IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 April 2023, accepted 12 May 2023, date of publication 24 May 2023, date of current version 12 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3279408

RESEARCH ARTICLE

Azeroth: Auditable Zero-Knowledge Transactions in Smart Contracts

GWEONHO JEONG[1], NURI LEE[2], JIHYE KIM[2,3], (Member, IEEE), AND HYUNOK OH[1,3], (Member, IEEE)

[1]Department of Information Systems, Hanyang University, Seoul 04763, South Korea
[2]Electronics and Information System Engineering Major, Kookmin University, Seoul 02707, South Korea
[3]Zkrypto, Seoul 04763, South Korea

Corresponding authors: Jihye Kim (jihyek@kookmin.ac.kr) and Hyunok Oh (hoh@hanyang.ac.kr)

This work was supported in part by the Institute for Information and Communications Technology Promotion (IITP) funded by the Korean Government (MSIT), in part by the Blockchain Privacy Preserving Techniques Based on Data Encryption (50%) under Grant 2021-0-00518, in part by the Study on Cryptographic Primitives for SNARK (50%) under Grant 2021-0-00727, and in part by the Klaytn Foundation.

ABSTRACT With the rapid growth of the blockchain market, privacy and security issues for digital assets are becoming more important. In the most widely used public blockchains, such as Bitcoin and Ethereum, all activities on user accounts are publicly disclosed, which violates privacy regulations such as EU GDPR. Encryption of accounts and transactions may protect privacy, but it also raises issues of validity and transparency. While encrypted information can protect privacy, it cannot alone verify the validity of a transaction. Additionally, encryption makes it difficult to meet anti-money laundering regulations, such as auditability. In this paper, we propose Azeroth, an auditable zero-knowledge transfer framework. Azeroth connects a zero-knowledge proof to an encrypted transaction, enabling it to check its validation while protecting its privacy. Azeroth also allows authorized auditors to audit transactions. Azeroth is designed as a smart contract for flexible deployment on existing blockchains. We implement the Azeroth smart contract, and execute it on various platforms including an Ethereum testnet blockchain, and measure the time to show the practicality of our proposal. The end-to-end latency of a privacy-preserving transfer takes about 4.4s. In particular, the client's transaction generation time with a proof only takes about 0.9s. The security of Azeroth is proven under the cryptographic assumptions.

INDEX TERMS Blockchain application, zero-knowledge proof, SNARK, privacy-preserving payment.

I. INTRODUCTION

With the widespread adoption of blockchains, various decentralized applications (DApps) and digital assets used in DApps are becoming popular. Unlike traditional banking systems, however, the blockchain creates privacy concerns about digital assets since all transaction information is shared across the network for strong data integrity. Various studies have been attempted to protect transaction privacy by utilizing cryptographic techniques such as mixers [2], [22], [23], ring signatures [28], homomorphic encryption [7], zero-knowledge proofs [5], [7], [19], [27].

In the blockchain community, the zero-knowledge proof (ZKP) system is a widely used solution to resolve the conflict between privacy and verifiability. The ZKP is a proof system that can prove the validity of the statement without revealing the witness value; users can prove arbitrary statements of encrypted data, enabling public validation while protecting data privacy. For instance, the well-known anonymous blockchain ledger Zerocash [5] operates on the UTXO model, secures transactions while leveraging zero-knowledge proofs [17] to ensure transactions in a valid set of UTXOs. Zether [7] based on the account model encrypts accounts with homomorphic encryption and provides zero-knowledge proofs [9] to ensure valid modification of encrypted accounts. Zether builds up partial privacy for

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

unlinkability between sender and receiver addresses, similar to Monero [28].

As asset transactions on the blockchain increase, the demand for adequate auditing capabilities is also increase. Moreover, if transaction privacy is protected without proper regulation, financial transactions can be abused by criminals and terrorists. Without the management of illegal money flows, it would also be difficult to establish a monetary system required to maintain a sound financial system and enforce policies accordingly. Recently, Bittrex[1] delisted dark coins such as Monero [28], Dash [13], and Zerocash [5]. In fact, many global cryptocurrency exchanges are also strengthening their distance from private and anonymous cryptocurrencies as recommended by the Financial Action Task Force (FATF) [14]. Thus, we need to find a middle ground of contradiction between privacy preservation and fraudulent practices. This paper focuses on a privacy-preserving transfer that provides auditability for auditors while protecting transaction information from non-auditors.

Auditable private transfer can be designed by using encryption and the ZKP. A sender encrypts a transaction so that only its receiver and the auditor can decrypt it. At the same time, the sender should prove that the ciphertext satisfies all the requirements for the validity of the transaction. In particular, we utilize zk-SNARK (Zero-Knowledge Succinct Non-interactive ARgument of Knowledge) [17], [18], [26] to prove *arbitrary* functionalities for messages, including encryption. Although the encryption check incurs non-negligible overhead for the prover, it is essential for the validity and auditability of the transaction. Indeed, without a proof for encryption as in Zerocash [5], even if a ciphertext passes all other transaction checks, there always exists a possibility that an incorrectly generated ciphertext, either accidentally or intentionally, will be accepted, resulting in the loss of the validity and auditability of the transaction.

In this paper, we propose Azeroth, an auditable zero-knowledge transaction framework based on zk-SNARK. The Azeroth framework provides privacy, verifiability, and auditability for personal digital assets while maintaining the efficiency of transactions. Azeroth preserves the original functionality of the account-based blockchain as well as providing additional zero-knowledge feature to meet standard privacy requirements.

There are two main privacy considerations in a transfer transaction: confidentiality (concealing a balance and a transfer amount) and anonymity (concealing a sender and a recipient). Cryptographic primitives(e.g., encryption, commitment) can provide confidentiality and anonymity. Specifically, we employ dual accounts that constitute an encrypted account in addition to a plain account, similar to Blockmaze [19]. A user may execute deposit/withdrawal transactions between its own plain/encrypted accounts. At the same time, the user may send some encrypted value from its accounts to the accumulator (implemented as a Merkle tree). The owner of the encrypted value may receive it from the accumulator to the user's accounts. Additionally, to hide the function type, we construct a single transaction to execute all these functions of deposit/withdrawal/send/receive simultaneously. Information about the transaction can be inferred only from the plain account state transition; if the plain account state keeps the same, it cannot learn even the function type as well as the value amount. We also add auditability by utilizing two-recipient encryption so that an authorizer and receiver can decrypt the transaction. The auditor's ability can be distributed through a threshold scheme. In order to have this strong anonymity with auditability, the most challenging part is how to adopt and implement zk-SNARKs for this complex relation proof. Azeroth is devised using encryption for two recipients (i.e., the recipient and the auditor) so that the auditor can audit all transactions. Still, the auditor's capability is limited to auditing and cannot manipulate any transactions. Azeroth enhances the privacy of the transaction by performing multiple functions such as deposit, withdrawal, and transfer in one transaction. For real-world use, we adopt a SNARK-friendly hash algorithm to instantiate encryption for efficient proving time and execute experiments on various platforms.

Contributions: The contributions of this paper are summarized as follows.

