Research Article

A Semidecentralized PKI System Based on Public Blockchains with Automatic Indemnification Mechanism

Gwan-Hwan Hwang ¹, Tao-Ku Chang ², and Hung-Wen Chiang¹

¹Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei, Taiwan

²Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien 97401, Taiwan

Correspondence should be addressed to Tao-Ku Chang; tkchang@mail.ndhu.edu.tw

Received 23 May 2021; Revised 27 July 2021; Accepted 21 October 2021; Published 31 October 2021

Academic Editor: Ruhul Amin

Copyright © 2021 Gwan-Hwan Hwang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The PKI framework is a widely used network identity verification framework. Users will register their identity information with a certification authority to obtain a digital certificate and then show the digital certificate to others as an identity certificate. After others receive the certificate, they must check the revocation list from the CA to confirm whether the certificate is valid. Although this architecture has a long history of use on the Internet, significant doubt surrounds its security. Because the CA may be attacked by DDoS, the verifier may not obtain the revocation list to complete the verification process. At present, there are many new PKI architectures that can improve on the CA's single point of failure, but since they still have some shortcomings, the original architecture is still used. In this paper, we proposed a semidecentralized PKI architecture that can easily prevent a single point of failure. Users can obtain cryptographic evidence through specific protocols to clarify the responsibility for the incorrect certificate and then submit the cryptographic evidence to the smart contract for automatic judgment and indemnification.

1. Introduction

The term public key infrastructure (PKI) refers to both hardware and software, as well as the policies, processes, and procedures necessary to create and deal with digital certificates, using asymmetric cryptography [1]. PKIs are a *sine qua non* for security in any e-commerce business environment. In the context of the Internet of Things (IoT), PKIs let users identify other trusted people, devices, and services. More than ever, cutting edge business applications are using PKI technology to facilitate online authentication and ensure compliance with increasingly strict regulations for protecting the security of online data.

In a PKI system, a Certificate Authority (CA) is responsible for issuing digital certificates. Whenever a client generates a public-private key pair, both the public key and the end-user's information are sent to the CA, which then creates a digital certificate comprising both the public key belonging to the user and the certificate attributes, which are used to verify the correctness of the information. The CA

signs the certificate with its private key, which legitimizes each certificate. Figure 1 shows how a person named Gillian Lin obtains a digital certificate from a CA.

In addition to verifying the CA's signature, the validity of the certificate also requires confirmation as to whether the certificate has been revoked. There are two methods to revoke the certificate. The first is called the Certificate Revocation List (CRL) [2], whereby the CA periodically publishes a revocation list. The verifier needs to download the complete revocation list to check whether the certificate is in the list. The second one is the Online Certificate Status Protocol (OCSP) [3], whereby the CA creates a certificate status database. The verifier only needs to provide the certificate information to the CA, whereupon the CA will return the latest status of the certificate to the verifier, based on the database. The current certificate status can therefore be "Good" or "Revoked" or "Unknown."

Current PKI and blockchain-based PKI [4] still have some problems, which are discussed in Section 2. To address these problems, we proposed a semidecentralized PKI

FIGURE 1: The process of obtaining a public key certificate from CA.

architecture based on the public blockchain. The user obtains cryptographic evidence using a specific agreement to clarify the type of certificate error and responsibility for the error and then submits the evidence to the smart contract for automatic judgment and indemnification, which can avoid the difficulties the user would otherwise encounter if he or she wanted to complain when the CA is not in the same country as the user.

The main contributions of this paper are as follows: (1) The proposed scheme could protect CA from attacks, such as DDoS attack. (2) The high-trust public blockchain could not cope with the issuance of a large number of certificates. The problem is insufficient bandwidth. We overcome this shortcoming by using the data structure of the TP-Merkle tree to enable the public blockchain-based PKI to be able to handle the issuance and revocation of all certificates. (3) Moreover, when the CA uploads the root hash of the TP-Merkle tree to the contract, the certificate information will be hashed, which can fully comply with the GDPR requirement for pseudonymization. (4) Finally, we make the smart contract as truly a trusted third party. A deposit system is established through smart contracts with TP-Merkle trees to generate cryptographic evidence and establish a fully-automated indemnification system.

The remainder of this paper is organized as follows. Section 2 gives an overview of related work. Section 3 describes the problems of the traditional PKI and blockchain-based PKI. Section 4 presents the system architecture and the proposed protocols. The implementation details and experimental results are presented in Section 5, and conclusions about the work described in the paper are drawn in Section 6.

2. Related Work

Decentralized public key infrastructure (DPKI) is used to solve the problems associated with the centralized PKI. DPKI is divided into two categories. One is trust networks, such as PGP [5], which is the most widely used software

package for e-mail and file protection. It establishes a decentralized trust model, wherein each party acts as a user and as a certification authority (CA); all users can be introducers to the web of trust, generate their key pairs, distribute their own public keys, and certify those of other users. The other category is to use a blockchain [6] technique called blockchain-based PKI. Bitcoin is a cryptocurrency with decentralized controls that was created in 2009. Blockchain was invented by Satoshi Nakamoto [7]. It provides a peer to peer network. Data stored in the blockchain is immutable and cannot be changed easily, and its ledger storing the transactions is visible to everyone. Using blockchain to store and manage certificates can avoid single-point failure.

Currently there are two types of blockchain-based PKI: decentralized and semidecentralized. The former was introduced by Mustafa Al-Bassam [8] and combines web of trust and blockchain technology to allow users to store certificates and other people's signatures in the blockchain. It is similar to PGP, but the storage location is different. The latter was introduced by Karen Lewison and Francisco Corella [9–13], and it retains the CA, which stores the certificate and revocation information in the blockchain. The user only needs to check the certificate information in the blockchain, without communicating with the CA.

As cryptocurrency is secured by cryptography, virtual or digital currency has the advantage of being practically impossible to double-spend or counterfeit. The bulks of cryptocurrencies [7, 14] are blockchain-based and are actually decentralized networks comprising distributed ledgers that are enforced by a wide-ranging network of computers. Cryptocurrencies can greatly simplify the direct transferring of funds between entities as there is no need for assistance from any bank, credit card issuing company, or other trusted third party. Security for cryptography transfers is provided by employing public keys and private keys in conjunction with various types of incentive systems, such as Proof of Work [7] or Proof of Stake [15].

