PISTIS: Issuing Trusted and Authorized Certificates with Distributed Ledger and TEE

Zecheng Li, Haotian Wu, Lap Hou Lao, Songtao Guo, *Senior Member, IEEE,* Yuanyuan Yang, *Fellow, IEEE,* and Bin Xiao, *Senior Member, IEEE*

Abstract—The security of HTTPS fundamentally relies on SSL/TLS certificates issued by Certificate Authorities (CAs), which, however, are vulnerable to be compromised to issue unauthorized certificates (i.e., certificates issued without domains' permission). Current countermeasures such as Certificate Transparency (CT) can only detect unauthorized certificates rather than preventing them. In this paper, we present PISTIS, a framework for issuing authorized and trusted certificates with the distributed ledger and Trusted Execution Environment (TEE) technology. In PISTIS, TEE nodes validate whether the domain in a requested certificate passes the domain ownership validation (i.e., under corresponding applicants' control) and submit attested results to a smart contract in the distributed ledger. The smart contract issues a certificate to the applicant when an attested result shows a pass. Therefore, PISTIS can ensure its issued certificates are authorized due to the domain ownership validation mechanism in the TEE. Furthermore, as the issued certificates are stored in a Merkle Patricia Tree (MPT) in PISTIS, they are trusted and can be verified by a normal user easily. The security of PISTIS is formally proved in the Universally Composable (UC) framework. Compared with state-of-the-art, PISTIS avoids potential damages by preventing unauthorized certificates from issuing.

Index Terms—distributed ledger, blockchain, smart contract, trusted execution environment (TEE), certificate issuance.

1 INTRODUCTION

SSL/TLS certificates issued by Certificate Authorities (CAs) form the security foundation of HTTPS connection. Clients establish HTTPS connections with a website only when its server provides a valid certificate to prove its identity. However, current CAs are vulnerable to be compromised to issue *unauthorized certificates*, which arises from the fact that a compromised CA might issue certificates without domains' permission. As an example, in 2011, attackers compromised the private key of DigiNotar and maliciously issued unauthorized certificates for Google, which were used to launched Man-in-the-Middle (MitM) attacks against Google services [1]. Similar incidents happened dozens of times [2], [3].

To mitigate this problem, some countermeasures such as HTTP Public Key Pinning (HPKP) [4] and Certificate Transparency (CT) [5] have been proposed. HPKP is a straightforward way through which a server provides a list of trusted public keys to clients. CT builds a log system to monitor the operation of certificates, which enables clients to detect unauthorized certificates. In addition, researchers proposed some other countermeasures, such as Accountable Key Infrastructure (AKI) [6] to disperse centralized trust, Attack Resilient Public Key Infrastructure (ARPKI) [7] & PoliCert [8] to log certificate operations, and Certificate Issuance and Revocation Transparency (CIRT) [9] to provide an efficient certificate verification method.

However, these countermeasures have some common limitations. What they can do is to detect unauthorized certificates or reduce the probability of unauthorized certificate problems. In most cases, attacks have already caused damage before unauthorized certificates are detected. In addition, when an unauthorized certificate is reported, browser vendors typically add the corresponding CA to the blacklist. However, some compromised CAs are too big to fail. For example, although Symantec was reported to issue unauthorized certificates to `Google.com` in 2015 [10], it was unrealistic to block Symantec immediately because it had controlled more than 10% of active certificates by that time. Blacklisting Symantec will block millions of websites at the same time.

We observe that three reasons account for the problem of unauthorized certificates: 1. Centralized CAs might be compromised; 2. Centralized CAs can issue certificates without domains' permission (i.e., unauthorized); 3. Centralized CAs' operations are opaque. Furthermore, we find that people typically trust a CA based on its identity. However, this trust relationship is fragile especially when the trusted CA might be compromised. In this case, we consider addressing this issue through a complete certificate issuance process design, which restricts only the domain owner can apply for a certificate related to that domain. A certificate is issued only when its applicant passes domain validation. We also ensure that CAs are hardly compromised. The advent of distributed ledger (i.e., blockchain) makes our idea feasible.

Blockchain reproduced smart contract, whose execution is immutable, transparent, and deterministic. Considering building a CA based on smart contract, we find that: 1.

- *Z. Li, H. Wu, L. Lao, and B. Xiao are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: {cszcli,cshtwu,cslhlao,csbxiao}@comp.polyu.edu.hk*
- *S. Guo is with the College of Computer Science, Chongqing University, China. E-mail: guosongtao@cqu.edu.cn*
- *Y. Yang is with the Department of Computer Science, Stony Brook University, NY, USA. Email: yuanyuan.yang@stonybrook.edu*

Manuscript received April XX, 20XX; revised August XX, 20XX. (Corresponding author: B. Xiao.)

the immutable feature makes it difficult for an attacker to compromise a CA based on smart contract, which eliminates the problem of traditional CAs being manipulated in the event of key breaches; 2. the transparent feature let people trust a CA based on its logic rather than its identity, and the blockchain becomes the log of certificates' operation history naturally; 3. the deterministic feature allows CA developers to implement autonomous and complete CA logic through sound contract design. In PISTIS, we hard coded that each certificate applicant should prove the ownership of the domain related to its requested certificate to the smart contract before certificate issuance. This process is called domain (ownership) validation. However, it is challenging to conduct domain validation in smart contracts solely.

Smart contracts work on overlay networks (i.e., blockchain P2P network), which are only accessed by blockchain nodes, and cannot communicate with domain servers directly. In this case, traditional domain validation methods via DNS or email cannot work. It is also infeasible for external validators to pass validation results to a smart contract directly. Due to the public nature of blockchain, anyone can pass information to smart contracts so that a smart contract cannot tell the authenticity of provided validation results. In this case, we need a mechanism that can conduct domain ownership validation and transfer domain validation results to smart contracts in a trusted way. Inspired by Town Crier [11], which employs TEE as smart contract's trusted information source, we intend to address this challenge with the TEE technology.

A TEE provides a fully isolated environment that prevents other software applications, operating system, and host from learning or tampering with the internal data and code of applications (i.e., enclaves) running inside the TEE. The attested execution model of TEE enables others to check whether an attested result is outputted by an expected program. We leverage TEE to validate whether a domain is under the control of its corresponding certificate applicant. Once the domain related to a requested certificate passes ownership validation, TEE nodes pass validation results to the smart contract. Then, after verifying the attested validation results, the smart contract can issue a certificate as request to the applicant.

In this paper, we present PISTIS[1], a framework for issuing authorized and trusted certificates with distributed ledger and TEE. PISTIS sets distributed ledger as the root of trust. Clients can decide whether to trust a smart contract on the distributed ledger based on its logic. PISTIS employs two contracts: authority contract $\mathcal{C}_{\text{AC}}$ and verification contract $\mathcal{C}_{\text{VC}}$. $\mathcal{C}_{\text{AC}}$ is for certificate issuance and revocation, and $\mathcal{C}_{\text{VC}}$ is for certificate validity verification.

PISTIS is designed to be TEE-agnostic (i.e., the underlying TEE implementation is changeable and upgradeable). PISTIS combines any desired underlying blockchain system with TEE-based execution. Anchored in a formal security model expressed as a cryptographic ideal functionality, PISTIS's design supports rigorous analysis of its security properties. The main challenges of this solution, such as TEE failures and DNS failures are discussed in Section 3.1.

1. In Greek mythology, PISTIS was the personification of trust and reliability.

Our main contributions are summarized as follows:

- **Authorized and trusted certificate issuance:** PISTIS can ensure all its issued certificates are authorized and trusted. The issuance is authorized because PISTIS leverages TEE nodes to validate the actual control of registrants over domains related to requested certificates. The attested validation results provided by TEE nodes allow the PISTIS smart contract to issue certificates to registrants who have passed the domain validation. As these PISTIS-issued certificates are recorded on the blockchain, they are trusted. Details are discussed in Section 4.3.
- **Efficient certificate verification:** PISTIS maintains a Merkle Patricia Tree (MPT), which stores latest certificates' states, to enable efficient certificate verification. In this tree, the key is a Fully Qualified Domain Name (FQDN), and the value field stores the state of the corresponding certificate. For the client, PISTIS provides web3.js scripts, which are executed in browsers and can verify the validity of PISTIS-issued certificates. Detailed information are discussed in Section 4.3.
- **Formal modeling and security analysis:** We formally model the PISTIS protocol as $\mathbf{Prot}_{\text{PISTIS}}$ in Section 4 and give its ideal functionality $\mathcal{F}_{\text{PISTIS}}$ in Section 5.1. We prove the security of PISTIS in the Universally Composable (UC) framework [12] in Section 5.2. By showing that $\mathbf{Prot}_{\text{PISTIS}}$ UC-realizes the ideal functionality $\mathcal{F}_{\text{PISTIS}}$, PISTIS can be seamlessly integrated in more complex cryptographic protocols such as HTTPS.
- **Implementation and evaluation:** We implemented a prototype of PISTIS based on Ethereum and Intel SGX, which could issue X.509 certificates according to the specification. We tested the gas consumption, storage overhead, and verification latency of PISTIS. The experiment results demonstrate that PISTIS can provide an efficient certificate verification service to end users.

2 BACKGROUND

In this section, we introduce the background knowledge and some commonly-used terms in this paper.

CA. The Certificate Authority (CA) is an organization that issues digital certificates, whose purpose is to prove that the entity and public key listed in a certificate are bound. The CA signs the certificates it issues, which prevents attackers from forging or tampering with these certificates. Users can determine the authenticity of a certificate by verifying its CA's signature. In this case, people trust a CA and then use the certificates it issues.

