SkyEye: A Traceable Scheme for Blockchain

Tianjun Ma[†‡§], Haixia Xu[†‡§*], Peili Li[†§]
[†]State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
[‡]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[§]Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing, China
{matianjun,xuhaixia,lipeili}@iie.ac.cn

***Abstract*—Many studies focus on the blockchain privacy protection. Unfortunately, the privacy protection brings regulatory issues (e.g., countering money-laundering). Tracing users' identities is a critical step in addressing blockchain regulatory issues. In this paper, we propose SkyEye, a traceable scheme for blockchain. SkyEye can be applied to the blockchain applications that satisfy the following conditions: (I) The users have public and private information, where the public information is generated by the private information; (II) The users' public information is disclosed in the blockchain data. SkyEye enables the regulator to trace users' identities. The design of SkyEye leverages some cryptographic primitives, including chameleon hash and zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK). Moreover, we demonstrate the security of SkyEye under specific cryptographic assumptions. Finally, we implement two prototypes of SkyEye, and evaluate the running time and related data storage requirements by performing the aforementioned prototypes.**

I. Introduction

The blockchain was first introduced in Bitcoin [29], and quickly became the supporting technology of decentralized cryptocurrencies such as Litecoin [1], PPcoin [23], and Nextcoin [3]. The blockchain integrates multiple technologies (e.g., cryptography and peer-to-peer networking) and includes a variety of features: distributed, decentralized, anonymity, transparency, and so on. Today, the blockchain is not only applied in decentralized cryptocurrencies, but also has broad applications in other fields, including defense, finance, and smart contract.

The blockchain can be considered a distributed database that only appends data (e.g., transactions). The data is stored in the block that contains the block header and block body. Every block header includes the hash of the previous block, forming a chain. The strategy of appending a block to the blockchain uses a consensus mechanism such as proof of work (POW) [29], proof of stake (POS) [6], [14], [22], or practical byzantine fault tolerance (PBFT) [11]. In many blockchain applications, every user generally has public/private information (e.g., the public key address and the signature private key for each user in Bitcoin [29], more details about the public/private information are described in Section II-A).

Many studies focus on the blockchain privacy protection [8], [32]. Unfortunately, the privacy protection brings regulatory issues. On one hand, if a user's private information is lost or stolen, the user loses control of the data corresponding to the private information forever. For example, if the Bitcoin's user loses the signature private keys in his wallet, there is no way to recover the coins in this wallet. In other words, the user loses the coins controlled by these signature private keys forever. On the other hand, strong privacy protection in the blockchain facilitates many criminal activities (e.g., ransomware [2], money laundering). CipherTrace's second quarter 2019 cryptocurrency anti-money laundering report shows that the total amount of funds that cybercriminals directly steal, scam, and misappropriate from users and trading platforms is approximately $4.3 billion in aggregate for 2019. These regulatory issues not only present a serious threat to the interests of users, but also have seriously hindered the development and application of the blockchain.

We stress that tracing users' identities is a critical step in addressing blockchain regulatory issues. When each user's identity in the blockchain data is determined, the regulator can conduct some regulatory operations (such as Big Data analysis) to decide who should be punished or who should own the lost data. Although there has been progress in designing traceable mechanisms, such as zkLedger [30] and several others [5], [15], [18], [21], these approaches are designed for specific application environments and do not seem to have been extended to other applications; see Section VIII for more details.

A. Contributions

The main contributions of this paper are the following.

First, we introduce the notion of a traceable scheme for blockchain and formalize the security properties to be satisfied, namely *identity proof indistinguishability and identity proof unforgeability*.

Second, we propose SkyEye, a traceable scheme for blockchain. SkyEye can be applied to the blockchain applications that satisfy the following conditions: (I) The users have public and private information, where the public information is generated by the private information; (II) The users' public information is disclosed in the blockchain data. These blockchain applications are called the **SkyEye-friendly blockchain applications**. SkyEye requires the user to register only once, and enables the regulator to trace users' identities. In our design strategy, we add identity proofs, associated with the users' private information, to the blockchain data. SkyEye is designed by using some cryptographic primitives (including chameleon hash [25] and zk-SNARK [19]). In

*The corresponding author.

addition, we demonstrate the security of SkyEye under specific cryptographic assumptions.

Finally, we implement two prototypes of SkyEye: $SkyEye_H$ and $SkyEye_S$. These correspond to the two primary ways of generating public and private information in the blockchain applications. The first way is through a pseudorandom function, and the second way is using elliptic curve scalar multiplication. We evaluate the running time and related data storage requirements by performing $SkyEye_H$ and $SkyEye_S$. Our evaluation results illustrate that using an i7 processor, a 16 GB RAM desktop machine, and a Merkle tree depth of 34, the time taken by a verifier to verify a user's identity proof is nearly 4.6 ms in the first way and less than 25 ms in the second way.

B. Paper Organization

The remainder of this paper is organized as follows. Section II provides the background. Section III provides key ideas in SkyEye design and an overview of SkyEye. Section IV defines the algorithm and security of the traceable scheme for blockchain. Section V details SkyEye. Section VI describes our implementation and the evaluation results. We discuss remaining issues of SkyEye and future work in Section VII. We discuss related work in Section VIII and summarize this paper in Section IX.

II. Background

A. SkyEye-friendly blockchain applications

We use $\boldsymbol{B_s}$ to denote SkyEye-friendly blockchain applications. Next, we describe the blockchain data in $\boldsymbol{B_s}$ and an overview of $\boldsymbol{B_s}$.

1) The Blockchain Data in $\boldsymbol{B_s}$: $\boldsymbol{B_s}$ satisfies two conditions: (I) The users have public and private information, where the public information is generated by the private information; (II) The users' public information is disclosed in the blockchain data.

We use equation $pub = gen(priv)$ to describe the generation relation in the condition (I), where pub denotes the public information, $priv$ denotes the private information, and $gen(\cdot)$ denotes the generation algorithm between pub and $priv$, which has one-wayness, i.e., it is easy to compute pub using the private information $priv$ but is hard to invert. In many blockchain applications, every user generally has private information that corresponds to public information. For example, the public key address and the signature private key in Bitcoin [29] are the user's public/private information. In Zerocash [8], $(sn,\ (a_{sk}, \rho))$ is the user's public/private information, where sn is the serial number, a_{sk} is the address private key, and ρ is the random number used to generate the serial number. The public information is generated by the private information via a cryptographic method, such as the pseudorandom function, or elliptic curve scalar multiplication.

According to the condition (II), the blockchain data in $\boldsymbol{B_s}$ can be divided into two parts: one part is the users' public information, such as the input/output addresses in Bitcoin [29], and the other part is the data contents, such as the payment amount and the executable contract code. Therefore, the blockchain data in $\boldsymbol{B_s}$ can be represented as the equation $\boldsymbol{data_{B_s}} = [(pub_i)_{i\in\{1,\ldots,n\}}, C]_{crytool}$, where $(pub_i)_{i\in\{1,\ldots,n\}}$ denotes the set of the users' public information, n is the number of the users' public information in the blockchain data, C denotes the data contents, and $crytool$ denotes the cryptographic tools (e.g., digital signature) that guarantee blockchain features such as tamper-resistance and privacy protection.

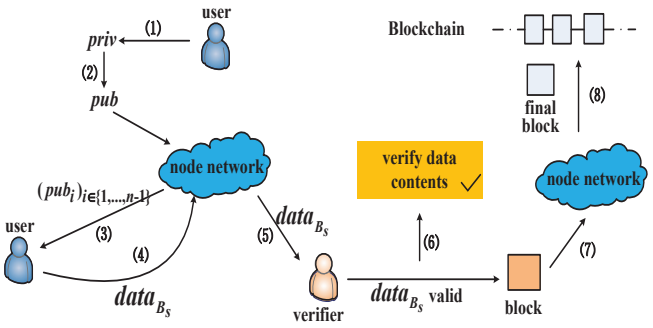

Fig. 1: Overview of $\boldsymbol{B_s}$.

For example, Bitcoin [29], Ethereum [34], and RScoin [13] are the applications that belong to $\boldsymbol{B_s}$. In these blockchain applications, the public key address and the signature private key are the user's public/private information, where the public key address is generated by the signature private key. Moreover, the user's public key address is disclosed in the blockchain data.

2) $\boldsymbol{B_s}$: An overview of $\boldsymbol{B_s}$ is shown in Figure 1. In (1) and (2), the user generates the $(pub, priv)$, and publishes pub to the node network. In (3), for creating data, the user obtains others' $(pub_i)_{i\in\{1,\ldots,n-1\}}$ from the node network. In (4), the user creates $\boldsymbol{data_{B_s}} = [(pub_i)_{i\in\{1,\ldots n\}}, C]_{crytool}$, where the pub_n denotes the user's pub, and publishes $\boldsymbol{data_{B_s}}$ to the node network. In (5) and (6), a verifier receives $\boldsymbol{data_{B_s}}$ from the node network and verifies data contents. If the verification is successful, $\boldsymbol{data_{B_s}}$ is valid and is added to the block generated by the verifier. In (7), the block is published in the node network by the verifier. In (8), according to a consensus mechanism, the nodes in the network select a final block and add it to the blockchain.

