the first work to propose a decentralized public audit certificate management framework. Through secure and efficient certificate operation queries, CertChain can resist certificate forgery and tamper attacks effectively.
2) To avoid centralization in practice, we design a distributed dependability-rank based consensus protocol to achieve trust dispersing.
3) To solve the mandatory traversal problem, we propose a new data structure called `CertOper` to record certificates operations. The `CertOper` is stored in block for operations forward traceability and efficient query.

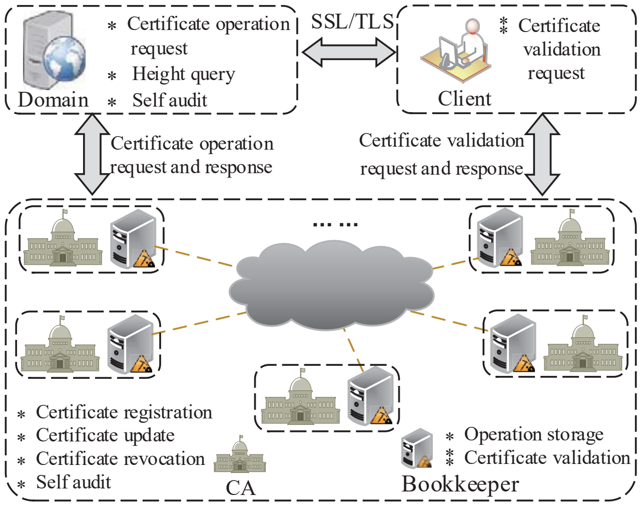

Fig. 1. Framework of CertChain

4) Considering the block size limitation issue, to achieve realtime certificate validation, we exploit a revocation checking method based on Dual counting bloom filter (**DCBF**) which can markedly eliminate false positives and ensure the practicability of our work.
5) We analyze the security of CertChain in theory. Besides, we implement a proof-of-concept prototype and evaluate the performance of CertChain in practice.

This paper is organized as follows. In section II, we give a description about the system model, threat model, and design goals. We describe the details of our CertChain in section III, and analyze its security properties in section IV. We describe the implementation of our prototype in section V, and evaluate the CertChain by comparing it with other schemes. Section VI reviews the literature related to the traditional PKI. Finally, we conclude in section VII.

II. Problem Statement

A. System Model

In our system, there are four kinds of entities: **client**, **domain**, **CAs**, and **bookkeepers**, as show in Fig. 1. A client is the entity who intends to establish TLS connections with a domain, while domain usually refers to a website, which gets a certificate from a CA for secure connections. CAs, besides signing and issuing certificates as in traditional PKI, need to generate and sign certificate operations. To support public audit service, we introduce the bookkeepers to store the operations in blocks and maintain the blockchain. The blockchain works in a permission mode which means that only authorized nodes can participate in certificate management.

In details, a domain requests a certificate operation from a CA, such as certificate registration, update, or revocation. After the CA finishes the requested certification operation, it signs the operation and broadcasts it to all bookkeepers. A client then can validate a certificate with the assistance of bookkeepers. There are two points worth noting here: 1) To speed up the certificate checking process, bookkeepers arrange and record all revocation information to **DCBF** stored in one block; 2) A CA couples with a unique bookkeeper and they share the dependability-rank (defined in III-C) that is prepared for consensus protocol design. By querying the blockchain, a CA manager can detect whether there exists forged or tampered certificates for malicious domains. A domain can also check whether its name is impersonated. We call these two processes as *self audit*.

B. Threat Model

Generally, an adversary attacks CertChain for three goals: (1) to issue a certificate for a malicious domain without being detected; (2) to insert, delete, or tamper the certificate operations for making clients' certificate validation failure; (3) to control the blockchain by attacking some bookkeepers.

From the practical perspective, we assume that an active adversary is able to manipulate a victim's web traffic, and it can also compromise any entity. Furthermore, it can eavesdrop, tamper, and forge messages among entities which communicate with each other in untrusted networks. However, we make some standard cryptographic assumptions. For example, the adversary is not able to forge signatures without getting a principal's private key. Additionally, we assume that an adversary cannot control more than 51% bookkeepers in blockchain.

C. Design Goals

- **Consensus fairness**. By dynamic accommodation, each bookkeeper has a similar probability to generate blocks for recording certificate operations.
- **High query efficiency**. All operations of a specified certificate can be traced without traversing the whole blockchain. Particularly, the process of certificate validation only requires to check the header block which records the latest revocation information of all certificates.
- **Intrusion tolerance**. Through *self audit*, for a CA, even if all its defense mechanisms are ineffective, it can still detect misbehavior effectively by querying blockchain, and then take actions to prevent attacks from deteriorating.