SSL/TLS. Since the HTTP protocol originally used on the Internet was plain text, transmission content would be sniffed and tampered. The SSL/TLS protocol was designed to preserve the integrity and confidentiality by encrypting data.

TEE. A Trusted Execution Environment (TEE) isolates a preserved memory and conducts attested execution via dedicated CPU instructions. TEE guarantees the confidentiality and integrity of stored code and data. Popular TEE

implementations include Intel SGX, ARM TrustZone, and AMD Secure Execution Environment.

Intel SGX. Intel Software Guard Extensions (SGX) protects selected code and data from disclosure and modification. Developers can divide an application into a CPU-enhanced *enclave* and a host application that manages it, which can improve security even in attacked platforms. Benefiting from TEE, developers can enable identity and record privacy, secure browsing and digital rights management protection, or any high-security application scenario that requires secure storage of confidential or protected data.

Ethereum. Ethereum is recognized as the second-generation blockchain system. From the science perspective, it is a globally accessible state machine with a built-in virtual machine that allows users to change its global state. From the engineering perspective, it is a distributed ledger with built-in smart contract mechanism that allows users to develop applications atop it. In Ethereum, there are two kinds of accounts: Externally Owned Accounts (EOA) and contract accounts. EOAs are controlled by end users with a unique key pair and blockchain address. By contrast, contract accounts are empowered by contract programs, account balances and persistent storage in the form of key/value pairs. Ethereum has its built-in currency called ether.

Smart Contract. The smart contract can be recognized as a constant program that runs deterministically atop Ethereum. Smart contracts are typically created by EOAs in the form of contract creation transactions. An EOA can invoke a smart contract by sending a transaction with specified function names and parameters to the contract's address. Compared with Bitcoin scripting system, Ethereum smart contract languages (i.e., Solidity, Vyper) is Turing-complete. Ethereum introduced gas to measure the computational overhead of smart contract execution. The invoking entity should pay for the consumed gas in ether, which is calculated by multiplying *gasprice* and the number of gas used. Ethereum transactions set limit on gas with two parameters: *gasprice* and *gaslimit*. The *gasprice* indicates how expensive the user is willing to pay for each gas. The higher the *gasprice*, the faster the transaction is mined in a new block. The *gaslimit* sets a limit on how many gases a transaction can use. Once a transaction consumes more gases than *gaslimit*, all execution and state changes are reverted.

Web3.js Blockchain Read/Write. In blockchain, reading and writing data are different from traditional methods. Writing data to blockchain requires you to put data as payload in transactions. Only when transactions are confirmed, data inside them is confirmed. The reading operation can be conducted in several ways. We can use Ethereum Web3.js to acquire data immediately. We can also use return value in smart contract to acquire blockchain data.

3 PISTIS DESIGN

3.1 Challenges

Before diving into the specifics of PISTIS, we first describe and address the fundamental pitfalls that arise when hybridizing smart contracts and TEE. Note that designing such a protocol that integrates smart contract with TEE to issue certificates is non-trivial, which requires us to resolve the following challenges.

3.1.1 TEE Failures

Though ensuring the integrity and confidentiality of enclave execution, TEE is not a panacea. We first consider the **availability** of TEE. In this paper, we do not make honest host assumptions. By contrast, hosts may be malicious. A malicious host can drop messages, abort execution, or exhaust the hardware resource (e.g., conduct computation-heavy work). Furthermore, even honest hosts may encounter power loss, which makes the TEE unavailable. PISTIS resolves this problem by employing a cluster of TEE nodes, where a TEE is easily replaced. We also assume that the adversary can compromise all but one TEE.

We then consider the breach of TEE **confidentiality**. Recent work has demonstrated the feasibility of extracting secrets from TEE enclaves via side-channel attacks [13]. PISTIS addresses this problem in two ways. First, PISTIS is designed to be TEE-agnostic so that vulnerable TEE can be upgraded with patched version promptly to resolve the discovered vulnerabilities. Second, PISTIS leverages constant memory consumption and execution time enclaves to defend against side-channel attacks.

We finally consider the replay attack that may be launched by malicious hosts. Replay attack is to confuse the TEE states by re-sending previous messages. PISTIS is fault-tolerant to this problem since all states are stored on the distributed ledger. As long as PISTIS smart contract maintains a correct state, the stateless TEE will not confuse smart contract execution.

3.1.2 DNS Failures

Before issuing certificates, the domain ownership should be validated via DNS, which, however, has been demonstrated insecure [14]. In PISTIS, we address this problem by conducting ownership validation from multiple vantage points. Specifically, TEE nodes are deployed in different ASes. Our approach is based on the assumption that the adversary cannot control the majority of the Internet, which was also made in [14].

3.2 Distributed Ledger as a Root of Trust

In traditional PKI, root CAs act as the root of trust. They issue intermediate certificates for commodity CAs to issue certificates. In PISTIS, we adopt the distributed ledger as a root of trust. We construct two contracts, an authorization contract $\mathcal{C}_{\text{AC}}$ and a verification contract $\mathcal{C}_{\text{VC}}$. $\mathcal{C}_{\text{AC}}$ takes charge of validating the domain and issuing certificates. Specifically, $\mathcal{C}_{\text{AC}}$ is a stateful smart contract that allows concurrent state transitions of different registrants. In the stateful contract, an execution either reaches the next state or revert, which protects PISTIS from malicious certificate squatting. The state transition is depicted in the bottom left of Fig. 1. $\mathcal{C}_{\text{VC}}$ provides two interfaces: tree update and certificate verification. The tree update allows the authorization contract $\mathcal{C}_{\text{AC}}$ to update certificates' states, and the certificate verification enables clients to verify certificates.

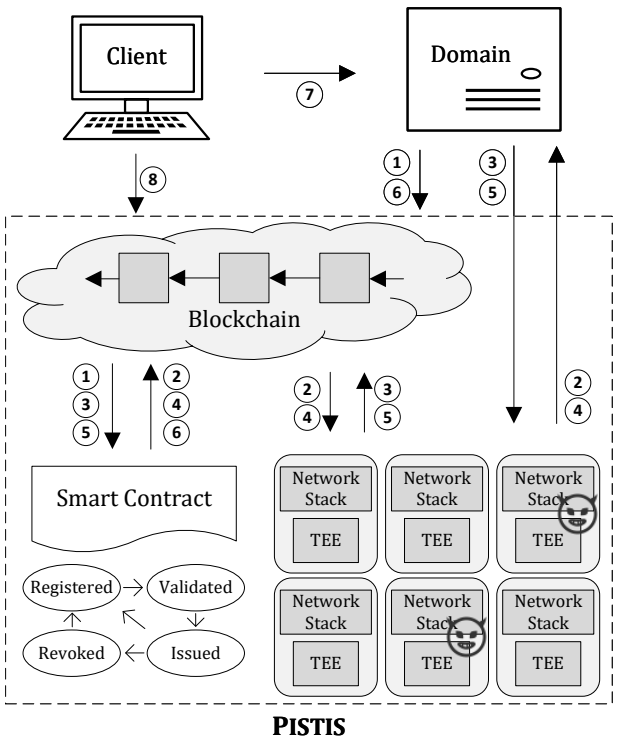


Fig. 1. Architecture and workflow of PISTIS. PISTIS consists of three parts: blockchain, smart contract and TEE nodes. A domain server can request a certificate from PISTIS. A client can verify a certificate provided by the domain server via the blockchain in PISTIS.

3.3 Architecture and Workflow

Fig. 1 illustrates the architecture of PISTIS. As it shows, PISTIS consists of blockchain, smart contract, and TEE nodes (some of them might be malicious). A domain, or accurately its owner can request a certificate from PISTIS via the blockchain. A client can verify the authenticity of PISTIS-issued certificates by querying the blockchain when connecting to a domain. Then, we discuss the workflow:

Certificate Request. Similar to certificate signing request (CSR), an applicant should first register an entry in PISTIS smart contract (Step ①), which contains a fully qualified domain name, an entity name, and a public key.

Domain Validity Validation. PISTIS only issues certificates to domains with `Expiry Date` longer than n seconds, which is an adjustable parameter that equals to the validity of PISTIS-issued certificates. The smart contract invokes the TEE to check the validity of requested domain by posting an invocation transaction onto the blockchain. When a TEE node receives latest block and parses the invocation transaction, it conducts the validation (Step ②). A TEE node typically queries the whois database for the corresponding `Expiry Date` and converts it to a Unix timestamp. By subtracting the timestamp of latest block from `Expiry Date`, the TEE node can get the remaining validity of this domain. The smart contract processes the certificate issuance request only when domain's valid period is longer than a predefined length (i.e., 3 months). The validation results will be put on the blockchain as a transaction that calls PISTIS smart contract to update the corresponding domain's state (Step ③).

Domain Ownership Validation. PISTIS smart contract invokes TEE nodes to validate whether an applicant controls the provided domain. Specifically, smart contract posts an invocation transaction onto the blockchain until TEE nodes receive it (Step ④). PISTIS adopts a secure 2-party computation mechanism to conduct the validation (Step ⑤). We require that the domain server should be equipped with a TEE processor because the secure 2-party computation can only be achieved when both parties are equipped with TEE, which has been proved in [15]. Once a registrant passes the domain ownership validation, TEE nodes can upload the validation result onto the blockchain as transactions that calls the smart contract (Step ⑤). Then, PISTIS smart contract can confirm that this domain is valid and under the control of the registrant. Following above steps, PISTIS can ensure its issued certificates are authorized.