Smart contracts are actually segments of code that carry out general-purpose computations, and one blockchain platform that is used to host and execute them is Ethereum. Written in Solidity, smart contracts author the digital tokens that can be used as proxies for money or other valuable assets, ownership shares, evidence of membership, and so on. Not only are these tokens tradeable, but also the so-called smart contracts can determine the rules for how and to whom they are distributed, for example, by limiting the supply of tokens or otherwise monopolizing their issuance. Further, every execution of a smart contract happens in public and the source code for the contract is often available.

Smart contracts are irreversible and immutable. Therefore, reasoning about the correctness of smart contracts before deployment is critical. There are also concerns that smart contracts are vulnerable to hacker attacks. One infamous example is the attack against the crowdfunding project Decentralized Autonomous Organization (DAO) in 2016 [16]. Using formal verification, it is possible to perform an automated mathematical proof that your source code fulfills a certain formal specification. A large number of surveys and schemes addressing smart contract analysis have been published. These include a review of security vulnerabilities [17–21], verification approaches and tools [22–29], formal specification and modeling techniques [30–32], and languages for smart contract development [33, 34]. As mentioned above, developers can apply these schemes to analyze and verify both the runtime safety and the functional correctness of the smart contracts.

3. Problems of the Traditional PKI and Blockchain-Based PKI

PKI has been widely used as the answer to many Internet security problems. However, there are still some problems with it [4]:

- (1) An unknown attacker is able to infiltrate the CA infrastructure and completely control all of the certificate-issuing servers during the operation, and they may also have issued some fake certificates that have not yet been identified. An example can be seen in the hack of DigiNotar in 2011 [35].
- (2) Many CAs have a low threshold for issuing certificates. Essentially, anyone can get certificates as long as they pay. This means that malicious users will pretend to be normal websites or services in the early going and lurk for a while. Then, after some time has passed, they will turn to phishing or a website containing a virus to launch malicious attacks [36].
- (3) CAs have a problem known as single-point failure. Here, single-point does not refer to a device, but rather refers to the CA itself. Sometimes it is not possible to connect to the CA server to check whether a certificate has been revoked because of various problems, such as DDoS attacks.
- (4) In addition, a CA is centralized architecture, and revocation of certificates after they have been issued

is usually opaque. It will take a long time to file a complaint when a user's certificate is maliciously or accidentally revoked because the user cannot provide substantive evidence to prove their innocence.

There are some successful cases or implementations of decentralized public key infrastructure (DPKI) to solve the problems caused by centralized PKI. These are divided into two categories. One is a trust network, such as Pretty Good Privacy (PGP) [5]; the other is blockchain-based PKI that is decentralized [8] or semidecentralized [9]. The semi-decentralized architecture looks perfect to solve long-standing problems. However, there are some emergent problems that make blockchain-based PKI not suitable to replace the current PKI:

- (1) The first problem is the performance issue and bandwidth limitation. Blockchain transaction speed is much slower. For example, the approximate average TPS of the Bitcoin [7] blockchain is about 5—although this may vary at times. Ethereum [14] can handle roughly double that amount. In comparison, Let's Encrypt [37], which is an open certificate authority (CA), can issue about 1 million certificates per day.
- (2) The second problem is the lack of a supervision mechanism. While storing certificates in the blockchain solves the problem of single-point failure when verifying the certificates, the process of issuing and revoking them is still a black-box centralized operation. In other words, users still cannot obtain useful evidence to prove whether there is a problem with the certificate information that the CA transfers in the blockchain.
- (3) The third problem is a contradiction of the General Data Protection Regulation (GDPR) [38]. The features of blockchain contradict two rules. One is personal data pseudonymization; the other is the right to erasure. GDPR explicitly recommends pseudonymization of personal data as one of several ways to reduce risks from the perspective of the data subject. To directly store certificates to blockchains, such as in [8, 9], is similar to publishing certificates with personal information on the Internet. The GDPR mandates that “the data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay.” In fact, data stored in blockchains are immutable and cannot be erased easily.

While PKI and DPKI are both used to solve the problem of identity verification, they both have their respective shortcomings. The goal of this paper is to integrate the strengths of PKI and DPKI to establish a more secure and convenient identity verification architecture. The biggest problem with PKI is single point of failure and the opaqueness of processes. Since blockchain technology is the best avenue for surmounting these problems, this paper will

focus on solving problems specifically related to blockchain-based PKI.

4. The Proposed Blockchain-Based PKI with a Fault-Tolerant Mechanism

4.1. System Architecture. Blockchain-based PKI with a fault-tolerant mechanism is a different PKI architecture from what we have seen in the past. Compared with traditional PKI, the system process contains two more steps: audit and appeal. The revocation, storage, and verification of certificates are also carried out in different ways. The system architecture is shown in Figure 2. The proposed scheme involves the following entities: web owner, CA, web user, and smart contract. In this paper, we refer to the web owner and web user as “owner” and “user,” respectively. The web owner and CA have the following public and private key pairs: (Pri(owner), Pub(owner)) and (Pri(CA), Pub(CA)), respectively. $[O]_{\text{Pri}(x)}$ is used to denote a digital signature of data object O that is generated by the private key of a subject x , and data objects within square brackets that are separated by commas are first connected and then have cryptographic operations performed on them:

- (i) Web owner: the web owner is the owner of the web certificate and the entity that announces the certificate to users for verification. It also plays the role of auditing and handling error complaints.
- (ii) CA: CA is the certificate authority, responsible for handling owner’s certificate-related requests. Such requests include certificate application, certificate revocation, and certificate replacement. It is responsible for uploading the certificate information to the blockchain.
- (iii) Web user: the web user is a person who visits the website and is responsible for checking whether the certificate is valid.
- (iv) Smart contract: the smart contract is deployed in a public blockchain by the CA, responsible for handling certificate storage, automatic appeal and ruling, and automatic indemnification.