III. CertChain: Public and Efficient Certificate Audit Based on Blockchain

A. Overview of CertChain

In this paper, we design a certificate management system based on a four-layer blockchain architecture that includes data layer, network layer, extension layer, and application layer. The system architecture of CertChain is exhibited in Fig. 2. In *data layer*, to retain the history certificate operations, we design a new data structure called `CertOper` to express certificate operations, which is stored in blockchain in the form of Merkle Hash Tree (MHT). We also present a dual counting bloom filter (**DCBF**) for all revoked certificates with economic storage and efficient query. In *network layer*, since the size of a `CertOper` is about the same as a transaction

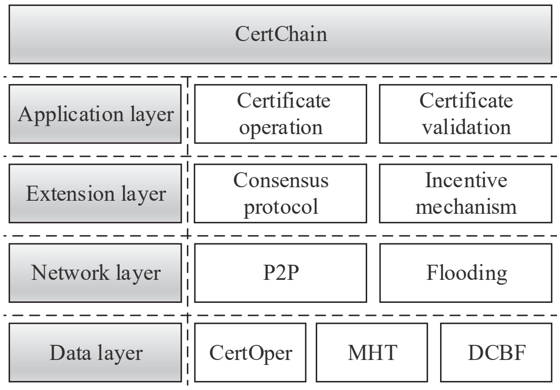

Fig. 2. Architecture of CertChain

in cryptocurrency system, the existing network protocol and transmission mechanisms are well compatible with our system. In *extension layer*, we design a distributed dependability-rank based consensus protocol to disperse trust, and in incentive mechanism among CAs and bookkeepers. In *application layer*, we propose a distributed certificate management system including certificate operations and certificate validation. Except the network layer, we will give an elaborate description about these layers in a bottom-up order.

B. Data layer

1) `CertOper` definition: Referring to the X.509 public key certificate standard, we define a new data format called `CertOper` that is used to express a concrete certificate operation requested by a domain. All the fields in a `CertOper` are explained as follows.

- `Version Number`, `Signature Algorithm ID`, `Signature Value`, `Extension Field` are the same as X.509 certificate.
- `Subject Name`: the name of a domain who requests a certificate operation;
- `Operator Name`: the name of a CA who signs a certificate and generates this data structure;
- `Operation Type`: three types of certificate operations including registration, update, revocation.
- `Timestamp & NotAfter`: the generation time of an operation, and the expiration time used by bookkeepers to clear the expired revocation information.
- `Current Certificate Hash`: the hash of the subject domain's certificate which is used for the process of certificate validation.
- `Last Operation Height`: if the operation type is registration, this field is null. Otherwise, it is filled in the block height of the subject's last certificate operation. Due to this filed, as shown in Fig. 3, the `CertOper` in blockchain can provide forward traceability.

2) DCBF-Dual Counting Bloom Filter: We present a revocation checking method that utilizes the **DCBF** to eliminate false positives. It has the property of economic space and efficient query.

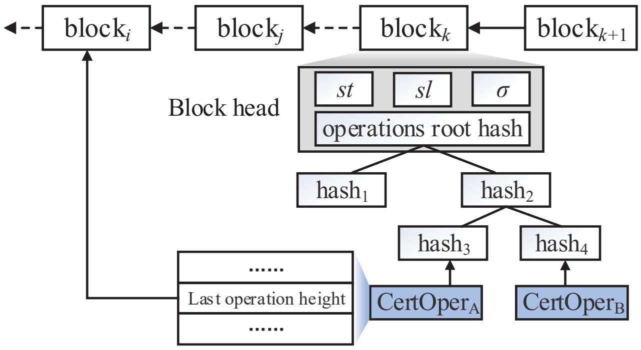

Fig. 3. Traceability of certificate operations in blockchain

Bloom filter [16] is a space-efficient probabilistic data structure used to check whether an element is a member of a set. In formulation, a Bloom filter is an array of m bits for representing a set $S = \{x_1, \dots, x_n\}$ of n elements. Initially all the bits in the filter are set to zero. There are k hash functions, $h_i(x), 1 \le i \le k$ used to map items $x \in S$ to random number uniform in range $1, \dots, m$. An element $x \in S$ is inserted into the filter by setting the bits $h_i(x)$ to one for $1 \le i \le k$. To test the set membership of an element y, we need to check every bits $h_i(y)$. If any of them are zero, y is definitively not in the set. In addition, the standard bloom filter dose not support elements deletion. This function can be achieved by counting bloom filter (CBF). In CBF, every bit is replaced by a counter. When a hash of an element is mapped to a counter, it increases by one. On the contrary, the counters corresponding to the hash of deleted elements decrease by one.

