Hindawi
Security and Communication Networks
Volume 2021, Article ID 1974493, 17 pages
https://doi.org/10.1155/2021/1974493

Research Article

DIIA: Blockchain-Based Decentralized Infrastructure for Internet Accountability

Pengkun Li,[1] **Jinshu Su**,[2] **Xiaofeng Wang**,[1] **and Qianqian Xing**[1]

[1]*College of Computer, National University of Defense Technology, Changsha 410073, China*
[2]*Science and Technology on Parallel and Distributed Laboratory, National University of Defense Technology, Changsha 410073, China*

Correspondence should be addressed to Xiaofeng Wang; xf_wang@nudt.edu.cn

Received 21 April 2021; Revised 24 June 2021; Accepted 30 June 2021; Published 19 July 2021

Academic Editor: Leandros Maglaras

Copyright © 2021 Pengkun Li 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 Internet lacking accountability suffers from IP address spoofing, prefix hijacking, and DDoS attacks. Global PKI-based accountable network involves harmful centralized authority abuse and complex certificate management. The inherently accountable network with self-certifying addresses is incompatible with the current Internet and faces the difficulty of revoking and updating keys. This study presents DIIA, a blockchain-based decentralized infrastructure to provide accountability for the current Internet. Specifically, DIIA designs a public-permissioned blockchain called TIPchain to act as a decentralized trust anchor, allowing cryptographic authentication of IP addresses without any global trusted authority. DIIA also proposes the revocable trustworthy IP address bound to the cryptographic key, which supports automatic key renewal and efficient key revocation and eliminates complexity certificate management. We present several security mechanisms based on DIIA to show how DIIA can help to enhance network layer security. We also implement a prototype system and experiment with real-world data. The results demonstrate the feasibility and suitability of our work in practice.

1. Introduction

Accountability in the network is a means to identify the sources of traffic for two purposes: to selectively filter abusive or noncompliant traffic from malicious sources on a per-destination basis, while permitting traffic from others to proceed, and to report and disconnect abusive machines before they attack others [1]. The lack of accountability in current Internet architecture facilitates malicious or infected hosts arbitrarily spoof their source addresses to evade responsibility for their malicious packets [2]. The spoofed IP source renders the trigger of distributed denial of service (DDoS) attacks concealed [3]. Prefix hijacking can make web service worldwide unreachable and take a country offline [4, 5].

The crucial problem to achieve Internet accountability is how to bind an IP address or prefix to its owner authentically. Cryptography-based mechanisms have attracted more attention [1]. Numerous existing proposals rely on a centralized public key infrastructure (PKI) and certificates to bind an IP address or prefix to its owner's public key. For instance, resource PKI (RPKI) [6] assigns an IP address block to a public key via a resource certificate. IPA (IP made Accountable) [7] uses DNSSEC as a PKI to certify IP addresses' ownership. However, PKI-based proposals involve the issues of centralized trust and complex certificate management. First, centralized authorities evidently constitute central points of failure, which has been shown to introduce serious security risks and limit the deployability of PKI [8–11]. A misbehaving authority can compromise the security of the entire system or even specific zones by arbitrarily revoking or maliciously substituting certificates under its control [9, 10]. Potential misconfigurations and attacks on authorities also severely hinder the deployment of such PKI [11]. Second, a single trust root is difficult to scale to the current Internet with diverse legal jurisdictions because mutually distrusting and competing nation states cannot reach a consensus on a single root of trust [12]. Third,

certificate management brings additional complexity and costs, which further limits the scalability and deployability of the infrastructure.

To address these problems, researchers propose the self-certifying address to achieve accountability [13, 14]. The self-certifying address is essentially the entity's public key, so it no longer needs a trusted certificate authority to bind the address to the cryptographic key. However, the self-certifying address abandons the current aggregatable IP address space to a flat address space, which turns out to be incompatible with the current Internet protocol and exacerbates the routing scalability problem. In addition, the complexity involved in key updates and the inherent inability to be revoked make it impractical on the current Internet.

Therefore, in this study, we aim for a secure solution that provides decentralized trust, easier key management, and scales to the Internet without changing the current Internet architecture. Identity-based cryptography (IBC) is a feasible cryptographic mechanism to achieve our goal. IBC is a type of public key cryptography in which a public key takes the form of the entity's identity [15]. Based on IBC, using the IP address as its public key can eliminate the use of certificates without changing the current Internet address structure. However, the practical application of the IBC system on the Internet suffers from two challenges. (1) Trust management. IBC uses a trusted party named key generation center (KGC) to create private keys for users. Traditional approaches assume that all users share the same KGC [16], which requires the establishment of a system of global trust and is difficult to implement in practice. Some schemes use hierarchical IBC [17] or DNSSEC [18] to build a multidomain IBC system on the Internet. Unfortunately, these schemes still rely on a global trust root and are vulnerable to single-point failure attacks. (2) Key revocation. Using the IP address as the public key makes it burdensome to revoke the public key when the corresponding private key is compromised. Revoking a public key in IBC also means revoking the identity. It is impractical to prohibit the use of the IP address once the corresponding private key is compromised.

To cope with the aforementioned challenges, this study proposes the Decentralized Infrastructure for Internet Accountability (DIIA). DIIA employs and extends the IBC mechanism to cryptographically authenticate IP addresses and prefixes without introducing any global trusted authority. It provides trusted domain isolation and autonomous control of key distribution of the IP addresses under the jurisdiction of each domain. A public-permissioned blockchain called TIPchain is designed as a decentralized trust anchor in DIIA to support verification of IP addresses in different domains, eliminating reliance on a global trusted authority. As a result, no single third party can undermine the security of the entire system or a specific domain. Furthermore, DIIA supports flexible key revocation by introducing the revocable trustworthy address as the public key of an IP address. With a novel key revocation and status verification mechanism built on TIPchain, the trustworthy address (public key) can be easily revoked without changing the corresponding IP address.

In summary, the main contributions of this work are the following:

(i) We present DIIA, a blockchain-based decentralized infrastructure for the authentication of IP addresses. Without any global trusted authority, DIIA provides autonomous, efficient, and secure key management for large-scale Internet endpoints.

(ii) To eliminate reliance on global trusted authority, we design TIPchain, a globally certifiable public-permissioned blockchain, for managing the authorized addresses allocation and system parameters distribution of trusted domains.

(iii) We design the trustworthy address to facilitate flexible and efficient key revocation. We also propose a trustworthy address revocation and status verification mechanism, which can efficiently revoke a trustworthy address (public key) and verify its revocation status at a trivial cost.

(iv) We present several security modules based on DIIA to show how DIIA achieves network layer accountability.

(v) We implement a proof-of-concept prototype of DIIA based on the Hyperledger Indy project and evaluate its performance based on real-world data. We also conduct a security analysis to demonstrate the security guarantees provided by DIIA.

The rest of the study is organized as follows. Section 2 reviews the related work on achieving Internet accountability. Section 3 introduces the design goal and threat model of our system. We detail the proposed DIIA system in Section 4 and describe the TIPchain design in Section 5. In Section 6, we present several security modules based on DIIA. The performance evaluation and security analysis of our system are presented in Section 7. Finally, Section 8 concludes the study.

2. Related Work

In this section, we review previous proposals to achieve network layer accountability on the Internet. This study follows the definition of network layer accountability similar to that in [1, 7, 13], i.e., the ability to identify the source of a specific packet and take countermeasures in case of misbehaviors. We mainly concern cryptography-based mechanisms which can establish an unforgeable association between the entity and the packet sent through the cryptographic mechanism. The basis of cryptography-based mechanisms is to securely bind an entity's identity to its cryptographic key [7].

2.1. PKI-Based Proposals. Many proposals rely on an extern PKI and certificates to securely establish the binding between network identities and public keys. IPA [7] uses the DNSSEC hierarchy as a PKI to bind the IP prefix to its owner's public key, bringing accountability to the Internet. IPA enables

several security building blocks for accountability, such as the security routing protocol and source authentication system. RPKI [6] constructs a cryptographic hierarchy of authorities that assign an IP prefix to a public key via a resource certificate. It is a prerequisite for many routing security mechanisms and accountability mechanisms [19–21]. Passport [22] establishes shared keys between autonomous systems (ASes) through secure BGP, achieving AS-based source accountability. End-to-end source accountability at the network layer can be achieved through the IPsec protocol [23]. However, complex certificate management and expensive handshake make it challenging to deploy IPsec to the entire Internet. Rothenberger et al. proposed PISKES [20] that allows hosts to obtain a symmetric key locally to perform source address authentication. PISKES requires RPKI to bootstrap its symmetric key infrastructure. Pappas et al. proposed forwarding accountability for Internet reputability (FAIR) [21] which holds transit ASes accountable for the traffic they forward. FAIR also relies on RPKI to enable entities to authenticate the public keys of cooperating ASes. The availability of these PKI-based mechanisms is related to the security and deployability of the PKI on which they depend. However, the centralized trust architecture of PKI has posed severe challenges in terms of security and deployability [8–11].

With the emerging blockchain technology [24–26], researchers have proposed blockchain-based PKIs to alleviate the problem of centralized authority in PKI. Some proposals use the distributed ledger as the public logs to record certificates and operations [27, 28] to monitor the behavior of authorities. However, these external monitoring systems further increase the complexity of certificate management and the cost of certificate verification. Decentralized PKIs (DPKI) based on blockchain [29–31] use decentralized datastores to store and manage public keys, eliminating the dependence on centralized authorities. Unfortunately, DPKI is not suitable for our scenario since the identifiers (IP address and prefix) are not managed by the users themselves.

