BKI: Towards Accountable and Decentralized Public-Key Infrastructure with Blockchain

Zhiguo Wan[1(✉)], Zhangshuang Guan[1], Feng Zhuo[1], and Hequn Xian[2]

[1] School of Computer Science and Technology, Shandong University, Jinan, Shandong, China
wanzhiguo@sdu.edu.cn, gzs_1994@163.com, 2906719340@qq.com

[2] College of Computer Science and Technology, Qingdao University, Qingdao, Shandong, China
xianhq@126.com

Abstract. Traditional PKIs face a well-known vulnerability that caused by compromised Certificate Authorities (CA) issuing bogus certificates. Several solutions like AKI and ARPKI have been proposed to address this vulnerability. However, they require complex interactions and synchronization among related entities, and their security has not been validated with wide deployment. We propose an accountable, flexible and efficient decentralized PKI to achieve the same goal using the blockchain technology of Bitcoin, which has been proven to be secure and reliable. The proposed scheme, called BKI, realizes certificate issuance, update and revocation with transactions on a special blockchain that is managed by multiple trusted maintainers. BKI achieves accountability and is easy to check certificate validity, and it is also more secure than centralized PKIs. Moreover, the certificate status update interval of BKI is in seconds, significantly reducing the vulnerability window. In addition, BKI is more flexible than AKI and ARPKI in that the number of required CAs to issue certificates is tunable for different applications. We analyze BKI's security and performance, and present details on implementation of BKI. Experiments using Ethereum show that certificate issuance/update/revocation cost 2.38 ms/2.39 ms/1.59 ms respectively.

Keywords: Blockchain · PKI · Security

1 Introduction

Public key infrastructure (PKI) plays a critical role for network security, e.g. SSL/TLS for secure web communication, public key crypto-based security protocols. The security of the PKI is of paramount importance to applications relying on it. However, traditional PKIs suffer from a well-known vulnerability in case

Z. Wan—This work was supported by the National Natural Science Foundation of China under Grant 61370027.

of compromised or malicious CAs. That is, a compromised or malicious CA may issue a certificate for some domain, which can use it to launch impersonation or Man-in-the-Middle attacks.

Many attacks have demonstrated the serious vulnerability of traditional PKIs. Recently, fraudulent certificates have been issued for domains of Google.com, Yahoo.com, mozilla.org from well-known CAs [1]. Such bogus certificates may have been used by the adversary to eavesdrop communication. This vulnerability is because current PKIs lack mechanisms to detect and prevent CA misbehavior.

To counter against this problem, different approaches have been proposed recently to make certificate issuance transparent and accountable. Among them, the public log-based schemes have been the most effective approach to achieve this goal. Recent advanced proposals, AKI [1], ARPKI [2], EICT [3] and DTKI [4], all follow this methodology. Schemes including ARPKI and DTKI even have formal proofs to ensure their security. Although these solutions have solved the vulnerability of traditional PKIs to some extent, they have several drawbacks for their complex operations and interactions. To prevent misbehavior, CAs and other entities (e.g. log servers) need to monitor each other's activities, incurring too much communication cost for the PKI system.

We turn to Bitcoin [5] for a better solution for this problem. Bitcoin has proved itself an overwhelming success over other alternatives as a digital cryptocurrency. The underlying blockchain technology of Bitcoin has gained tremendous attention, and it has been used in many different fields, ranging from decentralized storage, crowd funding, equity trading to notary services.

Observing the similarity between the PKIs with a public log as in AKI and ARPKI and the Bitcoin system, we take advantage of the blockchain technology to design a full-fledged decentralized PKI, referred to as **BKI**. In our design, the certificate issuance for one's public key is realized by creating a transaction. Since for each transaction to be valid, it must be signed by the senders. This is similar for CAs to certify a public key by generating signatures over the public key along with other related data.

To reduce trust on a single CA as AKI, a user can request certificates from multiple CAs and combine the certificates together to obtain his final certificate. The user needs to choose at least k CAs he trusts to certify his public key, where k is the minimum number of CAs needed for generating a valid certificate and it can be set as a system-wide parameter. With the blockchain technology, this can be implemented with a transaction which has multiple CAs as the senders and the certificate applicant as the receiver. The first advantage is BKI has a smaller certificate status update interval, which is determined by the block generation speed. For blockchain like Ethereum [6,7], a block is generated every 12 s, which is much smaller than the update interval in AKI or ARPKI. Secondly, BKI has a special Merkle Patricia Tree used for certificate status checking. This makes BKI more efficient than AKI and ARPKI in terms of certificate storage cost and verification cost. Thirdly, system parameters in BKI like the number of required CAs to issue a certificate are tunable according to different security requirements.