Certificate Issuance and Revocation. Once a domain passes the validity and ownership validation, PISTIS smart contract can issue a certificate to it. The issuance (Step ⑥) and revocation (Step ⑥) are two publicly available functions that can be invoked by valid registrants. PISTIS smart contract publishes issued certificates onto the blockchain, which allows clients to verify their validity and therefore trust these certificates (Step ⑥).

Certificate Verification. PISTIS provides an efficient way to verify its certificates. When a client connects to a website that is protected by a PISITS-issued certificate (Step ⑦), it can leverage the web3.js script to query the authenticity of provided certificate via blockchain (Step ⑧).

3.4 Threat Model

We assume applicants are semi-honest, namely they behave honestly only when registering certificates for their own domains. However, during other domain owners' application, they may behave maliciously. We assume TEE hosts are malicious. They may delay, abort, relay arbitrary messages, or replay previously transmitted messages to enclaves. Besides, we assume that active TEE enclaves are normally trustworthy, but a subset of TEE enclaves may suffer from arbitrary integrity or confidentiality breaches. They may be compromised by other parties (i.e., malicious domain owners) or external attackers who want to violate the certificate issuance security.

We assume the underlying blockchain satisfies three properties: liveness, consistency, and immutability. Liveness means a transaction will be included in the blockchain in a fixed time period. Consistency guarantees that all users have the same view on the state of the blockchain eventually. Immutability implies that once a transaction is confirmed k times, it cannot be reverted.

We consider two types of adversaries: internal byzantine adversary and external adversary. For byzantine adversary, they may control the operating system and network stacks of TEE nodes, which can reorder, replay, drop transmitted messages, and schedule processes arbitrarily. For external adversary, they observe global network traffic, and may reorder or delay messages arbitrarily.

4 THE PISTIS PROTOCOL

In this section, we specify $\mathbf{Prot}_{\text{PISTIS}}$, which aims to realize a Universal Composable (UC) [12] ideal functionality $\mathcal{F}_{\text{PISTIS}}$.

$\mathbf{Prot}_{\text{PISTIS}}$ utilizes digital signature scheme Σ(Gen,Sign,Verf), a symmetric encryption scheme $\mathcal{SE}$(Gen,Enc,Dec), and an asymmetric encryption scheme $\mathcal{AE}$(Gen,Enc,Dec).

4.1 Blockchain Model

We define the underlying blockchain as a general-purpose append-only ledger $\mathcal{F}_{\mathbf{B}}$ that maintained by common blockchain protocols. The blockchain is comprised as a chain of blocks that store transactions. We assume overlay semantics that associate with blockchain data:

- $\mathcal{F}_{\mathbf{B}}$.account: it is used to generate a new account with an address on the blockchain.
- $\mathcal{F}_{\mathbf{B}}$.latest(n): it is used to download the latest n blocks. By default, $n = 1$.
- $\mathcal{F}_{\mathbf{B}}$.post(tx): it is used to broadcast a transaction onto the blockchain. The broadcasted transaction will be included in δ blocks.

4.2 TEE Model

We adopt the attested execution model formalized in [15] and define our TEE as an ideal functionality $\mathcal{G}_{att}$. Similar to the notation in [15], a party that loads en enclave into TEE with an "install" message. A party that invokes the TEE with a "resume" call.

4.3 Formal Specification of the Protocol

In this section, we give the formal protocol of PISTIS, which is depicted in Fig. 2.

PISTIS Registration: An applicant $\mathcal{P}_i$ who wants to request a certificate from PISTIS should first create a blockchain account and register its domain in the $\mathcal{C}_{\text{AC}}$. In this phase, $\mathcal{P}_i$ can invoke $\mathcal{F}_{\mathbf{B}}$.post to broadcast a transaction to invoke $\mathcal{C}_{\text{AC}}$'s registration function. $\mathcal{C}_{\text{AC}}$ will mark $\mathcal{P}_i$'s blockchain address as `registered` and store its provided domain in the contract.

Domain Validation: In this phase, the contract $\mathcal{C}_{\text{AC}}$ needs to ensure that the registered domain is valid (i.e., its validity is longer than n seconds) and under the control of applicant $\mathcal{P}_i$.

For the **validity** validation, $\mathcal{C}_{\text{AC}}$ triggers TEE nodes by invoking the $\mathcal{F}_{\mathbf{B}}$.post function to broadcast a transaction that contains domain name and validity validation instructions.

For the **ownership** validation, we refer to the Automated Certificate Management Environment (ACME) protocol [14] and propose a *challenge-proof* protocol, which is illustrated in Fig. 3.

First, the applicant $\mathcal{P}_i$ invokes the ownership validation function by posting a transaction with a triple tuple $\{$*address*, pk_{acct}, *domain*$\}$. Contract $\mathcal{C}_{\text{AC}}$ checks whether the sender has registered an account. If so, $\mathcal{C}_{\text{AC}}$ generates a verification code $\sigma_{\mathcal{P}_i}$ and sends it to the applicant $\mathcal{P}_i$ as a return value.

Then, the applicant $\mathcal{P}_i$ puts the *proof* as a `TXT` resource record in its authoritative DNS server in the form of `challenge.<domain> IN TXT proof`. $\mathcal{P}_i$ can invoke the PISTIS to check whether the *proof* matches the *challenge*.

The $\mathcal{C}_{\text{AC}}$ then invokes TEE nodes to generate DNS query packets to `challenge.<domain>`. After receiving the `TXT` response, TEE nodes can extract the *proof* and respond to the PISTIS with a transaction. The PISTIS checks whether the received *proof* matches its corresponding *challenge*. If so, the domain passes the validation, and $\mathcal{C}_{\text{AC}}$ mark the domain as `validated`.

Certificate Issuance: After passing the domain validation, applicant $\mathcal{P}_i$ can invoke the $\mathcal{C}_{\text{AC}}$ to issue a certificate. $\mathcal{P}_i$ should generate a public key for its certificate and post this public key onto the blockchain. Then, $\mathcal{C}_{\text{AC}}$ generates a certificate for $\mathcal{P}_i$ and broadcasts it to the blockchain.

$\mathcal{C}_{\text{VC}}$ maintains a MPT to store the states of its issued certificates. Specifically, we set the domain name as the *key* of MPT. States of domain certificates (i.e., issued, revoked) are stored as *value*.

We make some adjustments to traditional MPT by indexing from the top-level domains (TLD). This is because domain names show similarity in their TLDs, and it is better to group similar domain names in a sub-tree. For example, `google.com` and `gmail.com` have identical TLD so that they can be categorized into one sub-tree. For the value part, there are two types. A valid certificate is given a concatenation in the form of `CertID||valid`. For a revoked certificate, the value part is in the form of `CertID||revoked`.

PISTIS $\mathcal{C}_{\text{VC}}$ contract provides three operation functions `Insert`, `Update`, and `Get`. The first two functions enable $\mathcal{C}_{\text{AC}}$ to update certificates' states, and the last function allows clients to verify certificates.

`Insert`. The insert function is operated as follows. First, we should find the node with the same top-level domain. Then, the TLD node's pointer will lead us to its branch node with 26 characters. After that, we check the first letter of the required domain and find its sub-tree. If this sub-tree is empty, we could insert the left letter as a leaf node after it. Otherwise, we go to the second letter and its sub-tree. We follow the letters one by one recursively until we find an empty sub-tree or run out of all letters in the domain name.

`Update`. The update operation is similar to the insert operation. The difference is when we update the state of a certificate, it has been inserted into the MPT. Accordingly, we could search the corresponding node from TLD nodes until we find the same longest prefix node of the domain name. We conduct this operation recursively until we match all letters in the domain. Finally, we could update the state of the found domain.

Certificate Revocation: In PISTIS, certificate revocation can be triggered in three ways:

1) A certificate expires once the latest block's timestamp exceeds its validity. In this case, $\mathcal{C}_{\text{VC}}$ will update the state of this revoked certificate as invalid automatically.
2) A domain owner who owns the private key can sign a revocation request to PISTIS to invalidate the certificate. In this case, the domain owner $\mathcal{P}_i$ should invoke the certificate revocation function directly by sending a transaction that contains its address, public key, domain, and certificate id.
3) A new applicant $\mathcal{P}_j$, who could prove its ownership of the domain related to a valid certificate, can invalidate that certificate. We consider this scenario because the ownership of a domain may change, while the previously-issued certificate is still valid.