The system process consists of four phases: request, clearing, auditing and appeal, and verification. The first three phases perform a supervision mechanism. The following sections will describe in detail how the system achieves the purpose of supervision. In the “verification” phase, a special layered verification is applied, which reduces the CA’s burden. A user can also distinguish whether the revocation list or other verification information they obtain is correct. Even if the information contains some error, the user can obtain the correct information from other sources. We refer to this as the fault tolerance of the certificate verification. In addition, in order to make the appeal easier, we have changed the possible status of the certificate from three statuses (Good, Revoked, and Unknown) to four statuses (Add, Renew, Pause, and Revoked), which are described as follows:

- (i) The “Add” status indicates that the certificate is valid and is a newly added certificate
- (ii) The “Renew” status indicates that the certificate has been renewed or changed from status “Pause” to status “Renew”
- (iii) The “Pause” status indicates that there are problems with the certificate, but it still has a chance to return to “Renew” status
- (iv) The “Revoked” status indicates that the certificate has been revoked permanently

4.2. Transaction Positioned Merkle Tree. In order to solve the bandwidth problem of blockchain, we use a tree data structure called a Transaction Positioned Merkle Tree (TP-Merkle tree) [39] to store certificates. This data structure can compress millions of certificates into a 32-byte data type. Data structures of the Merkle tree type rely on a unique signature and a set of messages for authentication, but they allow the intended verifier to establish the authenticity of a single message without compromising the secrecy of the other messages by disclosing them. The Merkle tree associated with a set of messages $M = \{m_1, \dots, m_n\}$ is built by bottom-up recursive computation. Initially, for every message $m \in M$, a distinctive leaf that contains the hash value of m is added to the tree. Subsequently, the value associated with each internal node is equal to $h(h_l || h_r)$, with $h_l || h_r$ denoting the concatenation of all the hash values associated with the left and right child nodes and $h()$ being a hash function. Thus the root node of the resulting binary hash tree is the digest of all the messages, and thus it can be digitally signed by using a standard signature technique.

The TP-Merkle tree shown in Figure 3 is based on a Merkle tree [40] that adds a position function and its tree height is determined in advance before storing data. The leaf node can store multiple data, and each data must be stored in a key-value pair, for example, (key, value). The key called “IndexValue” is the parameter used for positioning, and the value is the actual data. All the key-value pairs under the leaf node are called the list of key-value pairs. When passing IndexValue into the index Function Γ function, we get the specified leaf node position. The index function Γ is expressed as follows:

$$\Gamma(\text{IndexValue}) = \text{SHA256}(\text{IndexValue}) \bmod 2^{N-1}, \quad (1)$$

where N is the height of the tree. 2^{N-1} is used to calculate the total number of nodes of the tree. Using a modulus operation, the position will not exceed the total number of nodes of the tree. It is convenient to find the corresponding data through IndexValue when the data are stored in this way.

4.3. Merkle Proof: Slice. Slice, as shown in Figure 4, is important evidence for verification when using the TP-Merkle Tree. Each leaf node corresponds to a slice and takes the form shown in the following example, where a new certificate information is placed in a leaf node with Index = 3, and the set of all points marked with “X” is called Slice.

FIGURE 2: System architecture.

FIGURE 3: Data structure of TP-Merkle tree with the height being four.

FIGURE 4: The slice of a leaf node with Index=3.

The slice contains the leaf node itself, as well as the hash values of all sibling nodes and parent nodes. When verifying the data, it is only necessary to calculate the slice from bottom to top. It is possible to determine whether the data are correct by comparing it with the previously recorded root hash. In other words, it is not necessary to store all the hash values, but only to get the slice where the data in question are located. It is possible to store millions of certificates on a TP-Merkle Tree. A certificate owner can confirm the location of its certificate through the Index Function. After the location is known, it is possible to obtain the corresponding slice for auditing.

For example, the height of a TP-Merkle tree is 9. The value of $\Gamma(\text{IndexValue}) = \text{SHA256}(\text{IndexValue}) \bmod 2^{9-1}$ is 96, where $\text{IndexValue} = "436b32a70e6fe9e9f5c6fa93908d2bc6f8f3ef50711b169c962d28c8fddbe78e235."$ Then the key-value pair $[\text{hash}(\text{IndexValue}), \text{hash}(\text{the binary value of certificate} + \text{status})]$ is stored in the leaf node with index being 96, where $\text{hash}(\text{IndexValue}) = "4d8bcd404ec930517e5e2cda4ea22ef1599697ac353a6a4af1532a356e42076"$ and $\text{hash}(\text{the binary value of certificate} + \text{status}) = "[B@5a39699c."$ Figure 5 shows Slice (96) of the TP-Merkle tree whose height is nine. We can derive the root hash of a TP-Merkle tree if we have one of the slices of it. This is because we have hash values of all the internal nodes from a leaf

FIGURE 5: A slice of a TP-Merkle tree that is nine nodes high.

node to the root node and all the children of these internal nodes in a slice of a TP-Merkle tree.

4.4. Protocols for the Request Phase. In the traditional PKI architecture, the owner of the web certificate cannot obtain effective evidence to supervise the CA. To solve this problem, we employ the idea from proof of violation (POV) for cloud storage systems [41]. POV schemes are solutions for obtaining mutual nonrepudiation between users and the service provider in the cloud. The relationship between cloud service providers and users is exactly the same as the relationship between CA and the owner of the web certificate, but the original POV protocol is not suitable for PK; we need to redefine the protocols. In this section, we define request protocols. A complete interactive process will generate two general messages: request (M_{request}) and reply (M_{reply}). The message formats are defined as follows:

$$M_{\text{request}} = \{[\text{Operation}, \text{Pub}(\text{owner}), \text{IndexValue}, \text{CO}]_{\text{Pri}(\text{owner})}\},$$

$$M_{\text{reply}} = \{[M_{\text{request}}, \text{result}, H(\text{Certificate}), \text{Status}]_{\text{Pri}(\text{CA})}\}, \quad (2)$$

where operation represents the service to be performed. $\text{Pub}(\text{owner})$ is the PKI public key of the owner. IndexValue is the certificate index value, which will be passed into the Γ function for establishing position. CO , which stands for clearance order, represents the clearing number on the TP-Merkle tree. It is one of the important sources of evidence for any appeal, which will be explained in detail in the following section. $H(\text{Certificate})$ is the hash value of the certificate, which represents the certificate to be used for the interaction. Status represents the status of the certificate. According to the owner's requirements for certificates, the basic message is expanded into three different request protocols: "apply certificate," "change status," and "replace key."

(i) Apply certificate: applying for a certificate involves the following steps.