2.2. Self-Certifying Identifiers. Unlike PKI-based approaches, some proposals use self-certifying identifiers to achieve Internet accountability. The self-certifying identifier of an entity is derived from the public key of that entity. It creates a natural binding between the identity and the public key, thereby avoiding an external certificate authority to attest to this bind. Accountable Internet Protocol (AIP) [13] introduces the self-certifying address to replace the IP address to provide network layer accountability. However, AIP changes the addressing structure and requires a new routing protocol, so it is difficult to deploy on the current Internet. MobilityFirst [32] and Host Identity Protocol (HIP) [33] create a new protocol layer above the network layer and use self-certifying identifiers as the new protocol layer's namespace to allow accountability and seamless mobility. Both of them require a global system to map a self-certifying identifier to a network address securely. In addition to the deployment difficulty, self-certifying identifiers also suffer from the difficulty in revoking or updating keys.

2.3. IBC-Based Proposals. Identity-based cryptography is a public key cryptographic mechanism that uses a user's identity as the public key, which avoids the hassle of public key certification in traditional public key cryptography. Some proposals use the IP address as the public key and generate the corresponding private key through the IBC system [16–18, 34–37], eliminating the need to distribute certificates. Unfortunately, existing IBC-based proposals suffer from scalability and trust management problems regarding the practical deployment of the IBC system on the Internet.

In IBC, private keys are generated by a trusted party called the key generation center (KGC), which produces a set of cryptographic system parameters for key generation. To use the IBC key, the user needs to know the corresponding identity and the public part of system parameters. Some IBC-based proposals, such as T-IP [16], assume that all users in the system will share the same KGC, so that everyone knows the global system parameters. However, deploying a global KGC means establishing a system with global trust. Such a system also holds low scalability as everyone needs to obtain the private key from the same KGC. iPKI [17] utilizes hierarchical IBC (HIBC) [38] to construct an infrastructure for managing private keys of IP addresses. In iPKI, private keys are generated by a set of hierarchically organized KGCs, which can achieve better scalability. However, in the hierarchical organization, high-level KGCs can recover private keys of low-level KGCs, resulting in a system still requiring global trusted authority. In order to distribute public system parameters between different domains with different KGCs securely, an external authority (such as PKI) is generally needed to attest to the parameters. Smetters and Durfee proposed DNSIBC [18] that uses the DNSSEC infrastructure to distribute the public parameters of different domains. Essentially, these schemes all rely on a centralized trust and are vulnerable to single-point-failure attacks.

3. Problem Statement

3.1. Design Goals. Our goal in this work is to design a system to enable accountability in the current Internet by providing cryptographically verifiable IP addresses or prefixes. This ability allows hosts or domains to prove their ownership of the addresses they claim to have. The main objectives we want to achieve are as follows:

(i) Decentralized trust model: we aim for a solution that does not rely on any global trusted authority (such as centralized PKI) to control the trust in IP address authentication. Authentication relies on local trust managed by the authority of the network that controls the IP address. This enables limiting the scope of authorities and preventing single points of failure.

(ii) Efficient key update and revocation: an IP address's key can be rapidly updated and revoked without affecting the normal use of the IP address even after key loss or compromise.

(iii) Scalability: the system should scale to the global Internet and adapt to the heterogeneity of today's Internet partition and address jurisdiction.

(iv) Deployability: the system should be compatible with the current Internet and is gradually deployed on the Internet. The deployment of the system should not affect the existing Internet infrastructure or require changes to the current protocols. Early adopters should obtain immediate security benefits from deployment.

3.2. Threat Model. We consider an adversary that aims to spoof an IP address or prefix by obtaining or forging the cryptographic key to pass the authentication. We assume that the attacker can compromise any entity and learn all the cryptographic keys of the compromised entity. He can also control the entity to send any messages. We also assume that the attacker can passively eavesdrop on messages and actively tamper with the communication by injecting, dropping, delaying, or altering packets.

However, we assume the adversary has no effective method to break cryptographic primitives. Also, we do not consider directly destroying the IBC algorithm to obtain the key, which is out of the scope of this study. We also assume that there exists a secure channel between end hosts and the IP-KGC within the same trusted domain to distribute the cryptographic keys. Additionally, we assume that an adversary cannot control more than one-third of the blockchain nodes.

4. The Proposed DIIA

4.1. Background

4.1.1. Identity-Based Cryptography. Since DIIA is built on the IBC system, we first briefly review the notion of IBC. The concept of IBC was first proposed by Shamir in 1984 [15]. Compared to public key cryptography in which the public/private key pair is randomly generated by the user, the public key in IBC is calculated from the user's identity and the system's public parameters, while the private key can only be generated by a KGC. Like traditional asymmetric cryptography, IBC contains classic asymmetric schemes, such as identity-based signature (IBS) [39] and identity-based encryption (IBE) [40]. An IBS scheme consists of following four algorithms:

(i) $\text{Setup}(k) \longrightarrow (\text{params}, \text{msk})$: executed by the KGC. It takes a security parameter k as input and returns the system parameters params and a master secret key msk.

(ii) $\text{Extract}(\text{params}, \text{msk}, \text{id}) \longrightarrow \text{prk}_{\text{id}}$: executed by the KGC. It takes params, msk, and the user's identity $\text{id} \in \{0,1\}^*$ as input and outputs the user's private key prk_{id}.

(iii) $\text{Sign}(\text{params}, \text{prk}_{\text{id}}, M) \longrightarrow \sigma$: takes params, the user's private key prk_{id}, and a message M as input and outputs a digital signature σ on message M.

(iv) $\text{Verify}(\text{params}, \text{id}, M, \sigma) \longrightarrow (\text{true/false})$: takes the params, the user's identity id, a message M, and a digital signature σ as input and outputs true if the digital signature is valid or false if it is invalid.

Similar to the IBS scheme, an IBE scheme is also composed of four algorithms. In addition to the same Setup and Extract algorithms as the IBS scheme, there are two other algorithms:

(i) $\text{Encrypt}(\text{params}, \text{id}, M) \longrightarrow C$: takes params, id, and a message M as input and outputs a ciphertext C.

(ii) $\text{Decrypt}(\text{params}, \text{prk}_{\text{id}}, C) \longrightarrow M$: takes params, prk_{id}, and the ciphertext C as input and outputs the message M.

There is a plethora of constructions for IBS and IBE. Due to page limits, we do not include their concrete implementations in this study. In fact, DIIA does not specify which IBC algorithm to use. Each trusted domain in DIIA can autonomously choose its IBC algorithms and system parameters according to its security policy.

4.1.2. Merkle Tree and Its Variants. A Merkle tree is a tree in which each leaf node is labeled with the hash of data, and each nonleaf node is labeled with the hash of the labels of its children [41]. The structure of the tree can efficiently prove that a leaf node (or its data) is in the tree by providing the Merkle proof; for each node on the path from the leaf node to the tree root, the Merkle proof contains the hashes of its sibling nodes that are required to recompute its parent hash. The Merkle proof can be verified by recomputing the root hash along the path from the leaf to the root. The size of a Merkle proof is proportional to the log of the number of leaf nodes of the tree.

The Merkle Patricia tree (MPT) structure is a combination of the Merkle tree and Patricia tree, which constructs a tree of hashes on a dynamic list of key-value pairs. In a MPT, the value is stored in the node, and each level of the path from the root to the node represents one or more characters of the corresponding key. Keys with common prefix share path in the tree. Like Merkle trees, each nonleaf node in a MPT contains the hash of its children. Therefore, the root hash can reflect any changes to the list as it hashes the entire list. Through Merkle proof, the MPT structure can prove the existence of a specific value in the list.

The sparse Merkle tree (SMT) is another variant of the Merkle tree, which can also be used to accumulate a list of key-value pairs into a single root hash. A SMT is a perfect Merkle tree that reserves a distinct leaf node for every possible key. Unlike regular Merkle trees, each value occupies a fixed leaf node in the tree determined by its key. The leaf nodes corresponding to nonmember keys in the tree contain a default NULL value. The SMT is said sparse because if the list contains only sparsely distributed keys, the vast majority of leaves represent nonmembers (containing NULL value). This results in a construction in which root nodes of empty subtrees (all leaves of the subtree contain NULL) with the same height will derive identical constant

digests. Based on this structural feature, the SMT can be effectively represented [42]. The SMP can use the Merkle proof to provide convenient proofs of noninclusion: the proof that a key is not included in the list is the proof that the value for the key is NULL.

4.2. System Architecture. The DIIA design assumes that the Internet consists of a number of independently administrated networks, just like the current Internet. Each administrated network can be decomposed into one or more trusted domains, each of which is allocated an IP address range. In DIIA, a trusted domain is treated as an independent trust unit responsible for managing cryptographic keys of addresses under the jurisdiction of the domain. Such scoping of trust within a trusted domain can improve system security, as the compromise of the trust of a domain cannot harm operations outside the domain. Moreover, DIIA maintains a blockchain called TIPchain to guarantee the global verifiability of IP addresses in different domains. Specifically, TIPchain is responsible for managing the authorized addresses allocation and system parameters distribution of trusted domains. It provides a consensus state of the mapping between the IP addresses controlled by the domain and its system parameters, which are indispensable for authenticating addresses of the domain.

Figure 1 shows a high-level overview of DIIA architecture. There are three kinds of entities in DIIA:

(i) TIPchain nodes: TIPchain nodes cooperatively maintain the global ledger and state of TIPchain. They receive domains' system parameters from IP-KGC, encapsulate the parameters into transactions, and writes transactions into blocks. Besides, TIPchain provides query services for all Internet users. By sending query requests to TIPchain nodes, users can obtain the system parameters for authenticating a certain IP address.

(ii) IP-KGC: each trusted domain operates an IP-KGC, which manages the private keys of hosts connected to the domain through IBC. Specifically, IP-PKG selects the IBC algorithm and generates system parameters according to its domain's security policy and generates a private key for the host based on the host's IP address. Besides, IP-KGC records its system parameters on TIPchain by sending a transaction request to TIPchain nodes. Its system parameters will be acquired by other domains for authenticating the IP addresses of its domain.

(iii) Host: the host is connected to a trusted domain and desires to communicate with others. He can obtain a private key from the IP-KGC of his domain and use the key for authenticating his IP address to the communication peer.