In this paper, the certificates are divided into two sets: valid certificates set and revoked certificates sets. We utilize two CBFs (CBF_1, CBF_2) to record the certificates in these two sets respectively. The new or updated certificates are inserted into CBF_1. If a certificate is revoked, it should be deleted from CBF_1 and inserted into CBF_2. In this way, comparing the query results from both CBFs, we can determine whether a certificate is a false positive element, and then judge the accurate status of this certificate by checking the related operations in blockchain.

C. Extension layer

Based on Ourboros [17], we design our consensus protocol by defining dependability-rank as the measurement of CA's trust and the probability of leader elected process. In this layer, we describe the *block and blockchain*, *consensus protocol* and the *incentive mechanism*.

1) The block and blockchain: The blockchain is maintained by bookkeepers. We will give definitions about some concepts.

***Definition** 1:* **Dependability-rank.** Dependability-rank d_i is used to express the dependability degree of a bookkeeper or a CA via evaluating their behavior.

***Definition** 2:* **Genesis block.** The genesis block B_0 contains the list bookkeepers identified by their public-keys and respective dependability-rank value $\{(vk_1, d_1), \dots, (vk_n, d_n)\}$.

***Definition** 3:* **Block.** A block B_i generated at a slot $sl \in \{sl_1, \dots, sl_R\}$ contains the current state $st \in \{0,1\}^\lambda$, operations root hash $op \in \{0,1\}^*$, the slot number sl and a signature $\sigma = Sign_{sk_i}(st, op, sl)$ computed under sk_i corresponding to the bookkeeper U_i generating the block.

***Definition** 4:* **Blockchain.** A *blockchain* is a sequence of blocks $B_0, B_1, \dots, B_n$. It holds that for each block B_i, the state st_i is equal to $H(B_{i-1})$, where H is a collision resistant hash function. The length of a *blockchain* $len(\mathcal{C})$ is the number of blocks. The latest block of the *blockchain* is called head block denoted $head(\mathcal{C})$.

2) The dependability-rank based consensus protocol: The consensus protocols in blockchain are responsible for solving two problems. The one is the process of *leader election* that elects one of bookkeepers to generate the next block. The other is dealing with fork. In this paper, we design a dependability-rank based consensus protocol which not only solves the two problems mentioned above but also stimulates CAs and bookkeepers to work positively and legally.

***Definition** 5:* **Leader Selection.** A bookkeeper U_i is selected to be the leader with probability p_i of its dependability-rank where $p_i = \frac{d_i}{\Sigma_{j=1}^n d_j}$. Leader selection process flips a p_1-biased coin to check whether the first bookkeeper is selected; then, for all $j \ge 2$, it flips a $(1-p_1)\cdots(1-p_{j-1})p_j$ biased coin to check whether the j-th bookkeeper is selected.

Dependability-rank based consensus protocol

This is a protocol run by bookkeepers $U_1, \dots, U_n$ interacting themselves over a sequence of slots $S = sl_1, \dots, sl_t$. Before establishing the blockchain, via negotiating, each corresponding CA gets their initialized dependability-rank d_i through the percentage of the number of certificates issued by itself accounting for the total certificates. The protocol proceeds as follows:

1) **Initialization** When the protocol starts, each bookkeeper broadcasts their public key and dependability-rank (pk_i, d_i). Then all of them get their genesis block B_0 which is used to initialize the blockchain $\mathcal{C} = B_0$. The initial state is $st = H(B_0)$.
2) **Chain extension** For every slot $sl \in S$, every bookkeeper U_i performs the following steps:
   a) collects all valid blockchains into a blockchain set $\mathbb{C}$, verifying that for every block B_i in each blockchain $\mathcal{C}_j$, it holds that $Verify_{pk_l}(\sigma', (st', op', sl)) = 1$ where pk_l is the verification key of the U_l generating the related block. U_i calls the function $maxvalid(\mathcal{C}, \mathbb{C})$ to select the new blockchain $\mathcal{C} \in \mathbb{C}$ and set state $st = H(head(\mathcal{C}))$.
   b) if U_i is selected as the leader in slot sl_k, it generates a new block $B = (st, op, sl_k, \sigma)$ where st is current state, op is the root hash of operations and a signature $\sigma = Sign_{sk_i}(st, op, sl_k)$. U_i extends $\mathcal{C}$ by appending B and broadcasts $\mathcal{C}$.