- (1) The owner sends an apply certificate request $\text{ACM}_{\text{request}}$, where $\text{ACM}_{\text{request}} = \{[\text{Apply}, \text{Pub}(\text{owner}), \text{Web IP}, \text{IndexValue}, \text{CO}]_{\text{Pri}(\text{owner})}\}$ to CA, where Web IP is the IP address of the website on which the owner applies for the certificate.
- (2) The CA verifies the signature of $\text{ACM}_{\text{request}}$.
- (3) The CA checks whether the Web IP is registered; otherwise, it generates a new digital certificate.
- (4) The CA generates a key-value pair, where key-value pair = $\{[\text{IndexValue}, H(\text{Certificate}) | \text{Add}]_{\text{Pri}(\text{CA})}\}$, where IndexValue is the key, and the hash value and status of the certificate is the value. The key-value pair stores the leaf node, the location of which is calculated by Γ (IndexValue) function.
- (5) The CA generates and sends the corresponding $\text{ACM}_{\text{reply}}$, where $\text{ACM}_{\text{reply}} = \{[\text{ACM}_{\text{request}}, H(\text{Certificate}), \text{Add}]_{\text{Pri}(\text{CA})}\}$, and $\{[\text{Certificate}]_{\text{Pri}(\text{CA})}\}$ to the owner.

- (6) The owner verifies the signature of $\text{ACM}_{\text{reply}}$.
- (7) The owner saves $\text{ACM}_{\text{reply}}$ as evidence for subsequent appeals.

(ii) Change status: the change status protocol is used when the owner renews the certificate or wants to suspend or revoke the certificate. Note that to revoke the certificate, it is necessary to change the status to "Pause" first, which means that the owner must perform the change status request twice, for example, (Add $\rightarrow$ Pause $\rightarrow$ Revoked) or (Renew $\rightarrow$ Pause $\rightarrow$ Revoked). The change status protocol involves the following steps:

- (1) The owner sends a change status request $\text{CSM}_{\text{request}}$, where $\text{CSM}_{\text{request}} = \{[\text{Change}, \text{Status}, \text{Pub}(\text{owner}), H(\text{Certificate}), \text{IndexValue}, \text{CO}]_{\text{Pri}(\text{owner})}\}$ to CA.
- (2) The CA verifies the signature of $\text{CSM}_{\text{request}}$.
- (3) The CA calculates the location by IndexValue and then goes to the corresponding position on the TP-Merkle tree to check whether the certificate and status exist and are correct.
- (4) The CA changes the status of the certificate stored on the TP-Merkle tree according to $\text{CSM}_{\text{request}}$.
- (5) The CA generates and sends a corresponding $\text{CSM}_{\text{reply}}$, where $\text{CSM}_{\text{reply}} = \{[\text{CSM}_{\text{request}}, \text{Status}]_{\text{Pri}(\text{CA})}\}$, to the owner.
- (6) The owner verifies the signature of $\text{CSM}_{\text{reply}}$.
- (7) The owner saves $\text{CSM}_{\text{reply}}$ as evidence for subsequent appeals.

(iii) Replace key: if the owner suspects that the private key is leaked or has other security concerns, he/she can submit a request to the CA to replace the public key. The replace key protocol involves the following steps:

- (1) The owner sends a replace key request $\text{RKM}_{\text{request}}$, where $\text{RKM}_{\text{request}} = \{[[\text{Replace}, \text{Pub}(\text{owner_old}), \text{Pub}(\text{owner_new}), H(\text{Certificate_old}), \text{IndexValue}, \text{CO}]_{\text{Pri}(\text{owner_old})}]_{\text{Pri}(\text{owner_new})}\}$ to CA, where $\text{Pub}(\text{owner_old})$ is the original public key. $\text{Pub}(\text{owner_new})$ is the new public key. $\text{RKM}_{\text{request}}$ must be signed with both new and old public keys.
- (2) The CA verifies the signature of $\text{RKM}_{\text{request}}$.
- (3) The CA calculates the location by IndexValue and then goes to the corresponding position on the TP-Merkle tree to check whether the certificate and status exist and are correct.
- (4) The CA creates a new certificate ($\text{Certificate}_{\text{new}}$) using $\text{Pub}(\text{owner_new})$.
- (5) The CA updates the original certificate hash value and status on the TP-Merkle tree.
- (6) The CA generates and sends a corresponding $\text{RKM}_{\text{reply}}$, where $\text{RKM}_{\text{reply}} = \{[\text{RKM}_{\text{request}}, \text{Result}, H(\text{Certificate}_{\text{new}}), \text{Renew}]_{\text{Pri}(\text{CA})}\}$, and $\{[\text{Certificate}_{\text{new}}]_{\text{Pri}(\text{CA})}\}$ to the owner.
- (7) The owner verifies the signature of $\text{RKM}_{\text{reply}}$.

- (8) The owner saves RKM_{reply} as evidence for subsequent appeals.

4.5. Clearing Phase. In the request phase, the CA collects a lot of certificate information and stores the information on the TP-Merkle tree. At this time, the certificate information only exists in the CA's server. When the CA collects a certain number of certificates, or a period of time elapses, CA will calculate the root hash of the current TP-Merkle tree and upload the root hash to the smart contract. After that, the owner can request verification information, such as slice, from the CA for auditing. Finally, the CA will upload the entire tree to the public platform for everyone to download. The clearing process involves the following steps:

- (1) The CA calculates the root hash of the current TP-Merkle tree.
- (2) Upload CO and root hash to contract.
- (3) The Owner can use its IndexValue and CO to request the slice and list of key-value pairs from CA. The request message is $\{[CO, Slice, List\ of\ key-value\ pairs]_{Pri(CA)}\}$
- (4) The CA announces the entire TP-Merkle tree to the interplanetary file system (IPFS) [42]. The IPFS is a distributed system for storing and accessing files, websites, applications, and data. It is used as a bulletin board because it is a P2P structure, so there is no need to worry about data loss, and it can be easily downloaded by anyone. It can be replaced by other P2P file systems, such as BitTorrent [43].

There is no mandatory requirement for a time interval to perform clearing. Considering the timeliness of certificate verification, clearing is performed once a day. After clearing, the original TP-Merkle tree will not be cleared, and leaf nodes on the tree will continue to be added or modified. This means that once the certificate is uploaded, it will always exist, the status will only be changed upon the request of the owner. Therefore, the root hash obtained by clearing daily and the TP-Merkle tree obtained from the announcement can be regarded as the daily snapshot of the TP-Merkle tree. The owner can confirm the status of the tree in different periods checking the CO and the root hash. It also means that the owner can appeal to the CA in any time.