Contributions. In this paper we propose a novel solution for the vulnerability of PKIs based on the blockchain technology of Bitcoin [5] to deliver an accountable and efficient decentralized PKI. Our proposal, called BKI, takes advantage of the blockchain technology to implement a decentralized certificate log, on which it gracefully achieves certificate issuance, revocation and update. Main contributions of our work include:

– We propose BKI, an accountable and decentralized PKI built from the blockchain technology of Bitcoin to conquer the vulnerability of traditional PKIs. BKI achieves short certificate status update interval of tens of seconds, as compared to 1 h update interval in AKI/ARPKI, significantly reducing the vulnerability window and improving security. In addition, BKI is flexible in that system parameters like the number of CAs to issue a certificate are tunable.
– A detailed comparison between AKI and BKI is provided to highlight the advantages of our proposal. We also give a detailed discussion and a thorough analysis of BKI on its security, efficiency and flexibility.
– A prototype of BKI is implemented using Ethereum blockchain for performance evaluation. Experiments show that certificate issuance/update/revocation cost 2.38 ms/2.39 ms/1.59 ms respectively.

Organization. The rest of the paper is organized as follows. Background knowledge about log-centric PKIs and blockchain are presented in next section. Then We define the problem to be solved in this paper in Sect. 3. After that, we describe BKI in detail in Sect. 4, followed by discussion and analysis on security, efficiency and availability of BKI. Details on the implementation of the prototype of BKI are described in Sect. 6, and we also evaluate its performance there. Finally, concluding remarks are given in the end.

2 Background

Quite a few schemes have been proposed to deal with the vulnerability in current PKIs. Readers are referred to [2] for a comprehensive review of these works, which are categorized into client-centric, CA-centric, and domain-centric approaches. However, there is a special group of schemes built on the certificate log idea, which should be categorized as the log-centric approach. Here we review the schemes from the log-centric approach since they are very close to our proposal.

Log-centric PKIs. Certificate transparency (CT) [8] employs the Merkle hash tree structure to build an append-only log to record all registered certificates. After registering one's certificate with the log server, each domain is given a non-repudiable audit proof that its certificate is on the append-only log. This audit proof and the certificate are both provided to the client for validation. However, CT is only designed to make certificate issuance transparent, and it does not has revocation function. Hence it cannot detect or prevent registration of bogus certificates generated by compromised CAs. To amend this problem, Revocation Transparency (RT) [9] was proposed to implement certificate revocation.

Enhanced Issuance and Revocation Transparency (EICT) [3] combines the idea of CT and RT to achieve a more efficient transparent PKI. Although EICT can detect bogus certificates issued by compromised CAs, it cannot prevent them from happening as it lacks monitoring mechanisms. DTKI [4] further extends EICT by using multiple logs and maintainers, and web users collectively monitor the logs to detect misbehavior. Although the public logs are monitored by web users, it is still possible that some fraudulent certificates escape monitoring and stay on the logs for quite some time. Sovereign Keys (SK) [10] uses a timeline server to maintain an append-only and read-only certificate log, and that log is mirrored to avoid performance bottleneck. However, SK cannot prevent attacks due to compromised sovereign keys. Meanwhile, the mirrors is assumed to be trusted, which is a strong assumption.

Accountable Key Infrastructure (AKI) [1] intends to solve the single point of failure of CAs with public certificate logs. The log is organized using a lexicographic Merkle tree, such that certificate revocation can be done efficiently. Meanwhile, CAs and other entities monitor each other frequently to detect and prevent misbehavior. Attack Resilient PKI (ARPKI) [2] enhances AKI with stronger security guarantees and formal treatment. Using a system model similar to AKI, ARPKI specifies more details about log synchronization, validation and monitoring.

Bitcoin and Blockchain. Bitcoin [5] is a completely decentralized digital currency without relying on any trusted party. All participants of the Bitcoin system are connected by Internet and form a P2P network. They follow a suite of protocols to maintain coin generation, coin transmission, transaction verification and data synchronization etc. More importantly, the underlying technology of Bitcoin, called blockchain technology, has been a very useful tool in many areas, like decentralized storage, crowd funding, equity trading, notary services etc. This work is also an example of application of the blockchain technology in a new area (Fig. 1).