- Framework: We design a privacy-preserving framework Azeroth on an account-based blockchain model, including encryption verifiability and auditability. Moreover, since Azeroth constructed as smart contract does not require any modifications to the base ledger, it advocates flexible deployment, which means that any blockchain models supporting smart-contract can utilize our framework.
- Security: We revise and extend the security properties of private transactions: ledger indistinguishability, transaction non-malleability, balance, and auditability, and prove that Azeroth satisfies all required properties under the security of underlying cryptographic primitives.
- Implementation: We implemented and tested Azeroth on existing account-based blockchain models, such as Ethereum [8]. According to our experiment, it takes 4.38s to generate a transaction in a client and process it in a smart contract completely on the Ethereum testnet. While Azeroth additionally supports encryption verifiability and auditability, it performs better than other schemes through implementation optimization. To show the practicality of our scheme, we implement the client side on various devices, including Android/iOS mobile phones. For the details, refer to section VI.

Organizations: The paper is comprised of the following contents: First, we provide the related works concerning our proposed scheme in Section II. In Section III, we describe preliminaries on our proposed system. In Section IV, we explain data structures utilized in Azeroth.

[1] https://global.bittrex.com/

Afterward, we elucidate the overview and construction. In section Section V, we define the security model and describe the security proofs related to it. Section VI shows the implementation and the experimental results. Finally, we conclude in Section VII.

II. RELATED WORK

The blockchain has been proposed for integrity rather than privacy. Various schemes have been proposed to provide privacy in blockchain along several lines of work.

A mixing service(or tumbler) such as CoinJoin [22], Möbius [23], and Tornado Cash [2] offers for mixing identifiable cryptocurrency transfer with others to obscure the trail back to the transfer's source. Thus, the mixer supports personal privacy protection for transactions on the blockchain. However, since the mixers take heed of anonymity, the possibility of a malevolent problem exists.

Zerocash [5] provides a well-known privacy-preserving transfer on UTXO-model blockchains. In Zerocash, a sender makes new commitments that hide the information of the transaction (i.e., value, receiver), which is open only to the receiver. The sender then proves the correctness of the commitments using the zero-knowledge proof. As an extension to a smart contract, Zeth [27] sorts the privacy problem out by implementing Zerocash into a smart contract. Zeth creates the anonymous coin within the smart contract in the form of underlying the UTXO model. Thus, operations and mechanisms in Zeth are almost the same as in Zerocash. ZEXE [6], which extends Zerocash with scripting capabilities, supports functional privacy such that nobody knows which computation is executed offline. Hawk [21] is a privacy-preserving framework for building arbitrary smart contracts. However, there exists a manager entrusted with the private data that generates a zk-SNARK proof to show that the process is executed correctly.

Blockmaze [19] proposes a dual balance model for account-model blockchains, consisting of a public and private balance. To hide the internal confidential information, they employ zk-SNARK when constructing the privacy-preserving transactions. Thus, it performs within the transformation between the public balance and the private balance to disconnect the linkage of users. Blockmaze is implemented by revising the blockchain core algorithm, restricting its deployment to other existing blockchains.

Quisquis [15] is a new anonymity system without a trusted setup. However, it has the possibility of a front-running attack by updating the public keys in anonymity set before the transaction is broadcasted. Moreover, this system is a standalone cryptocurrency that does not support deployment on any smart contract platform.

Zether [7] accomplishes privacy protection in the account-based model using ZKPs (Bulletproofs [9]) and the ElGamal encryption scheme. While Zether is stateful, it does not hide the identities of parties involved in the transaction. Moreover, since the sender should generate a zero-knowledge proof for the large user set for anonymity, it has limitations to support a high level of anonymity. Diamond [12] proposes "many out of many proofs" to enhance proving time for a subset of a fixed list of commitments. Nevertheless, the anonymity corresponding to all system users is not supported.

Among the privacy-preserving payment systems, some approaches allow auditability. Solidus [10] is a privacy-preserving protocol for asset transfer in which banks play a role as an arbitrator of mediation. Solidus utilizes ORAM to support updating accounts without revealing the values but cannot provide a dedicated audit function. Specifically, it can only offer auditing by revealing whole keys to an auditor and opening transactions.

zkLedger [24] and FabZK [20] enable anonymous payments via the use of homomorphic commitments and NIZK while supporting auditability. However, since these systems are designed based on organizational units, there is the problem of performance degradation as the number of organizations increases. Thus, they are practical only when there are a small number of organizations due to performance issues.

PGC [11] aims for a middle ground between privacy and auditability and then proposes auditable decentralized confidential payment using additively-homomorphic public-key encryption. However, the anonymity set size should be small in the approach. The work [3] proposes a privacy-preserving auditable token management system using NIZK. However, the work is designed for enterprise networks on a permissioned blockchain. Moreover, whenever transferring a token, a user should contact the privileged party called by Certifier, which checks if the transaction is valid.

Table 1 summarizes and compares some notable schemes described in the related works. Our research demonstrates the provision of both anonymity and confidentiality, along with the capability of auditability. Furthermore, our research displays a relatively lower transaction fee compared to the main related works, however, with limitations to a non-transparent setting.

III. PRELIMINARIES

In this section, we describe notations for standard cryptographic primitives.

A. NOTATIONS

Let λ be the security parameter. We denote randomly choosing $\leftarrow_{\$}$ as the standard notation. Let $\mathbb{F}$ denote a finite field and $\mathbb{G}$ denote a group. Given a security parameter 1^{λ}, a relation generator $\mathcal{RG}$ returns a polynomial time decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^{\lambda})$. For $(x, w) \in \mathcal{R}$, we say w is a witness to the statement (I/O) x is in the relation. Also, we utilize collision-resistant hash (CRH), a commitment scheme (COM), and pseudorandom function (PRF). Given an input x, we denote CRH as $y \leftarrow \mathsf{CRH}(x)$. For the commitment scheme, we define the commitment cm for a message u and an opening o as $\mathsf{cm} \leftarrow \mathsf{COM}(u; o)$. We notate the PRF output for a seed k and an input x as $\mathsf{PRF}_k(x)$.

TABLE 1. Comparison of Azeroth to main related works: The term 'Platform' indicates that it has its ledger. The notation '△' implies that it is difficult to support high anonymity, meaning that the anonymity set can only be limited to a small size.

System		Anonymity	Confidentiality	Auditability	Gas Fee	Solution Approach
Permissionless Blockchain		✗	✗	✓	low	n/a
Platform	Zcash	✓	✓	✗	n/a	zk-SNARK
	Monero	✓	✓	✗	n/a	Ring Signature
	Blockmaze	✓	✓	✗	n/a	zk-SNARK
	Quisquis	✓	✓	✗	n/a	NIZK
	zkLedger	✓	✓	✓	n/a	NIZK
Smart Contract	Zeth	✓	✓	✗	moderate	zk-SNARK
	Zether	△	✓	✗	high	NIZK
	PGC	△	✓	✓	high	NIZK
	Our Work	✓	✓	✓	moderate	zk-SNARK

B. ZK-SNARK

As described in [17] and [18], given a relation $\mathcal{R}$, a zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK) is composed of a set of algorithms $\Pi_{\mathsf{snark}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerProof}, \mathsf{SimProve})$ that works as follows.