3) Incentive mechanism: It is well-known that CAs' economic benefit is derived from the domains' certificates operations. Based on **Dependability-rank based consensus protocol**, we present an *incentive mechanism* that takes the economic benefit and misbehavior into consideration. We define that every CA shares the dependability-rank d_i with the corresponding bookkeeper. According to the percentage of the valid certificates issued by each CA every quarter, all CAs' dependability-rank $D = \{d_1, \ldots, d_i, \ldots, d_n\}$ is initialized. The dependability-rank not only affects the probability of leader election among bookkeepers in consensus protocol, but also determines the operator of a domain's certificate operation. The latter directly affects the CA's economic benefits. Any domain selects a CA that has the maximum dependability-rank. If a CA issues a valid certificate, generates related operation, or reports an entity's misbehavior, it will be rewarded with dependability-rank. The bookkeeper related to a CA that has the maximum dependability-rank is elected to be the leader with the highest probability. The elected CA will consume some dependability-rank. Moreover, if a CA that is detected having some misbehavior, such as signing illegal certificate or issuing forge revocation information due to leakage of private key, it will be punished in the form of a reduction in dependability-rank, as well as the bookkeeper's misbehavior, such as omitting certificate operations.

D. Application layer

In this layer, we design the details of three types of certificate operations and certificate validation.

*1) **Certificate registration**:* The domain A sends a certificate registration request $RegReq = \{Cert_A, CA_i, Reg\}$ to a CA. (The CA is chosen according to the dependability-rank. More details are described in III-C3). As the operator, this CA checks the identity of the domain off-line and then signs the certificate. After that, by calling **algorithm** 1, it generates the corresponding `CertOper` with the operation type of certificate registration. This `CertOper` is broadcasted among all bookkeepers, and it will be recorded in blockchain by the leader. The hash of this certificate will be inserted into CBF_1. After about six slots (one slot one block), CA return the signed certificate and height back to domain A. The height is used for the third steps in Certificate Validation.

*2) **Certificate update**:* When the expiration date (*NotAfter time*) of a certificate is coming, the domain should request for certificate update without changing other information. Certificate update is similar with the process of certificate registration. The CA who receives the update request generates a `CertOper` with the type of certificate update. The field of `Last operation height` in this `CertOper` is filled with the related height value. The remaining process is the same with certificate registration.

*3) **Certificate revocation**:* Certificate revocation can be requested by a domain when its identity information is modified or its private key is leaked.

After receiving a certificate revocation request $RevReq$, the corresponding CA generates `CertOper` with the type of certificate revocation, then broadcasts it among bookkeepers. A bookkeeper firstly checks the `operation type` and extracts the `current certificate hash` from this `CertOper` if it is a revocation operation. And then, it puts this `CertOper` and `current certificate hash` into corresponding buffers respectively. The bookkeeper elected to be the leader deletes all the revoked certificates in revocation buffer from CBF_1 and inserts them into CBF_2. When a block is generated, the **DCBF** consisting of CBF_1 and CBF_2 as well as all `CertOper` are stored in this block.

*4) **Certificate validation**:* When a client C is ready to establish a SSL/TLS connection with domain A, the certificate $Cert_A$ with the related height value $height$ is provided by domain A in handshake protocol. Then, the client needs to initiate the process of $CertVal$ with four steps: (1) verify the signature; (2) check the expiration date; (3) check whether the

Algorithm 1 `CertOper` Generation

Input: $Cert_A$, $OperaType$, sk_{CA}.
Output: `CertOper`.

```
procedure OPERGEN(Cert_A, OperaType sk_CA)
    SubjectName, NotAfter ← Extract{Cert_A};
    if Reg = OperType then
        h(Cert_A) = null;
        Last_Oper_h = null ;
    else
        h(Cert_A) ← Hash(Cert_A);
        Last_Oper_h ← query(SbjectName, Timestep)
    σ ← Sign(Operation  content, sk_CA)
    return CertOper;
end procedure
```

Algorithm 2 Certificate Validation

Input: $Cert_A$, $height$, pk_{CA}.
Output: $b \in \{1, 0\}$. (The certificate is valid or not.)

```
procedure CERTVAL(Cert_A, height, pk_CA)
    if 1 ← Verify(Cert_A, pk_CA) then
        T_NotAfter ← Extract{Cert_A};
        if T_Current < T_NotAfter then
            send VerReq → a bookkeeper;
        else   return 0;
    else   return 0;
    if 1 ← BCHECK(hash_Cert_A, height) then return 1;
    else   return 0;
end procedure

procedure BCHECK(Cert_A, height)
    B_i ← Search{C, height};
    hash_Cert_A ← Hash(Cert_A);
    if 1 ← OperCheck(B_i, hash_Cert_A) then
        if 0 ← RevCheck(head(C), hash_Cert_A) then
            return 1;
    else   return 0;
end procedure
```