Blockchain. The blockchain is the core data structure of Bitcoin, containing all coin generation and transaction information. As its name implies, it is a chain of blocks, starting from the very first block with ID number 0 to the latest block. This chain of blocks serves as the decentralized ledger and it is maintained by peers of the Bitcoin P2P network. Each block is chained to the previous block by containing a hash of the previous block, and it also contains some transactions in its header.

A blockchain can be permissionless like Bitcoin or Ethereum [6,7], or can be permissioned like Hyperledger [11], which is maintained by privileged parties. Permissionless blockchains can use consensus mechanisms like Proof of Work (PoW), Proof of Stake (PoS) [12,13] and Delegated Proof of Stake (DPoS) [14], while permissioned blockchains can use Practical Byzantine Fault Tolerance (PBFT) [15], which is more scalable and efficient.

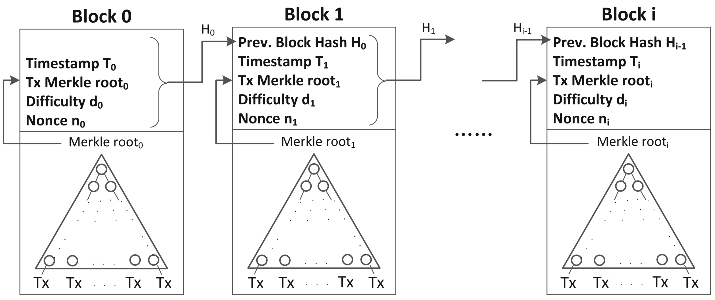


Fig. 1. Blockchain of Bitcoin: a block header contains a hash of its previous block header, a time stamp, a Merkle root computed from all transactions in this block, the current difficulty, a nonce which is the proof of work.

Transaction. A transaction is defined as follows:

$$\mathcal{TX} = \{TX\_ID, TX\_ID_{old}; Input: S_1, S_2, \ldots, S_k; \\ Output: PK_{R_1}, PK_{R_2}, \ldots, PK_{R_k}\},$$

where TX_ID is the identity of this transaction, TX_ID_{old} is the preceding transaction from which the Bitcoins are from, S_i is the sender of the transaction, R_i is the receiver and PK_{R_i} is R_i's public key. Each transaction references a previous transaction from which it spends Bitcoins to new destinations. Our certificate transaction has at most one preceding transaction, and it extends the Bitcoin transaction with two fields: domain name and expiry time. All senders must sign the transaction so as to be verified successfully by others.

3 Problem Definition

We attempt to tackle the vulnerability of single point of failure in traditional PKIs using the blockchain technology. The new PKI should be able to effectively detect misbehavior of compromised CAs and thereby prevent further damages. It should also be simple and efficient in certificate maintenance. More importantly, it must be highly secure since it is the basis for PKC-based security protocols. In this section, we first describe the system model, assumptions and the adversary model of our PKI system. Then the design goals of our proposed PKI are presented.

System Model. In BKI there are four types of participants: CAs, blockchain log maintainers (BLMs), certificate owners and clients. Figure 2 illustrates the system architecture of BKI and their interactions in certificate management.

– **Certificate Authorities.** CAs are responsible for identity verification and certificates issuance for users. Similar to ARPKI and DTKI, CAs are not fully trusted and compromised CAs may generate fraudulent certificates to impersonate users.