B. Cryptographic Preliminaries

The cryptographic building blocks in our construction include the following: chameleon hash scheme, zk-SNARK, and public key encryption. Below, we informally describe these notions.

Chameleon hash scheme. Compared with the traditional hash scheme, the chameleon hash scheme has a special property: the user who knows the trapdoor can easily find collision. A chameleon hash scheme $Chash = (\mathcal{G}_{chash}, \mathcal{K}_{chash}, \mathcal{H}_{chash}, \mathcal{CF}_{chash})$ is described below:

- $\mathcal{G}_{chash}(\lambda) \rightarrow pp_{chash}$. Given a security parameter λ, $\mathcal{G}_{chash}$ returns the public parameters pp_{chash}.

• $\mathcal{K}_{chash}(pp_{chash}) \rightarrow (pk_{chash}, sk_{chash})$. Given the public parameters pp_{chash}, $\mathcal{K}_{chash}$ returns a pair of public/private keys (pk_{chash}, sk_{chash}), where sk_{chash} is also known as the trapdoor.

• $\mathcal{H}_{chash}(pk_{chash}, m, r) \rightarrow CH$. Given the public key pk_{chash}, a message m, and a random number r, $\mathcal{H}_{chash}$ returns a chameleon hash CH about m.

• $\mathcal{CF}_{chash}(sk_{chash}, m, m', r) \rightarrow r'$. Given the trapdoor sk_{chash}, two messages m, m', and the random number r, $\mathcal{CF}_{chash}$ returns r' such that $Chash(pk_{chash}, m, r) = Chash(pk_{chash}, m', r')$.

A chameleon hash scheme satisfies three secure properties: (i) *collision resistance*; (ii) *trapdoor collision*; and (iii) *semantic security*. More details are available in [4], [25].

There is a relationship between the public key pk_{chash} and the trapdoor sk_{chash}, which we refer to as the generation relationship. As in [25], the $pp_{chash} = (p, q, g)$, where p, q are prime numbers such that $p = kq + 1$, and the order of g is q in $\mathbb{Z}_p^*$. The public key $pk_{chash} = h$ is computed as follows: $h = g^x$ mod p, where $x \in \mathbb{Z}_q^*$ is the trapdoor. Let equation $pk_{chash} = chash\_gen(sk_{chash})$ describe this relation, where $chash\_gen(\cdot)$ denotes the generation algorithm between the pk_{chash} and the sk_{chash}.

Zero-knowledge succinct non-interactive arguments of knowledge. Let $\mathcal{R}_{AC} = \{(x, w) \in \mathbb{F}^n \times \mathbb{F}^h | AC(x, w) = 0^l\}$ be an NP relation, where $\mathbb{F}$ denotes a finite field, and $AC : \mathbb{F}^n \times \mathbb{F}^h \rightarrow \mathbb{F}^l$ denotes an $\mathbb{F}$-arithmetic circuit. The language for $\mathcal{R}_{AC}$ is $\mathcal{L}_{AC} = \{x \in \mathbb{F}^n | \exists w \in \mathbb{F}^h \ s.t. \ AC(x, w) = 0^l\}$. A zk-SNARK scheme $NIZK = (\mathcal{K}_{nizk}, \mathcal{P}_{nizk}, \mathcal{V}_{nizk})$ corresponds to the language $\mathcal{L}_{AC}$, which is described below:

• $\mathcal{K}_{nizk}(\lambda, AC) \rightarrow (pk, vk)$. Given a security parameter λ and an $\mathbb{F}$-arithmetic circuit AC, $\mathcal{K}_{nizk}$ returns a pair of proving/verification keys (pk, vk).

• $\mathcal{P}_{nizk}(pk, x, w) \rightarrow \pi$. Given the proving key pk, a statement x, and a witness w, $\mathcal{P}_{nizk}$ returns a proof π for a statement x using a witness w.

• $\mathcal{V}_{nizk}(vk, x, \pi) \rightarrow \{0, 1\}$. Given the verification key vk, the statement x, and the proof π, $\mathcal{V}_{nizk}$ returns 1 if verification succeeds, or 0 if verification fails.

A zk-SNARK scheme satisfies five secure properties: (i) *completeness*; (ii) *soundness*; (iii) *succinctness*; (iv) *proof of knowledge*; and (v) *perfectly zero knowledge*. More details are available in [8], [10], [19].

Public key encryption. A public key encryption scheme $Enc = (\mathcal{G}_{enc}, \mathcal{K}_{enc}, \mathcal{E}_{enc}, \mathcal{D}_{enc})$ is described below:

• $\mathcal{G}_{enc}(\lambda) \rightarrow pp_{enc}$. Given a security parameter λ, $\mathcal{G}_{enc}$ returns the public parameters pp_{enc}.

• $\mathcal{K}_{enc}(pp_{enc}) \rightarrow (pk_{enc}, sk_{enc})$. Given the public parameters pp_{enc}, $\mathcal{K}_{enc}$ returns a pair of public/private keys (pk_{enc}, sk_{enc}).

• $\mathcal{E}_{enc}(pk_{enc}, m) \rightarrow c$. Given the public key pk_{enc} and a message m, $\mathcal{E}_{enc}$ returns a ciphertext c.

• $\mathcal{D}_{enc}(sk_{enc}, c) \rightarrow m$. Given the private key sk_{enc} and the ciphertext c, $\mathcal{D}_{enc}$ returns a message m, or returns $\perp$ if decryption fails.

The public encryption scheme Enc satisfies a security property: ciphertext indistinguishability under adaptive chosen ciphertext attack (IND-CCA2 security). More details are provided in [12].

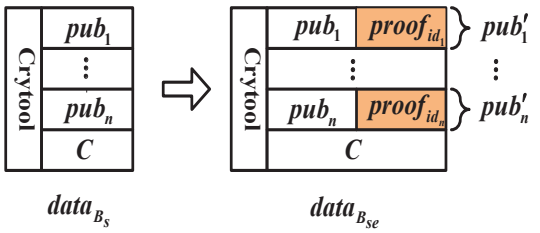

Fig. 2: Design idea.

III. Key ideas and SkyEye Overview

In this section, we provide key ideas in SkyEye design and an overview of SkyEye.

A. Key Ideas

Notation. Let $\boldsymbol{B_{se}}$ denote the $\boldsymbol{B_s}$ using SkyEye. We use u to denote a user, id_u to denote the u's identity and $proof_{id_u}$ to denote the u's identity proof. Let $(pub_u, priv_u)$ denote the u's public/private information, $(pk_{chash_u}, sk_{chash_u})$ denote the chameleon hash public/private key pair that generated by the user u, and CH_{id_u} denote the chameleon hash value of identity id_u. The $pk_{chash_u}||CH_{id_u}$ denotes the concatenation of pk_{chash_u} and CH_{id_u}, where $||$ denotes the concatenate symbol. The $MT = (rt; pk_{chash_1}||CH_{id_1}, ..., pk_{chash_n}||CH_{id_n})$ denotes a Merkle tree, where rt denotes the root of the Merkle Tree, and $(pk_{chash_1}||CH_{id_1}, ..., pk_{chash_n}||CH_{id_n})$ denotes the leaf nodes in the Merkle tree. Let (pk_{reg}, sk_{reg}) denote the encryption public/private key pair of the regulator.

As shown in Figure 2, our design idea is that we add identity proofs to $\boldsymbol{data_{B_s}}$. The blockchain data in $\boldsymbol{B_{se}}$ can be represented as the equation $\boldsymbol{data_{B_{se}}} = [(pub_i, proof_{id_i})_{i \in \{1,...,n\}}, C]_{crytool}$, where the $proof_{id_i}$ denotes the identity proof of the user whose identity is id_i, and the other variables are the same as those in the equation $\boldsymbol{data_{B_s}}$. The $(pub_i, proof_{id_i})$ can be viewed as the new public information pub_i' of the user whose identity is id_i.

The identity proof is the core of SkyEye. The two purposes of the identity proof are to prove the user's legitimacy and to achieve tracing. Next, we briefly describe the identity proof according to the above two purposes. More details are described in Section V.

1) Proving the user's legitimacy: We assume that the user u has generated $(pub_u, priv_u)$, $(pk_{chash_u}, sk_{chash_u})$ and $CH_{id_u} = \mathcal{H}_{chash}(pk_{chash_u}, id_u, r)$, where r is the random number sampled by u.