- $\mathsf{Setup}(\lambda, \mathcal{R}) \rightarrow \mathsf{crs} := (\mathsf{ek}, \mathsf{vk}), \mathsf{td}$: The algorithm takes a security parameter λ and a relation $\mathcal{R}$ as input and returns a common reference string crs containing an evaluating key ek and a verification key vk, and a simulation trapdoor td.
- $\mathsf{Prove}(\mathsf{ek}, x, w) \rightarrow \pi$: The algorithm takes an evaluating key ek, a statement x, and a witness w such that $(x, w) \in \mathcal{R}$ as inputs and returns a proof π.
- $\mathsf{VerProof}(\mathsf{vk}, x, \pi) \rightarrow \mathsf{true}/\mathsf{false}$: The algorithm takes a verification key vk, a statement x, and a proof π as inputs, and returns true if the proof is correct, or false otherwise.
- $\mathsf{SimProve}(\mathsf{ek}, \mathsf{td}, x) \rightarrow \pi_{\mathsf{sim}}$: The $\mathsf{SimProve}$ algorithm takes a evaluating key ek, a simulation trapdoor td, and a statement x as inputs, and returns a proof π_{sim} such that $\mathsf{VerProof}(\mathsf{vk}, x, \pi_{\mathsf{sim}}) \rightarrow \mathsf{true}$.

Its properties are completeness, knowledge soundness, zero knowledge, and succinctness, as described below.

1) COMPLETENESS

The honest verifier always accepts the proof for any pair (x, w) satisfying the relation $\mathcal{R}$. Strictly, for ${}^{\forall}\lambda \in \mathbb{N}$, ${}^{\forall}\mathcal{R}_{\lambda}$, and ${}^{\forall}(x, w) \in \mathcal{R}_{\lambda}$, it holds as follow.

$$\Pr\left[\begin{matrix}(\mathsf{ek}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Setup}(\mathcal{R}); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w)\end{matrix} \middle| \mathsf{true} \leftarrow \mathsf{VerProof}(\mathsf{vk}, x, \pi)\right] = 1$$

2) KNOWLEDGE SOUNDNESS

Knowledge soundness says that the prover must know a witness if the honest prover outputs a proof π. Such knowledge can be extracted with a knowledge extractor $\mathcal{E}$ in polynomial time. To be more specific, if there exists a knowledge extractor $\mathcal{E}$ for any PPT adversary $\mathcal{A}$ such that $\Pr\left[\mathsf{Game}^{\mathsf{KS}}_{\mathcal{RG},\mathcal{A},\mathcal{E}} = \mathsf{true}\right] = \mathsf{negl}(\lambda)$, an argument system Π_{snark} has knowledge soundness.

$\mathsf{Game}^{\mathsf{KS}}_{\mathcal{RG},\mathcal{A},\mathcal{E}} \rightarrow \mathsf{res}$

$(\mathcal{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^{\lambda})$; $(\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}), \mathsf{td}) \leftarrow \mathsf{Setup}(\mathcal{R})$;

$(x, \pi) \leftarrow \mathcal{A}(\mathcal{R}, \mathsf{aux}_R, \mathsf{crs})$; $w \leftarrow \mathcal{E}(\mathsf{transcript}_{\mathcal{A}})$;

Return $\mathsf{res} \leftarrow (\mathsf{VerProof}(\mathsf{vk}, x, \pi) \wedge (x, \pi) \notin \mathcal{R})$

3) ZERO KNOWLEDGE

Simply, a zero-knowledge means that a proof π for $(x, w) \in \mathcal{R}$ on Π_{snark} only has information about the truth of the statement x. If a simulator exists such that the following conditions hold for any adversary $\mathcal{A}$, we say that Π_{snark} is zero-knowledge in Pr, as shown at the bottom of the next page.

4) SUCCINCTNESS

An arguments system Π is *succinctness* if it has a small proof size and fast verification time.

$$|\pi| \leq \mathsf{Poly}(\lambda)(\lambda + log|w|)$$
$$\mathsf{Time}_{\mathsf{VerProof}} \leq \mathsf{Poly}(\lambda)(\lambda + log|w| + |x|)$$

C. SYMMETRIC-KEY ENCRYPTION

We use a symmetric-key encryption scheme SE, a set of algorithms $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, which operates as follows.

- $\mathsf{Gen}(1^{\lambda}) \rightarrow \mathsf{k}$: The Gen algorithm takes a security parameter 1^{λ} and returns a key k.
- $\mathsf{Enc}_{\mathsf{k}}(\mathsf{msg}) \rightarrow \mathsf{sct}$: The Enc algorithm inputs a key k and a plaintext msg and returns a ciphertext sct.
- $\mathsf{Dec}_{\mathsf{k}}(\mathsf{sct}) \rightarrow \mathsf{msg}$: The Dec algorithm inputs a key k and a ciphertext sct. It returns a plaintext msg.

The encryption scheme SE satisfies ciphertext indistinguishability under chosen-plaintext attack **IND-CPA** security and key indistinguishability under chosen-plaintext attack **IK-CPA** security.

D. PUBLIC-KEY ENCRYPTION

We use a public-key encryption scheme $\mathsf{PE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which operates as follows.

- $\mathsf{Gen}(1^{\lambda}) \rightarrow (\mathsf{sk}, \mathsf{pk})$: The Gen algorithm takes a security parameter 1^{λ} and returns a key pair $(\mathsf{sk}, \mathsf{pk})$.

- $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{msg}) \rightarrow \mathsf{pct}$: The Enc algorithm takes a public key pk and a message msg as inputs and returns a ciphertext pct.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{pct}) \rightarrow \mathsf{msg}$: The Dec algorithm takes a private key sk and a ciphertext pct as inputs. It returns a plaintext msg.

The encryption scheme PE satisfies ciphertext indistinguishability under chosen-plaintext attack **IND-CPA** security and key indistinguishability under chosen-plaintext attack **IK-CPA** security.

*Remark:*To prove that encryption is performed correctly within a zk-SNARK circuit, we need random values used in encryption as a witness. We denote the values as aux. Depending on the context in our protocol, we denote the encryption such that it also outputs aux as a SNARK witness as follows.

$$(\mathsf{pct}, \mathsf{aux}) \leftarrow \mathsf{PE.Enc}_{\mathsf{pk}}(\mathsf{msg})$$

Supporting the audit function allows the trusted auditor to decrypt the message for the encrypted message. Thus, we need encryption with two recipients. We denote its encryption with a user public key *pk* and an auditor public key apk as follows.

$$(\mathsf{pct}, \mathsf{aux}) \leftarrow \mathsf{PE.Enc}_{\mathsf{pk},\mathsf{apk}}(\mathsf{msg})$$

IV. CONSTRUCTION OF Azeroth SCHEME

This section describes the data structures used in our proposed scheme Azeroth, referring to the notion. Subsequently, we present an overview of the Azeroth system, detailing its core techniques and providing a concrete description of its construction. The overview comprehensively explains the core function's overall structure and functionality.

A. DATA STRUCTURES

1) LEDGER

All users can access the ledger denoted as L, which contains the information of all blocks. Additionally, L is sequentially expanded out by appending new transactions to the previous one (i.e., for any $\mathsf{T}' < \mathsf{T}$, L_{T} always incorporates $\mathsf{L}_{\mathsf{T}'}$).

2) ACCOUNT

There are two types of accounts in Azeroth: an externally owned account denoted as EOA, and an encrypted account denoted as ENA. The former is the same one as in other account-based blockchains (e.g., Ethereum), and the latter is an account that includes a ciphertext indicating an amount in the account. EOA is maintained by the blockchain network and interacts with the smart contract, while A smart contract manages ENA registration and updates; users cannot see the value in ENA without its secret key.