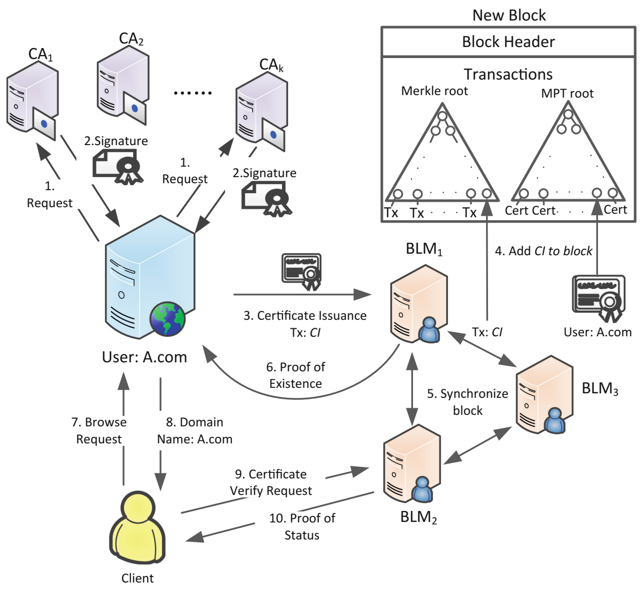


Fig. 2. BKI system architecture and certificate management

– **Blockchain-based Log Maintainers (BLMs).** Blockchain-based log maintainers (BLMs) are responsible for maintaining the blockchain, and each transaction in the blockchain-based log is a certificate transaction. Just like the Bitcoin blockchain, our blockchain-based log is also an append-only log, and the log is synchronized among BLMs to ensure it is up to date.
– **Certificate Owners.** Certificate owners can be of any type user of the PKI, e.g. domain owners or PGP users. They are also referred to as the users of the PKI. They need to provide their credentials to the CAs to obtain signatures for their public keys. Once the user certificate is added into the blockchain-based log, the user can use it to construct secure connections.
– **Clients.** Clients are those who need to verify validity of certificates. For example, a browser that needs to verify a SSL certificate when it visit a SSL-secured web site is a client of BKI.

Adversary Model. We assume that the adversary has full control over the communication channels between infrastructure entities of the PKI and users, i.e. the adversary can eavesdrop, modify and inject any message in the system. Furthermore, it can compromise and control some infrastructure entities, e.g. CAs or log maintainers. But the number of CAs compromised by the adversary is limited to a threshold k. However, the adversary is assumed to have limited computation resources and cannot break the cryptosystem used in our proposal. For the same reason, the adversary cannot produce proofs of work with extraordinary speed, and thus cannot generate new blocks as many as possible.

Design Goals. Our proposed scheme aims to achieve the following design goals:

- **Resilience.** The security of the new PKI should not rely on a single entity, and any entity is not completely trusted unconditionally. In the case of any entity being compromised, the new PKI should be able to detect its misbehavior immediately, and henceforth prevent any attacks, e.g. damages due to fraudulent certificate issued by compromised CAs.
- **Accountability.** All certificate operations are public and accountable, and everyone (users and certificate authorities) can check the blockchain to monitor certificate issuance, update and revocation.
- **Efficiency.** Communication efficiency is of importance for delay-sensitive applications such as web browsing. Hence it is important to reduce communication delay for operations including certificate verification.
- **Flexibility.** CA selection and security parameters should be flexible in the new PKI, so that the user can choose what they trust and prefer in certificate management.

4 BKI: The Blockchain-Based Decentralized PKI

In this section, we first give an overview of the proposed scheme, which helps readers to understand the underlying design principles. Then we describe our BKI in detail.

4.1 The Design of BKI

BKI consists of the following algorithms: initialization, certificate issuance, certificate verification, certificate update, and certificate revocation.

Initialization. In the initialization phase, each CA generates its own public key and private key pair (PK_{CA}, SK_{CA}), and publishes its public key on its website (or other secure places) such that every user can verify its pubic key. Blockchain log maintainers form a peer-to-peer network to maintain a blockchain which contains transactions about certificates, just like the Bitcoin system.

Similar to the peer-to-peer network of the Bitcoin system, the P2P network in our BKI has a specific communication protocol to exchange data on transactions, blocks and the blockchain. It enables BLMs, CAs and users to verify transactions, construct blocks and synchronize the blockchain with each other. The blockchain in BKI is open to everyone, so that anyone can monitor the activities of BLMs and any misbehavior will be detected at once.

Certificate Issuance. Certificate issuance is implemented by signing a transaction for a certificate requester, and this type of transaction is referred to as *certificate issuing (CI)* transaction. Specifically, a user u first generates his own public key and private key pair (PK_u, SK_u), and then request a certificate by asking k CAs to sign a transaction containing the public key PK_u for him. k can be system-wide parameter or a number chosen by u. A larger k means better security, so it is required that $k \geq 3$ for security.