4.3. Key Generation. In DIIA, each trusted domain runs an IP-KGC to manage the cryptographic keys of the hosts in the domain. When a host is connected to the domain, the IP-PKG creates a private key for the host based on the host's IP address. Generating cryptographic keys includes two phases: domain initialization and key generation.

4.3.1. Domain Initialization. Before the trusted domain can generate cryptographic keys for its hosts, the IP-KGC must be initialized first to generate the cryptographic system parameters. The following steps are required.

First, the domain administrator selects the IBC algorithm to be used according to his security policy. He then initializes the IP-KGC through the Setup function of the chosen algorithm, which generates the system secret key msk and the public system parameters params for a specified security level k:

$$(\text{params}, \text{msk}) = \text{Setup}(k). \tag{1}$$

The msk is only required for generating private keys for hosts and should be stored securely by the IP-KGC. A strong key management system should be adopted to protect the msk because a compromise of the msk means that an attacker can spoof addresses in the domain arbitrarily. The public parameters are required for signature and encryption operations and need to be made available to the local hosts within the domain. As the last step of system initialization, the trusted domain publishes the public parameters on the TIPchain through the PUB_PARMS transaction (Section 5.1), so that hosts within remote domains can acquire the public parameters when needed.

4.3.2. Trustworthy Address as Public Key. After the system parameters initialization has been completed, the IP-KGC can generate private keys for hosts in its trusted domain. The private key is derived from the IP address assigned to the host, thereby forming a natural binding between the IP address and its private key. However, the IP address cannot be directly used to generate the private key for the following reasons. First, since the IP address is not fixed on the host in the network with a dynamic address assignment, when an address is assigned to a host, the corresponding private key may be maliciously stored by an assailant who obtained the same IP address before. In addition, revoking the key in the IBC system also means revoking the corresponding identity. This means that the compromised key cannot be revoked unless the corresponding IP address is abandoned. Similarly, each key update needs to assign a new address to the host.

To address the above limitations, the IP-KGC derives the private key from an extension of the IP address, which we call the trustworthy address. The trustworthy address extends the IP address with other temporary information to support flexible and efficient key management. For an IP address or prefix ip, the corresponding trustworthy address tip has the following form:

$$\text{tip} = \text{ip} \| \{\text{host}\} \| \text{val\_time} \| \text{exp\_time}, \tag{2}$$

where host contains the information about the current owner of the IP address, which binds the address to its owner. It is optional and can contain diverse information in different cases. val_time specifies the time after which the

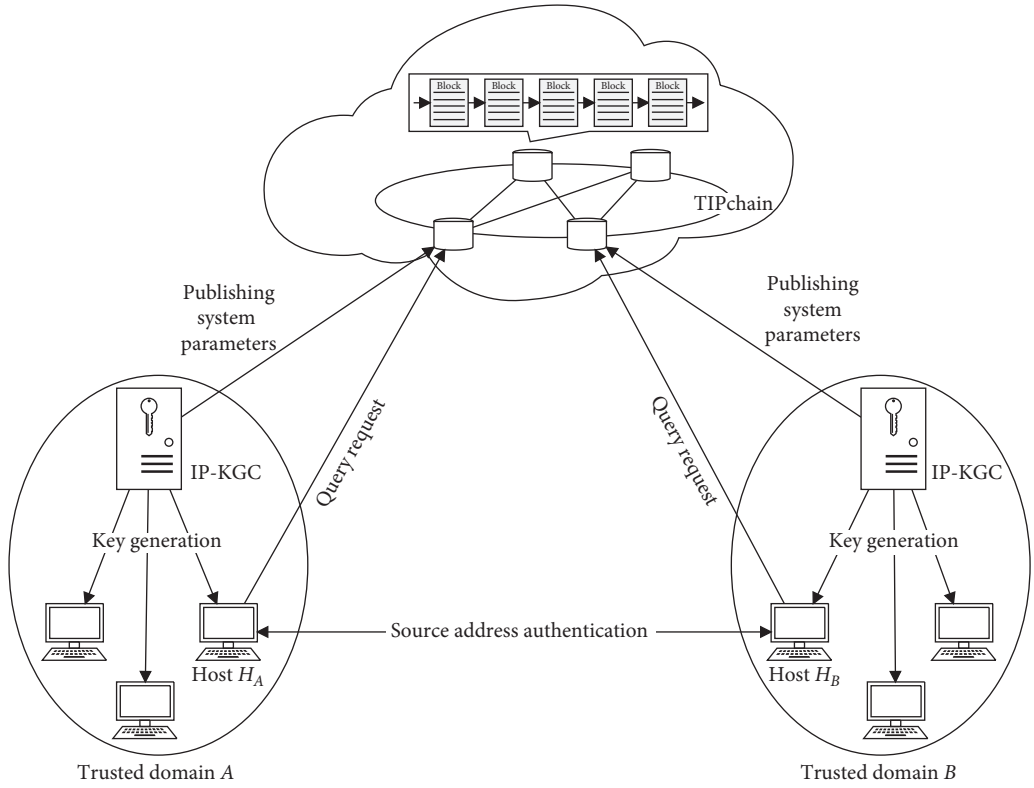

Figure 1: Overview of DIIA architecture.

private key is valid. It is usually set to the time when the private key is generated but may also be set to a future time. exp_time denotes the actual expiration time of the key. (exp_time – val_time) and thus represents the validity period of the key.

The private key corresponding to an IP address is derived from the trustworthy address through the Extract algorithm:

$$\text{prk}_{\text{ip}} = \text{Extract}(\text{params}, \text{msk}, \text{tip}). \quad (3)$$

The trustworthy addresses can greatly simplify key management. Key renewal is the same as distributing new keys. When the key is about to expire, the host only needs to send a new key request to the IP-PKG. IP-KGC will reply with a key derived from the trustworthy address with new temporal information attached. Furthermore, the trustworthy address supports implicit key revocation, and the expired key will automatically become invalid. The validity period of private keys is determined by the IP-KGC according to its policy configuration, based on the trade-off between the frequency of key renewal and the chance of compromise.

4.4. Authenticating an IP Address. With the trustworthy address and corresponding private key, the host can prove that he has the claimed IP address by showing his knowledge of the private key. The authentication process exploits the IBS or IBC mechanism. Figure 2 shows a basic authentication mechanism based on IBS with a simple example. In this example, a host H_A desires to communicate with a host H_B, and H_B wants to authenticate the IP address ip_{H_A} of H_A.

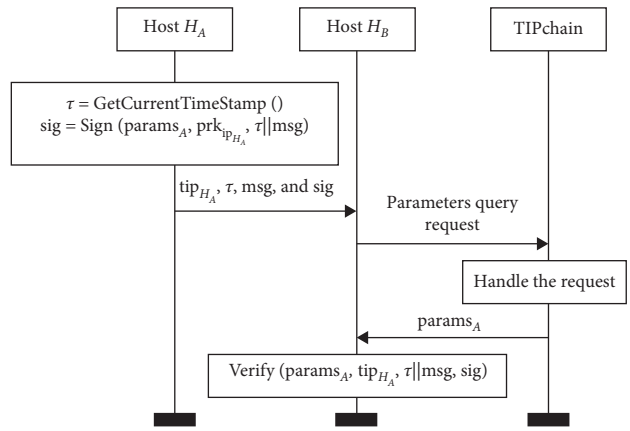

Figure 2: Procedure for IP address authentication based on IBS.

We assume that H_A has obtained a trustworthy address tip_{H_A} and the corresponding private key $\text{prk}_{\text{ip}_{H_A}}$ from the IP-KGC of his trusted domain A.

H_A initializes the authentication process by generating a signature using his private key and sending it to H_B:

$$\text{sig} = \text{Sign}\left(\text{params}_A, \text{prk}_{\text{ip}_{H_A}}, \tau \| \text{msg}\right), \quad (4)$$

where τ denotes a timestamp of the current time. msg is an unnecessary field for authentication, but can be added for other purposes.

Upon receiving the authentication message from H_A, H_B first checks whether τ is a recent timestamp and whether

tip_{H_A} has expired. H_B then looks up the system parameters of the trusted domain A in its local cache. If found, H_B can directly verify the signature from H_A. Otherwise, H_B sends a parameters query request to TIPchain for the system parameters for verifying ip_{H_A}. TIPchain queries the latest system parameters of the domain A from the ledger state (Section 5.2) and sends $params_A$ back to H_B. After receiving the response from TIPchain, H_B can Verify the received signature through the Verify algorithm. If the signature is correct, H_B assured that H_A owns ip_{H_A}.

Note that as the system parameters are domain-specific, users only need to request the parameters of that domain once when verifying the IP addresses from a trusted domain. The authentication mechanism described above is straightforward and basic, and it can be used as a foundation to build more complex mechanisms. We describe several DIIA-based security mechanisms in Section 6, which together can mitigate route hijacking, source address spoofing, and DoS attacks.

4.5. Trustworthy Addresses Revocation and Status Verification. Although the trustworthy address as the public key can support implicit key revocation, a user may revoke his trustworthy address before it expires. This may occur if the IP address is assigned to a new host or the corresponding private key is compromised. However, traditional revocation methods, such as revocation lists or online status checking, require a trusted third party to ensure the authenticity of IP-KGC. In addition, these methods pose a large burden on IP-KGC or verifiers.

DIIA uses a distributed authenticated data structure based on Merkle trees to provide efficient key revocation. Each trusted domain maintains a local trustworthy address state tree, which can be regarded as a summary of the status of trustworthy addresses. The state tree can provide the address holder with a succinct proof of the validity of his trustworthy address, with which the address holder can prove that his trustworthy address has not been revoked. Meanwhile, the trusted domain announces the root of its state tree on TIPchain, allowing anyone to verify the validity of a trustworthy address easily. This revocation mechanism has several benefits:

(1) IP-KGC is freed from the heavy burden of responding to revocation checking requests from all Internet users. Instead, it only needs to respond to proof requests from users in its domain. Besides, since the scope of the state tree only covers the authorized addresses of a domain, an in-memory implementation of the state tree [43] can be used to improve the update and response performance.