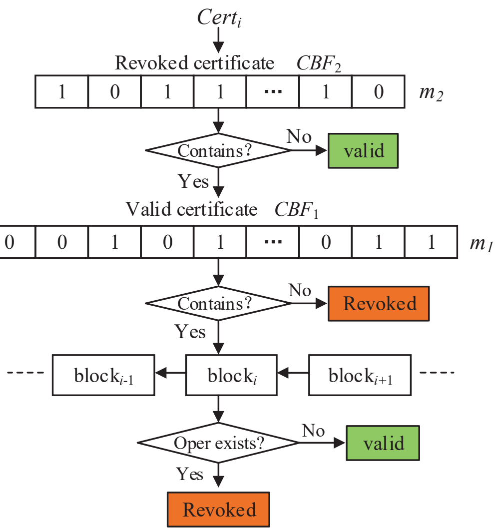

Fig. 4. Process of revocation checking

corresponding certificate operation is stored in blockchain; (4) check the certificate status (valid or revoked). The certificate validation process is shown as **algorithm** 2.

The first two validation steps are the same with them in traditional PKI. Therefore, we emphatically describe the last two steps which are specific in our scheme. The client C can launch a verification request $VerReq = \{hash_{Cert_A}, height\}_{sk_C}$ to a bookkeeper. The bookkeeper executes procedure BCHECK($hash_{Cert_A}, height$) by looking up the related certificate operation in blockchain and querying the **DCBF** for the status of $Cert_A$ from the latest block. The revocation checking steps in **DCBF** are shown as Fig. 4. Then, it gives a response to the client. If the related certificate operation is not stored in blockchain or the certificate status is revoked, the TLS/SSL connection will be terminated.

IV. Security Analysis

***Theorem** 1:* In CertChain, the certificate operation can be traced efficiently and certificate revocation checking can be fed back efficiently without false positives under **DCBF**.

Proof: We can find that, as shown in Fig. 3, for a given certificate operation, we can get the history certificate operations efficiently without traversing the blockchain. The reason is that the `CertOpers` provide certificate operation chains for all domains whose certificate operations are recorded in blockchain. For certificate revocation checking, it is well-known that, compared with traditional data structure exploited by CRL and OCSP, bloom filter is a space-economic and query-efficient data structure. However, false positives brought by bloom filter is unacceptable in certificate validation. In CertChain, since all the certificates operations are recorded in blockchain, it is easy to insert all valid certificates into CBF_1. When a certificate is revoked, it should be deleted from CBF_1 and inserted into CBF_2. As shown in Fig. 4, through two queries from CBF_2 and CBF_1, we can get the accurate status of certificates. Though there is a small number of certificates reported ambiguity by both CBFs because of the false positive rate, we can determine the status of these certificates through querying the blockchain.

Therefore, in Cerchain, the query of certificate operations and revocation can be fed back efficiently. ■

***Theorem** 2:* By self and public audit, CertChain can tolerate the failure of defense mechanisms implemented in CAs or bookkeepers under the threat model in section II-B.

Proof: Note that the process of certificate validation launched by a client includes the existences checking for the certificate operations in blockchain. According to this characteristics of CertChain, all the corresponding operations must be recorded in blockchain to receive self and public audit, otherwise the certificates cannot pass the clients' certificate validation. Therefore, even if an attacker compromises a CA and gets its private key to issue a certificate and operation on blockchain, the manager of CA can detect this certificate by traversing the blockchain. The certificate operations and revocation information recorded in blockchain are also verified by all other bookkeepers. Thus, the compromised bookkeepers would be detected quickly. ■

Next, we analyze CertChain's security against various attacks.

DoS attacks: In practice, CertChain should be hosted on DoS resilient infrastructure, likes a CDN. This makes it extremely difficult for an attacker to prevent CAs or bookkeepers from generating or recording legitimate certificate operations. However, DoS attacks cannot be defended absolutely. Assume that a power attacker could block access to any CA or bookkeeper by DoSing it. In CertChain, since the same kind of entities such as CAs or bookkeepers are in parallel, the entity attacked by DOS-ing could be replaced by others. But in traditional PKI, if the CA or OCSP provider is under attack, the result would be serious. Therefore, compared with traditional PKI, CertChain can provide sustainable service under the inescapable DoS attacks.

Rogue certificates or operations: In CertChain, all operations should be recorded in blockchain, which are available for public audit. In section III-D4, certificates validation processes include the existence of the relevant certificate operations. In other words, if an adversary compromises a CA and issues a certificate for a vicious domain, it should generate a related operation stored in blockchain for public and self audit. The CA's manager can check the issued operations from blockchain and will find the illegal certificate operations.