An *unsigned CI* transaction is defined as follows:

$$\begin{aligned}&\mathcal{CI} = \{TX\_ID, NULL, DN, ET; Input: CA_1, CA_2, \ldots, CA_k;\\ &Output: PK_u\}\ or\ \{PK_{CA_1}, PK_{CA_2}, \ldots, PK_{CA_k}\}_k^t\},\end{aligned} \tag{1}$$

where CA_i is the identity of a CA, PK_u is the public key of user u, DN is the domain name and ET is the expiry time. DN and ET are not allowed to be modified in any case. TX_ID_{old} is set to $NULL$ as it is the first transaction. For this transaction to be valid in a blockchain, all senders must sign the transaction. The actual effect is that all CAs sign user u's public key with their private keys, meaning that the user's public key is certified by these CAs. To this end, u needs to approach each CA with his credential and each CA should check the credential before signing the transaction. This can be done with a secure out-of-band channel. The outputs include public keys, domain name and expired time of user u and t CAs. It is demanded that either user u or any t out of the k CAs can "spend" the output of this transaction. When the certificate needs to be revoked, any t of the k CAs can do it since they are in the output of the *CI* transaction.

The certificate issuing process proceeds as follows:

- **Step 1.** User u selects k CAs as his certificate issuing authorities. Then u creates an unsigned version of the *CI* transaction of (1), and sends it to each of the k CAs along with his credential *cre* for certification using an out-of-band channel.

$$\mathcal{CI}, cre \tag{2}$$

- **Step 2.** Upon receiving the request from u, CA_i verifies u's credential, signs the transaction with its private key, and then returns the signed transaction to u.

$$Sig_{CA_i}(\mathcal{CI}) \tag{3}$$

- **Step 3.** After collecting all k signed transactions, u can merge $Sig_{CA_i}(\mathcal{CI}), i = 1, 2, \ldots, k$ into a final *CI* transaction as showed below:

$$\mathcal{CI}, \{Sig_{CA_i}(\mathcal{CI})\}_{i=1}^k. \tag{4}$$

  Then u publishes the *CI* transaction to the P2P network for verification.
- **Step 4.** If $\mathcal{CI}$ is verified successfully, BLM will check whether the domain name is registered in the blockchain, and update the state of MPT by checking new transactions in this new block only when the domain name is not occupied, then $\mathcal{CI}$ will be added into a new candidate block waiting for confirmation by some consensus algorithm.

– **Step 5.** Then the block containing the CI transaction will be appended into the blockchain after successful consensus, and the block will be synchronized throughout the P2P network. Subsequently, everyone can check the blockchain and verify the CI transaction of u.
– **Step 6.** Finally, the BLM contacted by the user sends a response back to the user. If the domain name is not occupied, this response will include the header of the block containing $\mathcal{CI}$ and a proof-of-existence of $\mathcal{CI}$ in the block. The proof-of-existence consists of the hash values on the Merkle tree which can prove that $\mathcal{CI}$ is on the Merkle tree. Otherwise, user will receive an error message.

Certificate Verification. This step utilize MPT [16] combines the advantages of Patricia tree and Merkle tree. It can not only perform efficient keyword query like Patricia tree, but also implement efficient verification of data at leaf nodes like Merkle tree. Ethereum's MPT uses the public keys of accounts as keys and treat balances as values at leaf nodes, while we use $\mathsf{SHA256}(DomainName)$ as keys, and treat the public keys and the expiration time as values at leaf nodes.

After a user, e.g. a domain owner, gets his certificate issued by BKI, the certificate issuance is also recorded on the blockchain maintained by BLMs. If a client intends to establish a secure connection with the domain server, then he needs to ensure that: (1) the certificate is indeed recorded on the blockchain; (2) the certificate has not been revoked. To this end, he follows the procedure below:

– **Step 7 and 8.** The client sends a request to the certificate owner, who will respond with the domain name of the domain owner. But whether this domain is valid or has been revoked is unknown, so the client needs to contact BLMs to verify this.
– **Step 9 and 10.** The client contacts BLMs to verify that a certificate corresponding to the domain name is on the blockchain and has not been revoked. On receiving the request, any BLM can check the MPT on whether a certificate corresponding to the domain name is on the MPT, and whether the certificate is revoked or not. Finally, the BLM returns a proof-of-status of the domain certificate to the client. The proof-of-status consists of the domain's certificate along with the hash values on the MPT which can prove that the certificate is on the MPT.

Certificate Update. The user can update his certificate whenever he feels necessary, without requesting help from any CA. This is achieved by the user generating a certificate update (CU) transaction, which "spends" the output of his CI transaction to his new public key. In order for the CAs to revoke the updated certificate, the output of the newly generated CU transaction should contain the same k CAs as the user's CI transaction.