Step 1: user registration. To prove the user's legitimacy, there must be something (similar to a certificate) that can indicate the user's legitimacy. In SkyEye, this is done through user registration. Here, we briefly introduce user registration in SkyEye. More details on user registration appear in Section V-A2.

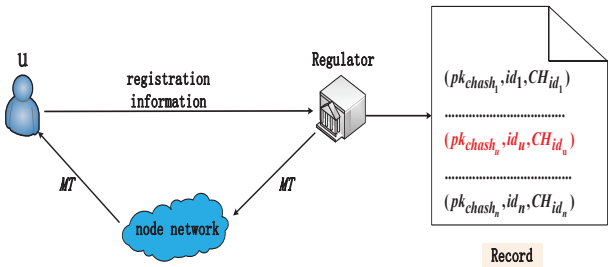

Fig. 3: User registration.

As shown in Figure 3, the user u sends registration information (C_{info}, π_{info}) to the regulator, where the C_{info} is the ciphertext that is the encryption of the plaintext $(pk_{chash_u}, id_u, CH_{id_u})$ under the regulator's public key pk_{reg} and the π_{info} is the zk-SNARK proof that is used to prove: *"I know* (sk_{chash_u}, r) *which can generate the* pk_{chash_u} *and* CH_{id_u}*"*.

If the verification of the (C_{info}, π_{info}) is successful, the regulator stores $(pk_{chash_u}, id_u, CH_{id_u})$, and adds $pk_{chash_u}||CH_{id_u}$ to the Merkle tree MT. The regulator publishes the Merkle tree MT at the right time. The registration of u is successful only if the u's $pk_{chash_u}||CH_{id_u}$ appears in the Merkle tree MT.

The Merkle tree MT can be regarded as a credential of proving the user's legitimacy. In other words, to prove the u's legitimacy, the user u must prove that his or her $pk_{chash_u}||CH_{id_u}$ appears in the Merkle tree MT. Therefore, the $proof_{id_u}$ generated by the user u must be able to prove the following.

"I know (sk_{chash_u}, id_u, r) *that can generate the* pk_{chash_u} *and* CH_{id_u}*, and the* $pk_{chash_u}||CH_{id_u}$ *appears as a leaf of the Merkle tree* MT *with the root rt"*.

Step 2: establishing the binding relationship between the pub_u and $proof_{id_u}$. Although the $proof_{id_u}$ described above can prove the u's legitimacy, an issue remains. It can be seen from Figure 4 that the $(pub_u, proof_{id_u})$ needs to be published in the node network. The adversary who has registered with the regulator can generate an identity proof $proof_{id_{adv}}$ and publish the $(pub_u, proof_{id_{adv}})$ to the node work. At this time, a pub_u corresponds to two different users' identity proofs (i.e., $proof_{id_u}$ and $proof_{id_{adv}}$). This presents a great obstacle to trace. We need to establish a relation between the pub_u and the $proof_{id_u}$ that only the user u can generate the identity proof $proof_{id_u}$ corresponding to the pub_u.

The key idea behind establishing this relation is that we establish the binding relationship between the $priv_u$ and $proof_{id_u}$. Because the pub_u is generated by the $priv_u$, there is binding relationship between the pub_u and $proof_{id_u}$.

We leverage the special property of chameleon hash scheme (i.e., the user who knows the trapdoor can easily find collision) to establish the binding relationship between the $priv_u$ and $proof_{id_u}$. Given the private information $priv_u$, the user u who knows the trapdoor sk_{chash_u} can easily find r' such that $CH_{id_u} = \mathcal{H}_{chash}(pk_{chash_u}, priv_u, r')$. To achieve the binding

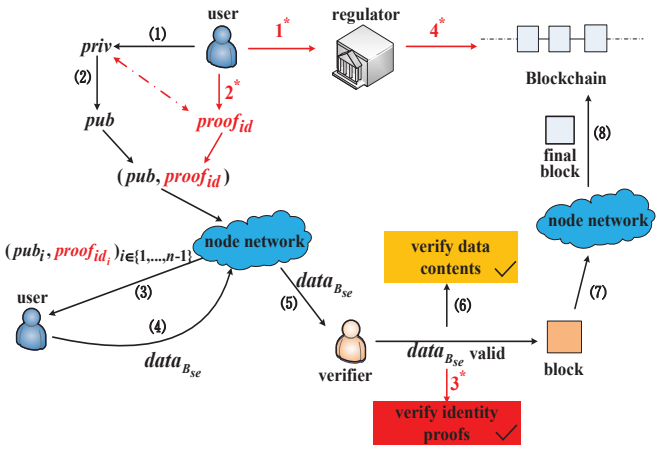

Fig. 4: Overview of $\boldsymbol{B_{se}}$. The red lines **1***, **2***, **3*** and **4*** represent the operations of the SkyEye scheme. The red line **1*** denotes user registration. The red line **2*** denotes generating identity proof. The red line **3*** denotes verifying identity proof. The red line **4*** denotes tracing.

of $priv_u$ and $proof_{id_u}$, we require the identity proof $proof_{id_u}$ to prove the following.

1. The public information pub_u is generated by the private information $priv_u$.
2. I know $(sk_{chash_u}, priv_u, r')$ which can generate the pk_{chash_u} and CH_{id_u};
3. The $pk_{chash_u}||CH_{id_u}$ appears as a leaf of a Merkle tree with the root rt.

This binding relationship between $proof_{id_u}$ and $priv_u$ ensures that only the user u who knows the private information $priv_u$ can generate the identity proof $proof_{id_u}$, and others cannot forge the identity proof corresponding to the pub_u.

Moreover, the special property of chameleon hash scheme makes the user u just register once, and then can generate identity proofs without involving the regulator.

2) Achieving tracing: To achieve tracing, we add C_{id_u} which is the ciphertext of the pk_{chash_u} under the regulator public key pk_{reg} to the $proof_{id_u}$, and require the $proof_{id_u}$ to prove that the plaintext corresponding to the ciphertext C_{id_u} is pk_{chash_u}. Because the regulator has the record $[(pk_{chash_1}, id_1, CH_{id_1}), ..., (pk_{chash_n}, id_n, CH_{id_n})]$, the regulator can decrypt the C_{id_u}, obtain the pk_{chash_u}, and determine the identity of the user u based on the record. More details about the identity proof are provided in V-A3.

Remark. We use id to denote the user's real identity (such as identity card number or mobile phone number). Moreover, there are many ways to verify the user's identity in reality, such as, face recognition, identity card, or short message service (SMS) verification. Therefore, we assume there is an efficient way of verifying the id in the user registration of SkyEye.

B. SkyEye Overview

As can be seen from Figure 4, SkyEye's application strategy in $\boldsymbol{B_s}$ is that the user generates the identity proof $proof_{id}$ cor-

responding to the user's public information pub, and publishes the $(pub, proof_{id})$ to the node network.

From Figure 4, it can also be seen that the SkyEye scheme mainly has the following operations between the regulator, users, and verifiers.

• *User registration.* The user generates registration information and sends it to the regulator. The regulator is responsible for the verification of registration information.

• *Generating identity proof.* The user who registers successfully can generate the identity proof $proof_{id}$. There is a binding relationship between the $proof_{id}$ and the $priv$.

• *Verifying identity proof.* Different from traditional verification process in the blockchain, the verifier (e.g., the miner) verifies identity proofs in addition to verifying data contents. If the data contents and identity proofs are simultaneously verified successfully, the data will be added to the block generated by the verifier.

• *Tracing.* The regulator traces the users' true identities in the blockchain data.

Remark. The SkyEye scheme provides an alternative tracing strategy for $\boldsymbol{B_s}$. If a blockchain application that does not belong to $\boldsymbol{B_s}$ wants to use SkyEye, this application must modify some rules to make it belong to $\boldsymbol{B_s}$.

IV. Definition of a Traceable Scheme

A. Definition

A traceable scheme for blockchain is a tuple of polynomial-time algorithms Π = *(Setup,* Gen_{info}*,* Ver_{info}*,* Gen_{proof}*,* Ver_{proof}*, Trace)* described below:

• *Setup(*λ*)* $\rightarrow$ *pp*. Given a security parameter λ, the *Setup* returns public parameters pp. This algorithm is executed by a trusted party and is done only once. The public parameters pp are published and made available to all parties.

• Gen_{info}*(pp,id)* $\rightarrow$ *reginfo*. Given the public parameters pp and a user identity id, this algorithm returns the registration information *reginfo*.

• Ver_{info}*(pp,reginfo,*sk_{reg}*)* $\rightarrow$ *b*. Given the registration information *reginfo*, the public parameters pp and the regulator private key sk_{reg}, this algorithm returns a bit b. If verification succeeds, this algorithm returns 1; otherwise, it returns 0.