4.6. Auditing Phase. In the request and clear phases, the owner obtains $M_{request}$, a list of key-value pairs, and a slice, which together are called cryptographic evidence. The owner can check the correctness of these pieces of evidence to determine whether the CA operations are correct. The detailed steps are as follows:

- (1) The owner compresses the CO and root hash of the slice given by the CA with those on the contract to confirm that the slice belongs to the TP-Merkle tree and they are from the same day.
- (2) The owner computes the slice with the list of key-value pairs and then checks whether it is equal to the

root hash. If the first two steps do not reveal any error, it means that the TP-Merkle tree is correct and the owner can trust the status of the certificate stored on it.

- (3) The owner checks whether the certificate's status in the list of key-value pairs is consistent with that recorded in M_{reply} .

The owner can find errors through auditing, which can be divided into two categories. The first is cryptographic evidence errors; the second is process errors in the CA's operation. There are many different errors under each category, as shown in Figure 6.

Most auditing errors can be appealed, but there is no way to appeal errors in IPFS and contract. In the case of "CA does not upload evidence," when the CA uploads evidence to IPFS and the contract, it cannot obtain cryptographic evidence by a specific communication protocol. As for "CA upload error," only IPFS upload errors cannot be appealed. If the root hash on the contract is wrong, the owner can prove it using the slice. However, if the owner wants to verify the entire TP-Merkle tree on IPFS, it is necessary to calculate all the leaf nodes. Since the contract has a limitation on the amount of uploaded data, the owner cannot pass the entire wrong tree into the contract for verification.

Fortunately, we can assume that the two situations mentioned above will not likely occur. Although the owner cannot appeal, they can still know whether the CA has an error. Therefore, regardless of whether the CA has uploaded errors or has not uploaded errors, the CA will lose the trust of the owner, which will cause subsequent business problems. In addition, users can also sign relevant contracts in real life to restrict the CA through real laws.

4.7. Appeal Phase. This section will explain in detail the appeal process for any errors, including what evidence the owner must upload and how the contract is judged:

- (i) The certificate disappears from the TP-Merkle tree: in the third step of the auditing phase, when the owner checks the certificate, they cannot find the certificate data, which means that the key-value pair is missing. There are two possibilities; the first is that the CA did not upload it to the TP-Merkle tree when applying, and the other is that the CA accidentally deleted the key-value pair at some time. The appeal steps are as follows:
 - (1) The owner sends $\{ACM_{reply}, slice, list\ of\ key-value\ pairs\}$ to the contract.
 - (2) The contract verifies whether the CA's signature on ACM_{reply} is valid.
 - (3) The contract checks whether the CO and root hash are the same as those stored on the contract.
 - (4) The contract computes the slice with the list of key-value pairs and then checks whether it is equal to the root hash.

FIGURE 6: The errors of auditing.

- (5) The contract checks whether the certificate stored in ACM_{reply} also appears in the list of key-value pairs.
 - (6) If the certificate is not in the list of key-value pairs, the contract adjudges that the CA lost the certificate. In such a case, the contract will transfer cryptocurrency to the owner as indemnification.
- (ii) The CA changes the incorrect certificate's status: the owner sends $CSM_{request}$ to the CA, whereupon the CA agrees to the request and returns CSM_{reply} . In the third step of the auditing phase, if the owner checks the certificate and finds that the status of the certificate is different between the key-value pair and in $CSM_{request}$, the appeal steps are as follows:
- (1) The owner sends $\{CSM_{reply}, \text{slice, list of key-value pairs}\}$ to the contract.
 - (2) The contract verifies whether the CA's signature on CSM_{reply} is valid.
 - (3) The contract checks whether the CO and root hash are the same as those stored on the contract.
 - (4) The contract computes the slice with the list of key-value pairs and then checks whether it is equal to the root hash.
 - (5) The contract checks whether the status of the certificate stored in the list of key-value pairs is the same as the one stored in CSM_{reply} .
 - (6) If the result of step (5) is negative, then the contract adjudges that the CA updated the incorrect certificate's status. In such a case, the contract will transfer cryptocurrency to the owner as indemnification.
- (iii) CA revokes a certificate error: this means the CA did not follow the normal procedures for revoking the certificate (Pause first, Revoked). If the owner finds that the certificate's status changed directly from "Add" or "Renew" to "Revoked" during the auditing phase, the owner can appeal based on the evidence of the two slices with the connected the CO. The appeal steps are as follows:
- (1) The owner sends $\{\text{two slices, list of key-value pairs}\}$ to the contract.
 - (2) The contract checks whether the COs of the two slices are connected.
 - (3) The contract checks whether the two pairs of CO and root hash are consistent with those stored on the contract.
 - (4) The contract computes the slices with the list of key-value pairs and then checks whether they are equal to the root hash.
 - (5) The contract compresses the status of the certificates stored in the two lists of key-value pairs and checks whether the statuses are "Pause" and "Revoked", respectively.
 - (6) If the result of step (5) is negative, then the contract adjudges that the CA did not follow the normal procedures to revoke the certificate. In such cases, the contract will transfer cryptocurrency to the owner as indemnification.
- (iv) Data in M_{reply} are incorrect: this refers to a situation where the owner receives any kind of M_{reply} , but the data in M_{reply} are not the same as the data in $M_{request}$. The owner can appeal based on the evidence of the difference between $M_{request}$ and M_{reply} , and the appeal steps are as follows:
- (1) The owner sends $\{M_{reply}, \text{slice, list of key-value pairs}\}$ to the contract.
 - (2) The contract verifies whether CA's signature on M_{reply} is valid.
 - (3) The contract checks whether the value of the result field in M_{reply} is "Accept."
 - (4) The contract compresses and checks whether the statuses in M_{reply} and $M_{request}$ are the same.
 - (5) If the result of step (4) is negative, then the contract adjudges that the M_{reply} from CA is

incorrect. The contract will transfer cryptocurrency to the owner as indemnification.

- (v) CA uploads incorrect evidence: the root hash of the slice obtained by the owner is inconsistent with that on the contract. There are two possibilities. The first is that the CA uploaded the wrong root hash to the contract, and the second is that the slice sent by the CA was incorrect. The reasons in both cases are grounds for appeal because only the CA can deploy the contract and the slice is signed by the CA. Therefore, if the data are not synchronized, we can reasonably infer that it must be the CA's own operating error. The owner can appeal based on the evidence of the slice, and the appeal steps are as follows:

- (1) The owner sends {slice, list of key-value pairs} to the contract.
- (2) The contract verifies whether the signatures on slice and list of key-value pairs are valid.
- (3) The contract checks whether CO and root hash are the same as those stored on the contract.
- (4) If the result of step (3) is negative, then the contract adjudges that the CA uploaded incorrect evidence. The contract will transfer cryptocurrency to the owner as indemnification.