(2) Verifiers can easily verify whether a trustworthy address is valid without access to a complete revocation list. Besides, the computational cost of validating a trustworthy address is trivial.

(3) Each domain only records its local trustworthy address state root (a constant-sized hash) on TIPchain, which minimizes the storage cost of the blockchain.

Specifically, IP-KGC can maintain a validity state tree or a revocation state tree through Merkle tree variants. The validity state tree contains a list of valid trustworthy addresses, which is maintained through the MPT structure. The path from root to a leaf in the tree represents an IP address, and the corresponding leaf node contains the latest trustworthy address assigned to the IP address. IP-KGC inserts each newly assigned trustworthy address into the tree with the corresponding IP address as the key and deletes each revoked or expired trustworthy address from the tree. With the validity state tree, proving that a trustworthy address was valid reduces to proving the existence of a certain leaf in the tree.

The revocation state tree is constructed through the SMT structure, which maintains a collection of only the revoked trustworthy addresses. In the revocation state tree, a path still represents an IP address. However, the tree contains a distinct leaf node for every IP address in the domain's authorized address space. If an IP address's trustworthy address is revoked before it expires, it will be inserted into the leaf node corresponding to the IP address. All other leaves in the tree contain a NULL value, indicating that the corresponding IP addresses do not have a revoked trustworthy address. The revocation state tree can provide proof of nonrevocation: a trustworthy address has not been revoked if the leaf node corresponding to the IP address contains a NULL value.

The trusted domain chooses one of the two state trees to maintain its trustworthy address state. It then publishes the root hash of its state tree on TIPchain on a periodic or as-needed basis through the TRUST_ADDR_STATE transaction (Section 5.1). When validating a trustworthy address, the prover requests the IP-PKG for a validity proof or nonrevocation proof and sends the proof (standard Merkle proof) to the verifier. The verifier can then readily verify the validity of the trustworthy address with the state root published on the ledger by performing a simple proof verification.

5. TIPchain Design

In this section, we detail the design of TIPchain, the central component of DIIA. Taking best practices from Ethereum [44], TIPchain is designed to consist of a distributed ledger and a global state. The ledger records transactions that specify the authorized addresses and public system parameters of trusted domains and organizes them as blockchain. TIPchain also maintains a global state that holds the current values of the ledger data. The ledger state makes it easy to retrieve the current system parameters of a domain rather than having to traverse the entire ledger.

5.1. Transactions. There are four types of transactions in TIPchain. The transaction types and the specific data fields of each type are illustrated in Figure 3.

5.1.1. ID_REG Transaction. TIPchain is built on decentralized identities [29–31]. The ID_REG transaction creates a

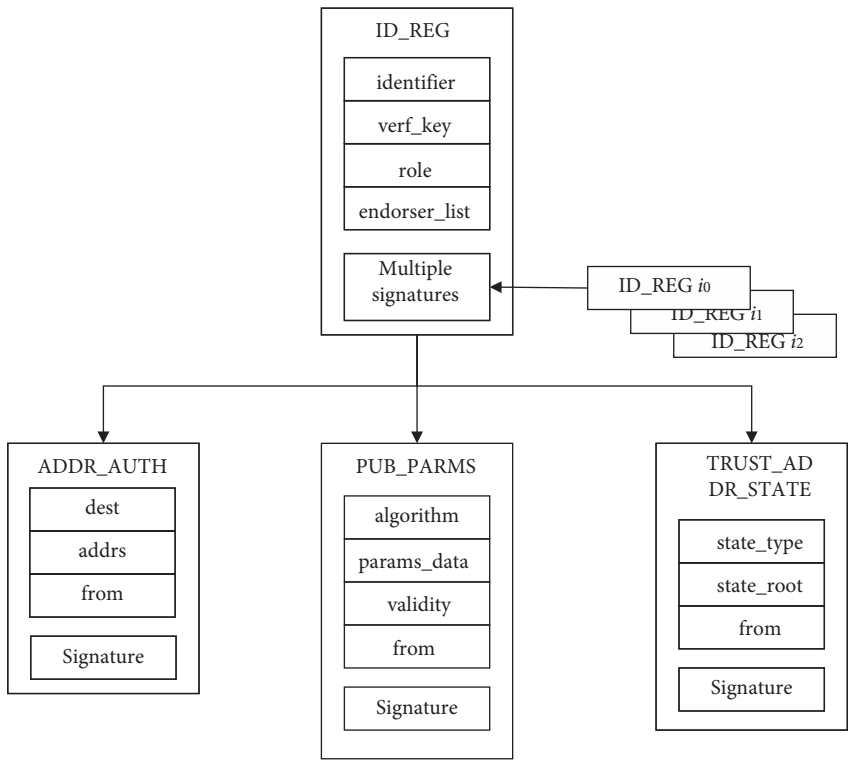

Figure 3: Transaction types and their specific data fields in TIPchain.

new identity record for a specific participant in TIPchain, allowing the participant to write transactions on the blockchain. The identifier field indicates the unique identifier of the participant in TIPchain. Similar to the account address in the traditional blockchain, the identifier is used to identify the initiator of a transaction (the from field in each transaction). Each identifier in TIPchain is explicitly associated with a public/private key pair, where the public part is contained in the ver_key field. A transaction initiator uses its private key to sign its transaction, and the TIPchain node will use the corresponding verification key in ver_key to verify the signature before committing the transaction.

The role field indicates the role of the participant that the ID_REG transaction is being created for. It can have the following values currently: IR, TA, and NONE. IR represents Internet registries responsible for the distribution of network addresses. An identifier with the IR role can send the ADDR_AUTH transaction to authorize address space to another identifier. TA represents trusted domains responsible for managing trustworthy addresses and keys. An identifier with the TA role can write PUB_PARMS and TRUST_ADDR_STATE transactions to publish the domain's system parameters and trustworthy address state. An identifier's role can be changed to NONE, which means to stop it from making any further write operations.

In order to create an initial ID_REG transaction (the first transaction for the creation of an identifier), the participant should designate several entities that he trusts (whose identifiers have been recorded on TIPchain) as its endorsers. The identifiers of these endorsers are contained in the endorser_list field. In addition to being signed by the identifier owner, this initial transaction must also be signed by all endorsers. Endorsers assist in the key recovery of the participant's verification key. This may occur when the participant has lost his key or an attacker has compromised his key and replaced it with a new key under the attacker's control. If an event occurs, the participant can contact his endorsers to sign a new ID_REG transaction to reset his verification key.

When an ID_REG transaction is submitted, if there is no ID_REG transaction for the specified identifier yet, then this transaction is considered to create a new identifier. If there is already an ID_REG transaction with the specified identifier, this is considered an update of that identifier. The participant can update its key (key rotation) by signing a new ID_REG transaction with its old private key and specifying the verf_key field with a new key. In addition, an identifier's endorsers can cooperate to sign a new ID_REG transaction to reset the verf_key, endorser_list, or role value of the identifier.

5.1.2. ADDR_AUTH Transaction. The ADDR_AUTH transaction is used to authorize IP addresses ownership to an identifier. The dest field contains the destination identifier. The addrs field indicates the address block to be allocated to the dest, and this field is represented by the IP prefix. The creation of the ADDR_AUTH transaction follows the current allocation and registration process of Internet addresses. An identifier with role of IR can suballocate portions of its authorized address space to other identifiers. For

example, if a regional IR (RIR) allocates an address block to a national IR (NIR)/local IR (LIR), the RIR will write an ADDR_AUTH transaction in which dest contains the identifier of the NIR/LIR and addrs contains the allocated addresses. Finally, the trusted domain (identifier with role of TA) will be assigned an address block through the ADDR_AUTH transaction. An ADDR_AUTH transaction is valid only if the transaction initiator has ownership of the authorized addresses and has not allocated any subspace of the authorized addresses to any other identifiers.

To revoke the allocated addresses, a special identifier with a value of hash ($\varnothing$) is used in the dest field. In addition, the ADDR_AUTH transaction for revocation must be signed by both identifiers of the assignor and assignee. This regulation prevents the compromised IR from unilaterally revoking the allocated address space and then reallocating the address space to a trusted domain that he forged or is under his control.

5.1.3. PUB_PARMS Transaction. The PUB_PARMS transaction is used for the registration and update of the public system parameters of the trusted domain. A trusted domain specifies the information about its system parameters in the transaction and signs the transaction with the private key corresponding to its identifier. The algorithm field carries the algorithm identifier, which specifies an IBC algorithm. The actual cryptographic parameters are contained in the params_data field. The specific structure of params_data depends on the specified IBC algorithm. The validity field defines the lifetime of the public parameters. It should be set to an interval adequate to protect the corresponding master secret from being compromised. When the validity period is about to end, the domain should generate and publish new system parameters or retain the current parameters unchanged and only extend its validity period.

A trusted domain may generate multiple sets of system parameters for different algorithms, such as algorithms for authentication and encryption, respectively. The algorithm field is used to identify different system parameters issued by the identical identifier on TIPchain. Consequently, the public parameters update is carried out by creating a new PUB_PARMS transaction with the identical initiator and algorithm as the original transaction. The trusted domain can extend the validity period of the public parameters by specifying only the validity field. In this case, the unspecified algorithm and params_data values remain unchanged.

It should be noted that the ledger does not store the specific system parameters. TIPchain uses a separate parameters database to store the actual parameters data. The key of each entry in the database is calculated by hashing the parameters data:

$$\text{key} = \text{Hash}(\text{algorithm} \| \text{params\_data} \| \text{validity}). \quad (5)$$

The PUB_PARMS transaction recorded on the ledger only contains the hash of its specific parameters data, which greatly reduces the ledger size.