• $Gen_{proof}(pp, pub, priv, CH_{id}, pk_{chash}, sk_{chash}, r, rt, path_{id}) \rightarrow proof_{id}$. Given the public parameters *pp*, a user's public/private information $(pub, priv)$, the chameleon hash CH_{id}, the chameleon hash public/private key (pk_{chash}, sk_{chash}), the random element r for computing the chameleon hash, the Merkle tree root rt, and the path $path_{id}$ from $pk_{chash}||CH_{id}$ to rt, Gen_{proof} returns the user's identity proof $proof_{id}$.

• Ver_{proof}*(pp,pub,*$proof_{id}$*)* $\rightarrow$ *b*. Given the public parameters pp, a user's public information pub and the user identity proof $proof_{id}$, Ver_{proof} returns a bit b. If the verification of $proof_{id}$ succeeds, this algorithm returns 1; otherwise, it returns 0.

• *Trace(*$\boldsymbol{data_{B_{se}}}$*)* $\rightarrow$ *ID*. Given the blockchain data $\boldsymbol{data_{B_{se}}}$, *Trace* returns the identity set ID for $\boldsymbol{data_{B_{se}}}$.

B. Security

We assume that in $\boldsymbol{B_s}$ relevant cryptographic techniques (e.g., digital signatures) have been used to ensure that the blockchain data generated by the users cannot be tampered with. Therefore, the users' identity proofs added to the blockchain data also cannot be tampered with. We also assume that the regulator is trusted and has an efficient way of verifying user identity. Therefore, the goals of the adversary are to forge the user identity proof and to distinguish two distinct user identity proofs. The security of a traceable scheme must satisfy the following security properties:

• **Identity proof indistinguishability.** This property requires that even if the adversary can adaptively induce honest parties to perform operations of his choice, the identity proof reveals no information except for some public information, such as public addresses and serial numbers. In other words, even if the adversary queries two different honest parties(one identity is id_0, and the other identity is id_1), no polynomial-time adversary can distinguish between the identity proofs $proof_{id_0}$ and $proof_{id_1}$. The meaning of this property is that if the blockchain is indistinguishable, adding the identity proofs to the blockchain data does not affect the indistinguishability in the blockchain.

• **Identity proof unforgeability.** This property requires that even if the adversary can adaptively induce honest parties to perform operations of his choice, no polynomial-time adversaries can forge the identity proof of honest parties. This property ensures that the adversary cannot forge the honest user's identity proof to create blockchain data for evading supervision.

We defer formal definition of each property to Appendix A.

Definition 1 *A traceable scheme* $\prod$ = *(Setup,* Gen_{info}*,* Ver_{info}*,* Gen_{proof}*,* Ver_{proof}*, Trace) is secure if it satisfies identity proof indistinguishability and identity proof unforgeability.*

V. Construction

A. SkyEye Construction

1) SkyEye Initialization: The public parameters pp created by the *Setup* algorithm include the following information: the zk-SNARK proving/verification key (pk_{info}, vk_{info}) used to generate and verify the zk-SNARK proof π_{info} for the NP relation R_{info} (see Section V-A2 for details), the zk-SNARK proving/verification key (pk_{proof}, vk_{proof}) used to generate and verify the zk-SNARK proof π_{proof} for the NP relation R_{proof} (see Section V-A3 for details), the regulator public key pk_{reg} for public key encryption, and the public parameters pp_{chash} of the chameleon hash scheme. Once the algorithm $Setup$ returns *pp*, the pp will be published and made available to all parties. (See *Setup* algorithm in Algorithm 1 for specific operations.)

2) User Registration: The user registration is divided into two parts: the generation of user registration information and the verification of user registration information.

Algorithm 1 SkyEye Construction

Setup

Input: security parameter λ;
Output: public parameters pp;
construct arithmetic circuit AC_{info} for relation R_{info} at security λ;
construct arithmetic circuit AC_{proof} for relation R_{proof} at security λ;
$(pk_{info}, vk_{info}) = \mathcal{K}_{nizk}(\lambda, AC_{info})$;
$(pk_{proof}, vk_{proof}) = \mathcal{K}_{nizk}(\lambda, AC_{proof})$;
compute $pp_{enc} = \mathcal{G}_{enc}(\lambda)$;
compute $(pk_{reg}, sk_{reg}) = \mathcal{K}_{enc}(pp_{enc})$;
compute $pp_{chash} = \mathcal{G}_{chash}(\lambda)$;
return $pp := (pk_{info}, vk_{info}, pk_{proof}, vk_{proof}, pk_{reg}, pp_{chash})$;

***Gen*$_{info}$**

Input:
public parameters pp,
user identity id;
Output:
registration information $reginfo$;
$(pk_{chash}, sk_{chash}) = \mathcal{K}_{chash}(pp_{chash})$;
randomly sample r;
compute $CH_{id} = \mathcal{H}_{chash}(pk_{chash}, id, r)$;
set $x_{info} = (id, pk_{chash}, CH_{id}), w_{info} = (sk_{chash}, r)$;
$\pi_{info} = \mathcal{P}_{nizk}(pk_{info}, x_{info}, w_{info})$;
set $C_{info} = \mathcal{E}_{enc}(pk_{reg}, x_{info})$;
store $(id, pk_{chash}, sk_{chash}, r, CH_{id})$;
return $reginfo = (C_{info}, \pi_{info})$;

***Ver*$_{info}$**

Input:
public parameters pp;
registration information $reginfo$,
regulator's private key sk_{reg},
Output: bit b;
parse $reginfo$ as (C_{info}, π_{info});
$x_{info} = \mathcal{D}_{enc}(sk_{reg}, C_{info})$;
parse x_{info} as $(id, pk_{chash}, CH_{id})$;
if id not valid **then**
  return b=0;
end if
if $\mathcal{V}_{nizk}(vk_{info}, x_{info}, \pi_{info}) = 0$ **then**
  return b=0;
else
  store $(pk_{chash}, id, CH_{id})$;
  publish $pk_{chash}||CH_{id}$ via the Merkle tree MT;
  return b=1;
end if

***Gen*$_{proof}$**

Input:
public parameters pp,
user public/private information $(pub, priv)$,
chameleon hash CH_{id},
chameleon hash public/private key (pk_{chash}, sk_{chash}),
random element r for computing chameleon hash,
Merkle tree root rt,
path $path_{id}$ from $pk_{chash}||CH_{id}$ to rt;
Output:
user identity proof $proof_{id}$;
compute $r' = \mathcal{CF}_{chash}(sk_{chash}, id, priv, r)$;
randomly sample rn for encrypting;
compute $C_{id} = \mathcal{E}_{enc}(pk_{reg}, pk_{chash}, rn)$;
set $u_{proof} = (rt, pk_{reg}, C_{id})$;
set $x_{proof} = (pub, u_{proof})$,
$w_{proof} = (path_{id}, CH_{id}, sk_{chash}, pk_{chash}, priv, r', rn)$;
compute $\pi_{proof} = \mathcal{P}_{nizk}(pk_{proof}, x_{proof}, w_{proof})$;
set $proof_{id} = (u_{proof}, \pi_{proof})$;
return $proof_{id}$

***Ver*$_{proof}$**

Input:
public parameters pp,
user public information pub;
identity proof $proof_{id}$;
Output:
bit b;
parse $proof_{id}$ as (u_{proof}, π_{proof})
set $x_{proof} = (pub, u_{proof})$
if $(\mathcal{V}_{nizk}(vk_{proof}, x_{proof}, \pi_{proof}) = 0)$ **then**
  return b=0;
else
  return b=1;
end if

Trace

Input:
blockchain data $\boldsymbol{data_{B_{se}}}$;
Output:
identity set ID for $\boldsymbol{data_{B_{se}}}$;
set $ID = \phi$;
get ciphertext set $C = \{C_{id_i}\}_{i \in \{1,\ldots,n\}}$ from $\boldsymbol{data_{B_{se}}}$, where n is the number of the users' public information in $\boldsymbol{data_{B_{se}}}$;
for (each $C_{id_i} \in C$) **do**
  compute $pk_{chash_i} = \mathcal{D}_{enc}(sk_{reg}, C_{id_i})$;
  search $(pk_{chash}, id, CH_{id})$ records, get id_i corresponding to pk_{chash_i};
  put id_i in ID;
end for
return ID;

As shown in Algorithm 1, A user generates his own (pk_{chash}, sk_{chash}) based on the chameleon hash public parameters pp_{chash}, then computes the chameleon hash value CH_{id} of identity id, and stores $(id, pk_{chash}, sk_{chash}, r, CH_{id})$. At this point, the user can produce a zk-SNARK proof π_{info} for the following NP relation, which we call R_{info}:
Given $x_{info} = (id, pk_{chash}, CH_{id})$, I know $w_{info} = (sk_{chash}, r)$ such that:

♦ The chameleon hash private key matches the chameleon hash public key: $pk_{chash} = chash\_gen(sk_{chash})$.