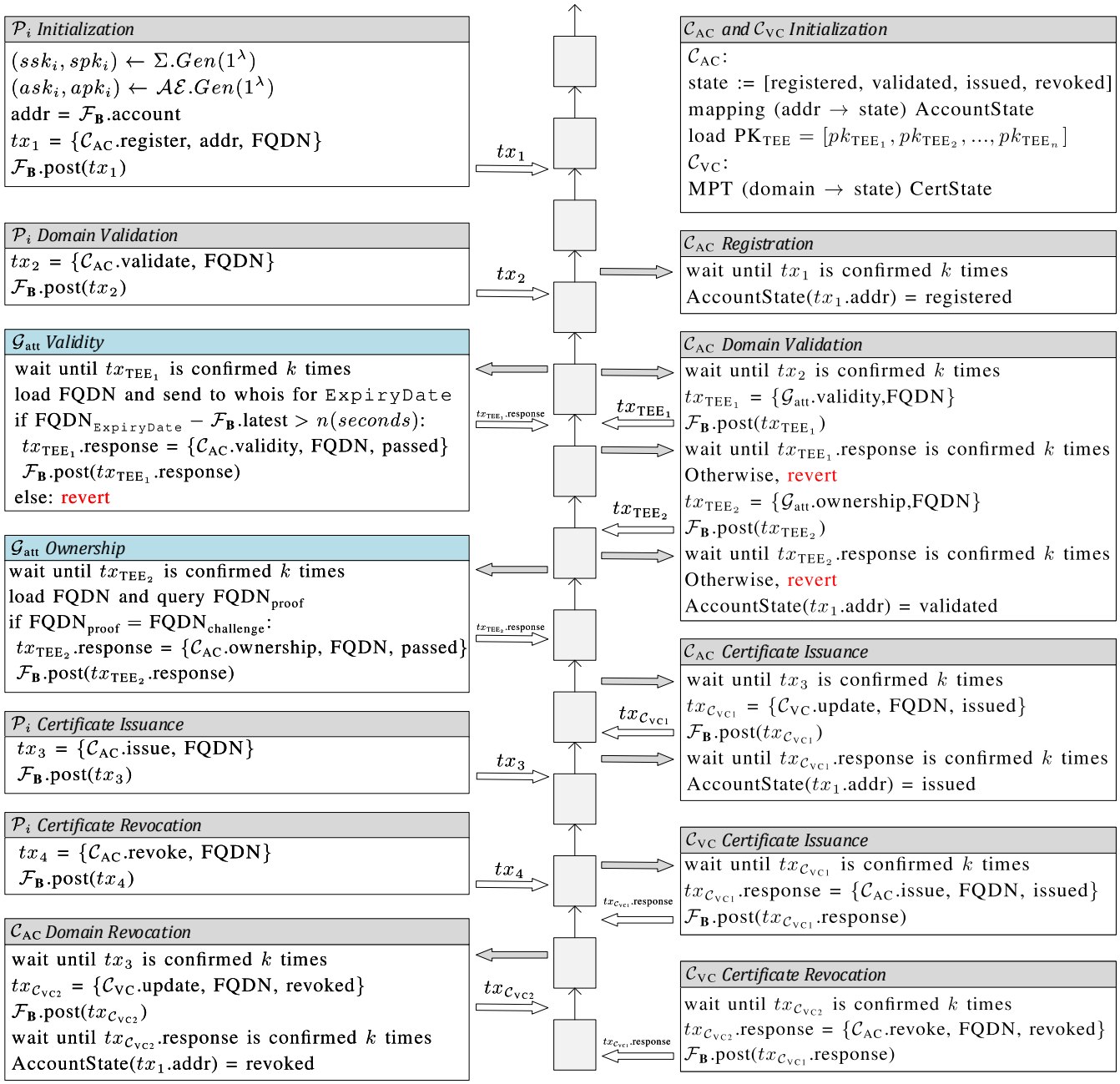


Fig. 2. A formal specification of PISTIS protocol. Gray arrows indicate reading blockchain data. White arrows indicate broadcasting a transaction onto the blockchain.

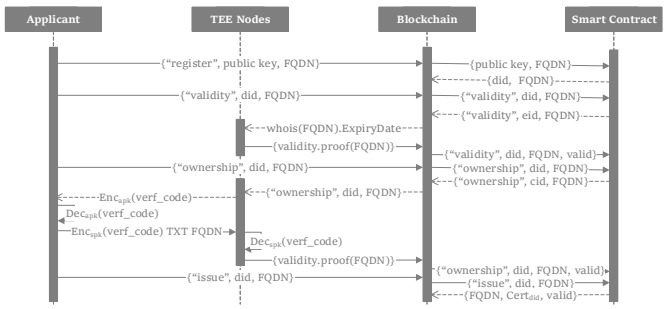


Fig. 3. The *challenge-proof* protocol. Under TEE nodes, dashed line means the communication does not involve the enclave while the thick solid line represents that enclaves are involved.

In this case, the new domain owner should explicitly invoke the PISTIS certificate revocation function. Then, PISTIS invokes the domain authorization function, and once the new domain owner passes the ownership validation, $\mathcal{C}_{AC}$ issues a new certificate and invokes $\mathcal{C}_{VC}$ to update certificate state.

Certificate Verification: When a client connects to a website protected by a PISTIS-issued certificate, he/she could verify whether the certificate is valid before establishing a HTTPS connection. Only when the certificate passes verification, the client believes that the server is the authentic one. Otherwise, the browser should halt the connection. In PISTIS, certificate verification is conducted by the `Get` operation provided by contract $\mathcal{C}_{VC}$.

`Get`. The `Get` operation is a getter function provided by the underlying blockchain, which enables clients to acquire data directly. Specifically, the underlying blockchain implementation traverses the MPT to acquire the required state for users.

5 Security Analysis

In this section, we formally prove the security of PISTIS in the Universally Composable (UC) framework [12].

5.1 Ideal Functionality

The idea functionality of PISTIS is specified in Fig. 4 as $\mathcal{F}_{\text{PISTIS}}$. $\mathcal{F}_{\text{PISTIS}}$ allows applicants (each applicant is denoted by a unique id $\mathcal{P}_i$) to request for certificates. Following the convention in [12], we set an information leakage function ℓ to capture the allowed information leakage from the encryption. We also use the standard delayed output [12] to model the power of network adversary.

Applicants can send messages to $\mathcal{F}_{\text{PISTIS}}$ to invoke PISTIS registration, domain validation, certificate issuance, and certificate revocation, which will update the corresponding domain's state. Clients (i.e., environment $\mathcal{Z}$) can also query $\mathcal{F}_{\text{PISTIS}}$ for the state of a domain's certificate.

$\mathcal{F}_{\text{PISTIS}}(\ell, \mathcal{P}_i)$

Parameter: leakage function $\ell : \{0,1\}^* \to \{0,1\}^*$

On receive ("register", FQDN) from $\mathcal{P}_i$:
- did $\leftarrow \{0,1\}^\lambda$
- notify $\mathcal{A}$ of ("register", $\mathcal{P}_i$, did, FQDN)
- Storage[did] := (FQDN, $\vec{0}$)
- send a public delayed output ("receipt", did) to $\mathcal{P}_i$

On receive ("validity", did, eid) from $\mathcal{P}_i$:
- notify $\mathcal{A}$ of ("validity", $\mathcal{P}_i$, did, eid)
- (FQDN, st) := Storage[did]; abort if not found
- (outp, st) := whois(FQDN, did)
- notify $\mathcal{A}$ of ($\ell(outp)$, did, eid)
- update Storage[did] := (FQDN, st)
- send a public delayed output ("receipt", did) to $\mathcal{P}_i$

On receive ("ownership", FQDN, did, eid) from $\mathcal{P}_i$:
- notify $\mathcal{A}$ of ("ownership", $\mathcal{P}_i$, did, eid)
- (FQDN, st) := Storage[did]; abort if not found
- send a secret delayed message (challenge, did) to $\mathcal{P}_i$
- notify $\mathcal{A}$ of (ℓ(challenge), did, eid)
- update Storage[did] := (FQDN, st)
- send a public delayed output ("receipt", did) to $\mathcal{P}_i$

On receive "issue", FQDN) from $\mathcal{P}_i$:
- notify $\mathcal{A}$ of ("issue", $\mathcal{P}_i$, did, eid)
- (FQDN, st) := Storage[did]; abort if not found
- update Storage[did] := (FQDN, st)
- send a public delayed output ("issued", did) to $\mathcal{P}_i$

On receive "revoke", FQDN) from $\mathcal{P}_i$:
- notify $\mathcal{A}$ of ("revoke", $\mathcal{P}_i$, did, eid)
- (FQDN, st) := Storage[did]; abort if not found
- update Storage[did] := (FQDN, st)
- send a public delayed output ("revoked", did) to $\mathcal{P}_i$

On receive ("read", did) from $\mathcal{Z}$:
- (FQDN, st) := Storage[did]; abort if not found
- send st to $\mathcal{Z}$

Fig. 4. The ideal functionality of PISTIS.

5.2 Security Proof

Intuitively, a PISTIS-issued certificate being authorized and trusted means that an adversary cannot convince the PISTIS to accept a response that differs from the expected content obtained from the specified domain.

Theorem 1. *(Security of $\boldsymbol{Prot}_{\text{PISTIS}}$). Assume Σ_{TEE} is existentially unforgeable under chosen message attacks (EU-CMA), and $\mathcal{AE}$ is INC-CPA secure. Then $\boldsymbol{Prot}_{\text{PISTIS}}$ securely realizes $\mathcal{F}_{\text{PISTIS}}$ in the ($\mathcal{G}_{att}$,$\mathcal{F}_{\mathbf{B}}$)-hybrid model, for static adversaries.*

Proof. Let $\mathcal{Z}$ be an environment and $\mathcal{A}$ be a "dummy adversary" (i.e., acts as a "transparent channel" between the environment $\mathcal{Z}$ and protocol [12]). To show that $\mathbf{Prot}_{\text{PISTIS}}$ UC-realizes $\mathcal{F}_{\text{PISTIS}}$, we specify below a simulator Sim such that no environment $\mathcal{Z}$ can distinguish an interaction between $\mathbf{Prot}_{\text{PISTIS}}$ and $\mathcal{A}$ from an interaction between $\mathcal{F}_{\text{PISTIS}}$ and Sim. That is, Sim satisfies

$$\forall \mathcal{Z},\ \text{EXEC}_{\mathbf{Prot}_{\text{PISTIS}},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F}_{\text{PISTIS}},\text{Sim},\mathcal{Z}}$$

5.2.1 Construction of Sim

Sim works as follows: if a message is sent by an honest applicant to $\mathcal{F}_{\text{PISTIS}}$, Sim emulates appropriate real world "network traffic" for $\mathcal{Z}$ with information obtained from $\mathcal{F}_{\text{PISTIS}}$. If a message is sent by a corrupted party, Sim extracts the input and interacts with the corrupted party with the help of $\mathcal{F}_{\text{PISTIS}}$. We provide further details on the processing of specific messages.