5.1.4. TRUST_ADDR_STATE Transaction. The trusted domain uses the TRUST_ADDR_STATE transaction to announce its trustworthy address state. The state_root field contains the root hash of the current state tree. The state_type field defines the trustworthy address validation strategy, and it has the following values: VALIDITY_STATE and REVOC_STATE. VALIDITY_STATE represents the validity state tree while REVOC_STATE represents the revocation state tree (Section 4.5).

5.2. Ledger State. In addition to the ledger that records all valid transactions, TIPchain also maintains a global state, which can be regarded as a projection of ledger data. The state is built from the ledger transactions and is constantly updated as new transactions are recorded. It can provide efficient data retrieval for query requests without having to traverse the entire ledger to search.

TIPchain maintains the global state as a pair of MPTs. The first MPT, which we call StateTree, constructs a mapping between participants and their specific data (similar to the world state in Ethereum [44]). In StateTree, the path is the hash of a participant's identifier, Hash(identifier), and the leaf node contains the following contents:

(i) Authorized_Addresses: (prefix_1, prefix_2, ...), the authorized address space of the identifier that contains a list of address prefixes.

(ii) Public_Parameters: ((alg_1, ParmHash_1), (alg_2, ParmHash_2), ...), a list of mapping from the algorithm identifier to the hash of the specific public parameters data.

(iii) Address_State_Root: (stateType, stateRoot), a mapping from trustworthy address validation strategy to the root hash of trustworthy address state.

Newly committed transactions will cause updates of StateTree. For the ADDR_AUTH transaction, the identifier in dest field is used as the key, and the IP prefix carried in addrs is inserted into Authorized_Addresses. For the PUB_PARMS or TRUST_ADDR_STATE transaction, the identifier of the transaction author is used as the key, and the corresponding parameters hash or trustworthy address state root is updated into Public_Parameters or Address_State_Root of the corresponding leaf node in StateTree.

StateTree keeps the current value of each identifier. The system parameters and trustworthy address state of a trusted domain can be easily obtained from StateTree. However, as the main service of DNS is to look up the IP address for a domain name, the vast majority of DIIA requests will look up the corresponding public parameters or trustworthy address state for a certain IP address or prefix.

In order to efficiently meet such demand, TIPchain maintains another MPT called RetrieveTree, which stores the mapping between address prefixes and identifiers. In RetrieveTree, the path represents the IP address prefix, and the node contains an identifier as its value. The IP prefix represented by the path from the root to a node is the authorized addresses of the identifier corresponding to the node. RetrieveTree is updated with confirmed

ADDR_AUTH transactions. When an ADDR_AUTH transaction is written on the ledger, a new (prefix, identifier) pair is inserted into the tree. When an identifier assigns a subprefix of its authorized address space to another identifier, the node contains the identifier in the tree will become the parent node of the newly inserted node. Therefore, in addition to the hash of its children, an inner node in RetrieveTree may contain an identifier with the role of IR. Each identifier with the role of TA is associated with a leaf node in RetrieveTree.

RetrieveTree can retrieve the latest identifier responsible for an IP address. The get(key) method of RetrieveTree is slightly different from StateTire because it does not necessarily find a value that exactly matches the query key. RetrieveTree performs the get(key) method in the way of longest prefix match. When searching in the RetrieveTree with an IP address as the key, the search path may pass through multiple nodes containing identifiers. The identifiers contained in the node at the bottom of the path will be returned. If the role of the identifier is TA, the trusted domain corresponding to the identifier owns the IP address being queried.

5.2.1. System Parameters and Trustworthy Address State Query. With the ledger state, TIPchain can efficiently process query requests. Users who want to query the system parameters or trustworthy address state corresponding to an IP address send a query request to the TIPchain node. After validating the request, the TIPchain node directly looks up the corresponding value from the ledger state. It first uses the target IP address as the key to retrieve the identifier of the trusted domain responsible for this address from RetrieveTree. Then, it uses the retrieved identifier to look up the corresponding value from StateTree. If the user is querying the system parameters, the node will also look up specific public parameters data from the parameters database according to the parameters hash. TIPchain also generates a state proof for the query result, which contains the root hash of StateTree and the Merkle proof of the value. With the help of state proof, the user can verify the query result before accepting it. Algorithms 1 and 2 show the pseudocode for the TIPchain node to look up the public parameters and for the user to verify the query result, respectively.

5.3. Block and Genesis Block. TIPchain nodes collect transactions and generate blocks appended to the blockchain. Each block consists of a header and a sequence of transactions, which are hashed and organized as a Merkle tree. A block header includes the following: (1) the block number, an integer indicating the position of the block in the blockchain, (2) the hash of the previous block header that links the present block to its parent, (3) the hash of the root node of the Merkle tree constructed from the included transactions, (4) the hash of the root node of the StateTree and RetrieveTree after all transactions are executed and applied, and (5) the timestamp indicating when the block was created.

In the genesis block of TIPchain, several ID_REG transactions are included to create identifiers for the initial Internet registries including IANA and five RIRs. Each of these transactions is signed by all initial Internet registries. That is, the identifiers in the genesis block are self-certified by the initial Internet registries cooperatively. The genesis block also contains ADDR_AUTH transactions signed by the IANA to allocate addresses to RIRs.

5.4. Consensus Algorithm. Blockchain nodes run the consensus algorithm to achieve distributed consensus on the validity and order of transactions. There are various consensus algorithms for blockchains [45], such as proof-of-work (PoW), proof-of-stake (PoS), delegated PoS (DPoS), and Practical Byzantine Fault Tolerance (PBFT). PoW [46] is the most well-known consensus method for public blockchains such as Bitcoin and Ethereum. It relies on the computing power competition between distributed nodes to ensure the consistency and security of blockchain data. It is more suitable for permissionless blockchains that allow any node to join and leave freely. However, as a consensus algorithm for public infrastructure, PoW is too costly since the computing power competition consumes a lot of computing resources and electric energy. In addition, it has low performance in terms of transaction latency and throughput [47].

In this study, we propose to build TIPchain as a public-permissioned blockchain [48]. The nodes constituting the TIPchain network are distributed around the world and operated by trusted institutions in multiple countries or organizations. We envision that the initial node operators include IANA, all RIRs, and several NIRs and LIRs distributed in different regions and countries. All the nodes cooperate and monitor each other to maintain TIPchain. Taking advantage of the permission of nodes, we can choose a consensus algorithm that is energy-efficient and computationally sustainable and has been publicly vetted. In the initial implementation of TIPchain, we adopt Redundant Byzantine Fault Tolerance (RBFT) [49], belonging to a family of PBFT, as the consensus algorithm of TIPchain, which can provide better performance and deterministic transaction finality [47].

However, the RBFT consensus algorithm has poor scalability in terms of the number of blockchain nodes [47]. To overcome this hurdle, we divide TIPchain nodes into two categories like other permissioned blockchains: a limited number of validator nodes, which accept transactions and run the consensus protocol to validate new transactions, and a large number of observer nodes, which maintain read-only copies of the TIPchain to handle query requests. Validator nodes are operated by the abovementioned trusted institutions, while each trusted domain can run an observer node to process query requests from local hosts. Division of node functions

Input: QueryRequest, the query request for specific public parameters, which indicates the target IP address IP_{Target} and the expected IBC algorithm Alg_{Target}
Output: result
(1) identifier←RetrieveTree. get (IP_{Target})
(2) path←Hash (identifier)
(3) value, StateProof←StateTree. getWithProof (path)
(4) ParmHashList←Extract{value}
(5) ParmHash←ParmHashList. get (Alg_{Target})
(6) **if** ParmHash ≠ NULL then
(7) PubParm←ParmStore. get (ParmHash)
(8) **else**
(9) PubParm←NULL
(10) result← (PubParm, identifier, value, StateProof)
(11) **return** result

Algorithm 1: Pseudocode to look up the public parameters for a query request.

Input: result, the query result returned from Algorithm 1.
Output: $b \in$ {**true**, **false**}. (The public parameters is valid or not.)
(1) Alg, $T_{Validity}$←Extract{PubParm}
(2) **if** $Alg_{Target} \neq$ Alg **or** IsExpired ($T_{Validity}$) = **true then**
(3) **return false**
(4) AuthAddrs, ParmHashList←Extract{value}
(5) **if** IsWithin (IP_{Target}, AuthAddrs) = **false then**
(6) **return false**
(7) ParmHash←ParmHashList. get (Alg)
(8) **if** Hash (PubParm) = ParmHash **then**
(9) **if** VerifyProof (identifier, value, StateProof) = **true then**
(10) **return true**
(11) **else return false**
(12) **else return false**

Algorithm 2: Pseudocode to verify the validity of the public parameters queried from TIPchain.

distributes the request load in different trusted domains and enables query requests to be handled locally, thereby greatly reducing the network latency of requests.

6. Use of DIIA

In this section, we describe how DIIA enables various lightweight security modules that can jointly enhance network layer security.

6.1. In-Band Route Origin Verification. DIIA can generate corresponding cryptographic keys for IP prefixes. Using the prefix key, the AS can provide evidence that it is authorized to originate a specific prefix. We describe an in-band route origin verification (IBROV) mechanism through which route advertisements can be authenticated as originating from authorized ASs, thereby effectively mitigating routing hijacking. Compared with the route origin validation that relies on RPKI, IBROV introduces trivial overheads of verification because it eliminates complicated credential management and does not require extra out-of-band credentials transmission.