♦ The chameleon hash is computed correctly: $CH_{id} = \mathcal{H}_{chash}(pk_{chash}, id, r)$.

The Gen_{info} algorithm outputs registration information *reginfo*, which consists of the ciphertext C_{info} and zk-SNARK proof π_{info}. The ciphertext C_{info} is the encryption of x_{info} that uses the regulator public key pk_{reg}.

The Ver_{info} algorithm shown in Algorithm 1 is used to verify the user's registration information *reginfo* by the regulator. The verification operations in algorithm Ver_{info} include verifying the identity id and verifying the zk-SNARK proof π_{info}. If the above two operations are verified successfully, the regulator stores $(pk_{chash}, id, CH_{id})$, and then publishes $pk_{chash}||CH_{id}$ stored in the Merkle tree MT in which the root is denoted by rt. Meanwhile, this algorithm returns 1.

3) Generating and Verifying Identity Proof: The Gen_{proof} shown in Algorithm 1 is used to generate the identity proof for each user. Assume a user has generated the public/private information $(pub, priv)$. According to the known trapdoor sk_{chash}, the user can calculate a value r' such that $CH_{id} = \mathcal{H}_{chash}(pk_{chash}, priv, r')$. Next, the user computes ciphertext $C_{id} = \mathcal{E}_{enc}(pk_{reg}, pk_{chash}, rn)$, where pk_{reg} is the public key of the regulator, and rn is the random number used for encryption. Finally, the user produces a zk-SNARK proof π_{proof} for the following NP relation, which we term R_{proof}:
Given a statement $x_{proof} = (pub, rt, pk_{reg}, C_{id})$, I know $w_{proof} = (path_{id}, CH_{id}, sk_{chash}, pk_{chash}, priv, r', rn)$ such that:

♦ The private information matches the public information: $pub = gen(priv)$.

♦ The chameleon hash private key matches the chameleon hash public key: $pk_{chash} = chash\_gen(sk_{chash})$.

♦ The chameleon hash CH_{id} is computed correctly: $CH_{id} = \mathcal{H}_{chash}(pk_{chash}, priv, r')$.

♦ The ciphertext C_{id} corresponds to the plaintext pk_{chash}: $C_{id} = \mathcal{E}_{enc}(pk_{reg}, pk_{chash}, rn)$.

♦ The $pk_{chash}||CH_{id}$ appears as a leaf of a Merkle tree with the root rt.

The Ver_{proof} algorithm in Algorithm 1 is used to verify the user's identity proof $proof_{id}$. The verification operation verifies the zk-SNARK proof π_{proof}. This algorithm returns 1 if and only if the above operation verifies successfully.

4) Tracing: As shown in Algorithm 1, the algorithm $Trace$ is used to trace the blockchain data $\boldsymbol{data}_{\boldsymbol{B_{se}}}$. The regulator obtains pk_{chash} by decrypting every ciphertext C_{id}, and according to the record that stores each user's chameleon hash public key pk_{chash}, identity id, and chameleon hash CH_{id}, the regulator can determine the true identities of the users in the $\boldsymbol{data}_{\boldsymbol{B_{se}}}$. This algorithm returns the users' identity set ID.

B. SkyEye Security

Theorem 1 *Assuming that the $Chash$ scheme is collision resistant, trapdoor collision and semantic security, the NIZK scheme is perfectly zero-knowledge and simulation sound extractable, the encryption scheme Enc satisfies IND-CCA2 security, and $gen(\cdot)$ has one-wayness property. Our scheme $\prod$ = (Setup, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof}, Trace) described in Algorithm 1 is a secure (cf. Definition 1) traceable scheme.*

We provide the proof of Theorem 1 in Appendix B.

VI. Implementation and Evaluation

A. Implementation

There are two main ways of generating public and private information in blockchain applications. One is through the pseudorandom function (e.g., Zerocash [8], Hawk [24]), i.e. $pub = PRF_{priv}(s)$, where PRF denotes the pseudorandom function, pub is the pseudorandom number, $priv$ is the private key used to generate pub, and s is the uniform seed. The other way is to use elliptic curve scalar multiplication (e.g., Bitcoin) to generate the public and private information, i.e., $pub = priv \cdot G$, where $priv$ is a scalar, G is a base point on the elliptical curve, and pub is a point on the elliptical curve. We use $SkyEye_H$ to represent the scheme that generates public and private information in the first way, and $SkyEye_S$ to represent the scheme that generates public and private information in the second way. We use the C++ programming language to implement the prototype of the above two different schemes based on the zk-SNARK library, libsnark [9].

There are some cryptographic building blocks in $SkyEye_H$: the pseudorandom function, chameleon hash scheme, hash function in the Merkle tree, public encryption scheme, and zk-SNARK scheme. For the chameleon hash scheme, we use the chameleon hash scheme proposed by Hugo Krawczyk and Tal Rabin [25]. For efficiency, we use the SHA256 compression function to implement the pseudorandom function and hash function in the Merkle tree, which is similar to the approach used in Zerocash [8]. We use the practical public key encryption scheme proposed by Cramer and Shoup [12], an IND-CCA2 secure public encryption scheme, as our encryption scheme. We use the scheme proposed by Parno et al. [31] as the zk-SNARK scheme. In the concrete implementation, we use the Barreto-Naehrig elliptic curve [7] that provides 128-bit security as the underlying curve of the zk-SNARK scheme. The implementation of the chameleon hash and public key encryption scheme is based on a prime field of 254 bits.

In $SkyEye_S$, the main cryptographic building block differs from the former in that the pseudorandom function is replaced by elliptic curve scalar multiplication. The chameleon hash scheme, public key encryption scheme, and zk-SNARK scheme are the same as those in the $SkyEye_H$. In the concrete implementation, we use the MNT4 elliptic curve

TABLE I: Performance of $SkyEye_H$

$SkyEye_H$		Configuration 1: intel(R) core(TM) i5-2450M @2.50GHz 4GB of RAM				Configuration 2: intel(R) core(TM) i7-6700 @ 3.40GHz 16GB of RAM			
		Tree depth							
		10	20	30	34	10	20	30	34
Setup	time(s)	69	114	156	175	37	61	83	94
	$\|pk_{info}\|$(KB)	480							
	$\|vk_{info}\|$(B)	574							
	$\|pk_{proof}\|$(MB)	90	149	209	231	90	149	209	231
	$\|vk_{proof}\|$(KB)	21							
Gen_{info}	time(ms)	475.7	494.3	530.1	538.5	231.5	248.0	266.2	272.4
	$\|\pi_{info}\|$(B)	287							
Ver_{info}	time(ms)	15.3	15.3	15.7	15.1	7.1	7.0	7.0	6.9
Gen_{proof}	time(s)	29	46	59	66	15	24	30	35
	$\|\pi_{proof}\|$(B)	287							
Ver_{proof}	time(ms)	10.1	10.2	10.1	10.2	4.5	4.5	4.8	4.6
$Trace$	time(ms)	0.13				0.075			

TABLE II: Performance of $SkyEye_S$

$SkyEye_H$		Configuration 1: intel(R) core(TM) i5-2450M @2.50GHz 4GB of RAM				Configuration 2: intel(R) core(TM) i7-6700 @ 3.40GHz 16GB of RAM			
		Tree depth							
		10	20	30	34	10	20	30	34
Setup	time(s)	187	296	403	451	101	162	220	244
	$\|pk_{info}\|$(KB)	661							
	$\|vk_{info}\|$(B)	667							
	$\|pk_{proof}\|$(MB)	105	174	243	268	105	174	243	268
	$\|vk_{proof}\|$(KB)	13							
Gen_{info}	time(ms)	1398.0	1413.3	1456.3	1523.9	754.9	772.6	787.8	793.8
	$\|\pi_{info}\|$(B)	337							
Ver_{info}	time(ms)	51.8	52.4	53.4	54.4	27.3	27.8	27.0	27.3
Gen_{proof}	time(s)	56	83	109	120	30	45	58	64
	$\|\pi_{proof}\|$(B)	337							
Ver_{proof}	time(ms)	47.8	47.9	48.0	47.9	24.9	24.9	24.7	24.9
$Trace$	time(ms)	0.14				0.09			

[28] as the underlying curve of the zk-SNARK scheme. The implementation of elliptic curve scalar multiplication is based on the MNT6 elliptic curve [28]. We implement the chameleon hash scheme and public key encryption scheme in a prime field of 298 bits. To improve efficiency, in the formation of the Merkle tree, because the length of the leaf node is 298 bits, two leaf nodes together cannot form 512 bits. Therefore, the upper node is generated by the leaf node using the standard SHA256. In addition, the data length of the node above the leaf node is 256 bits, so each node that is not generated through the leaf node is generated by the SHA256 compression function.