3) AUDITOR KEY

An auditor generates a pair of private/public keys (ask, apk) used in the public key system; apk is used when a user generates an encrypted transaction, while ask is used when an auditor needs to audit the ciphertext.

4) USER KEY

Each user generates a pair of private/public keys ($\mathsf{usk} = (\mathsf{k}_{\mathsf{ENA}}, \mathsf{sk}_{\mathsf{own}}, \mathsf{sk}_{\mathsf{enc}})$, $\mathsf{upk} = (\mathsf{addr}, \mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}})$).

- $\mathsf{k}_{\mathsf{ENA}}$: It indicates a secret key for an encrypted account of ENA in a symmetric-key encryption system.
- $(\mathsf{sk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{own}})$: $\mathsf{pk}_{\mathsf{own}}$ is computed by hashing $\mathsf{sk}_{\mathsf{own}}$. The key pair is used to prove the ownership of an account in a transaction. Note that $\mathsf{sk}_{\mathsf{own}}$ is also used to generate a nullifier, preventing double-spending.
- $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}})$: These keys are used in a public-key encryption system; $\mathsf{sk}_{\mathsf{enc}}$ is used to decrypt ciphertexts taken from transactions while $\mathsf{pk}_{\mathsf{enc}}$ is used to encrypt transactions.
- addr : It is a user address computed by hashing $\mathsf{pk}_{\mathsf{own}}$ and $\mathsf{pk}_{\mathsf{enc}}$.

5) COMMITMENT AND NOTE

We use a commitment scheme to construct a privacy-preserving transaction in which a commitment is utilized to hide sensitive information (i.e., amount, address). Our commitment is noted as follows.

$$\mathsf{cm} = \mathsf{COM}(\mathsf{v}, \mathsf{addr}; \mathsf{o})$$

To commit, it takes v, addr as inputs and runs with an opening o. v is the digital asset value to be transferred and addr is the address of a recipient. Once cm is published on a blockchain, the recipient with the opening key o and the value v from the encrypted transaction uses them to make another transfer. We denote the data required to spend a commitment as a note:

$$\mathsf{note} = (\mathsf{cm}, \mathsf{o}, \mathsf{v})$$

Note that each user privately stores his notes in his wallet for convenience.

$$\Pr\left[\begin{array}{c}(\mathcal{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda); (\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}), \mathsf{td}) \leftarrow \Pi.\mathsf{Setup}(\mathcal{R}) \\ : \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w); \mathsf{true} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi)\end{array}\right]$$
$$\approx$$
$$\Pr\left[\begin{array}{c}(\mathcal{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda); (\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}), \mathsf{td}) \leftarrow \mathsf{Setup}(\mathcal{R}) \\ : \pi_{\mathsf{sim}} \leftarrow \mathsf{SimProve}(\mathsf{ek}, \mathsf{td}, x); \mathsf{true} \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi_{\mathsf{sim}})\end{array}\right]$$

6) MEMBERSHIP BASED ON MERKLE TREE

We use a Merkle hash tree to prove the membership of commitments in Azeroth and denote the Merkle tree and its root as MT and rt, respectively. MT holds all commitments in L, and it appends commitments to nodes and updates rt when new commitments are given. Additionally, an authentication co-path from a commitment cm to rt is denoted as $\mathsf{Path}_{\mathsf{cm}}$. For any given time T, MT_{T} includes a list of all commitments and rt of these commitments. There are three algorithms related to MT.

- $\mathsf{true/false} \leftarrow \mathsf{Membership}_{\mathsf{MT}}(\mathsf{rt}, \mathsf{cm}, \mathsf{Path}_{\mathsf{cm}})$: This algorithm verifies if cm is included in MT rooted by rt; if rt is the same as a computed hash value from the commitment cm along the authentication path $\mathsf{Path}_{\mathsf{cm}}$, it returns true.
- $\mathsf{Path}_{\mathsf{cm}} \leftarrow \mathsf{ComputePath}_{\mathsf{MT}}(\mathsf{cm})$: This algorithm returns the authentication co-path from a commitment cm appearing in MT.
- $\mathsf{rt}_{\mathsf{new}} \leftarrow \mathsf{TreeUpdate}_{\mathsf{MT}}(\mathsf{cm})$: This algorithm appends a new commitment cm, performs hash computation for each tree layer, and returns a new tree root $\mathsf{rt}_{\mathsf{new}}$.

7) VALUE TYPE

A transaction includes several publicly visible or privately secured input/output asset values. In our description, pub and priv represent publicly visible and encrypted (or committed) values, respectively. "in" indicates the value to be deposited to one's account, and "out" represents the value to be withdrawn from one's account. We summarize the types of digital asset values as follows.

- $\mathsf{v}^{\mathsf{ENA}}$: The digital asset value is available in the encrypted account ENA.
- $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$: The digital asset value is to be publicly transferred from the sender's EOA and the digital asset value to the receiver's public account EOA, respectively.
- $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$: The digital asset value received anonymously from an existing commitment and the value sent anonymously to a new commitment in MT, respectively.

B. OVERVIEW

We construct Azeroth by integrating deposit/withdrawal transactions and public/private transfer transactions into a single transaction zkTransfer. Since zkTransfer executes multi-functions in the same structure, it improves function anonymity. One may try to guess which function is executed by observing the input/output values in zkTransfer. zkTransfer, however, reveals the input/output values only in EOA; the values withdrawn/deposited from/to ENA and the values transferred from/to MT are hidden. A membership proof of MT hides the recipient's address. As a result, the information that an observer can extract from the transaction is that someone's EOA value either increases or decreases; he cannot know whether the amount of difference is deposited/withdrawn to/from its own ENA or is transferred from/to a new commitment in MT. It is even more complicated because those values can be mixed in a range where the sum of the input values equals the sum of the output values.

zkTransfer implements a private transfer with only two transactions; a sender executes zkTransfer transferred to MT and a receiver executes zkTransfer transferred from MT. In zkTransfer, all values in ENA and MT are processed as ciphertexts. Whether remittance is between own or non-own accounts is hidden, so the linking information between the sender and receiver is protected.

Figure 1 illustrates the zkTransfer. The left box "IN" represents input values, and the right box "OUT" denotes output values. In zkTransfer, $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ are publicly visible values. $\mathsf{v}^{\mathsf{ENA}}$ is obtained by decrypting its encrypted account value sct. The updated $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}}$ is encrypted and stored as sct^* in ENA. The amount ($\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$) included in a commitment can be used as input if a user has its opening key; the opening key is delivered in a ciphertext pct so that only the destined user can correctly decrypt it. To prevent double spending, for each spent commitment, a nullifier is generated by hashing the commitment and the private key $\mathsf{sk}_{\mathsf{own}}$ and appended; it is still unlinkable between the commitment and the nullifier without the private key $\mathsf{sk}_{\mathsf{own}}$. Finally, zkTransfer proves that all of the above procedures are correctly performed by generating a zk-SNARK proof for transaction validity.

Auditability is achieved by utilizing public-key encryption with two recipients; all pct ciphertexts can be decrypted by an auditor and a receiver so that the auditor can monitor all the transactions. We note that ENA exploits symmetric-key encryption only for the performance gain, although ENA can also utilize the public-key encryption. Notice that without decrypting ENA, the auditor can still learn the value change in ENA by computing the remaining values from $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$, $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$, and $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$.