CA's private key leakage: The CAs are the main roles who are responsible for preventing from attacks and detecting misbehavior in CertChain. If an adversary gets a CA's private key, then it can issue certificates, generate operations, and revoke certificates. Although these operations will be broadcasted among bookkeepers, except the manager of this CA, no entity would detect forge operations before the malicious web server is reported to be dishonest. The reason is that all the certificates and operations issued by this CA can pass the signature verification with its public key. For an attacked CA, if its manager detects certificate operations in blockchain without identity check, it should issue the revocation information directed for these certificates and update its private key as soon as possible. In this case, the CA would not be punished in the form of a reduction in dependability-rank. However, if the malicious web server is reported by nodes in network before the manager solving this problem, the corresponding certificate operator will be seriously punished. Therefore, by self and public audit, Certchain can mitigate this type of attacks.

V. Experiment and Evaluation

A. Implementation

We develop a prototype implementation of CertChain. The main processes of CertChain are written in Javascript(node.js), HTML, CSS, and PHP. We implement the domain by extending an Apache HTTP server (version 2.4.27), and create CAs with OpenSSL. Bookkeepers' implementation is based on Ethereum [18], and the called API interfaces include 1) `web.eth.getBlock`, 2) `web.eth.getTransaction`, and 3) `web3.eth.contract`. Bookkeepers insert all `CertOper` and **DCBF** in Merkle hash trees implemented by SHA-512 which are stored in blockchain. We implement the client by Firefox Developer Edition, because it offers low-level APIs for obtain the certificate information. The prototype is composed by ten CAs with Intel celeron E4300 (2.6GHz) CPU, 4G RAM, and Ubuntu 14.04 64bit operation system. Correspondingly, ten bookkeepers are implemented with Inter(R) Xeon(R) CPU E5-2682 v4 @2.5GHz, 4GB RAM and Win server 2012 R2 Datacenter.

B. Result analysis

Firstly, we evaluate the basic characteristics of blockchain from three aspects, the speed of block generation, the average size of a block, and the capacity of a block. In Ethereum, the size of a block is limited to 2MB. The difficulty that determines the speed of block generation is adjustable. We set the difficulty as `0x160000`, so that blocks are generated about every 6.7s on average. We measure that the size of an empty block is about 2.6KB, and a single `CertOper` is about 1.8KB. We assume that we have one million certificates, and under 5% revoked certificates. Then we get the size of the **DCBF** about 412KB. All `CertOper` and **DCBF** are stored in a block. Therefore, one block maximally contains more than 500 `CertOpers`.

Secondly, we investigate how long it takes for this infrastructure to process a certificate operation initiated by a domain and a certificate validation initiated by a client. Measurements are given as the average over 100 test runs, and the results are presented in Table I. In a process of certificate operation, a CA needs to generate two signatures for the certificate and operation, which costs about 23ms. A bookkeeper who is the leader needs to generate the MHT containing all `CertOpers` and updates the latest certificate statuses in **DCBF**, and then stores them into a block. Therefore, Bookkeepers costs about 130ms. The first two steps in certificate validation initiated by a client cost about 9.5ms and the last two steps accomplished

TABLE I
Processing time (in milliseconds) required for three certificate operations and certificate validation

Operation	**CA**	**Bookkeeper** (leader)	**Client**
`RegReq`	23.21	130.54	–
`UpdReq`	23.05	131.36	–
`RevReq`	23.32	145.74	–
`CertVal`	–	23.86	9.50

TABLE II
Processing time (in milliseconds) of every step in certificate validation

	Client		**Bookkeeper**	
step detail	step 1 V_Sign	step 2 Ch_NotAfter	step 3 Query_**MHT**	step 4 Query_**DCBF**
time	6.12	2.01	13.66	6.77

by a bookkeeper cost about 23.86ms. In practice, we mostly care about the whole processing time of certificate validation which directly affects the clients' website experience. We measured the detail processing time from four steps as shown in Table II. The former two steps are the same as in traditional PKI. The latter two steps are executed by a bookkeeper. The processing time in the third step includes block lookup process according to the height and Merkle Hash Tree query. The last step is to check the status of certificate. Note that, to determine the status of a false positives certificate in **CBF**, extra time is cost to query blockchain. Fortunately, we make the false positive rate low enough, so the processing time of these two steps is nearly stable.

Thirdly, we give a comparison about the space cost and TLS handshake latency between Certchain and other schemes in certificate revocation. We measure the space requirement of CCSP, standard bloom filter, and counting bloom filter with different percentage of revoked certificates. As shown in Fig. 5, CCSP needs the smallest space and CBF requires the largest space. The reason is that for the lower percentage of revoked certificates, more than 99% bits in CCSP's bitmap are 0. After compressing, the space is small indeed. Based on the standard bloom filter, CBF replaces the bit with counter, so the space is large. However, compared the size of a block, this space requirement is acceptable. We also compare the TLS handshakes latency among OCSP, CCSP, and Certchain as shown in Fig. 6. In OCSP, the average latency is 250ms [15]. The latency in CCSP is a little higher than 120ms [12]. Compared with these two, under different CBF false positive rate, we find that the latency of CertChain is about 55ms. The query time increases with the false positives reducing, while this increase is too small. Compared with Certchain, CCSP cost more time, because it must extract the compressed bitmap when the client checks the status of a certificate. What's more, CertChain provides a distributed revocation checking service rather than a centralized server which is vulnerable under single-point-failure.