B. Evaluation

We evaluate the performance of every algorithm in the two aforementioned schemes in two different configurations: configuration 1, with an Intel i5 processor and 4 GB memory laptop; and configuration 2, with an Intel i7 processor and 16 GB memory desktop machine. The depth of the Merkle tree in our evaluation is 10, 20, 30, and 34, respectively. In other words, the maximum number of users which the Merkle tree supports is 2^{10}, 2^{20}, 2^{30}, and 2^{34}. This fully meets demand, because the current global population is about 7.5 billion, and 2^{34} reaches more than 17 billion. Moreover, we evaluate the performance of the $Trace$ algorithm under the condition that there are already 1024 successfully registered users at the regulator.

Table I and Table II illustrate the performance results of the $Setup$, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof} and $Trace$ algorithms in $SkyEye_H$ and $SkyEye_S$, respectively (the time in the two tables is the average of 10 runs per algorithm). In the two tables, time represents the running time of the algorithm, and $|\cdot|$ represents the data length. For example, the $|pk_{info}|$ represents the length of the proving key in the registration. Without loss of generality, using an i7 processor, a 16 GB memory desktop machine, and with a tree depth of 34 in Table I, we can obtain the results of the $SkyEye_H$ scheme:

- $Setup$ algorithm takes 94 s. The size of the proving key and verification key used for user registration are 480 KB and 574 B, respectively. and the size of the proving key and verification key used for user identity proof are 231 MB and 21 KB, respectively.
- Gen_{info} requires 272.4 ms, and the size of the zk-SNARK proof π_{info} is 287 B.
- Ver_{info} algorithm takes 6.9 ms.
- Gen_{proof} algorithm takes 35 s to generate a user's identity proof, and the size of the zk-SNARK proof π_{proof} is 287 B.

• Ver_{proof} algorithm takes 4.6 ms.

• $Trace$ algorithm takes 0.075 ms to trace a user's identity.

The tables reveal the following:

• In each configuration, the time required for verification by the regulator and the verifiers is small and does not substantially change as the depth of the tree changes. As shown in Table I, the regulator takes approximately 15 ms to verify the user registration information in configuration 1 and approximately 7 ms in configuration 2; and the time taken by a verifier to verify the user identity proof is approximately 10 ms in configuration 1 and approximately 5 ms in configuration 2. From Table II, we can observe that the time taken for verifying the user registration information is approximately 53 ms in configuration 1 and approximately 28 ms in configuration 2; and the time taken for verifying the user identity proof is approximately 48ms in configuration 1 and approximately 25 ms in configuration 2.

• Not all of the information in SkyEye must be on-chain. Only the information $proof_{id}$ generated by the Gen_{proof} algorithm is added to the user data. Furthermore, the size of the user's zk-SNARK proof π_{proof} in the $proof_{id}$ is dominant. As can be observed from the two tables, the length of the zk-SNARK proof π_{proof} will not change as the configuration environment and tree depth change. The size of π_{proof} is small, and the length is 287 B in $SkyEye_H$ and 337 B in $SkyEye_S$.

VII. Discussion and Future Work

In SkyEye, the centralization of the regulator is a major issue. The regulator can arbitrarily trace the identity of blockchain data without any restrictions and oversight.

From the data tracing process of the regulator, it can be seen that the regulator must first use its private key sk_{reg} to decrypt the ciphertext of each user's chameleon hash public key in the blockchain data. Therefore, we can restrict the regulator through the distributed key generation (DKG) protocol [20]. Specifically, the public/private key pair (pk_{reg}, sk_{reg}) is generated by a committee with a threshold of t through the DKG protocol. In this way, the pk_{reg} is made public, and each committee member has a share of sk_{reg}. The regulator submits the data and tracing evidence to the committee. If at least t+1 members of the committee accept the data and tracing evidence, the regulator will obtain the sk_{reg} from the committee.

However, this approach does not completely restrict the regulator. Even if the committee regularly updates the public/private key pair, as long as the regulator obtains the private key sk_{reg} in a cycle, it can trace not only the data submitted to the committee, but also all user data in this cycle. In future work, we will consider how to restrict the regulator to make the regulator only trace the data submitted to the committee.

VIII. Related work

Blockchain research focuses primarily on enhancing blockchain privacy protection [8], [32], improving blockchain scalability [16], [35], analyzing blockchain security [17], [26], and applying blockchain to other areas [27], [33]. However, research on traceable mechanisms is limited.

Narula, Vasquez, and Virza proposed zkLedger [30], the first distributed ledger system, that provides strong privacy protection, public verifiability, and practical auditing. zkLedger uses table-construction in the ledger. Each user identity corresponds to each column in the ledger. Therefore, the regulator can determine every user identity through the ledger. However, this traceable mechanism in zkLedger cannot be applied to environments with a large number of users and is used only for auditing digital asset transactions over some banks.

Defrawy and Lampkins [15] proposed a proactively-private digital currency (PDC) scheme that can provide privacy-preserving and accountability. In their scheme, the ledger is kept by a group of ledger servers. Every ledger sever has a balance ledger that contains a share of every user identity. Therefore, the regulator can determine every user identity through those ledger servers. However, their traceable mechanism does not seem to have been extended to other applications.

Ateniese and Faonio [5] constructed a scheme that provides certified Bitcoin addresses to enable Bitcoin users to trade with certifiable users authenticated by the trusted certificate authority. The regulator can determine every user identity through the authority. However, if a user wants to use a new certified address for each transaction, the user must contact the certificate authority to obtain a certified address. This reduces the efficiency of the entire system and exerts considerable pressure on the certificate authority when the number of users is large. Moreover, their approach only applies to Bitcoin.

Garman, Green, and Miers [18] designed new decentralized anonymous payment (DAP) systems to address the regulatory issue by adding privacy preseving policy-enforcement mechanisms that guarantee regulatory compliance, allow selective user tracing, and admit tracing of tainted coins. The regulator can determine every user identity through the identity escrow policy. However, the DAP system are based on Zerocash [8].

The traceable mechanisms proposed above can only be applied to specific application environments and do not seem to have been extended to other applications. We propose SkyEye, a traceable scheme for blockchain. Our scheme can be applied to a class of blockchain applications, which is denoted by $\boldsymbol{B_s}$.

IX. Conclusion

In this paper, we design SkyEye, a traceable scheme for blockchain. SkyEye can be applied to the blockchain applications that satisfy the following conditions: (I) The users have public and private information, where the public information is generated by the private information; (II) The users' public information is disclosed in the blockchain data. SkyEye just requires the user to register only once, and enables the regulator to trace users' identities. Moreover, we implement two different SkyEye prototypes: $SkyEye_H$ and $SkyEye_S$. Our evaluation results show that even if the number of users

is very large, the registration information and identity proof are verified quickly.

APPENDIX A
SECURITY OF THE TRACEABLE SCHEME

We describe *identity proof indistinguishability* and *identity proof unforgeability*. Every property is formalized as an experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The behavior of the honest user with identity id is realized by the oracle O_{id}, and the behavior of the regulator is realized by the oracle O_{reg}. We assume that the honest users and adversary in the experiment have already registered successfully in the regulator, i.e., they can generate any identity proof. Below, we describe how O_{id} and O_{reg} work.

Oracles O_{id} and O_{reg} are initialized by challenger $\mathcal{C}$ using the public parameters pp. The O_{reg} stores: (1) **Record**, a set of information used to trace true identities of all registered users; (2) the encryption public/private key pair (pk_{reg}, sk_{reg}). The O_{reg} accepts different queries, which are described below:

- $Q = (judge, proof_{id_1}, proof_{id_2})$, the O_{reg} determines whether $proof_{id_1}$ and $proof_{id_2}$ belong to the same user, and sends the result to the inquirer.
- $Q = (chashset, proof_{id})$, the O_{reg} sends the chameleon hash set P_{chash} to the inquirer. The P_{chash} includes the chameleon hash of the user who generates the $proof_{id}$.

The O_{id} stores: (1) **RegPriInfo**, the secret information used to generate registration information; (2) **IdProof**, a set of identity proofs generated by the user whose identity is id; (3)**IdProofPriInfo**, the set of evidence that the user uses to generate the identity proofs. The O_{id} accepts different queries, which are described below:

- $Q = (genidproof)$, The adversary is not aware of the private information $priv$. The oracle $\mathcal{O}_{id}$ first randomly selects $priv$, and then generates the public information pub. Finally, the oracle $\mathcal{O}_{id}$ calls the Gen_{proof} algorithm to generate the identity proof $proof_{id}$, and sends the $(pub, proof_{id})$ to the inquirer.
- $Q = (genidproof, priv)$, The adversary knows the private information $priv$, and the oracle $\mathcal{O}_{id}$ uses the $priv$ selected by the adversary to generate the public information pub and then calls the Gen_{proof} algorithm to generate the identity proof $proof_{id}$. Finally, sends the $(pub, proof_{id})$ to the inquirer.
- $Q = (genidproof, pub_i)$, here, $pub_i \in T_{pub}$, and $T_{pub} = \{pub_1, ..., pub_n\}$ is the public information set of the user whose identity is id. The oracle $\mathcal{O}_{id}$ calls the Gen_{proof} algorithm to generate the identity proof $proof_{id}$, and sends the $(pub_i, proof_{id})$ to the inquirer.