Suppose user u has the following *CI* transaction:

$$\begin{aligned}\mathcal{CI} = \{&TX\_ID, NULL, DN, ET; Input: CA_1, CA_2, \ldots, CA_k;\\ &Output: PK_u \ or \ \{PK_{CA_1}, PK_{CA_2}, \ldots, PK_{CA_k}\}_k^t\}\end{aligned}$$

Then the *CU* transaction is as follows:

$$\begin{aligned}\mathcal{CU} = \quad &\{TX\_ID', TX\_ID, DN, ET; Input: u;\\ &Output: PK_u' \ or \ \{PK_{CA_1}, PK_{CA_2}, \ldots, PK_{CA_k}\}_k^t\}\end{aligned}$$

The transaction $\mathcal{CU}$ references the transaction ID, i.e. TX_ID, of the transaction $\mathcal{CI}$, indicating it updates the public key PK_u in $\mathcal{CI}$ to PK_u'. Note that there is only one sender u in the input of $\mathcal{CU}$. The output must contain the same CAs as those in $\mathcal{CI}$, so that the same CAs can collaborate to revoke $\mathcal{CU}$. Furthermore, user u can continue to update his public key by creating a new *CU* transaction in the same way.

Whenever the user's public key is updated, his old pubic key is invalidated and can not be used anymore. All the user's public keys and their update ordering are recorded in the blockchain. Anyone can check the blockchain to obtain one's latest public key.

Certificate Revocation. Certificate revocation is necessary when a user's private key is compromised. In this case, an adversary may use the user's private key for malicious purposes, or update the user's public key to a new one so that the user's old public key certificate is invalidated. Thus, the user should seek the help of CAs to revoke the old public key certificate.

At least t CAs from the output of the *CI* or *CU* transaction should collaborate to accomplish the revocation task. Suppose the t CAs that decide to revoke the user's old public key are CA_{i_1}, CA_{i_2},..., CA_{i_t}, where $i_1, i_2, \ldots, i_t \in \{1, 2, \ldots, m\}$. They generate the following certificate revocation (*CR*) transaction to revoke the user's certificate in $\mathcal{CI}$:

$$\mathcal{CR} = \{TX\_ID'', TX\_ID, DN, ET; Input: CA_{i_1}, CA_{i_2}, \ldots, CA_{i_t}; Output: NULL\}$$

5 Discussion and Analysis

In this section, we first compared BKI with ARPKI and AKI, and then analyze its security.

5.1 Comparison with ARPKI and AKI

There are some similarities between BKI and AKI (also ARPKI). Both BKI and AKI are based on a synchronized certificate log to manage certificate; both utilize multiple CAs to reduce trust on a single CA and remove single point of failure in CAs; Both employs multiple signatures to generate certificates for users. However, there are a number of differences in design making BKI more preferable.

Certificate Management. Merkle hash trees (forming a chain) are used by AKI and ARPKI to manage certificates, while certificate operations are recorded as transactions on blockchain in BKI. AKI relies on the Merkle tree to add new certificate, remove certificate and produce proof of absence of a certificate. In BKI, the Merkle hash tree is only used to obtain the Merkle root of transactions in a block, and Merkle hash trees from different blocks are independent from each other.

Another difference between BKI and AKI is that revoked certificates are removed from the latest Merkle hash tree in AKI, while revocation is realized as a revocation transaction in BKI. In order to make certificate verification more efficient, BKI employs Merkle Patricia Tree (MPT) to record the latest status of certificates. Therefore, it is more efficient to check certificate validity in BKI. Moreover, a complete life cycle of a certificate can be easily obtained from the blockchain in BKI.

Trustworthy Timestamping. BKI also inherits one additional advantage of the blockchain technology, trustworthy timestamping. Each block is generated by the consensus algorithm in fixed intervals, thus all transactions in that block are timestamped accordingly. These timestamps are trustworthy and can be used to prove when a certificate is issued, updated or revoked. The Merkle hash tree update interval in AKI is about one hour, while the block generation interval can be seconds in BKI, i.e. the timestaming precision can be seconds in BKI.

Flexibility. BKI is more flexible than AKI and ARPKI in that parameters can be chosen by certificate owners. The least number of CAs required for certificate issuance is variable for different applications in BKI. The number needs to be higher for sensitive applications like online banking, and it can be smaller for online forums. The least number of CAs required for certificate update can also be chosen differently for different applications.

5.2 Security Analysis

Due to space limit, we provide here an informal discussion on security of our proposed scheme from the following aspects. Rigorous formal treatment of BKI is left as our future work.