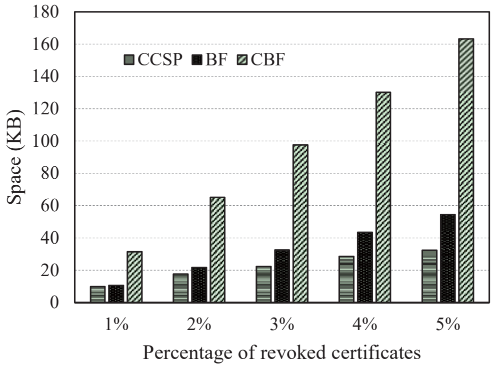

Fig. 5. Space required to store revocation status for one million certificates.

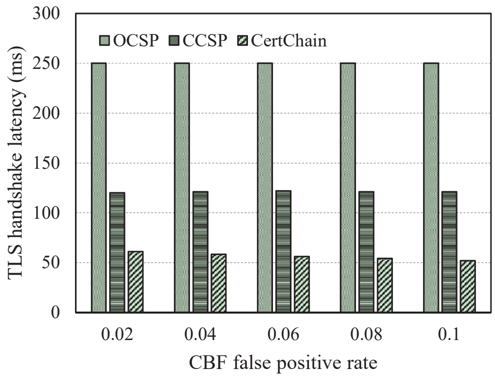

Fig. 6. TLS handshake latency with different CBF false positive rate.

Finally, from ten pairs of CAs and bookkeepers, we investigate the relationship between the number of certificates issued by a CA and blocks generated by the relative bookkeeper. As shown in Table III, we find that the ratio of certificates issued by CAs are uniform as well as the number of blocks. In other words, our scheme achieve the goal of consensus fairness.

TABLE III
COMPARISON BETWEEN THE NUMBERS OF CERTIFICATES ISSUED BY CAS AND BLOCKS GENERATED BY THE RELATIVE BOOKKEEPERS

No.	Num_Cert	Num_Block	**No.**	Num_Cert	Num_Block
1	18	147	6	20	154
2	19	150	7	19	153
3	20	151	8	21	150
4	18	147	9	20	155
5	19	150	10	19	147

VI. Related Work

A. CA-based trust disperse

The core of traditional PKI is the ecosystem of CA which is responsible for issuing and maintaining SSL certificates. However, recent compelling real-world attacks have demonstrated existing CA's vulnerability. Multi-cooperation and limited-policy are the most common methods to solve this problem.

In Cosigning [5], Syta et al. propose a multi-signature scheme that adapted a scale of thousands witness. However, there are some problems not being taken into consideration. For instance, in order to certify the correctness of a signature, a verifier needs to get a collective public key which is constituted of all the keys of witnesses who participate in signing the message. In other words, it has to verify the identity of the witness and its public key. In ARPKI [6], Basin et al. present an attack resilient PKI by signing, checking, sending certificates among multiple CAs in line. It improves the security of PKI, but it also brings in the problem of sustainable service because each CA is the target of Denial of Service (DoS) attack. In TriPKI [4], Jing et al. put forward a certificate management scheme based on threshold signature to resist single-point failure by introducing multiple CAs and DNSs and together with integrity log servers. Note that in practice, the CA companies are competitive not cooperative, even if the CAs are denoted servers in one company. Therefore, this method may be too expensive to implement. The authors of [4] and [9] take the idea of mutual verification into consideration to monitor the misbehavior. But sometimes, the introduced entities bring in new vulnerability or become the bottleneck. In fact, there exist some researches to improve the TLS PKI about CAs' misbehavior based on blockchain smart contact. Matsumoto and Reischuk present the IKP [19] that automates response to unauthorized certificates and provided incentives for CAs to behave correctly. However, this scheme neither provides certificate public audit nor handles certificates revocation checking.