In IBROV, the AS requires its IP-KGC to generate a cryptographic key for the IP prefix that it is authorized to originate. The prefix key establishes a firm binding between the public key, autonomous systems, and the prefix:

$$\begin{aligned} \mathrm{tip}_{\mathrm{prefix}} &= \mathrm{prefix} \| \mathrm{ASN} \| \mathrm{val\_time} \| \mathrm{exp\_time}, \\ \mathrm{prk}_{\mathrm{prefix}} &= \mathrm{Extract}\left(\mathrm{params}_{\mathrm{AS}}, \mathrm{msk}_{\mathrm{AS}}, \mathrm{tip}_{\mathrm{prefix}}\right). \end{aligned} \tag{6}$$

The AS then uses prk_{prefix} to sign a route origin authorization which consists of its AS number ASN, the validity period information, and a signature:

$$\mathrm{sig} = \mathrm{Sign}\left(\mathrm{params}_{\mathrm{AS}}, \mathrm{prk}_{\mathrm{prefix}}, \mathrm{ASN} \| \mathrm{val\_time} \| \mathrm{exp\_time}\right). \tag{7}$$

When the AS advertises an UPDATE message to announce the prefix, the signed route origin authorization is piggybacked as a path attribute of the UPDATE message. When BGP routers of other ASes receive the UPDATE message, they can authenticate the prefix by extracting the public key, i.e., tip_{prefix}, from the message and verifying sig. If sig is valid, the BGP routers accept the route originated from the AS. Then, they can cache the mapping between the prefix and its authorized AS to reduce the overhead of subsequent signature verification.

6.2. Interdomain Source Authentication. With the cryptographic key assigned to IP prefix, DIIA can provide support for cryptography-based interdomain source authentication mechanisms, e.g., passport [22]. Prefix keys can be used to establish authenticated symmetric keys between domains, which can further be used to authenticate the packet source address. For example, passport employs a distributed key exchange: each AS piggybacks a Diffie–Hellman (DH) public key in its BGP message and establishes shared keys with other ASs when receiving DH public keys contained in BGP messages from other ASs. With DIIA, an AS can sign its DH public value in the BGP message with its prefix key. Other ASes can confirm that the DH public key is indeed from the source AS by verifying the signature, thereby securely establishing a shared secret with the source AS to enable AS level source address authentication.

To authenticate the packet's source address, the source AS AS_S stamps a message authentication code (MAC) into every packet from its domain as an authentication mark. The MAC for a packet is computed using the shared key $\mathrm{key}_{\mathrm{AS}_S \longrightarrow \mathrm{AS}_D}$ between the source AS AS_S and the destination AS AS_D:

$$\mathrm{mark} = \mathrm{MAC}_{\mathrm{key}_{\mathrm{AS}_S \longrightarrow \mathrm{AS}_D}}(\mathrm{packet}). \tag{8}$$

When the destination AS receives a packet from another AS, it uses the packet's source address to look up the source AS and fetches the shared key with the source AS. Then, it recalculates the MAC using the shared key and compares it with the mark carried in the packet. If the recalculated MAC matches the one in the packet, it can validate the source AS of the packet.

6.3. Lightweight End-to-End Packet Authentication. The current Internet uses the IPsec protocol [23] to authenticate the host's source address and ensure the packet's integrity. However, IPsec has high connection latency and transmission overhead due to credential-based authentication and adds complicated security session states to the stateless IP [16]. With DIIA, we describe a lightweight end-to-end packet authentication mechanism that can resist address spoofing efficiently.

The source host H_S in the trusted domain S desires to authenticate its packets to his communication peer. He first randomly generates a local secret value sv_{H_S}. Then, H_S can use a pseudorandom function (PRF) [20] to derive different symmetric keys for different destinations from sv_{H_S}. To send packets to the destination host H_D in trusted domain D, H_S first generates a symmetric key:

$$\mathrm{key}_{H_S \longrightarrow H_D} = \mathrm{PRF}_{\mathrm{sv}_{H_S}}\left(\mathrm{ip}_{H_D}\right), \tag{9}$$

where the input to the PRF is the IP address of the destination H_D. Using the symmetric key, the source can calculate MAC, $\mathrm{MAC}_{\mathrm{key}_{H_S \longrightarrow H_D}}(\mathrm{packet})$, for each packet he sends to H_D. The MAC calculated covers the immutable part of the packet header and the packet payload.

To authenticate packets he sends, H_S also needs to transport the symmetric key to H_D. H_S computes an encrypted and signed key transport token using their trustworthy addresses:

$$\begin{aligned} \mathrm{ct} &= \mathrm{Encrypt}\left(\mathrm{params}_D, \mathrm{tip}_{\mathrm{ip}_{H_D}}, \mathrm{key}_{H_S \longrightarrow H_D}\right), \\ \mathrm{sig} &= \mathrm{Sign}\left(\mathrm{params}_S, \mathrm{prk}_{\mathrm{ip}_{H_S}}, \tau \| \mathrm{ct}\right), \\ \mathrm{token} &= \mathrm{ct} \| \mathrm{sig} \| \tau. \end{aligned} \tag{10}$$

H_S carries the token in the first packet he sends to H_D, avoiding extralatency for key agreement. After receiving the token, H_D decrypts the token to obtain the symmetric key and caches the key locally. Then, he can verify the MAC of packets from H_S.

7. Experiment and Evaluation

7.1. Implementation. We implemented a prototype of TIPchain based on the Hyperledger Indy project (Hyperledger Indy. https://www.hyperledger.org/use/hyperledger-indy) that provides a distributed ledger purpose built for decentralized identities. Indy-Node (Indy Node. https://github.com/hyperledger/indy-node) written in Python implements the main code of Indy, which contains the implementation of the ledger, state, and RBFT protocol. It also provides plugins that allow for creating custom transactions. Leveraging the plugins, we implement TIPchain transactions and the logic of validating and processing transactions. We also implement the ledger and ledger state of TIPchain and use RocksDB (RocksDB, https://rocksdb.org/) as a key-value database to perform the storage for TIPchain ledger and state. Furthermore, we implement the client programs of IP-KGC and host through Indy-SDK (Indy SDK. https://github.com/hyperledger/indy-sdk), which can generate and send transactions and query requests to TIPchain. Indy-SDK provides interfaces to the client to establish the connection with the blockchain and send requests to it. After receiving the client's request, TIPchain nodes will process the request and send a reply back to the client.

We deployed a local TIPchain network and the client on the machine with a 3.20 GHz Intel i7-8700 processor and 16 GB memory. The TIPchain network contains two nodes, each of which was run on a virtual machine instance with a processor core, 2 GB memory, and Ubuntu 16.04 64 bit operation system. The client and the blockchain network are connected through a local area network to minimize the influence of network delays on the measurement results.

7.2. Performance Evaluation. We evaluate DIIA in terms of the performance of processing transactions and the performance of query requests. We first bootstrap TIPchain with the data from the real world. We extract 70612 ASes and 517064 IPv4 prefix-to-AS mappings from the CAIDA pfx2as dataset (RouteView prefix-to-AS mappings (pfx2as), https://publicdata.caida.org/datasets/routing/routeviews-prefix2as/) that is derived from RouteView BGP monitors. We generate

an identifier for each AS through the ID_REG transaction and produce an ADDR_AUTH transaction for each IP prefix-to-AS mapping. Each identifier also generates a PUB_PARMS transaction and a TRUST_ADDR_STATE transaction. In total, around 720 k transactions are written on TIPchain in the bootstrap phase, and the storage cost is 2.7 GB for a TIPchain node.

7.2.1. Transaction Performance. We first evaluate the performance of processing transactions. During the transaction performance evaluation experiment, we configure the maximum number of transactions that a block can contain as 500 and the time interval for generating a block as three seconds. The number of endorsers in each ID_REG transaction is set to 3.

We perform an experiment in which the client sends a different number of transaction requests to the TIPchain node for each type of transaction. Requests are sent asynchronously, that is, all requests are sent without waiting for a reply from the blockchain. We measured the execution time, throughput, and average latency under different numbers of concurrent requests to evaluate the transaction performance. The execution time is the total time spent by TIPchain to execute and confirm all transactions in the dataset. It is the time taken from the submission of the first transaction to the confirmation of all transactions on TIPchain. The transaction latency is the difference between the completion time and the submission time of a transaction. We measure the average latency of all transactions in the dataset. Throughput is expressed as the number of valid transactions committed by TIPchain per second. It is measured as the average throughput over the execution time.

(1) Execution Time. Figure 4(a) explores the differences in execution time of different transaction types with varying number of transactions. The execution time grows as the number of transactions in the dataset increases. The execution time of the ID_REG, PUB_PARMS, and TRUST_ADDR_STATE transactions are similar, but the TRUST_ADDR_STATE transaction's execution time is slightly lower than the other two types of transactions. The execution time of the ADDR_AUTH transaction is consistently higher than that of the other types of transactions, and the gap between the execution times also grows larger as the number of transactions grows. It is mainly because executing the ADDR_AUTH transaction requires updating two state trees of the ledger state, and the size of RetrieveTree is much larger than StateTree (the storage for RetrieveTree is 1.4 GB, while the storage for StateTree is 0.7 GB after the bootstrap phase), while executing other types of transactions only involves StateTree. For the batch of 10,000 transactions, the ID_REG, PUB_PARMS, and TRUST_ADDR_STATE transactions take 55.95, 54.97, and 53.03 seconds, respectively, while the ADDR_AUTH transaction takes 73.76 seconds.

(2) Transaction Throughput. Figure 4(b) shows the plot of average throughput that each type of transaction experiences with a different number of concurrent transactions. After the number of concurrent transactions exceeds 2,000, the throughput of the four transaction types stabilizes. The throughput of the ID_REG, PUB_PARMS, and TRUST_ADDR_STATE transactions has slight differences. For the batch of 10,000 transactions, their throughput is 178.74, 181.93, and 188.58 transactions/s, respectively. The PUB_PARMS transaction's throughput is slightly lower than the TRUST_ADDR_STATE transaction since it needs to hash the parameters and store them in the parameters database. The throughput of the ADDR_AUTH transaction is the lowest among all transaction types. It is 135.57 transactions/s when the number of concurrent transactions is 10,000.