PISTIS Registration:

- If applicant $\mathcal{P}_i$ is honest, Sim obtains ($\mathcal{P}_i$, FQDN, eid) from $\mathcal{F}_{\text{PISTIS}}$ and emulates an execution of the "create" call.
- If applicant $\mathcal{P}_i$ is corrupted, Sim extracts FQDN from $\mathcal{Z}$. On behalf of $\mathcal{P}_i$, Sim sends {"register", FQDN} to $\mathcal{F}_{\text{PISTIS}}$ and instructs $\mathcal{F}_{\text{PISTIS}}$ to deliver the output.
- In both cases, Sim simulates the interaction between $\mathcal{F}_{\mathbf{B}}$ and $\mathcal{G}_{\text{att}}$, on behalf of the adversary or honest parties.

Domain Validity Validation:

- If applicant $\mathcal{P}_i$ is honest, Sim obtains ($\mathcal{P}_i$, FQDN, did, eid) from $\mathcal{F}_{\text{PISTIS}}$ and emulates an execution of the "validity" call. Specifically, Sim extracts the whois database and checks FQDN's validity. Then, Sim updates $\mathcal{F}_{\text{PISTIS}}$ with (FQDN, eid, st).
- If applicant $\mathcal{P}_i$ is corrupted, Sim extracts FQDN from $\mathcal{Z}$. On behalf of $\mathcal{P}_i$, Sim sends {"validity", FQDN} to $\mathcal{F}_{\text{PISTIS}}$ and instructs $\mathcal{F}_{\text{PISTIS}}$ to extract the whois database.

Domain Ownership Validation:

- If applicant $\mathcal{P}_i$ is honest, Sim obtains ($\mathcal{P}_i$, FQDN, did, eid) from $\mathcal{F}_{\text{PISTIS}}$ and emulates an execution of the "validity" call.
- If applicant $\mathcal{P}_i$ is corrupted, Sim extracts FQDN from $\mathcal{Z}$. On behalf of $\mathcal{P}_i$, Sim sends {"ownership", FQDN} to $\mathcal{F}_{\text{PISTIS}}$ and instructs $\mathcal{F}_{\text{PISTIS}}$ to deliver the output.

Certificate Issuance:

- If applicant $\mathcal{P}_i$ is honest, Sim obtains ($\mathcal{P}_i$, "issue", FQDN, eid) from $\mathcal{F}_{\text{PISTIS}}$ and emulates an execution of the "revoke" call.
- If applicant $\mathcal{P}_i$ is corrupted, Sim extracts FQDN from $\mathcal{Z}$. On behalf of $\mathcal{P}_i$, Sim sends {"issue", FQDN} to $\mathcal{F}_{\text{PISTIS}}$ and instructs $\mathcal{F}_{\text{PISTIS}}$ to deliver the output.

Certificate Revocation:

- If applicant $\mathcal{P}_i$ is honest, Sim obtains ($\mathcal{P}_i$, "revoke", FQDN, eid) from $\mathcal{F}_{\text{PISTIS}}$ and emulates an execution of the "revoke" call.
- If applicant $\mathcal{P}_i$ is corrupted, Sim extracts FQDN from $\mathcal{Z}$. On behalf of $\mathcal{P}_i$, Sim sends {"revoke", FQDN} to $\mathcal{F}_{\text{PISTIS}}$ and instructs $\mathcal{F}_{\text{PISTIS}}$ to deliver the output.

Public Read: On any call ("read", eid) from $\mathcal{P}_i$, Sim emulates a "read" message to $\mathcal{F}_{\mathbf{B}}$. If $\mathcal{P}_i$ is corrupted, Sim sends to $\mathcal{F}_{\text{PISTIS}}$ a "read" message on $\mathcal{P}_i$'s behalf and forwards the response to $\mathcal{A}$.

Corrupted Enclaves: Sim obtains eids of corrupted enclaves when $\mathcal{Z}$ corrupts them. In real world, $\mathcal{Z}$ could make corrupted enclaves unavailable at any time by dropping messages or terminating it. Sim relays all messages between a corrupted enclave and $\mathcal{Z}$. Sim terminates $\mathcal{F}_{\text{PISTIS}}$ once the emulated execution is aborted by $\mathcal{Z}$ prematurely.

5.2.2 Validity of Sim

We show that, in hrbrid settings, no environment $\mathcal{Z}$ can distinguish an interaction with $\mathbf{Prot}_{\text{PISTIS}}$ and $\mathcal{A}$ from an interaction with $\mathcal{F}_{\text{PISTIS}}$ and Sim. We consider the following sequence of hybrid settings, starting with the real protocol execution.

- Hybrid H_1 lets Sim to emulate $\mathcal{G}_{\text{att}}$ and $\mathcal{F}_{\mathbf{B}}$.
- Hybrid H_2 filters out the forgery attacks against Σ_{TEE}.
- Hybrid H_3 lets Sim emulate the issuance phase.
- Hybrid H_4 replaces the encryption of challenge with encryption of $\vec{0}$.

Hybrid H_1 proceeds as in the real world protocol, except that Sim emulates the behavior of $\mathcal{G}_{\text{att}}$ and $\mathcal{F}_{\mathbf{B}}$. To emulate $\mathcal{G}_{\text{att}}$, Sim generates a key pair (pk_{TEE}, sk_{TEE}) for Σ_{TEE}. Whenever $\mathcal{A}$ wants to communicate with $\mathcal{G}_{\text{att}}$, Sim records $\mathcal{A}$'s messages and faithfully emulates $\mathcal{G}_{\text{att}}$'s behavior. As to emulate $\mathcal{F}_{\mathbf{B}}$, Sim stores blockchain data internally.

From the perspective of $\mathcal{A}'$, H_1 is perfectly simulated as in the real world. In this case, $\mathcal{Z}$ cannot distinguish between H_1 and the real world execution.

Hybrid H_2 proceeds as in H_1, except for the following differences. If $\mathcal{A}$ invokes $\mathcal{G}_{\text{att}}$ with correct message, then for all sequential calls, Sim records a tuple where state is the output of contract and σ_{TEE} is the attestation under sk_{TEE}. Let Ω denote the set of all such tuples. Whenever $\mathcal{A}$ sends an attested output (st, σ_{TEE}) $\notin \Omega$ to $\mathcal{F}_{\mathbf{B}}$ or an honest party $\mathcal{P}_i$, Sim aborts.

The indistinguishability between H_1 and H_2 can be shown by the following reduction to the EU-CMA property of Σ: In H_1, if $\mathcal{A}$ sends forged attestations to $\mathcal{F}_{\mathbf{B}}$, signature verification by $\mathcal{F}_{\mathbf{B}}$ or honest party will fail with all but negligible probability. If $\mathcal{Z}$ can distinguish H_2 from H_1, $\mathcal{Z}$ and $\mathcal{A}$ can be used to win the game of signature forgery.

Hybrid H_3 is the same as H_2 but has Sim to emulate the certificate issuance. Sim emulates messages from G_{att} to F_B as described above. If P_i is corrupted, Sim sends ("issue", FQDN) to F_{PISTIS} as P_i.

It is clear that from the perspective of $\mathcal{A}$, Sim emulates G_{att} and F_B perfectly. In this case, $\mathcal{Z}$ cannot distinguish between H_3 and the real world execution.

Hybrid H_4 is the same as H_3 except that honest applicants also send messages to $\mathcal{F}_{\text{PISTIS}}$. If $\mathcal{P}_i$ is corrupted, Sim emulates real-world messages with the help of $\mathcal{F}_{\text{PISTIS}}$.

The indistinguishability between H_3 and H_4 can be reduced to the IND-CPA property of $\mathcal{AE}$. Having no knowledge of the secret key, $\mathcal{A}$ cannot distinguish encryption of $\vec{0}$ from encryption of other messages. Note that we don't require IND-CCA security because $\mathcal{A}$ does not have direct access to a decryption oracle.

□

6 Experiments and Evaluation

In this section, we conduct experiments to explore the feasibility of PISTIS by evaluating the performance of contracts and TEE nodes.

6.1 Contract Evaluation

We implement PISTIS contracts in Solidity and deploy them on the Ropsten testnet. We operate three nodes: a domain node, a TEE node, and a client node. All nodes are equipped with a blockchain endpoint. For client node, we install a MetaMask wallet, which is a web-based Ethereum wallet. The Geth wallet operates as the endpoint for the TEE node. It exchanges data between PISTIS contracts and the TEE node. On the client side, we inject certificate verification script into the PISTIS-protected websites. Only when PISTIS returns a valid answer, the browser recognizes the certificate as valid and establishes a HTTPS connection.

We first evaluate the performance of PISTIS contract. Three experiments are conducted to test the gas consumption, storage cost, and verification latency.

6.1.1 Gas Consumption

In this experiment, we test the gas consumption of most operations provided by PISTIS. The gas consumption is closely related to the sustainability and feasibility of PISTIS. Specifically, we measure the approximate computational steps (in Ethereum gas) and money cost (in USD) for each operation supported by the PISTIS contract. During the writing of this paper (i.e., December of 2020), an ether costs around 500 USD. For the gas price, we adopt 40 Gwei (1 Gwei = 10^{-9} ether), which is 0.00002 USD. For testing, we assume all strings are a maximum of 32 bytes, which is the basic storage unit in Ethereum. We also assume that the public keys for certificate verification are 2048-bit RSA keys. Table 1 shows the costs of various operations in PISTIS.