C. CONSTRUCTION

Azeroth consists of three components: Client, Smart Contract, and Relation. Client generates a transaction with a ciphertext, a commitment, and a proof. Smart Contract denotes a smart contract running on a blockchain. Relation represents a zk-SNARK circuit for generating a zkTransfer proof.

1) [Azeroth Client]

We define the term 'Client' to encompass not only users but also auditors and trusted parties in the context outside of the blockchain. Intuitively, our Client algorithm is a tuple of algorithms defined as stated below. We formally define the algorithm of the Client and describe it in fig. 2.

- $\mathsf{Setup}_{\mathsf{Client}}$: A trusted party runs this algorithm only once to set up the whole system. It also returns the public parameter pp.
- $\mathsf{KeyGenAudit}_{\mathsf{Client}}$: This algorithm generates an auditor key pair (ask, apk). It also outputs the key pair and a transaction $\mathsf{Tx}_{\mathsf{KGA}}$.

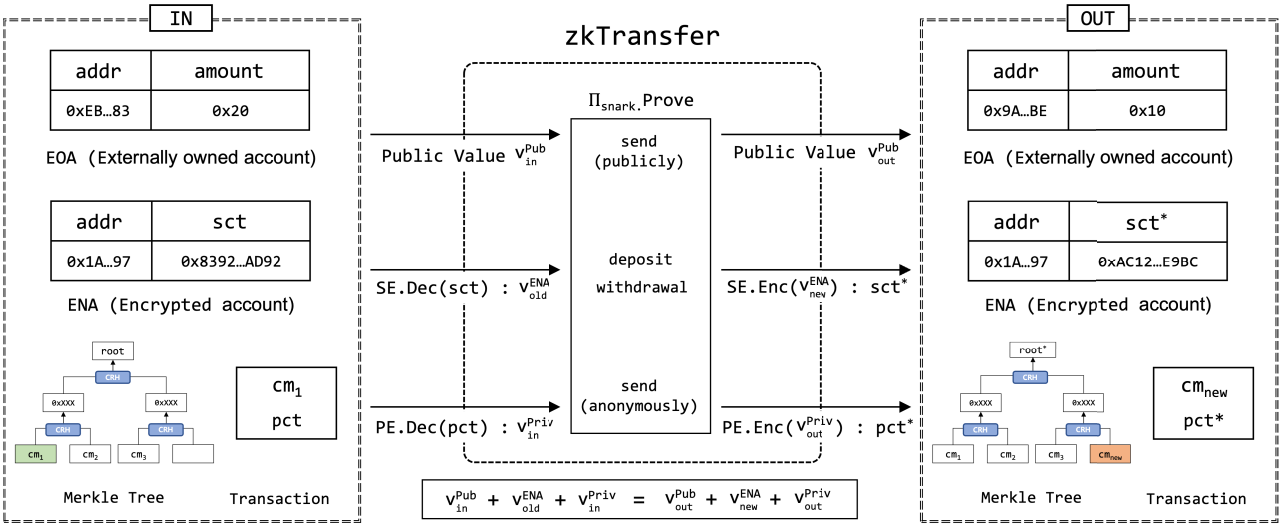


FIGURE 1. Overview of zk Transfer.

- $\mathsf{KeyGenUser}_{\mathsf{Client}}$: This algorithm generates a user key pair $(\mathsf{usk}, \mathsf{upk})$. It also returns a transaction $\mathsf{Tx}_{\mathsf{KGU}}$ to register the user's public key.
- $\mathsf{zkTransfer}_{\mathsf{Client}}$: A user executes this algorithm for transfer. The internal procedures are described as follows:
  i) Consuming $\mathsf{note} = (\mathsf{cm}, \mathsf{o}, \mathsf{v})$: It proves the knowledge of the committed value v using the opening key o and the membership of a commitment cm in MT and derives a nullifier nf from PRF to nullify the used commitment.
  ii) Generating $\mathsf{cm}_{\mathsf{new}}$: By executing $\mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$, a new commitment and its opening key are obtained. Then it encrypts $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}})$ via $\mathsf{PE.Enc}$ and outputs pct.
  iii) Processing currency: The sender's ENA balance is updated based on $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$ (from **note**), $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$, and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ or $\Delta v^{\mathsf{ENA}} = \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$.
  With prepared witnesses and statements, the algorithm generates a zk-SNARK proof and finally outputs a $\mathsf{zkTransfer}$ transaction $\mathsf{Tx}_{\mathsf{ZKT}}$.
- $\mathsf{RetreiveNote}_{\mathsf{Client}}$: This algorithm is a *sub-algorithm* computing a note used in $\mathsf{zkTransfer}_{\mathsf{Client}}$. It allows a user to find cm transferred to the user along with its opening key and its committed value. Then, a user decrypts the ciphertext using $\mathsf{sk}_{\mathsf{enc}}$ each transaction $\mathsf{pct} \in \mathsf{L}$ to $(\mathsf{o}, \mathsf{v}, \mathsf{addr}^*)$ and stores $(\mathsf{cm}, \mathsf{o}, \mathsf{v})$ as note in the user's wallet if addr^* matches its address addr.
- $\mathsf{Audit}_{\mathsf{Client}}$: An auditor with a valid ask runs this algorithm to audit a transaction by decrypting the ciphertext pct in the transaction.

2) [Azeroth Smart Contract]

The $\mathsf{Smart\ Contract}$ algorithm describes the processes involved in the deployment and specifies only the essential parts. The $\mathsf{Azeroth\ Smart\ Contract}$ comprises several algorithms, as described below. Note that within the algorithms, '*this*' refers to the $\mathsf{Azeroth}$ contract address, and $\mathsf{TransferFrom}$ is a simple public transfer function from the sender to the receiver. The specific and formal description is provided in fig. 3.

- $\mathsf{Setup}_{\mathsf{SC}}$: This algorithm deploys a smart contract and stores the verification key vk from zk-SNARK where vk is used to verify a zk-SNARK proof in the smart contract.
- $\mathsf{RegisterAuditor}_{\mathsf{SC}}$: This algorithm stores an auditor public key apk in $\mathsf{Azeroth}$'s smart contract.
- $\mathsf{RegisterUser}_{\mathsf{SC}}$: This algorithm registers a new encrypted account for address addr. The transaction is reverted if the address already exists in $\mathsf{List}_{\mathsf{addr}}$. Otherwise, it registers a new ENA and initializes it with zero amount.
- $\mathsf{zkTransfer}_{\mathsf{SC}}$: This algorithm checks the validity of the transaction and processes the transaction. A transaction is valid iff: Merkle root $\mathsf{rt}_{\mathsf{old}}$ exists in root list $\mathsf{List}_{\mathsf{rt}}$, a nullifier nf does not exists in the nullifier list $\mathsf{List}_{\mathsf{nf}}$, $\mathsf{addr}^{\mathsf{send}}$ exists, $\mathsf{cm}_{\mathsf{new}}$ does not exists in $\mathsf{List}_{\mathsf{cm}}$, and a proof π is valid in zk-SNARK. If the transaction is valid, the $\mathsf{cm}_{\mathsf{new}}$ is appended to MT, MT is updated, a new Merkle tree root $\mathsf{rt}_{\mathsf{new}}$ is added to $\mathsf{List}_{\mathsf{rt}}$ and the nullifier nf is appended to $\mathsf{List}_{\mathsf{nf}}$. The encrypted account is updated. And then the public amounts are processed; $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ is acquired from $\mathsf{EOA}^{\mathsf{send}}$, and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ is delivered to $\mathsf{EOA}^{\mathsf{recv}}$. If the transaction is invalid, it is reverted and aborted.