Compromise of CAs. We assume that a user must obtain k signatures from k different CAs of the n CAs in the system. Suppose the adversary can compromise a CA with probability p. Then the number of compromised CAs X follows binomial distribution $B(n, p)$. Then the probability of $X \geq k$ is as follows:

$$P(X \geq k) = \sum_{m=k}^{n} C_n^m p^m (1-p)^{n-m}$$

The relationship between P and n, k is showed in Fig. 3. We can see that, for any n, one can choose an appropriate k such that the $P(X \geq k)$ can be as small as possible. Actually, the probability of that a CA is compromised is far less than 0.05 in real life. So the security of BKI is guaranteed in the case of comprised CAs when we choose multiple CAs to manage certificates.

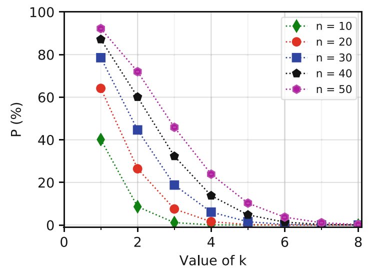


Fig. 3. The probability of forging a valid certificate by compromising at least k CAs for CA compromise probability $p = 0.05$.

Compromise of BLMs. The consensus algorithms used by BLMs in BKI have significant impact on security of BKI. Accordingly, the adversary can take different attack strategy against different consensus algorithms. For the Proof of Work (PoW), the adversary can launch "51%" attack only if he has more than 50% computation power of the whole system. For Practical Byzantine Fault Tolerance (PBFT), the adversary needs to compromise more than $(N-1)/3$ BLMs among all N BLMs. And when BKI uses Proof of Stake (PoS) or Delegated PoS (DPoS), the adversary can launch the "51%" attack, which requires more than 50% resources of the system or controlling more than 50% delegators. Compared with the attacks against CAs, these attacks may incur big cost, resulting less economic incentives for the adversary. So the adversaries may mainly aim at attacking CAs instead of BLMs.

Vulnerability Window. Vulnerability window is determined by the certificate status update interval (i.e. ILS update interval in AKI or ARPKI). When the certificate private key is leaked or the adversary succeeds in forging the certificate with the help of enough compromised CAs, the system should revoke the bogus certificate as soon as possible to reduce losses during revocation The vulnerability window in BKI is determined by block generation frequency, which can be only 10 min with Bitcoin or 12 s with Ethereum. From this perspective, BKI is more secure than AKI or ARPKI in that certificate status can be updated in seconds.

6 Implementation and Performance Evaluation

In this section, we describe our implementation of BKI and evaluate its performance. We implement our proposal using Ethereum, an open-source blockchain

Table 1. Average processing time(in ms) for certificate operations ($k = 3, t = 2$).

Time	Issue	Update	Revoke
TotalTime	2.385	2.387	2.386
SignTime	0.752	0.753	0.753
VerifyTime	1.633	1.633	1.633

Table 2. Average processing time(in ms) for certificate operations with different threshold.

Threshold	Issue	Update	Revoke
$k = 3, t = 2$	2.385	2.387	1.593
$k = 5, t = 3$	3.997	4.006	2.391
$k = 7, t = 4$	5.766	5.766	3.190
$k = 9, t = 5$	7.413	7.408	3.990
$k = 10, t = 8$	8.007	8.003	6.386

system that extends Bitcoin blockchain with smart contract functionality, which enables a simple and convenient implementation of BKI.

Implementation. BKI is implemented on Ethereum as a smart contract using a special javascript-like language called Solidity. A user initiates the smart contract to register with CAs, while CAs interact with the smart contract by sending certificate issuance, update and revocation transactions.

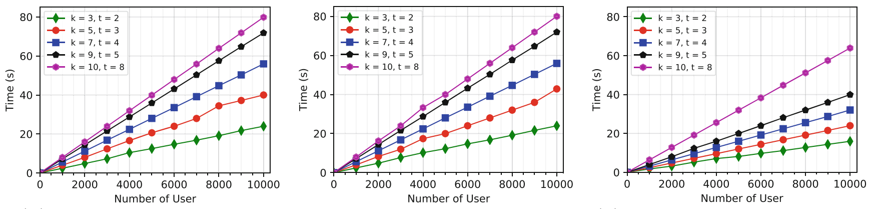
Our implementation supports multiple CAs (CA_1, CA_2, ..., CA_n) that provide certificate services to users. Each CA or user is represented by an address associated with a public/private key pair. With the smart contract implementation, a CA can generates a certificate issuance, update or revocation transaction that interacts with the smart contract. Note that each transaction is signed by its originator, so the user's certificate is issued, updated or revoked if enough transactions are received by the smart contract.