B. Log-based misbehavior monitor

In traditional PKI, certificates are held by domains and CAs. In other words, a certificate is only verified by the browser when a client access the website. To make the certificates public audit, Google firstly presented Certificate Transparency (CT) [7] for monitoring and auditing certificates. Through a system of certificates logs, monitors, and auditors, CT allows website users and domain owners to identity mistakenly or maliciously issued certificates and identify CAs that have gone rogue. The key idea of this scheme is introducing log servers that maintain append-only databases of certificates issued by CAs. These databases are implemented as merkle hash trees [20] which provide efficient proofs of a certificate's presence. Thus, an unauthorized certificate will be exposed to the public. Inspired by this idea, in [6, 7, 9, 10, 21], the authors introduce the log servers to record Certificates or revocation information. Unfortunately, log-based PKIs suffer from several other problems. First, log servers expands the attack surface of the whole system. Second, several log-based PKIs require all logs to periodically synchronize certificates, but they do not provide a secure synchronization protocol. Finally, log-based PKIs do not provide sufficiently incentive to record or monitor entities' behavior. Therefore, log-based PKIs may not work securely as expected.

C. Enhancing Certificates Revocation Service

Aimed at the problem of certificate revocation, some efforts are proposed recently, such as Google's CRLset [22] and

Mozilla's OneCRL [23]. Both of these two schemes would have significant difficulty in scaling to handle millions of certificates, because their data formats use 110 and 1,928 bits per revocation respectively. Additionally, they require users to place unconditional trust in Google and Mozilla, since these data structure are not auditable publicly. Chariton et al. present a Compressed Certificate Status Protocol (CCSP) [12] that is able to pack revocation information more than one million certificates in less than 10KB of space. CCSP introduces a new notion of signed collections, a *bitmap* used to record the revocation status of certificates. It utilizes two compression algorithms to achieve that it is possible in s bits to fit revocation information for more than s certificates. Larisch et al. propose a certificate revocation scheme [11] which aggregates revocation information and stored them in a space-efficient filter cascade data structure with neither false positive nor negative rate. Both of [11] and [12] update their latest status of revocation information by bitwise XOR within hours. For those certificates revoked in this period, they may be captured by attackers.

VII. Conclusion

To establish secure SSL/TLS connections, we propose a public and efficient certificate audit scheme based on blockchain (CertChain) in this paper. It applies characteristics of blockchain to provide a decentralized and tamper-proof public audit certificates management. Specially, we design a distributed dependability-rank based consensus protocol to avoid centralization in practice. We also propose a new data structure called `CertOper` that is stored in blockchain for forward traceability and public audit. To achieve economic space and efficient query for certificate revocation checking, we present a method that utilizes Dual counting bloom filter (**DCBF**) with eliminating false positives. The security proof and experimental results show that Certchain is suitable in practice.

References

[1] A. Langley, "Enhancing digital certificate security," 2013. [Online]. Available: http://googleonlinesecurity.blogspot.de/2013/01/enhancing-digital-certificate-security.html.

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

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

[4] C. Jing, Y. Shixiong, Y. Quan, D. Ruiying, and X. Guoliang, "Checks and balances- a tripartite public key infrastructure for secure web-based connections," in *IEEE INFOCOM*, 2017, pp. 2322–2330.

[5] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE S&P*, pp. 526–545.

[6] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *ACM CCS*, pp. 382–393.

[7] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," 2013, rfc6962. [Online]. Available: https://tools,ietf.org/html/rfc6962

[8] P. Eckersley, "Sovereign key cryptography for internet domains," *https://git.eff.org/public/sovereign-keys.git*, 2011.

[9] T. H.-J. Kim, L.-S. Huang, A. Perring, C. Jackson, and V. Gligor, "Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure," in *WWW*, pp. 679–690.

[10] M. D. Ryan, "Enhanced certificate transparency and end-to-end encrypted mail," in *NDSS*, 2014, pp. 1–14.

[11] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Crlite: A scalable system for pushing all tls revocations to all browsers," in *IEEE S&P*, 2017, pp. 539–556.

[12] A. A. Chariton, E. Degkleri, P. Papadopoulos, P. Ilia, and E. P. Markatos, "CCSP: a compressed certificate status protocol," in *IEEE INFOCOM*, pp. 1091–1098.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[14] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *IEEE S&P*, 2015, pp. 104–121.

[15] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's pki," in *ACM IMC*, 2015, pp. 183–196.

[16] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[17] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol." [Online]. Available: http://eprint.iacr.org/2016/889

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger (eip-150 revision)," 2017. [Online]. Available: http://gavwood.com/paper.pdf

[19] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *IEEE S&P*, 2017, pp. 2375–1207.

[20] G. Becker, "Merkle signature schemes, merkle trees and their cryptanalysis," *Tech. Rep.*, 2008.

[21] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible TLS certificate management," in *2014 ACM CCS*, pp. 406–417.

[22] A. Langley, "Revocation checking and chromes crl," 2012. [Online]. Available: https://www.imperialviolet.org/2012/02/05/crlsets.html

[23] M. Goodwin, "Revoking intermediate certificates: Introducing onecrl," http://mzl.la/1zLFp7M, 2015.