3) [Azeroth Relation]

The statement $\vec{x}$ and witness $\vec{w}$ of $\mathsf{Relation}$ $\mathcal{R}_{\mathsf{ZKT}}$ are as shown at the bottom of page 9.

Intuitively, we say that a witness $\vec{w}$ is valid for a statement $\vec{x}$, if and only if the following holds:

- If $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} > 0$, then $\mathsf{cm}_{\mathsf{old}}$ must exist in MT with given rt and Path.

$\mathsf{Setup}_{\mathsf{Client}}(1^\lambda, \mathcal{R}_{\mathsf{ZKT}})$

$(\mathsf{ek}, \mathsf{vk}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathcal{R}_{\mathsf{ZKT}})$
$G \leftarrow\$\ \mathbb{G}$ / Choose a generator
return $\mathsf{pp} := (\mathsf{ek}, \mathsf{vk}, G, 1^\lambda)$

$\mathsf{KeyGenAudit}_{\mathsf{Client}}(\mathsf{pp})$

$(\mathsf{ask}, \mathsf{apk}) \leftarrow \mathsf{PE.Gen}(\mathsf{pp})$
$\mathsf{Tx}_{\mathsf{KGA}} = \mathsf{apk}$
return $(\mathsf{apk}, \mathsf{ask}), \mathsf{Tx}_{\mathsf{KGA}}$

$\mathsf{Audit}_{\mathsf{Client}}(\mathsf{ask}, \mathsf{pct})$

$(\mathsf{o}, \mathsf{v}, \mathsf{addr}) \leftarrow \mathsf{PE.Dec}_{\mathsf{ask}}(\mathsf{pct})$
$\mathsf{msg} := (\mathsf{o}, \mathsf{v}, \mathsf{addr})$
return msg

$\mathsf{KeyGenUser}_{\mathsf{Client}}(\mathsf{pp})$

$(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \leftarrow \mathsf{PE.Gen}(\mathsf{pp})$
$\mathsf{k}_{\mathsf{ENA}} \leftarrow \mathsf{SE.Gen}(\mathsf{pp})$
$\mathsf{sk}_{\mathsf{own}} \leftarrow\$\ \mathbb{F}$
$\mathsf{pk}_{\mathsf{own}} \leftarrow \mathsf{CRH}(\mathsf{sk}_{\mathsf{own}})$
$\mathsf{addr} \leftarrow \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}} || \mathsf{pk}_{\mathsf{enc}})$
$\mathsf{usk} := (\mathsf{k}_{\mathsf{ENA}}, \mathsf{sk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}})$
$\mathsf{upk} := (\mathsf{addr}, \mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}})$
$\mathsf{Tx}_{\mathsf{KGU}} := \mathsf{addr}$
return $(\mathsf{usk}, \mathsf{upk}), \mathsf{Tx}_{\mathsf{KGU}}$

$\mathsf{RetreiveNote}_{\mathsf{Client}}(\mathsf{L}, \mathsf{usk}, \mathsf{upk})$

parse usk as $(\mathsf{k}_{\mathsf{ENA}}, \mathsf{sk}_{\mathsf{own}}, \mathsf{sk}_{\mathsf{enc}})$
parse upk as $(\mathsf{addr}, \mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}})$
for each $\mathsf{Tx}_{\mathsf{ZKT}} \in \mathsf{L}$ **do**
  parse $\mathsf{Tx}_{\mathsf{ZKT}}$ as $(\mathsf{cm}, \mathsf{pct}, \dots)$
  $(\mathsf{o}, \mathsf{v}, \mathsf{addr}^*) \leftarrow \mathsf{PE.Dec}_{\mathsf{sk}_{\mathsf{enc}}}(\mathsf{pct})$
  if $\mathsf{addr} = \mathsf{addr}^*$ **then**
    return note $= (\mathsf{cm}, \mathsf{o}, \mathsf{v})$
  endif
endfor
return $\perp$

$\mathsf{zkTransfer}_{\mathsf{Client}}(\mathsf{pp}, \mathsf{note}, \mathsf{apk}, \mathsf{usk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{EOA}^{\mathsf{recv}})$

parse $\mathsf{usk}^{\mathsf{send}}$ as $(\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{enc}}^{\mathsf{send}})$ / Parse sender's secret key
parse $\mathsf{upk}^{\{\mathsf{send},\mathsf{recv}\}}$ as $(\mathsf{addr}^{\{\mathsf{send},\mathsf{recv}\}}, \mathsf{pk}_{\mathsf{own}}^{\{\mathsf{send},\mathsf{recv}\}}, \mathsf{pk}_{\mathsf{enc}}^{\{\mathsf{send},\mathsf{recv}\}})$ / Parse both sender and receiver public keys
if note $\neq \perp$ **then**
  parse note as $(\mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}})$
else / Initial state without receiving zkTransfer from anyone
  $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} \leftarrow 0; \mathsf{o}_{\mathsf{old}} \leftarrow\$\ \mathbb{F}$ / Choose arbitrary values
  $\mathsf{cm}_{\mathsf{old}} \leftarrow \mathsf{COM}(\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}}; \mathsf{o}_{\mathsf{old}})$
endif
$\mathsf{sct}_{\mathsf{old}} \leftarrow \mathsf{ENA}[\mathsf{addr}^{\mathsf{send}}]; \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \leftarrow \mathsf{SE.Dec}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{sct}_{\mathsf{old}})$ / Compute ENA amount with own secret key via SE.Dec
$\mathsf{nf} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}}(\mathsf{cm}_{\mathsf{old}})$ / Generate a nullifier due to double-spending
$\mathsf{rt} \leftarrow \mathsf{List}_{\mathsf{rt}}.\mathsf{Top}$ / Get a Merkle tree MT's root
  $\mathsf{Path} \leftarrow \mathsf{ComputePath}_{\mathsf{MT}}(\mathsf{cm}_{\mathsf{old}})$ / Get an authentication co-path
$\mathsf{cm}_{\mathsf{new}} \leftarrow \mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$ / Compute a new commitment
$\mathsf{pct}_{\mathsf{new}}, \mathsf{aux}_{\mathsf{new}} \leftarrow \mathsf{PE.Enc}_{\mathsf{pk}_{\mathsf{enc}}^{\mathsf{recv}}, \mathsf{apk}}(\mathsf{o}_{\mathsf{new}} || \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} || \mathsf{addr}^{\mathsf{recv}})$
$\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} \leftarrow \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ / Compute a new ENA amount
$\mathsf{sct}_{\mathsf{new}} \leftarrow \mathsf{SE.Enc}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}})$ / Encrypt the new ENA amount
$\vec{x} = \left\{\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{upk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}}\right\}$
$\vec{w} = \left\{\mathsf{usk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{aux}_{\mathsf{new}}, \mathsf{Path}\right\}$
$\pi \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{ek}, \vec{x}, \vec{w})$ / Generate a zk-SNARK proof over $\mathcal{R}_{\mathsf{ZKT}}(\vec{x}, \vec{w})$
$\mathsf{Tx}_{\mathsf{ZKT}} := (\pi, \mathsf{rt}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{EOA}^{\mathsf{recv}})$
return $\mathsf{Tx}_{\mathsf{ZKT}}$

FIGURE 2. Azeroth's client(Client) algorithms.