As we can see, the issuance of a certificate with PISTIS may cost a domain owner around 5 dollars, which is less than most commercial certificate authorities' certificate issuance service. For end clients, the certificate verification costs nothing so that they do not need to pay to visit PISTIS-protected websites.

TABLE 1
The consumption of PISTIS operations.

Operation	Gas Cost (Unit)	Gas Cost (USD)
Account_Registration	38500	0.77
domain_Registration	41800	0.836
domain_Validation	52900	1.058
Challenge_Generation	65050	1.31
Challenge_Verification	3600	0.072
Certificate_Issuance	52500	1.05
Certificate_Revocation	4100	0.082
Certificate_Verification	0	0
CertID_Broadcast	1800	0.036

6.1.2 Storage Overhead

In this experiment, we investigate the storage overhead of PISTIS. Specifically, we investigate the on-chain storage for MPT, the account list, and transactions required for data transmission. For the MPT, we store the states of 11,239 certificates, which costs about 104 MB. For the account list, we insert 11,239 account entries, whose storage requirement is 8 MB. As the storage overhead increases linearly with the number of certificates, we can infer that the required storage space is in the TB level when there are hundreds of millions of certificates in the system. We think this requirement can be reached easily on current consumer computers.

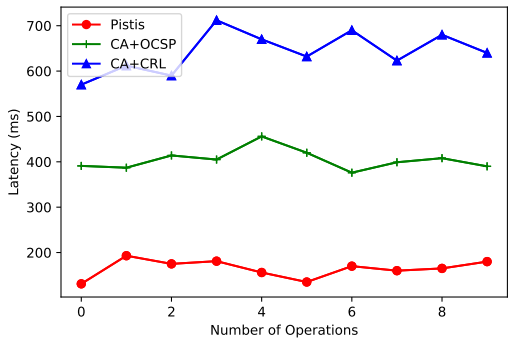


Fig. 5. The comparison of verification latency between PISTIS, traditional CA + OCSP, and traditional CA + CRL.

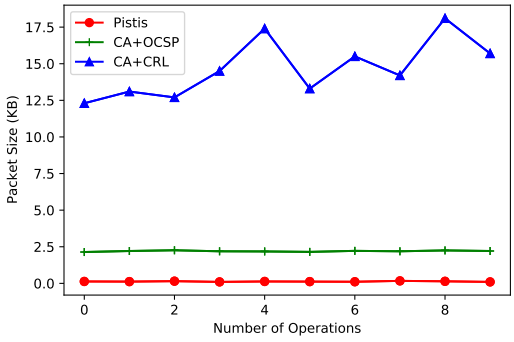


Fig. 6. The comparison of verification packet size between PISTIS, traditional CA + OCSP, and traditional CA + CRL

6.1.3 Verification Performance

In this experiment, we aim to test the performance of certificate verification in PISTIS and compare it with the latency of OCSP and CRL, two widely-used certificate verification methods. Typical certificate verification can be categorized into two parts. First, we need to verify the signature provided by the certificate and check whether it is assigned by a trusted CA. Second, we need to verify whether this certificate is revoked. A certificate will not pass the verification if either step fails.

The first step is conducted on the browser side. The main difference in verification latency lies in the certificate state checking. For PISTIS, we measure the latency from the time that a client sends a state verification transaction to the time that a response transaction is parsed, and the certificate's state is confirmed. For the OCSP and CRL, we use OpenSSL to send OCSP and CRL requests and record the corresponding latency.

We conduct the measurement ten times on PISTIS, OCSP, and CRL, respectively. Results are summarized in Fig. 5. We use the circle dots to represent the delay of PISTIS, the vertical lines to represent the delay of OCSP, and the triangle dots to represent the delay of CRL. As depicted in Fig. 5, the verification latency of PISTIS is the lowest, which fluctuates around 150 ms. The delay of OCSP ranks secondly with 400 ms certificate verification service. The CRL has the longest delay, nearly 700 ms, which results from the biggest packet they transmit during verification. In addition, we also measure the packet size during each operation and illustrate them in Fig. 6. The CRL needs to transmit around 15 KB data for certificate verification, while OCSP needs to transmit 2 KB. By contrast, PISTIS only needs to transmit a transaction for verification, whose size is around 0.1 KB.

6.2 TEE Evaluation

We evaluate the performance of PISTIS TEE on a server with i9-9900k CPU and 32GB memory. Our experiment results prove that PISTIS can easily meet the peak throughput of Ethereum network and can be deployed on current commodity servers.

6.2.1 Throughput

We first evaluate the throughput of a TEE node. We aim to explore to what extent can a TEE node processes transactions. The experiment results are illustrated in Fig. 7. We can see that a server with at most 30 enclave instances can handle up to 58 transactions per second. We also notice that when the number of enclave instances is less than 16, the throughput increases linearly.

6.2.2 Response Time

We define the response time of a TEE node as the interval between the time that a PISTIS Contract/Domain Owner sends a request to the enclave and the time that a PISTIS Contract/Domain Owner receives a response from the TEE node. The experiment results are presented in the Table 2. As we can see, the interaction between a enclave and PISTIS contracts are fast, which is because the TEE node is equipped with a blockchain endpoint. In this case, the interaction between the PISTIS contract and the enclave omit

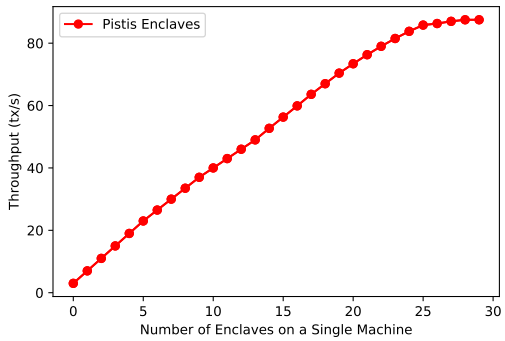


Fig. 7. The throughput of TEE node on a single machine with different numbers of enclave instances.

TABLE 2
The response time of TEE node operations.

Operation	t_{mean} (ms)	t_{max} (ms)	t_{min} (ms)
Protocol Trigger	1.1	4.21	0.36
DNS Query	1.06	4.17	0.28
DNS Response	86.4	280.2	46.7
Protocol Response	1.13	4.25	0.41
Total	89.69	292.83	47.75

the network latency. By contrast, the DNS query operation has the longest response time, which is because the communication latency between TEE nodes and domains.

7 Discussion

In this section, we discuss how to deploy, maintain, and upgrade PISTIS contracts and TEE nodes. We also discuss the limitations of the underlying infrastructure of PISTIS.

7.1 Deployment, Maintenance, and Upgrade

As PISTIS adopts a new way to issue and revoke certificates, its deployment, maintenance, and upgrade methods are different from traditional CAs. PISTIS contracts are written by its developers (e.g., a traditional CA company or open-source community that wants to issue certificates using PISTIS). These developers can deploy PISTIS contracts onto a chosen blockchain platform and operate TEE nodes to establish the communication channels between PISTIS and domains. Developers also monitor the state transition of PISTIS and check if there are any problems. If they want to add new functions, they can use a new contract to upgrade the PISTIS service. The older contracts will be destructed by calling the `selfdestruct` function and the new contract address will be advertised to target domain owners. We want to emphasize that these upgrades will not increase the possibility that PISTIS is manipulated by its developers. Since PISTIS is transparent, any deviation from its design principle will cause people not to trust it, which conflicts with the interests of developers.

Once developers deploy the PISTIS contract on the blockchain, domain owners can interact with it to request a certificate. After getting a PISTIS-issued certificate, a domain owner can set up its HTTPS-protected service. Clients can verify whether a certificate is valid by querying PISTIS blockchain. Specifically, a client needs to inject the verification script into the website to interact with the certificate verification contract $\mathcal{C}_{\text{VC}}$. Once a certificate passes verification, the client can establish a HTTPS connection with the PISTIS-protected website.

In this process, PISTIS will not be affected by whether people trust its developers. If developers leave a back door in the contract program, domain owners can discover it and reject PISTIS-issued certificates. Even developers are compromised during PISTIS's execution, the hybrid architecture of blockchain and TEE prevents their malicious behaviors as long as the PISTIS contract has no loopholes.

7.2 Infrastructure Limitations

PISTIS is constructed based on smart contract and TEE, which are both emerging technologies and have some limitations.

Blockchain Limitations. Blockchain has demonstrated to be vulnerable in some aspects. For example, the mining policy is not secure since attackers can withhold a newly-mined block and broadcast it until another miner finds a new block in the same height, which makes the computation power devoted by most honest miners invalid. In addition, bribery attack [16] demonstrates the possibility that attackers can manipulate the inclusion of transactions.

Blockchain platforms, especially those support smart contract, can only provide limited-throughput transaction processing service. In this case, PISTIS can only issue certificates at a limited rate. In our future work, we aim to extend PISTIS to some performant blockchain systems such as Hyperledger, Libra, or some experimental sharding systems.

Smart Contract Limitations. The smart contract also has some vulnerabilities and some even lead to serious consequences such as the DAO attack. Since the authorization of certificates constructs the foundation of secure web connections, we should alleviate the exposed vulnerabilities and protect PISTIS from the other potential vulnerabilities.