A. Identity proof indistinguishability

Identity proof indistinguishability is formalized by $Exp^{IDP-IND}_{\mathcal{A},\Pi}(\lambda)$, which is shown below:

1. The challenger $\mathcal{C}$ randomly samples $b \in \{0,1\}$, gets pp by running Setup(λ), and sends pp to adversary $\mathcal{A}$. Next, $\mathcal{C}$ initializes two separate oracles $\mathcal{O}_{id_0}$ and $\mathcal{O}_{id_1}$.

2. The adversary $\mathcal{A}$ issues queries $q_1, ..., q_m$, where q_i is one of the following:

- *Q and $Q^{'}$ are both genidproof queries. $\mathcal{C}$ forwards Q to $\mathcal{O}_{id_0}$, and forwards $Q^{'}$ to $\mathcal{O}_{id_1}$. $\mathcal{C}$ replies to $\mathcal{A}$ with $((pub_b, proof_{id_b}), (pub_{1-b}, proof_{id_{1-b}}))$, which is the two oracle answer.*
- *$\{Q, Q^{'}\} = \{(genidproof, priv), (genidproof, priv^{'})\}$, where $priv = priv^{'}$. $\mathcal{C}$ forwards Q to $\mathcal{O}_{id_0}$, and forwards $Q^{'}$ to $\mathcal{O}_{id_1}$. $\mathcal{C}$ replies to $\mathcal{A}$ with $((pub, proof_{id_b}), (pub, proof_{id_{1-b}}))$, which is the two oracle answer.*

3. At the end of the query, $\mathcal{A}$ sends $\mathcal{C}$ a guess $b^{'} \in \{0,1\}$. If $b = b^{'}$, $\mathcal{C}$ outputs 1; otherwise, $\mathcal{C}$ outputs 0.

Identity proof indistinguishability requires that the adversary $\mathcal{A}$ wins the above experiment with only negligible probability. Next, we formally define this property.

Definition 2 *A traceable scheme $\prod$ = (Setup, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof}, Trace) satisfies identity proof indistinguishability if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function $negl(\cdot)$ such that*

$$Adv^{IDP-IND}_{A,\Pi} \leq negl(\lambda), \tag{1}$$

where $Adv^{IDP-IND}_{A,\Pi} = Pr[Exp^{IDP-IND}_{\mathcal{A},\Pi}(\lambda) = 1] - 1/2$ is $\mathcal{A}$'s advantage in the experiment $Exp^{IDP-IND}_{\mathcal{A},\Pi}(\lambda)$.

B. Identity proof unforgeability

Identity proof unforgeability is formalized by an experiment $Exp^{IDP-UNF}_{\mathcal{A},\Pi}(\lambda)$, which is shown below:

1. The challenger $\mathcal{C}$ obtains pp by running Setup(λ), and sends pp to adversary $\mathcal{A}$. Next, $\mathcal{C}$ initializes two separate oracles $\mathcal{O}_{id}$ and $\mathcal{O}_{reg}$. Let $T_{pub} = \{pub_1, ..., pub_n\}$ be the public information set for the user whose identity is id.

2. The adversary $\mathcal{A}$ issues queries $q_1, ..., q_m$, where q_i is $(genidproof, pub_i)$, and $pub_i \in T_{pub}$. $\mathcal{C}$ forwards Q to $\mathcal{O}_{id}$, $\mathcal{C}$ replies to $\mathcal{A}$ with $(pub_i, proof_{id})$, which is the oracle $\mathcal{O}_{id}$'s answer.

3. At the end of the query, let $P = \{proof_1, ..., proof_m\}$ is the identity proof set that is generated by $\mathcal{O}_{id}$. $\mathcal{A}$ sends $(pub^*, proof^*_{id})$ to $\mathcal{C}$. $\mathcal{C}$ checks as follows:

- *If $proof^*_{id} \notin P \wedge Ver_{proof}(pp, pub^*, proof^*_{id}) = 1$, $\mathcal{C}$ proceeds as follows; otherwise it aborts.*
- *$\mathcal{C}$ sends $(judge, proof_{id^*}, proof_i)$ to $\mathcal{O}_{reg}$, where $i \in [1, m]$. If $proof_{id^*}$ and $proof_i$ belong to the user whose the identity is id, $\mathcal{O}_{reg}$ sends $c = 1$ to $\mathcal{C}$; otherwise it returns $c = 0$.*

If $proof^*_{id} \notin P \wedge Ver_{proof}(pp, pub^*, proof^*_{id}) = 1 \wedge c = 1$, $\mathcal{C}$ outputs 1; otherwise, $\mathcal{C}$ outputs 0.

The adversary $\mathcal{A}$ wins the above experiment if the $proof^*_{id}$ such that (i) $proof^*_{id} \notin P$; (ii) $Ver_{proof}(pp, pub^*, proof^*_{id}) = 1$; (iii) $proof_{id^*}$ belongs to the user whose identity is id. In other words, $\mathcal{A}$ can forge the identity proof of honest parties. Identity proof unforgeability requires that the adversary wins the above experiment with only negligible probability. Next, we formally define this property.

Definition 3 *A traceable scheme $\prod$ = (Setup, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof}, Trace) satisfies identity proof*

unforgeability if for all probabilistic polynomial time adversaries $\mathcal{A}$, there is a negligible function $negl(\cdot)$ such that

$$Adv_{A,\Pi}^{IDP-UNF} \leq negl(\lambda), \tag{2}$$

where $Adv_{A,\Pi}^{IDP-UNF} = Pr[Exp_{\mathcal{A},\Pi}^{IDP-UNF}(\lambda) = 1] - 1/2$ is $\mathcal{A}$'s advantage in the experiment $Exp_{\mathcal{A},\Pi}^{IDP-UNF}(\lambda)$.

APPENDIX B
PROOF OF THEOREM 1

A. Proof of identity proof indistinguishability

Theorem 2 *Assuming that the NIZK scheme is perfectly zero-knowledge and simulation sound extractable, the encryption scheme Enc satisfies IND-CCA2 security, then, our scheme $\prod$ = (Setup, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof}, Trace) described in Algorithm 1 satisfies identity proof indistinguishability.*

We prove the Theorem 2 through a sequence of hybrid experiments. Let q_m be the number of queries issued by the adversary $\mathcal{A}$.

$\boldsymbol{exp_{real}}$**.** The experiment exp_{real} is the same as the $Exp_{\mathcal{A},\Pi}^{IDP-IND}(\lambda)$.

$\boldsymbol{exp_1}$**.** The experiment exp_1 is the same as the experiment exp_{real} except that the challenger $\mathcal{C}$ simulates the $NIZK$. More precisely, $\mathcal{C}$ calls a polynomial-time simulator $S_{nizk}(\lambda, AC_{proof})$ to obtain $(pk_{proof}, vk_{proof}, tra)$, where tra is the trapdoor, instead of invoking $\mathcal{K}_{nizk}(\lambda, AC_{proof})$. When an oracle $\mathcal{O}_{id}$ sends a NIZK proof π_{proof} to $\mathcal{C}$, $\mathcal{C}$ replaces the real proof with a simulated proof by invoking $S_{nizk}(pk_{proof}, x_{proof}, tra)$, without using the witness. Because the NIZK scheme is perfectly zero-knowledge, the distribution of the simulated π_{proof} is identical to that of the proof computed in exp_{real}. Therefore, $Adv_{exp_{real}} = Adv_{exp_1}$.

$\boldsymbol{exp_{final}}$. The experiment exp_{final} is the same as the experiment exp_1 except that the challenger $\mathcal{C}$ replaces the C_{id} in $proof_{id}$ by encrypting a random string. More precisely, when an oracle $\mathcal{O}_{id}$ sends an identity proof $proof_{id}$ to $\mathcal{C}$, $\mathcal{C}$ replaces the C_{id} with a $C_{id}^{'}$ generated by $\mathcal{E}_{enc}(pk_{reg}, r, rn)$, where r is a random strings sampled uniformly from the plaintext space of the encryption scheme. Because the responses to the adversary $\mathcal{A}$ in exp_{final} are independent of the bit b. Therefore, $Adv_{exp_{final}} = 0$ in the experiment exp_{final}.