(3) Transaction Latency. Figure 4(c) shows the average latency of different transaction types with varying number of concurrent transactions. For a single transaction, the average latency of the ID_REG, ADDR_AUTH, PUB_PARMS, and TRUST_ADDR_STATE transactions are 63.18 ms, 65.19 ms, 64.38 ms, and 63.39 ms, respectively. It can be observed that as the number of concurrent transactions grows, the average latency of all types of transactions increases rapidly. Specifically, when the number of concurrent transactions grows from 1,000 to 10,000, the average latency of the four transaction types increases by 7.53 times, 7.29 times, 6.83 times, and 6.24 times, respectively. The transaction latency is greatly affected by the block size. Figure 5 explores the latency of individual PUB_PARMS transactions in the dataset where 5,000 transactions are deployed under different block sizes. It shows that transactions are confirmed in batches of block size. Transaction latency increases as the block size grows because larger blocks bring longer verification time and propagation delay.

7.2.2. Query Performance. In the following experiments, to evaluate the performance of query requests, we mainly focus on the common scenario in which the host queries the system parameters and trustworthy address state corresponding to an IP address or prefix. In TIPchain, read operations differ from transaction operations in which the former capture no changes to the ledger state. On receiving a query request, the TIPchain node first validates the request and then retrieves the requested value from the state. It first retrieves the identifier of a trusted domain responsible for the IP address from RetrieveTree. After that, it uses the identifier to look up the corresponding value and to generate a state proof from StateTree. After the host receives the result, it will use the state proof to verify the validity of the result.

(1) Query Latency. We first conduct an experiment to evaluate the query latency of a single request. In the experiment, the client sends a query request, receives a reply, and verifies the result. We measure the query latency on the client by the time from submitting the request to receiving the reply. We also measure the time for the TIPchain node to find the requested value and the time for the client to verify

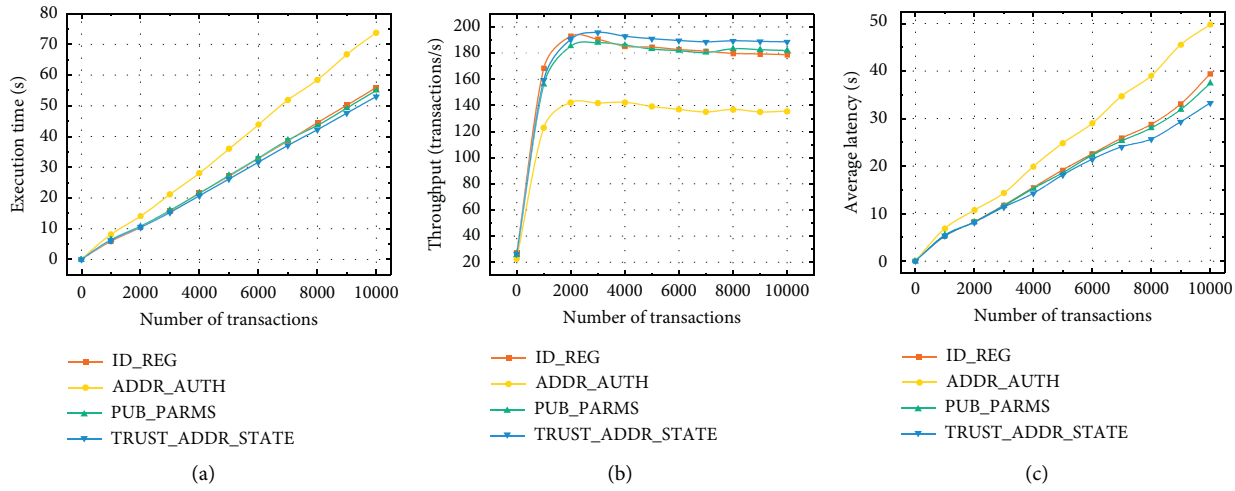

Figure 4: Transaction performance with varying number of transactions. (a) Execution time. (b) Throughput. (c) Average latency.

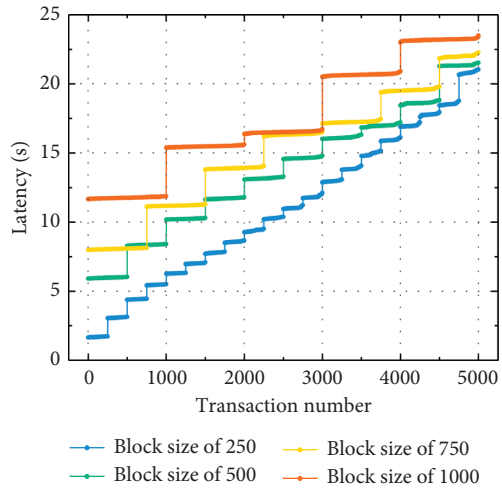

Figure 5: Latency of individual PUB_PARMS transactions with different block sizes.

Table 1: Query latency of a single request.

Request type	Latency (ms)	Node Find value (ms)	Client Verify result (ms)
System parameters	7.16	1.33	0.87
Trustworthy address state	6.77	1.26	0.87

the result. We conducted 2,000 measurements for each type of request and compute the average latency. The results are presented in Table 1. The latencies for querying the public system parameters and querying the trustworthy address state root are 7.16 ms and 6.77 ms, respectively. During the process, the TIPchain node spends 1.33 ms and 1.26 ms, respectively, to find the requested value and generate a state proof. Furthermore, it takes 0.87 ms for the client to verify the result.

(2) Query Throughput. Then, we measured the throughput of processing query requests under a large number of concurrent requests. In this experiment, the client asynchronously sends a batch of query requests to the blockchain node. We measure the total time taken by the TIPchain node to complete these query requests and then calculated the throughput. As Figure 6 shows, the throughput of querying the trustworthy address state is slightly higher than that of querying the system parameters. The difference is mainly due to the data size of system parameters and the requirement of searching the parameters database. Our prototype implementation is able to process around 635 requests for querying public system parameters and 661 requests for querying trustworthy address state root per

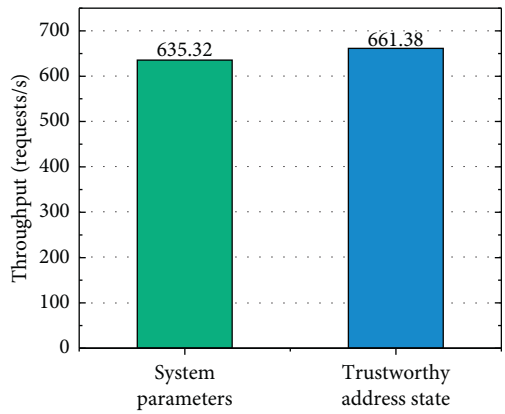

Figure 6: Throughput of processing query requests.

second. Deploying a TIPchain node on a commodity server and processing query requests in parallel can significantly improve the query throughput.

Note that hosts only need to query the system parameters of a trusted domain once since the parameters are global to the domain. By caching query results on the host, the frequency of querying public parameters will be significantly reduced. The introduction of observer nodes can also distribute query requests to different domains and reduce the network latency of the query requests.

7.3. Security Analysis. We analyze the security of DIIA under various attack scenarios as follows. When the entities involved were compromised, we evaluate the impact of the attack and present corresponding countermeasures. In this security analysis, we assume that TIPchain is append-only, so that a confirmed transaction will never be modified or deleted.

7.3.1. Compromised Host. The attacker who compromises a host can access the cryptographic key of the host's IP address. Then, the attacker can spoof the host's IP address by forging signatures for arbitrary messages. DIIA provides efficient key rotation and revocation to protect the host from compromising address keys. The host periodically requests new keys to reduce the risk of key compromise. If the private key is compromised, the host only needs to revoke the compromised key when requesting a new one. With the trustworthy address revocation and status verification mechanism, the attacker can no longer use the compromised key to generate valid signatures.

7.3.2. Compromised IP-KGC. Compromising the master secret of an IP-KGC is a more severe case than compromising the key of a host. An attacker gaining access to msk can recompute the private key that has already been issued or generate a new key for any IP address in the domain. However, compromised IP-KGC only implicates hosts in its trusted domain. Hosts residing in other trusted domains are not affected. Access to msk needs to be strongly protected, and only the IP-KGC is allowed to access it. This can be achieved by employing a hardened security device (e.g., the TPM [50]) or a trusted execution environment (e.g., the Intel SGX [51]). In addition, the trusted domain should update the master secret and public parameters periodically by sending a new PUB_PARMS transaction to TIPchain.

7.3.3. Compromised Trusted Domain. If the trusted domain's private key that is used for transaction signature is compromised, the attacker could publish fraudulent system parameters through the PUB_PARMS transaction. The attacker can then use the fraudulent master secret to generate new keys for the compromised domain's addresses. Furthermore, it could modify the trusted domain's verification key with another key pair through the ID_REG transaction to prevent the domain from updating its public parameters. The trusted domain can detect this attack by querying its public parameters from TIPchain with its identifier regularly. Once the domain observes the fraudulent PUB_PARMS transaction through self-audit, the domain will contact its endorsers to sign a new ID_REG transaction to reset its verification key. Simultaneously, the new ID_REG transaction transforms the role of the domain's identifier to NONE to prevent the fraudulent public parameters instantly.

7.3.4. Compromised Internet Registry. The attacker might compromise the private key of the Internet registry to send an ADDR_AUTH transaction, which authorizes an address prefix to an identifier under its control. By issuing public system parameters on TIPchain, the attacker can control the cryptographic keys of this address space. However, this has no effect on the IP addresses of existing trusted domains because the ADDR_AUTH transaction to authorize an assigned address prefix will not pass the verification. Meanwhile, the malicious identifier can be easily detected through public audits. Moreover, the attacker cannot revoke the address space that has been allocated to a trusted domain and then authorize it to its identifier since the ADDR_AUTH transaction for revocation requires the signature from the trusted domain.