4.8. Verification Phase. We designed a simpler verification method from the user's perspective, whereby the user gets the certificate verification information provided by the owner instead of the CA, which is impossible with the current PKI architecture. The proposed mechanism uploads the evidence to the contract, so that the user has the ability to identify whether the owner has forged information. Even if the information provided by the owner is incorrect, the user can directly treat the certificate as invalid or request the correct certificate information from the CA. The verification steps are as follows:

- (1) The owner obtains the slice and list of key-value pairs from CA in the daily clearing phase
- (2) The owner announces the verification information and certificate on the website
- (3) When the user visits the website, he/she will first check whether the root hash on the contract is the same as that on the website
- (4) The user calculates and checks whether the slice and list of key-value pairs are correct
- (5) The user verifies whether the certificate status is valid

This hierarchical verification architecture has many advantages. In Figure 7, we see that the CA does not need to handle a large number of certificate status requests from users. It only needs to process the requests from the second-tier owners, which can greatly reduce the burden on the server. The following analysis is to illustrate the difference between the traditional PKI architecture and the proposed PKI mechanism.

The burden of the traditional CA when verifying certificates is as follows:

$$\text{burden}_{\text{CA}} = N * S + M, \quad (3)$$

where N is the number of users, S is the number of times users visit the web page in one day, and M is the number of owners.

The burden of the proposed PKI mechanism when verifying certificates is as follows:

$$\text{burden}_{\text{CA}} = M + 1 + K, \quad (4)$$

where M is the number of owners, 1 represents the number of times that the CA uploads evidence, K represents the number of times that the CA provides filing support when owners give incorrect information to users.

This verification method is advantageous for the CA. As for the owner, it can immediately find out whether the CA has an operating error by requesting the daily slices. The user can also use different channels for certificate verification.

5. Experiment Results

We conducted a series of experiments to demonstrate the feasibility of the proposed PKI mechanism. First, we measured the number of collisions on the TP-Merkle tree for performing the index function and measured the storage required for the owner and CA according to TP-Merkle trees of different heights. Second, we conducted experiments to measure gas use associated with deploying the contract and the appeal process. Third, we measured the time required to generate a TP-Merkle tree and the time required to extract slices from a TP-Merkle tree.

First, for measuring the number of collisions on the TP-Merkle trees, we count collisions on TP-Merkle trees with different heights. The machine used in Tables 1 and 2 has four processors (Intel® Core™ i5-4570@3.20 GHz, 16 GB Memory, Windows 10) and is located in our university. A TP-Merkle Tree can compress data. According to Table 1, with different tree heights, the numbers of key-value pairs stored in each leaf node are also different. This means that the slice sizes extracted from trees of different heights are significantly different, which will affect the size of the data that owners and users need to request when verifying certificates. Since the Index Function is used to locate the position of each leaf node, we can analyze the number of key-value pairs according to the collisions of the Index Function with TP-Merkle trees of different heights.

We use the writeObject method of java.io.ObjectOutputStream in Java to output the TP-Merkle Tree into an object file and store it in the CA server. The CA stores the entire TP-Merkle Tree, and the client only needs to store the slice and list of key-value pairs. According to Table 2, we can easily observe that although there are few leaf nodes, the list of key-value pairs is huge when the tree height is less than or equal to 14, which indicates that the CA needs less storages and the client needs more storage. The situation is the opposite when the tree height is greater than 14. Considering the cost differential for storage space between

FIGURE 7: The hierarchical verification architecture.

TABLE 1: The number of collisions on TP-Merkle trees with different heights.

Height	Number of leaf nodes	Average number of collisions	The smallest number of collisions	The largest number of collisions
1	1	1,000,000.000	1,000,000	1,000,000
4	8	125,000.000	124,395	125,439
6	32	31,250.000	30,955	31,656
8	128	7,813.000	7,582	7,983
10	512	1,951.658	1,812	2,075
12	2,048	487.951	403	559
14	8,192	121.890	84	164
16	32,768	30.405	9	53
18	131,072	7.754	0	24
20	524,288	2.350	0	11
21	1,048,576	1.462	0	8

TABLE 2: Storage required for the owner and CA according to TP-Merkle trees of different heights.

Height	Client	Server (MB)
4	7.629 MB	136
8	488.786 kB	136
10	122.533 kB	136
12	31.760 kB	136
14	8.160 kB	137
18	2.660 kB	164
20	2.540 kB	251
21	2.670 kB	367

the CA and client, the tree with height 21 is the most appropriate.

In the second part of our experiment, we developed and deployed an Ethereum smart contract written in the Solidity programming language [44] to Ropsten Testnet [45] to measure how much gas the CA and the owner need to pay when deploying a contract and performing basic and appeal functions on the contract. When deploying smart contracts and performing functions, the entities involved need to pay. The CA is only responsible for designing and deploying

contracts. After deployment, miners are responsible for execution, and the CA does not need to pay additional costs. The owner just needs to pay for miners when performing the appeal functions. Tables 3 and 4 show the gas use associated with deploying the contract and performing its related functions.

From the CA's perspective, deploying a contract costs about \$11.55, but it only needs to be deployed one time, and uploading root hash every day only costs about \$ 0.22. The total cost for a year is therefore about \$ 81, which is a very small amount for an enterprise. From the owner's perspective, if there is no problem with the certificate, there is no need to appeal at all. Even if an appeal is really needed, it only costs about \$1.42, but the indemnification for a successful appeal would definitely be greater than the handling fee.

In the third part of our experiment, the time required to generate a TP-Merkle tree and the time required to extract slices from a TP-Merkle tree are shown in Table 5, Figure 8, Table 6, and Figure 9. The machine used in Tables 5 and 6 and Figures 8 and 9 has four processors (Intel® Core™ i5-8250@1.60 GHz, 8 GB Memory, Windows 10). For a TP-Merkle tree which contains less than 100,000 digital

TABLE 3: Gas use associated with deploying the appeal contract and performing its basic functions (we assume that the price of Ether is \$200).