- $\mathsf{pk}_{\mathsf{own}}^{\mathsf{send}} = \mathsf{CRH}(sk_{\mathsf{own}}^{\mathsf{send}})$.
- The user address $\mathsf{addr}^{\mathsf{send}}$ and $\mathsf{addr}^{\mathsf{recv}}$ are well-formed.
- $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{cm}_{\mathsf{new}}$ are valid.
- nf is derived from $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}$.
- $\mathsf{pct}_{\mathsf{new}}$ is an encryption of $\mathsf{cm}_{\mathsf{new}}$ via $\mathsf{aux}_{\mathsf{new}}$.
- $\mathsf{sct}_{\mathsf{new}}$ is an encryption of updated ENA balance.
- All amounts (e.g., $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$, ...) are not negative.

We detail the relations mentioned above and present them in fig. 4.

V. SECURITY

Following the similar model defined in [5] and [19], we define the security properties of Azeroth including *ledger indistinguishability*, *transaction non-malleability*, and *balance*, and define *auditability* as a new property.

Before describing the security definition for each property and its experiment, we assume that there exists a (stateful) Azeroth oracle $\mathcal{O}^{\mathsf{Azeroth}}$ answering queries from an adversary $\mathcal{A}$ who uses a challenger $\mathcal{C}$ in the role of performer for the experiment's sanity checks. First, we recount how

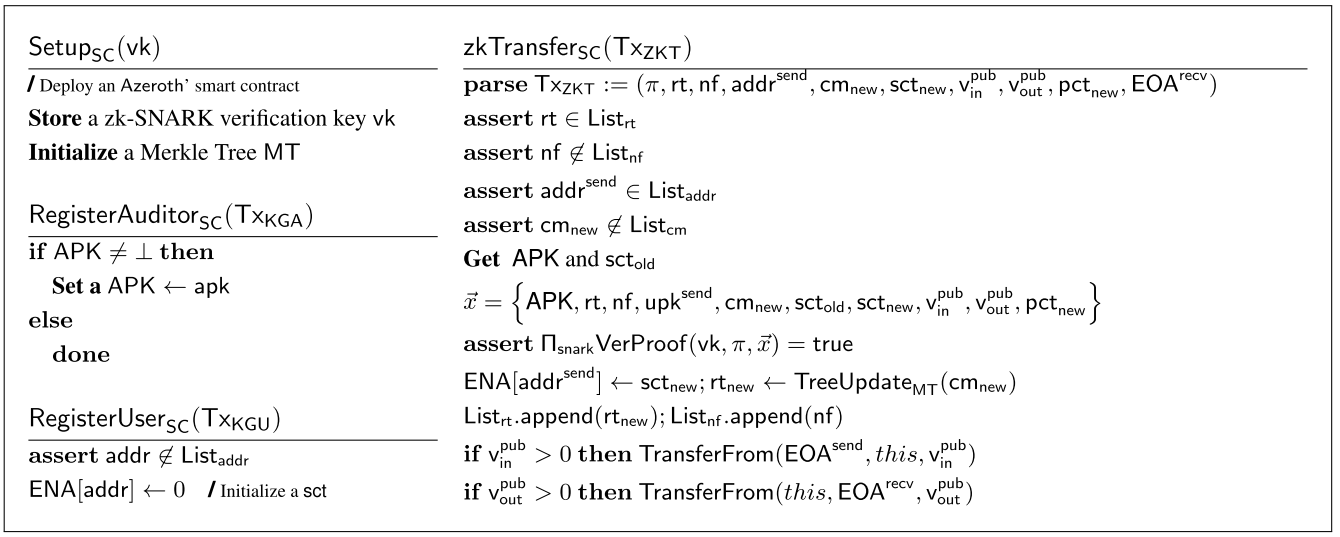


FIGURE 3. Azeroth's Smart Algorithms.

$\mathcal{O}^{\mathsf{Azeroth}}$ works. Given a list of public parameters pp, the oracle $\mathcal{O}^{\mathsf{Azeroth}}$, and auditor public key apk is initialized and retains its state of which it has the elements internally : [I] L, a ledger; [II] Acct, a set of account key pairs; [III] NOTE, a set of notes. In the beginning, all of the elements are empty. Note that, regarding **KeyGenAudit**, it is already presupposed that it has undergone the initialization process within the ledger. In the case of **RetreiveNote**, it serves as a subsidiary algorithm for readability purposes and, therefore, cannot be considered the primary algorithm. As such, the scope of the query is limited to $\mathcal{Q} = \{\mathsf{KeyGenUser}, \mathsf{zkTransfer}\}$. We describe each query type $\mathcal{Q}$ and how it works as follows.

$\mathcal{Q}$(KeyGenUser) :

i) Compute a key pair (usk = ($\mathsf{k_{ENA}}$, $\mathsf{sk_{own}}$, $\mathsf{sk_{enc}}$), upk = (addr, $\mathsf{pk_{own}}$, $\mathsf{pk_{enc}}$), $\mathsf{Tx_{KGU}}$) := KeyGenUser(pp)
ii) ADD the key pair (usk, upk) to Acct.
iii) Register the ENA address addr to L, and initialize ENA[addr] to 0.
iv) Add the KeyGenUser transaction $\mathsf{Tx_{KGU}}$ to L
v) Output the public key upk.

$\mathcal{Q}$(zkTransfer, note, upk^{send}, upk^{recv}, $\mathsf{v_{out}^{priv}}$, $\mathsf{v_{in}^{pub}}$, $\mathsf{v_{out}^{pub}}$, EOA^{recv})

i) Compute rt over all commitment in L
ii) Find usk^{send} in Acct. If no such key usk^{send}, then $\mathcal{O}^{\mathsf{Azeroth}}$ aborts.
iii) Get an auditor public key apk in L.
iv) Compute ($\mathsf{Tx_{ZKT}}$, note) := zkTransfer(note, apk, usk^{send}, upk^{send}, upk^{recv}, $\mathsf{v_{out}^{priv}}$, $\mathsf{v_{in}^{pub}}$, $\mathsf{v_{out}^{pub}}$, EOA^{recv}).
v) Add a new note note to NOTE.
vi) Add the zkTransfer transaction $\mathsf{Tx_{ZKT}}$ to L.
vii) Parse usk as ($\mathsf{k_{ENA}}$, $\mathsf{sk_{own}}$, $\mathsf{sk_{enc}}$).
viii) If any of the above operations fail, $\mathcal{O}^{\mathsf{Azeroth}}$ aborts. Otherwise, output $\perp$.

Remark: $\mathcal{O}^{\mathsf{Azeroth}}$ additionally provides adversaries with a way to directly add zkTransfer transaction to L. In other words, an adversary can use a zkTransfer query to cause $\mathsf{Tx_{ZKT}}$ in L, or if he has generated a key himself and knows all the information about the key, he can add $\mathsf{Tx_{ZKT}}$ to L without asking a zkTransfer query. We name its query as Insert.