Preliminary Experiment Results. We evaluate the performance of our proposed scheme based on our implementation. Extensive experiments are conducted on a laptop with Intel Core i5-4200U 1.60GHz*4 and 4GB RAM running 64bit Ubuntu 16.04. In each experiment, a user request his certificate from a given number of CAs, which generate appropriate transactions for the user. The time for certificate issuance, update and revocation is measured for evaluation, while transmission time is not considered. We run the experiments for 100,000 times and average measurements are presented in the following two tables.

In Table 1, the average processing time shows the time for CAs to issue/update/revoke a public key certificate for $k = 3$ and $t = 2$. k and t ($t < k$) denote the threshold operated certificate, and k is used for *Issue* and *Update*, and t is used for *Revoke*. TotalTime denotes the average time spent on signatures and verifications. SignTime denotes the average time spent on signatures, and VerifyTime denotes the average time spent on verification. *Issue*, *Update*, and *Revoke* are the certificate operations k (or t) CAs execute. From the table, we can see that a certificate operation (issue/update/revoke) requires around 2ms.

We provide results for the different value of k and t and give measurements as the average over 100,000 runs. And we present the result in Table 2. The average processing time shows the time for CAs to issue/update/revoke a public key certificate for different k and t. As k and t increase, the time cost also increases accordingly, approximately linear to k.

Figure 4 shows the total time cost for certificates issued, updated or revoked with different value of k and t for the different number of users. The abscissa represents the number of users, and the ordinate represents the total time required for the corresponding number of certificate operations. For given k and t, as the number of users grows, the total time required for certificate operation is growing linearly.



(a) Certificates Issued (b) Certificates Updated (c) Certificates Revoked

Fig. 4. The total time that CAs operate certificates.

7 Conclusion

We have proposed BKI, a PKI built from the blockchain technology of Bitcoin to address the vulnerability of traditional PKIs. We have provided in-depth discussion and analysis on security and performance issues of BKI. A prototype of BKI has been implemented on Ethereum blockchain, and comprehensive experiments have been conducted to evaluate its performance. It has showed that certificate operations can be accomplished in about 2 ms. We plan to work out a formal security proof for our proposed scheme as the future work.

References

1. Kim, T.H.J., Huang, L.S., Perring, A., Jackson, C., Gligor, V.: Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. In: Proceedings of the International World Wide Web Conference, pp. 679–690. ACM (2013)
2. Basin, D., Cremers, C., Kim, T.H.J., Perrig, A., Sasse, R., Szalachowski, P.: ARPKI: Attack resilient public-key infrastructure. In: Proceedings of ACM CCS 2014, pp. 382–393. ACM (2014)
3. Ryan, M.D.: Enhanced certificate transparency and end-to-end encrypted mail. In: Proceedings of NDSS (2014)
4. Yu, J., Cheval, V., Ryan, M.: DTKI: a new formalized PKI with no trusted parties (2014). http://arxiv.org/abs/1408.1023

5. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
6. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2014). http://gavwood.com/Paper.pdf
7. Buterin, V.: Ethereum white paper: a next generation smart contract and decentralized application platform (2013). https://github.com/ethereum/wiki/wiki/White-Paper
8. Laurie, B., Langley, A., Kasper, E.: Certificate transparency. IETF RFC 6962 (2013)
9. Laurie, B., Kasper, E.: Revocation transparency. Google Research, September 2012
10. Eckersley, P.: Sovereign key cryptography for internet domains (2011). https://git.eff.org/?p=sovereign-keys.git
11. Androulaki, E., Cachin, C., Christidis, K., Murthy, C., Nguyen, B., Vukolić, M.: Hyperledger fabric proposals: next consensus architecture proposal (2016)
12. King, S., Nadal, S.: PPCoin: Peer-to-peer crypto-currency with proof-of-stake (2012). http://peerco.in/assets/paper/peercoin-paper.pdf
13. Buterin, V.: Slasher: a punitive proof-of-stake algorithm (2014). https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm
14. Larimer, D.: Delegated proof-of-stake (DPOS). Bitshare whitepaper (2014)
15. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI. **99**, pp. 173–186 (1999)
16. Work2Heat: understanding the ethereum trie (2014). https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/