Function	Gas use	Gas price	ETH	USD
Deploy	2,310,253	Fast (25 Gwei)	0.057756325	11.55
		Slow (15 Gwei)	0.034653795	6.93
Upload root hash	44,029	Fast (25 Gwei)	0.001100725	0.22
		Slow (15 Gwei)	0.000660435	0.13
Verify slice	138,582	Fast (25 Gwei)	0.003464550	0.69
		Slow (15 Gwei)	0.002078730	0.41
Verify k-v pairs	28,564	Fast (25 Gwei)	0.000714100	0.14
		Slow (15 Gwei)	0.000428460	0.08
Verify request	39,143	Fast (25 Gwei)	0.000978575	0.19
		Slow (15 Gwei)	0.000587145	0.11
Verify reply	38,173	Fast (25 Gwei)	0.003817325	0.19
		Slow (15 Gwei)	0.000572595	0.11

Wei is the smallest denomination of Ether. 1 Wei = 10^{-18} Ether; 1 Gwei = 10^{-9} Ether.

TABLE 4: Gas use associated with performing appeal functions (we assume that the price of Ether price is \$200).

Objection	Gas use	Gas price	ETH	USD
The certificate disappears from the TP-Merkle tree	251,634	Fast (25 Gwei)	0.00629085	1.25
		Slow (15 Gwei)	0.00377451	0.75
CA changes an incorrect certificate's status	250,552	Fast (25 Gwei)	0.00626380	1.25
		Slow (15 Gwei)	0.00375828	0.75
CA revokes a certificate error	285,922	Fast (25 Gwei)	0.00714805	1.42
		Slow (15 Gwei)	0.00428883	0.85
CA uploads incorrect evidence	147,466	Fast (25 Gwei)	0.00368665	0.73
		Slow (15 Gwei)	0.00221199	0.44
Data in M_{reply} are incorrect	59,783	Fast (25 Gwei)	0.00149458	0.29
		Slow (15 Gwei)	0.00089675	0.17

Wei is the smallest denomination of Ether. 1 Wei = 10^{-18} Ether; 1 Gwei = 10^{-9} Ether.

TABLE 5: The time required to generate a TP-Merkle tree that is 21 nodes high.

# of digital certificates	Average time (ms)
1000	1455.7
10,000	1503.7
100,000	1963.9
1,000,000	6705.2

FIGURE 8: The time required to generate a TP-Merkle tree that is 21 nodes high with different numbers of digital certificates.

certificates, we only require about 1.96 s for generation. We store different numbers of digital certificates with different heights. We only needed less than one millisecond to extract a slice even when a TP-Merkle tree contained one million digital certificates.

According to related work, we compared the proposed scheme with other blockchain-based PKI schemes shown in Table 7. In our scheme, all certificates are stored securely in TP-Merkle trees and CA announces the entire TP-Merkle trees to IPFS, which is the operation of Step (4) of clearing phase in Section 4.5. CA will calculate the root hash (cryptographic evidence) of the current TP-Merkle tree and upload root hash to the smart contract. Because a TP-Merkle tree can compress one million certificates into a 32-byte data type, our scheme is better than other schemes in terms of space and communication cost.

TABLE 6: The time required to extract slices from TP-Merkle trees.

# of digital certificates	Height of the TP-Merkle tree	Average time (ms)
1000	10	0.72
10,000	15	0.78
100,000	18	0.44
1,000,000	21	0.76

FIGURE 9: The time required to extract slices from a TP-Merkle tree that is 21 nodes high with different numbers of digital certificates.

TABLE 7: Comparison of the proposed scheme and other blockchain-based PKI.

Blockchain-based PKI	All certificates stored in blockchains	Cryptographic proof of all certificates stored in blockchains	Certificates stored in IPFS	Avoid single-point failure	Real time	High scalability
Ours		V	V	V	V	V
[8, 10–13]	V			V		

6. Conclusion

In the past, blockchain-based PKI was not feasible because the high-trust public blockchain could not cope with the issuance of a large number of certificates, and the use of private blockchains could not satisfy the need for certificate security. We overcome this shortcoming by using the data structure of the TP-Merkle tree to enable the public blockchain-based PKI to be able to handle the issuance and revocation of all certificates. Moreover, when the CA uploads the root hash to the contract, the certificate information will be hashed, which can fully comply with the GDPR requirement for pseudonymization. A deposit system is established through smart contracts with TP-Merkle trees to generate cryptographic evidence and establish a fully-automated indemnification system. In addition to protecting the rights and interests of the certificate owner, our scheme can also reduce the probability of CA errors. Finally, we make the smart contract as truly a trusted third party. Web users are no longer limited to obtaining certificate revocation information from the CA, and the CA can directly reduce the demand on it from user connections.

In future work, we are extending the proposed scheme to implement any kind of certificate-issuing system, such as diploma certification for education institutions and professional certificates. The limitation of the proposed scheme that it is based on the public blockchain which will be limited to a platform for mining. This problem will be solved when

Ethereum completes the transition to Proof of Stake (PoS) from the Proof of Work (PoW) system with 2021.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Ministry of Science and Technology, Taiwan (Grant no. MOST 108-2221-E-003-004, 109-2221-E-003-013, and 110-2221-E-259-004).

References

- [1] Network Working Group, "RFC 4949 (Internet security glossary, version 2)," 2007, <https://www.rfc-editor.org/rfc/pdf/rfc4949.txt.pdf>.
- [2] Network Working Group, "RFC 5280. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," 2008, <https://datatracker.ietf.org/doc/html/rfc5280>.
- [3] Network Working Group, "RFC-6960, june 2013. X.509 Internet public key infrastructure online certificate status protocol-OCSP," 2013, <https://datatracker.ietf.org/doc/html/rfc6960>.