Next, we prove that no polynomial-time adversary can distinguish exp_1 from exp_{final} except with negligible probability(see below lemma).

Lemma 1 *After q_m queries, $|Adv_{exp_{final}} - Adv_{exp_1}| \leq q_m \cdot Adv_{enc}$, where Adv_{enc} denotes the adversary's advantage in the IND-CCA2 experiment.*

Proof sketch. We construct an algorithm $\mathcal{B}$, using $\mathcal{A}$ as a subroutine, to win the IND-CCA2 experiment. Define $\epsilon = Adv_{exp_{final}} - Adv_{exp_1}$.

For some $i \in \{1, ..., q_m\}$, when $\mathcal{A}$ issues an i-th query, $\mathcal{B}$ uses the NIZK extractor ε to obtain plaintext m corresponding to C_{id}. Then, $\mathcal{B}$ chooses a random string r that has the same length as m. $\mathcal{B}$ sends $(m_0, m_1) = (m, r)$ to the IND-CCA2 challenger and receives $C^* = \mathcal{E}_{enc}(pk_{reg}, r, m_{\overline{b}})$, where $\overline{b}$ is the bit chosen by the IND-CCA2 challenger. $\mathcal{B}$ replaces C_{id} included in $proof_{id}$ with C^*. We return b', which $\mathcal{A}$ outputs as the guess in the IND-CCA2 experiment. We know that when $\overline{b} = 0$, $\mathcal{A}$'s view of the interaction is distributed identically to that of exp_1. And when $\overline{b} = 1$, $\mathcal{A}$'s view represents the exp_{final} in which one ciphertext C_{id} has been replaced. Based on a standard hybrid argument over each of the q_m ciphertexts, we can conclude that over the randomness of the experiment, $\mathcal{B}$ must succeed in the IND-CCA2 experiment with the advantage of at least ϵ/q_m. Therefore, $|Adv_{exp_{final}-Adv_{exp_1}}| \leq q_m \cdot Adv_{enc}$.

B. Proof of identity proof unforgeability

Theorem 3 *Assuming that the $Chash$ scheme is collision resistant, trapdoor collision and semantic security, the NIZK scheme is perfectly zero-knowledge and simulation sound extractable, $gen(\cdot)$ is one-wayness, then, our scheme $\prod$ = (Setup, Gen_{info}, Ver_{info}, Gen_{proof}, Ver_{proof}, Trace) described in Algorithm 1 satisfies identity proof unforgeability.*

From experiment $Exp_{\mathcal{A},\Pi}^{IDP-UNF}(\lambda)$, we can observe that $\mathcal{A}$ succeeds only if it outputs $(pub^*, proof_{id}^*)$ such that: (i) $proof_{id}^* \notin P$; (ii) $Ver_{proof}(pp, pub^*, proof_{id}^*) = 1$; (iii) $proof_{id^*}$ belongs to the user whose identity is id. We define the two disjoint events which $\mathcal{A}$ succeeds: (i) $Event$, $\mathcal{A}$ succeeds, and $pub^* \in T_{pub}$; (ii) $\overline{Event}$, $\mathcal{A}$ succeeds, and $pub^* \notin T_{pub}$. Let ε be the NIZK extractor for $\mathcal{A}$.

Obviously, $Adv_{A,\Pi}^{IDP-UNF} = Pr[Event] + Pr[\overline{Event}]$. Define $\epsilon_1 = Pr[Event]$ and $\epsilon_2 = Pr[\overline{Event}]$.

When $Event$ occurs, we construct the algorithm $\mathcal{B}$. It uses $\mathcal{A}$ as a subroutine, and solves the one-wayness of $gen(\cdot)$. Let ε be the NIZK extractor for $\mathcal{A}$. The algorithm $\mathcal{B}$ works as follows.

1. $\mathcal{B}$ randomly selects $i \in \{1, ..., n\}$.
2. $\mathcal{B}$ performs the experiment $Exp_{\mathcal{A},\Pi}^{IDP-UNF}(\lambda)$ with $\mathcal{A}$ to obtain $(pub^*, proof_{id}^*)$.
3. $\mathcal{B}$ runs the $\varepsilon(vk_{proof}, \pi_{proof}^*)$ to obtain $w_{proof} = \{path_{id}^*, CH_{id}^*, pk_{chash}^*, sk_{chash}^*, priv^*, r^*, rn^*\}$.
4. If $pub^* = pub_i$, then $\mathcal{B}$ outputs $priv^*$; otherwise, $\mathcal{B}$ aborts.

Because the index i is selected at random, $\mathcal{B}$ succeeds with probability ϵ_1/n. Because of the one-wayness of the $gen(\cdot)$, ϵ_1 must be negligible in λ.

When $\overline{Event}$ occurs, we construct algorithm $\mathcal{Z}$. It uses $\mathcal{A}$ as a subroutine and finds collision for the chameleon hash scheme. $\mathcal{Z}$ sends $(chashset, proof_{id})$ to $\mathcal{O}_{reg}$, and obtains $P_{chash} = \{CH_{id_1}, ..., CH_{id_k}\}$ from the oracle $\mathcal{O}_{reg}$, where $k \ll \lambda$. The set P_{chash} includes the chameleon hash CH_{id} of the user whose identity is id. The algorithm $\mathcal{Z}$ performs as follows.

1. $\mathcal{Z}$ randomly selects $i \in \{1, ..., k\}$.
2. $\mathcal{Z}$ performs the experiment $Exp_{\mathcal{A},\Pi}^{IDP-UNF}(\lambda)$ with $\mathcal{A}$ to obtain $(pub^*, proof_{id}^*)$.
3. $\mathcal{Z}$ runs the $\varepsilon(vk_{proof}, \pi_{proof}^*)$ to obtain $w_{proof} = \{path_{id}^*, CH_{id}^*, pk_{chash}^*, sk_{chash}^*, priv^*, r^*, rn^*\}$.

4. If $CH^*_{id} = CH_{id_i}$, then $\mathcal{Z}$ outputs $(priv^*, r^*)$; otherwise, $\mathcal{B}$ aborts.

Because the index i is selected at random, $\mathcal{Z}$ succeeds with probability ϵ_2/k. Furthermore, because of the collision resistance of the chameleon hash scheme, ϵ_2 must be negligible in λ.

REFERENCES

[1] https://litecoin.org/.

[2] https://en.wikipedia.org/wiki/Ransomware.

[3] Nxt Whitepaper. https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf, july 2014.

[4] G. Ateniese and B. de Medeiros. On the key exposure problem in chameleon hashes. In *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, pages 165–179, 2004.

[5] G. Ateniese, A. Faonio, B. Magri, and B. de Medeiros. Certified bitcoins. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, pages 80–96, 2014.

[6] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 913–930, 2018.

[7] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, pages 319–331, 2005.

[8] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[9] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 781–796, 2014.

[10] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 315–333, 2013.

[11] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186, 1999.

[12] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 13–25, 1998.

[13] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[14] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 66–98, 2018.

[15] K. E. Defrawy and J. Lampkins. Founding digital currency on secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1–14, 2014.

[16] S. Dziembowski, S. Faust, and K. Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 949–966, 2018.

[17] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 89–103, 2015.

[18] C. Garman, M. Green, and I. Miers. Accountable privacy for decentralized anonymous payments. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 81–98, 2016.

[19] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[20] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 295–310, 1999.

[21] T. Hardjono, N. Smith, and A. S. Pentland. Anonymous identities for permissioned blockchains. https://petertodd.org/assets/2016-04-21/MIT-ChainAnchor-DRAFT.pdf, 2014.

[22] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 357–388, 2017.

[23] S. King and S. Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. https://peercoin.net/assets/paper/peercoin-paper.pdf, August 2012.

[24] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 839–858, 2016.

[25] H. Krawczyk and T. Rabin. Chameleon hashing and signatures. *IACR Cryptology ePrint Archive*, 1998:10, 1998.

[26] Y. Kwon, D. Kim, Y. Son, E. Y. Vasserman, and Y. Kim. Be selfish and avoid dilemmas: Fork after withholding (FAW) attacks on bitcoin. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 195–209, 2017.

[27] S. Matsumoto and R. M. Reischuk. IKP: turning a PKI around with decentralized automated incentives. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 410–426, 2017.

[28] A. M. M. Nakabayashi and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals*, 84(5):1234–1243, 2001.

[29] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2009.

[30] N. Narula, W. Vasquez, and M. Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 65–80, 2018.

[31] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 2013.

[32] T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2P mixing and unlinkable bitcoin transactions. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, 2017.

[33] A. Tomescu and S. Devadas. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 393–409, 2017.

[34] G. WOOD. Ethereum: A secure decentralized transaction ledger. http://gavwood.com/paper.pdf, Accessed: 2016-05-15.

[35] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 931–948, 2018.