TEE Limitations. The Intel SGX was proposed to provide trusted computation by isolating some memory parts and encapsulating programs in enclaves securely. However, some researchers have successfully launched side-channel attacks against the Intel SGX platforms [13]. Moreover, enclaves can offload some computation overhead from the PISTIS contract, which might lead to new attack vectors such as SgxPectre [17]. In this case, mitigation should be located in the PISTIS framework. We also consider the case that some TEE hosts are malicious. They can delay or drop correct DNS responses but cannot forge a valid proof of ownership validation. In this case, malicious hosts can only cause a small DoS attack. As we have assumed that at least one TEE is working, such attacks cannot affect the overall operation of PISTIS.

MitM Attack. We consider the possibility that MitM attacker may affect the system. We discuss attacks launched by two kinds of attackers: passive attackers that controls a large ISP, and active attackers that attempts to attract traffic from other networks. We run simulations with different number of TEE nodes. If the attacker is in the victim

domain's network, it can hijack requests from the domain owner and spoof responses. Most domains may have multiple nameservers and these nameservers are usually placed in different networks. This is following the best practice to avoid a single point of failure for domains. Furthermore, as nameservers of the same domain are hosted in different networks, an attacker can hardly hijack or spoof all responses. We also quantify the ability of an on-path attacker to intercept majority of DNS requests sent to a TEE node from the victim domain. The simulation evaluates all possible scenarios for an on-path attacker to cover almost all routes between the victim domain and the TEE node. Results show that it is impossible for a MitM attacker to acquire challenge value during the validation process.

8 Related Work

In this section, we discuss the related work of PISTIS, including blockchain-TEE hybridized systems, traditional countermeasures against unauthorized certificates, and blockchain-based PKI/CA. We finally compare PISTIS with state-of-the-art.

8.1 Blockchain-TEE Hybridized Systems

The combination of blockchain and TEE shows great potential due to their superior characteristics and complementary nature in areas such as privacy preserving and attested execution. Kaptchuk et al. explores the computational properties of using blockchain to store the state of stateless TEE [29]. Teechain [30] leverages TEE as the entry and exit point for off-chain payment channels and enables asynchronous execution of off-chain transactions. FASTKITTEN [31] employs TEE to execute arbitrarily complex smart contracts efficiently on cryptocurrencies, which are originally designed without smart contract support.

8.2 Traditional Countermeasures

Traditional countermeasures that aim to address the unauthorized certificate problem can be categorized into two types: client-side and server-side.

On the client-side, proposals such as HPKP [4] and Trust Assertions for Certificate Keys (TACK) [32] aim to establish a solid connection between the public key and the domain name. These solutions, however, require a domain to inform clients which keys are valid so that clients can distinguish valid certificates from unauthorized ones. Researchers also proposed community of trust [33], which acts as the basis of certificate verification. Syta et al. proposed CoSi [19], which employs a witness cosigning protocol to ensure that every statement is verified and publicly logged by a diverse group.

On the server-side, most countermeasures aim to establish log servers, which allows domain owners to record operations on their certificates. This also provides public accountability to clients. Sovereign Keys (SK) [34], Certificate Transparency (CT) [5], AKI [6], ARPKI [7], DTKI [18], and PoliCert [8] fall in this category. Researchers also focused on the notification of certificate revocation to prevent attackers from further involvement [35].

8.3 Blockchain-based PKI/CA

There are some proposals aiming to address the unauthorized certificate problem utilizing blockchain technology.

Fromknecht et al. proposed CertCoin [20], whose core idea is letting the public ledger as a "bulletin boards" for domains and their associated public keys. Mustafa proposed SCPKI [21], which is an alternative PKI system based on a decentralized and transparent design using the web-of-trust model and smart contracts. Catena [36] leverages Bitcoin as the log server and generates a transaction chain to prevent CAs from issuing contradicting certificates. By contrast, PISTIS not only preserves transactions to trace certificate state transitions, but also maintains a MPT to store latest states of certificates. IKP [22] aims to mitigate unauthorized certificates by incentivizing the CA, domain owner, and clients to report unauthorized certificates. IKP is designed to be compatible with current PKI so that certificate issuance and revocation are both conducted by traditional CAs. Cecoin [23] employs a Bitcoin-like blockchain to provide irreversible unforgeability and public verifiability in the CAs. Specifically, certificates are treated as currency and are circulated on the blockchain. BlockPKI [26] uses blockchain as the log server to make CA operations publicly visible and accountable.

Yakubov et al. introduced smart contracts to establish a blockchain-based PKI [37]. Each smart contract acts as a CA that takes charge of issuing and revoking certificates. CertChain [24] aims to enhance the security of PKI by recording certificate operations on the blockchain. However, it does not provide a feasible domain validation function, which leaves a door of unauthorized certificates. PBCert [25] explores the way to enable privacy-preserving in querying the latest states of certificates, which protects users from eavesdropping. CertLedger [27] utilizes the blockchain to implement certificate transparency and provides an efficient certificate verification method. SmartCert [28] generates smart contracts for certificates to automate the certificate validation.

8.4 Comparison

Differences between PISTIS and state-of-the-art are summarized in Table 3. The comparison is conducted in six dimensions: certificate authority's construction and type, certificate issuance, domain ownership validation, certificate validity verification, certificate revocation request and checking, and security analysis of the proposed system. The related work is sorted chronologically in the table, while we separate PISTIS, traditional solutions and blockchain-based solutions with dual horizontal lines.

For traditional solutions, all work relies on external trusted CA to issue certificates. As to the request initiation side, all systems can initiate registration request from both domain and CA side. We have emphasized that requests initiated from CA side enable attackers to compromise a CA and maliciously issue unauthorized certificates. By contrast, PISTIS uses smart contracts to build CAs, which are essentially source code that cannot initiate a certificate issuance request locally, namely from CA side. For ownership validation, most work are left empty because they rely on external trusted CA to conduct ownership

TABLE 3
Comparison between PISTIS and related work.

Name	Certificate Authority		Issuance		Ownership[2]		Validity		Revocation		Security Analysis
	Construction	Type	Domain	CA[1]	Validation	Method	Validation	Method	Request	Checking	
PISTIS	Smart Contract + TEE	Decentralized	✓	N/A	✓	TEE	✓	MPTP[3]	✓	web3.js	Formal
AKI [6]	External Trusted CA	Centralized	✓	✓	-	-	✓	ILS Proof[4]	✓	N/A	Informal
ARPKI [7]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP[3]	✓	N/A	Formal
PoliCert [8]	External Trusted CA	Centralized	✓	✓	-	-	✓	Log Proof	✓	Log Proof	Informal
DTKI [18]	External Trusted CA	Centralized	✓	✓	-	-	✓	Master Key	✓	N/A	Formal
DV++ [14]	External Trusted CA	Centralized	✓	✓	✓	Multi-Path	N/A	N/A	N/A	N/A	N/A
CoSi [19]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP	N/A	N/A	N/A
CertCoin [20]	On-Chain Transaction	Decentralized	✓	N/A	✓	Passive	✓	Passive	✓	N/A	N/A
SCPKI [21]	Smart Contract	Decentralized	N/A	N/A	N/A	N/A	✓	MTP	✓	N/A	N/A
IKP [22]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP	✓	N/A	N/A
Cecoin [23]	On-Chain Transaction	Decentralized	✓	N/A	✓	Passive	✓	Passive	N/A	N/A	Informal
Certchain [24]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP	✓	Bloom Filter	Informal
PBCert [25]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP	✓	OCSP	Informal
BlockPKI [26]	External Trusted CA	Centralized	✓	✓	✓	ACME	✓	Multi-Sig	N/A	N/A	Informal
CertLedger [27]	External Trusted CA	Centralized	✓	✓	-	-	✓	MPTP	✓	N/A	Formal
SmartCert [28]	External Trusted CA	Centralized	✓	✓	-	-	✓	MTP	✓	N/A	Informal

[1] This column indicates that a malicious or compromised CA can issue certificates for any domains without their permission.
[2] For systems that rely external trusted CA to issue certificates, we use "-" to indicate not covering ownership validation.
[3] MPTP is short for Merkle Patricia Tree Proof while MTP is short for Merkle Tree Proof.
[4] ILS is short for Integrity Log Server, which is proposed in AKI [6].

validation except DV++, which proposes a multiple vantage points domain validation mechanism. Allowing CA-side certificate issuance means that malicious or compromised CAs may issue certificates without domain permission, i.e. unauthorized certificates. PISTIS ensures that malicious or compromised CAs cannot issue certificates by combining this certificate issuance limitation. A certificate request can only be initiated from domain side and the domain must pass the ownership validation. In addition, all systems except DV++ proposed the corresponding certificate validity validation mechanism, that is, to detect whether a certificate is valid, whether it has been revoked or beyond its valid period. As to certificate revocation, DV++ and CoSi do not provide mechanism to revoke certificates and only PISTIS and PoliCert provide revocation checking mechanism. For security analysis, either formal or informal, most work provides security analysis. In general, traditional approaches were based on a trusted CA, with enhanced protection to avoid corruption caused by unauthorized certificates. However, as the CA might still be compromised to issue rogue certificates, traditional methods cannot prevent the issuance of unauthorized certificates as PISTIS.