- [4] C. Ellison and B. Schneier, "Ten risks of PKI: what you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000.
- [5] R. Z. Philip, *The Official PGP User's Guide*, MIT Press, Cambridge, MA, USA, 1996.
- [6] M. Swan, *Blockchain: Blueprint for a New Economy*, O'Reilly, Sebastopol, CA, USA, 2015.
- [7] S. Nakamoto, *Bitcoin: A Peer-To-Peer Electronic Cash System*, 2008, <https://bitcoin.org/bitcoin.pdf>.
- [8] M. Al-Bassam, "SCPki: a smart contract-based PKI and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, Abu Dhabi, United Arab Emirates, April 2017.
- [9] K. Lewison and F. Corella, "Backing rich credentials with a blockchain PKI," Tech Report, 2016.
- [10] A. Garba, Q. Hu, Z. Chen, and M. R. Asghar, "BB-PKI: blockchain-based public key infrastructure certificate management," in *Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Yanuca Island, Cuvu, Fiji, December 2020.
- [11] Y. Yu, Y. Wang, Y. Hu et al., "Blockchain-based PKI system and its application in Internet of Things," in *Proceedings of the 2020 IEEE 4th Conference on Energy Internet and Energy System Integration (EI2)*, Wuhan, China, November 2020.
- [12] V. Osmov, A. Kurbanniyazov, R. Hussain, A. Oracevic, S. M. A. Kazmi, and F. Hussain, "On the blockchain-based general-purpose public key infrastructure," in *Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–8, Abu Dhabi, United Arab Emirates, November 2019.
- [13] E. F. Kfoury, D. Khoury, A. AlSabeh, J. Gomez, J. Crichigno, and E. Bou-Harb, "A blockchain-based method for decentralizing the ACME protocol to enhance trust in PKI," in *Proceedings of the 2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 461–465, Milan, Italy, July 2020.
- [14] Ethereum, "The Ethereum blockchain explorer," 2021, <https://etherscan.io/>.
- [15] Proof of Stake (PoS), <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/>.
- [16] D. Siegel, "Understanding the DAO attack-CoinDesk," 2016, <https://www.coindesk.com/understanding-dao-hack-journalists>.
- [17] Nicola Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts," in *Proceedings of the 6th International Conference on Principles of Security and Trust*, pp. 164–186, New York, NY, USA, April 2017.
- [18] A. Groce, J. Feist, G. Grieco, and M. Colburn, "What are the actual flaws in important smart contracts (and how can we find them)?," 2019, <https://arxiv.org/abs/1911.07567>.
- [19] M. Alexander and M. Flatscher, "Security vulnerabilities in Ethereum smart contracts," in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services (iiWAS2018)*, pp. 375–380, Halifax, NS, Canada, August 2018.
- [20] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: attacks and protections," *IEEE Access*, vol. 8, pp. 24416–24427, 2020.
- [21] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential risks of Hyperledger Fabric smart contracts," in *Proceedings of the 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pp. 1–10, Hangzhou, China, February 2019.
- [22] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on Ethereum systems security," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–43, 2020.
- [23] T. Chen, X. Zhang, Z. Li et al., "A large-scale empirical study on control flow Identification of smart contracts," in *Proceedings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–11, Porto de Galinhas, Brazil, September 2019.
- [24] M. di Angelo and G. Salzer, "A survey of tools for analyzing Ethereum smart contracts," in *Proceedings of the 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pp. 69–78, Newark, CA, USA, April 2019.
- [25] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 530–541, New York, NY, USA, June 2020.
- [26] X. Feng, Q. Wang, X. Zhu, and S. Wen, "Bug searching in smart contract," 2019, <https://arxiv.org/abs/1905.00799>.
- [27] Y. Murray and D. A. Anisi, "Survey of formal verification methods for smart contracts on blockchain," in *Proceedings of the 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–6, Canary Islands, Spain, June 2019.
- [28] P. Praitheshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security analysis methods on Ethereum smart contract vulnerabilities: a survey," 2019, <https://arxiv.org/abs/1908.08605>.
- [29] J. Ye, M. Ma, T. Peng, Y. Peng, and Y. Xue, "Towards automated generation of bug benchmark for smart contracts," in *Proceedings of the 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 184–187, Xi'an, China, April 2019.
- [30] M. Zunino and Z. Roberto, "Formal models of Bitcoin contracts: a survey," *Frontiers in Blockchain*, vol. 2, 2019.
- [31] A. Imeri, N. Agoulmine, and D. Khadraoui, "Smart contract modeling and verification techniques: a survey," in *Proceedings of the 8th International Workshop on ADVANCES in ICT Infrastructures and Services (ADVANCE 2020)*, pp. 1–8, Cancun, Mexico, March 2020.
- [32] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: approaches and challenges to address vulnerabilities," *Computers and Security*, vol. 88, Article ID 101654, 2020.
- [33] D. Harz and W. Knottenbelt, "Towards safer smart contracts: a survey of languages and verification methods," 2018, <https://arxiv.org/abs/1809.09805>.
- [34] R. M. Parizi, A. Amritraj, and A. Dehghantanha, "Smart contract programming languages on blockchains: an empirical evaluation of usability and security," *Lecture Notes in Computer Science*, Springer, Cham, Switzerland, pp. 75–91, 2018.
- [35] PCWorld, "Comodo hacker claims credit for DigiNotar attack," 2011, <https://www.pcworld.idg.com.au/article/399812/>.
- [36] "Hashed out," 58% of Phishing Websites Now Use, <https://www.thesslstore.com/blog/58-of-phishing-websites-now-use-https/>, 2019.
- [37] Let's Encrypt stats, 2021, <https://letsencrypt.org/stats/>.
- [38] Eur-Lex, "Regulation (EU) 2016/679 of the European parliament and of the council," 2016, Council, 2016, <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.

- [39] G. H. Hwang, Po-H. Chen, C.-H. Lu, C. Chiu, H. C. Lin, and A. J. Jheng, "InfiniteChain: a multi-chain architecture with distributed auditing of sidechains for public blockchains," in *Proceedings of the International Conference on Blockchain*, pp. 47–60, Seattle, WA, USA, June 2018.
- [40] R. Merkle, *Secrecy, Authentication, and Public Key Systems. Electrical Engineering*, PhD Thesis, Stanford University, Stanford, California, 1979.
- [41] G.-H. Hwang, W.-S. Huang, and J. Z. Peng, "Real-time proof of violation for cloud storage," in *Proceedings of the 2014 6th IEEE International Conference on Cloud Computing Technology and Science*, pp. 394–399, Singapore, December 2018.
- [42] J. Benet, "IPFS-content addressed, versioned, P2P file system," 2014, <https://arxiv.org/abs/1407.3561>.
- [43] BitTorrent, 2021, <https://www.bittorrent.com/>.
- [44] Solidity, 2021, <https://docs.soliditylang.org/en/v0.7.5/>.
- [45] Ropsten Testnet, 2021, <https://ropsten.etherscan.io/>.