In summary, the adversary has no way of obtaining the cryptographic key of an IP address except by directly compromising the host or the IP-KGC of the trusted domain. In the worst case where an attacker compromises the IP-KGC, he is able to spoof the IP address of a host located within the trusted domain. However, end hosts located outside the trusted domain are not affected. This will incentivize trusted domains to protect IP-KGC security better to maintain their reputation. Furthermore, the public parameters must be recorded on TIPchain. As a result, the trusted domain can rapidly detect the illegal public parameters through self and public audits and take measures to prevent attacks from deteriorating.

8. Conclusion

In this study, we present DIIA, a blockchain-based decentralized infrastructure for achieving accountability on the

current Internet. DIIA is built from the IBC mechanism but does not introduce any global trusted authority. It employs TIPchain as a decentralized trust anchor to provide cryptographic authentication for the IP address and prefix, avoiding reliance on a global trusted authority. It also eliminates the complexity and cost of public key management presented in PKI-based approaches and provides efficient key management through the flexible design of trustworthy addresses. In particular, a lightweight key revocation and status verification mechanism is designed to efficiently revoke a public key and verify its revocation status with trivial cost. We present several DIIA-based security modules that can collectively deal with various network layer attacks. We have evaluated the performance of DIIA based on real-world data, and the experimental results on the prototype system demonstrate its feasibility and suitability in practice. We expect future work on the consensus algorithm of TIPchain to guarantee stronger scalability and security of the system.

Data Availability

The data used to support the findings of this study are included in this article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (62002374).

References

[1] L. He, Y. Liu, and G. Ren, "Network-layer accountability protocols: a survey," *IEEE Access*, vol. 6, pp. 66886–66902, 2018.

[2] CAIDA's Spoofer Project, "State of IP spoofing," 2021, https://spoofer.caida.org/summary.php.

[3] S. Day and M. Thompson, "Mirai attack on DYN internet infrastructure," 2016, https://coar.risc.anl.gov/mirai-attack-dyn-internet-infrastructure/.

[4] D. Madory, "Large European routing leak sends traffic through China telecom," 2019, https://blog.apnic.net/2019/06/07/large-european-routing-leak-sends-traffic-through-china-telecom/.

[5] Ripe Network Coordination Centre, "Youtube hijacking: a ripe NCC RIS case study," 2008, https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study.

[6] R. Bush and R. Austein, "The resource public key infrastructure (RPKI) to router protocol," 2017, https://rfc-editor.org/rfc/rfc8210.txt.

[7] A. Li, X. Liu, and X. Yang, "Bootstrapping accountability in the internet we have," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp. 155–168, USENIX Association, Boston, MA, USA, March 2011.

[8] B. Rothenberger, D. E. Asoni, D. Barrera, and A. Perrig, "Internet kill switches demystified," in *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6, Association for Computing Machinery, Belgrade, Serbia, April 2017.

[9] E. Heilman, D. Cooper, L. Reyzin, and S. Goldberg, "From the consent of the routed: Improving the transparency of the RPKI," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, pp. 51–62, Association for Computing Machinery, Chicago, IL, USA, August 2014.

[10] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg, "On the risk of misbehaving RPKI authorities," in *Proceedings of the 12th ACM Workshop on Hot Topics in Networks*, pp. 1–7, Association for Computing Machinery, College Park, MD, USA, November 2013.

[11] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman, "Are we there yet? On RPKI's deployment and security," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017*, San Diego, CA, USA, 2017.

[12] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The scion internet architecture," *Communications of the ACM*, vol. 60, no. 6, pp. 56–65, 2017.

[13] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable internet protocol (AIP)," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, pp. 339–350, Association for Computing Machinery, Seattle, WA, USA, 2008.

[14] D. Naylor, M. K. Mukerjee, and P. Steenkiste, "Balancing accountability and privacy in the network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 75–86, 2014.

[15] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of the 1984 Workshop on the Theory and Application of Cryptographic Techniques*, pp. 47–53, Springer, Santa Barbara, CA, USA, 1984.

[16] X. Wang, H. Zhou, J. Su, B. Wang, Q. Xing, and P. Li, "T-ip: a self-trustworthy and secure internet protocol," *China Communications*, vol. 15, no. 2, pp. 1–14, 2018.

[17] P. Chen, X. Wang, Y. Wu, J. Su, and H. Zhou, "POSTER: IPKI: identity-based private key infrastructure for securing BGP protocol," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1632–1634, Association for Computing Machinery, Denver, CO, USA, October 2015.

[18] D. K. Smetters and G. Durfee, "Domain-based administration of identity-based cryptosystems for secure email and IPSEC," in *Proceedings of the 12th Conference on USENIX Security Symposium*, USENIX Association, Washington, DC, USA, 2003.

[19] M. Lepinski and K. Sriram, "BGPsec protocol specification," 2017, https://rfc-editor.org/rfc/rfc8205.

[20] B. Rothenberger, D. Roos, M. Legner, and A. Perrig, "PISKES: pragmatic internet-scale key-establishment system," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pp. 73–86, Association for Computing Machinery, Taipei, Taiwan, October 2020.

[21] C. Pappas, R. M. Reischuk, and A. Perrig, "Fair: forwarding accountability for internet reputability," in *Proceedings of the 2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pp. 189–200, IEEE, San Francisco, CA, USA, November 2015.

[22] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: secure and adoptable source authentication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 365–378, USENIX Association, San Francisco, CA, USA, April 2008.

[23] K. Seo and S. Kent, "Security architecture for the internet protocol," 2005, https://rfc-editor.org/rfc/rfc4301.txt.

[24] C. Esposito, M. Ficco, and B. B. Gupta, "Blockchain-based authentication and authorization for smart city applications," *Information Processing & Management*, vol. 58, no. 2, 2021.

[25] Mamta, B. B. Gupta, K.-C. Li, V. C. M. Leung, K. E. Psannis, and S. Yamaguchi, "Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system," *IEEE/CAA Journal of Automatica Sinica*, 2021.

[26] G. N. Nguyen, N. H. L. Viet, M. Elhoseny, K. Shankar, B. B. Gupta, and A. A. A. El-Latif, "Secure blockchain enabled cyber-physical systems in healthcare using deep belief network with ResNet model," *Journal of Parallel and Distributed Computing*, vol. 153, pp. 150–160, 2021.

[27] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[28] S. Matsumoto and R. M. Reischuk, "IKP: turning a PKI around with decentralized automated incentives," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 410–426, IEEE, San Jose, CA, USA, May 2017.

[29] C. Allen, A. Brock, V. Buterin et al., "Decentralized public key infrastructure. a white paper from rebooting the web of trust," 2015, https://www.weboftrust.info/downloads/dpki.pdf.

[30] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: a global naming and storage system secured by blockchains," in *Proceedings of the 2016 USENIX Annual Technical Conference*, pp. 181–194, USENIX Association, Denver, CO, USA, June 2016.

[31] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.

[32] A. Venkataramani, J. F. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, "Mobilityfirst: a mobility-centric and trustworthy internet architecture," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 74–80, 2014.

[33] R. Moskowitz and M. Komu, "Host identity protocol architecture," 2019, https://datatracker.ietf.org/doc/html/draft-ietf-hip-rfc4423-bis-20.

[34] T. Markmann, T. C. Schmidt, and M. Wählisch, "Federated end-to-end authentication for the constrained internet of things using IBC and ECC," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 603-604, 2015.

[35] C. Schridde, M. Smith, and B. Freisleben, "TrueIP: prevention of IP spoofing attacks using identity-based cryptography," in *Proceedings of the 2nd International Conference on Security of Information and Networks*, pp. 128–137, Association for Computing Machinery, London, UK, October 2009.

[36] P. Li, J. Su, and X. Wang, "ITLS: lightweight transport-layer security protocol for IOT with minimal latency and perfect forward secrecy," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6828–6841, 2020.

[37] P. Li, J. Su, and X. Wang, "iTLS/iDTLS: lightweight end-to-end security protocol for iot through minimal latency," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pp. 166–168, Association for Computing Machinery, Beijing, China, August 2019.

[38] C. Gentry and A. Silverberg, "Hierarchical ID-based cryptography," in *Proceedings of the 2002 International Conference on the Theory and Application of Cryptology and Information Security*, pp. 548–566, Springer, Queenstown, New Zealand, 2002.

[39] F. Hess, "Efficient identity based signature schemes based on pairings," in *Proceedings of the 2002 International Workshop on Selected Areas in Cryptography*, pp. 310–324, Springer, St. John's, Canada, 2002.

[40] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of the 2001 Annual International Cryptology Conference*, pp. 213–229, Springer, Santa Barbara, CA, USA, 2001.

[41] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proceedings of the 1987 Conference on the Theory and Application of Cryptographic Techniques*, pp. 369–378, Springer, Santa Barbara, CA, USA, 1987.

[42] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees: caching strategies and secure (non-)membership proofs," in *Proceedings of the 21st Nordic Workshop on Secure Computer Systems (NORDSEC 2016)*, pp. 199–215, Springer, Oulu, Finland, 2016.

[43] S. Ponnapalli, A. Shah, A. Tai et al., *Scalable and Efficient Data Authentication for Decentralized Systems*, https://arxiv.org/abs/1909.11590v1, 2019.

[44] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," 2014, https://ethereum.github.io/yellowpaper/paper.pdf.

[45] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[46] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.

[47] M. Vukolić, "The quest for scalable blockchain fabric: proof-of-work vs. BFT replication," in *Proceedings of the 2015 International Workshop on Open Problems in Network Security*, pp. 112–125, Springer, Zurich, Switzerland, 2015.

[48] J. Ruiz, "Public-permissioned blockchains as common-pool resources," 2020, https://hal.archives-ouvertes.fr/hal-02477405/document.

[49] P.-L. Aublin, S. Ben Mokhtar, and V. Quéma, "RBFT: redundant byzantine fault tolerance," in *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, pp. 297–306, IEEE, Philadelphia, PA, USA, 2013.

[50] Trusted Computing Group, "TPM 2.0 library specification," 2019, https://trustedcomputinggroup.org/resource/tpm-library-specification/.

[51] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.