For blockchain-based solutions, we found that most of them rely on external trusted CAs to issue certificates. This reflects their log server based ideas, which are similar to traditional log server based solution. In these blockchain-based log server solutions, blockchain is used as a data carrier similar to the traditional log server to chain the history of certificate operations. By contrast, SCPKI relies on smart contracts to issue certificates. SCPKI does not issue full certificates, but binding relationships between public keys and identities. In addition, SCPKI cannot conduct domain ownership validation on its applicants, which indicates that SCPKI is incomplete. CertCoin and Cecoin are based on Namecoin and therefore use blockchain as their certificate authority. Furthermore, solutions that rely on external trusted CAs still offload certificate issuance and ownership validation to them. Only BlockPKI emphasizes that it carries out ownership validation through ACME method. In CertCoin and Cecoin, as the registration of certificates is based on the validity of relevant domains, it is only necessary for a domain owner to broadcast a certificate registration transaction based on its previous domain registration transaction. At this step, ownership validation of domains is simply a matter of looking up the transaction history on blockchain. For certificate validity validation methods, all listed solutions are equipped with a proper one. As to certificate revocation, BlockPKI issues short-lived certificates so that omits it while Cecoin does not provide revocation mechanism. In addition, only CertChain and PBCert provide revocation checking mechanism. For the security analysis, only CertLedger provides a formal analysis. The rest either only provide an informal analysis or no analysis at all. Generally speaking, the idea behind most blockchain-based work is similar to that of traditional work, namely log server mode. They simply use the blockchain as a substitute of the log server to record operations of certificates.

In summary, PISTIS focuses on fundamental issues of certificate issuance, i.e., mandatory domain ownership validation for applicants and disallowing CA-side initiated requests. By contrast, most existing work still focus on monitoring issued certificates and providing accountability in the form of log server, which cannot prevent issuing unauthorized certificates and even further MitM attacks.

9 CONCLUSION

In this paper, we propose PISTIS to address the problem that traditional CAs are vulnerable to be compromised to

issue unauthorized certificates. PISTIS leverages TEE and smart contract to ensure that only the domain owner can request a certificate for its domain, which guarantees all issued certificates are authorized. Previous work either only detects unauthorized certificate or decreases its possibility. PISTIS not only ensures its issued certificates are authorized, but also provides a new trust paradigm. Users can trust a CA based on its execution logic rather than its identity, and PISTIS-issued certificates are trusted as they are recorded on blockchain. To the best of our knowledge, this is the first attempt to prevent issuing unauthorized certificates. Our security analysis and experiment results prove the security and feasibility of PISTIS.

ACKNOWLEDGMENTS

This paper is partially supported by HK RGC GRF PolyU 15217321 and 15216220.

REFERENCES

[1] Wikipedia, "DigiNotar Unauthorized Certificates," 2011. [Online]. Available: https://en.wikipedia.org/wiki/DigiNotar

[2] S. M. Kerner, "Google Hit Again by Unauthorized SSL/TLS Certificates," 2015. [Online]. Available: https://www.esecurityplanet.com/browser-security/google-hit-again-by-unauthorized-ssltls-certificates.html

[3] D. GOODIN, "Google warns of unauthorized TLS certificates trusted by almost all OSes," 2013. [Online]. Available: https://arstechnica.com/information-technology/2015/03/google-warns-of-unauthorized-tls-certificates-trusted-by-almost-all-oses/

[4] C. Evans, C. Palmer, and R. Sleevi, "Public key pinning extension for HTTP," RFC Editor, RFC 7469, 2015.

[5] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," RFC Editor, RFC 6962, 2013.

[6] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor, "Accountable key infrastructure (AKI) a proposal for a public-key validation infrastructure," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 679–690.

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

[8] P. Szalachowski, S. Matsumoto, and A. Perrig, "PoliCert: Secure and flexible tls certificate management," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 406–417.

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

[10] S. Somogyi, "Google security blog: Improved digital certificate security," 2015. [Online]. Available: https://security.googleblog.com/2015/09/improved-digital-certificate-security.html

[11] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 270–282.

[12] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2001, pp. 136–145.

[13] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *USENIX Security Symposium*, 2018, pp. 991–1008.

[14] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 2060–2076.

[15] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 2017, pp. 260–289.

[16] S. Gao, Z. Li, Z. Peng, and B. Xiao, "Power adjusting and bribery racing: Novel mining attacks in the bitcoin system," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 833–850.

[17] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[18] J. Yu, V. Cheval, and M. Ryan, "DTKI: A new formalized PKI with verifiable trusted parties," *The Computer Journal*, vol. 59, no. 11, pp. 1695–1713, 2016.

[19] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities" honest or bust" with decentralized witness cosigning," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 526–545.

[20] C. Fromknecht, D. Velicanu, and S. Yakoubov, "Certcoin: A namecoin based decentralized authentication system," MIT, Tech. Rep., 2014.

[21] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, 2017, pp. 35–40.

[22] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 410–426.

[23] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, "Cecoin: A decentralized PKI mitigating mitm attacks," *Future Generation Computer Systems*, vol. 107, pp. 805–815, 2020.

[24] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: Public and efficient certificate audit based on blockchain for tls connections," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2060–2068.

[25] S. Yao, J. Chen, K. He, R. Du, T. Zhu, and X. Chen, "PBCert: Privacy-Preserving Blockchain-Based Certificate Status Validation Toward Mass Storage Management," *IEEE Access*, vol. 7, pp. 6117–6128, 2018.

[26] L. Dykcik, L. Chuat, P. Szalachowski, and A. Perrig, "BlockPKI: an automated, resilient, and transparent public-key infrastructure," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018, pp. 105–114.

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

[28] P. Szalachowski, "Smartcert: Redesigning digital certificates with smart contracts," *arXiv preprint arXiv:2003.13259*, 2020.

[29] G. Kaptchuk, M. Green, and I. Miers, "Giving state to the stateless: Augmenting trustworthy computation with ledgers," in *NDSS*, 2019, pp. 1–15.

[30] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 63–79.

[31] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *USENIX Security Symposium*, 2019, pp. 801–818.

[32] M. Marlinspike, "Trust assertions for certificate keys," 2013. [Online]. Available: http://tack.io/

[33] E. Osterweil, D. Massey, D. McPherson, and L. Zhang, "Verifying keys through publicity and communities of trust: Quantifying off-axis corroboration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 283–291, 2013.

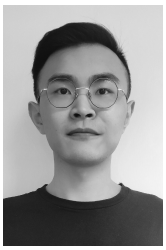
[34] P. Eckersley, "Sovereign keys: A proposal to make HTTPS and email more secure," 2012. [Online]. Available: https://www.eff.org/deeplinks/2011/11/sovereign-keys-proposal-make-https-and-email-more-secure

[35] W. Liu, H. Nishiyama, N. Ansari, J. Yang, and N. Kato, "Cluster-based certificate revocation with vindication capability for mobile ad hoc networks," *IEEE Transactions on parallel and distributed systems*, vol. 24, no. 2, pp. 239–249, 2012.

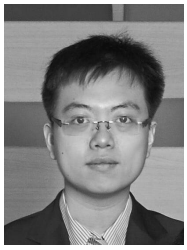
[36] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 393–409.

[37] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda *et al.*, "A Blockchain-based PKI Management Framework," in *The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018*, 2018.

Zecheng Li is currently pursuing his Ph.D. degree in the Department of Computing, The Hong Kong Polytechnic University under the supervision of Dr. Bin Xiao. He received his B.Eng. degree from the School of Information Science and Technology, Southeast University, Nanjing, China, in 2017. His research interest lies in the network security, blockchain security, and smart contract security.



Haotian Wu received the BSc and MSc degrees in computer science from Southeast University, Nanjing, China, in 2015 and 2018, respectively. He is currently working toward the PhD degree in the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests include mobile computing, blockchain systems, and data security.



LapHou Lao received his master degree in information technology from The Hong Kong Polytechnic University. He is currently pursuing his Ph.D. degree at The Hong Kong Polytechnic University. His research interests include blockchain, data privacy, and cryptography.

Songtao Guo received his B.S., M.S. and Ph.D. degrees in Computer Software and Theory from the Chongqing University, Chongqing, China, in 1999, 2003 and 2008, respectively. At present, he is a full professor at Chongqing University, China. His research interests include wireless sensor networks, wireless ad hoc networks and parallel and distributed computing. He has published more than 80 scientific papers in leading refereed journals and conferences. He has received many research grants as a Principal Investigator from the National Science Foundation of China and Chongqing and the Postdoctoral Science Foundation of China.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a SUNY Distinguished Professor of computer engineering and computer science and the Associate Dean for Academic Affairs in the College of Engineering and Applied Sciences at Stony Brook University, New York. Her research interests include data center networks, cloud computing and wireless networks. She has published over 380 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently the Associate Editor-in-Chief for IEEE Transactions on Cloud Computing. She has served as an Associate Editor-in-Chief and Associated Editor for IEEE Transactions on Computers and Associate Editor for IEEE Transactions on Parallel and Distributed Systems. She has also served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences. She is an IEEE Fellow.

Bin Xiao (S'01–M'04–SM'11) is a full professor at Department of Computing, The Hong Kong Polytechnic University, Hong Kong. Dr. Xiao received the B.Sc and M.Sc degrees in Electronics Engineering from Fudan University, China, and Ph.D. degree in computer science from University of Texas at Dallas, USA. After his Ph.D. graduation, he joined the Department of Computing of the Hong Kong Polytechnic University as an Assistant Professor. His research interests include AI and network security, data privacy, and blockchain systems. He published more than 180 technical papers in international top journals and conferences. Currently, he is the associate editor of IEEE IoTJ, IEEE TCC, IEEE TNSE, and Elsevier JPDC. He is the vice chair of IEEE ComSoc CISTC committee. He has been the symposium co-chair of IEEE ICC 2020, ICC 2018 and Globecom 2017, and the general chair of IEEE SECON 2018. He is a senior member of IEEE, the member of ACM and CCF.