1) LEDGER INDISTINGUISHABILITY

Informally, we define the ledger as *indistinguishable* if it satisfies a security property that prevents an adversary $\mathcal{A}$ from learning new information, even when the adversary can observe all public information adaptively interact with honest parties to execute Azeroth functions. More specifically, if there exist two ledgers L_0 and L_1 constructed by the adversary using oracle queries, $\mathcal{A}$ cannot distinguish between these ledgers. Additionally, note that the property implies the *unlinkability* between the sender and the receiver on the blockchain. Any information containing the users' details is not publicly available; hence, a ledger is distinguishable if a linkage between users is identified.

$$\vec{x} = (\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{upk^{send}}, \mathsf{cm_{new}}, \mathsf{sct_{old}}, \mathsf{sct_{new}}, \mathsf{v_{in}^{pub}}, \mathsf{v_{out}^{pub}}, \mathsf{pct_{new}})$$
$$\vec{w} = (\mathsf{usk^{send}}, \mathsf{cm_{old}}, \mathsf{o_{old}}, \mathsf{v_{in}^{priv}}, \mathsf{upk^{recv}}, \mathsf{o_{new}}, \mathsf{v_{out}^{priv}}, \mathsf{aux_{new}}, \mathsf{Path})$$

Azeroth relation $\mathcal{R}(\vec{x};\vec{w})$

$\vec{x} = \left\{\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{upk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}}\right\}$

$\vec{w} = \left\{\mathsf{usk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{aux}_{\mathsf{new}}, \mathsf{Path}\right\}$

parse $\mathsf{usk}^{\mathsf{send}}$ as $\left(\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{enc}}^{\mathsf{send}}\right)$

parse $\mathsf{upk}^{\{\mathsf{send},\mathsf{recv}\}}$ as $\left(\mathsf{addr}^{\{\mathsf{send},\mathsf{recv}\}}, \mathsf{pk}_{\mathsf{own}}^{\{\mathsf{send},\mathsf{recv}\}}, \mathsf{pk}_{\mathsf{enc}}^{\{\mathsf{send},\mathsf{recv}\}}\right)$

if $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} > 0$ **then**

  assert true $= \mathsf{Membership}_{\mathsf{MT}}(\mathsf{rt}, \mathsf{cm}_{\mathsf{old}}, \mathsf{Path})$

endif

assert $\mathsf{pk}_{\mathsf{own}}^{\mathsf{send}} = \mathsf{CRH}(\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}})$

assert $\mathsf{addr}^{\mathsf{send}} = \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}}^{\mathsf{send}} || \mathsf{pk}_{\mathsf{enc}}^{\mathsf{send}})$

assert $\mathsf{cm}_{\mathsf{old}} = \mathsf{COM}(\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}}; \mathsf{o}_{\mathsf{old}})$

assert $\mathsf{nf} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}}(\mathsf{cm}_{\mathsf{old}})$

assert $\mathsf{pct}_{\mathsf{new}}, \mathsf{aux}_{\mathsf{new}} = \mathsf{PE.Enc}_{\mathsf{pk}_{\mathsf{enc}}^{\mathsf{recv}}, \mathsf{apk}}(\mathsf{o}_{\mathsf{new}} || \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} || \mathsf{addr}^{\mathsf{recv}})$

assert $\mathsf{addr}^{\mathsf{recv}} = \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}}^{\mathsf{recv}} || \pi_{\mathsf{enc}}^{\mathsf{recv}})$

assert $\mathsf{cm}_{\mathsf{new}} = \mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$

if $\mathsf{sct}_{\mathsf{old}} = 0$ **then** $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \leftarrow 0$

else $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \leftarrow \mathsf{SE.Dec}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{sct}_{\mathsf{old}})$ **endif**

assert $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} \leftarrow \mathsf{SE.Dec}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{sct}_{\mathsf{new}})$

assert $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} = \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$

assert $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} \geq 0; \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} \geq 0; \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} \geq 0; \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}} \geq 0$

assert $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} \geq 0; \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \geq 0$

FIGURE 4. Azeroth's Relation.

We formally describe ledger indistinguishability using an experiment L-IND, as depicted in fig. 5. Before describing the experiment, we define the notion of *public consistency* for a pair of queries.

Definition 5.1 (Public Consistency): A pair of queries $(\mathcal{Q}, \mathcal{Q}')$ is publicly consistent if two queries $(\mathcal{Q}, \mathcal{Q}')$ must be the same type and publicly consistent in $\mathcal{A}$'s viewpoint.

- If $(\mathcal{Q}, \mathcal{Q}')$ are of type KeyGenUser, they are always publicly consistent. The same key can be generated in the special case of KeyGenUser.
- If $(\mathcal{Q}, \mathcal{Q}')$ are both of type zkTransfer, then $\mathcal{Q}, \mathcal{Q}'$ must be well-formed respectively and jointly consistent with respect to public information and $\mathcal{A}$'s view as follows.
  a) note in $\mathcal{Q}$ and $\mathcal{Q}'$ must appear in the ledger oracles' NOTE table.
  b) The notes in two queries are unspent, which means their serial number must not appear in a valid $\mathsf{Tx}_{\mathsf{ZKT}}$ transaction on the corresponding oracle's ledger.
  c) The sender addresses $\mathsf{addr}^{\mathsf{send}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must match the addresses of their note.
  d) The balance equation must hold.
  $$\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} = \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}} > 0$$
  e) The public values $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
  f) The receiver's external addresses $\mathsf{EOA}^{\mathsf{recv}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
  g) The transaction strings in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
  h) If the recipient's public key $\mathsf{upk}^{\mathsf{recv}}$ in $\mathcal{Q}$ is not in Acct, then $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal (and vice versa for $\mathcal{Q}'$).
  i) If any of note in $(\mathcal{Q}, \mathcal{Q}')$ is generated by an Insert query, both note in $(\mathcal{Q}, \mathcal{Q}')$ must be generated by an Insert.

Definition 5.2: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, RetreiveNote, zkTransfer,Audit) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is L-IND secure if the following equation holds:

$$\Pr\left[\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{L\text{-}IND}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

2) TRANSACTION NON-MALLEABILITY

Transaction non-malleability refers to the property in that a transaction cannot be modified in a way that changes the personal data, such as the secret key, associated with it. Intuitively, a transaction is non-malleable if no transaction can be constructed with incorrect personal data (i.e., secret key). We formalize this property with an experiment TR-NM, shown in fig. 5, where a PPT adversary $\mathcal{A}$ attempts to break a given Azeroth scheme. It is worth noting that $\mathcal{A}$ could also be an auditor trying to attack our scheme with their private key ask. We describe the TR-NM experiment in fig. 5.

Definition 5.3: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, RetreiveNote, zkTransfer,Audit) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is TR-NM secure if the following equation holds:

$$\Pr\left[\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{TR\text{-}NM}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

3) BALANCE

We define the balance property of Azeroth as the property that prevents an attacker from spending more than she has or receiving more than what is allowed. We formalize this property with an experiment BAL as shown in fig. 5.

Definition 5.4: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, RetreiveNote, zkTransfer,Audit) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is BAL secure if the following equation holds:

$$\Pr\left[\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{BAL}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

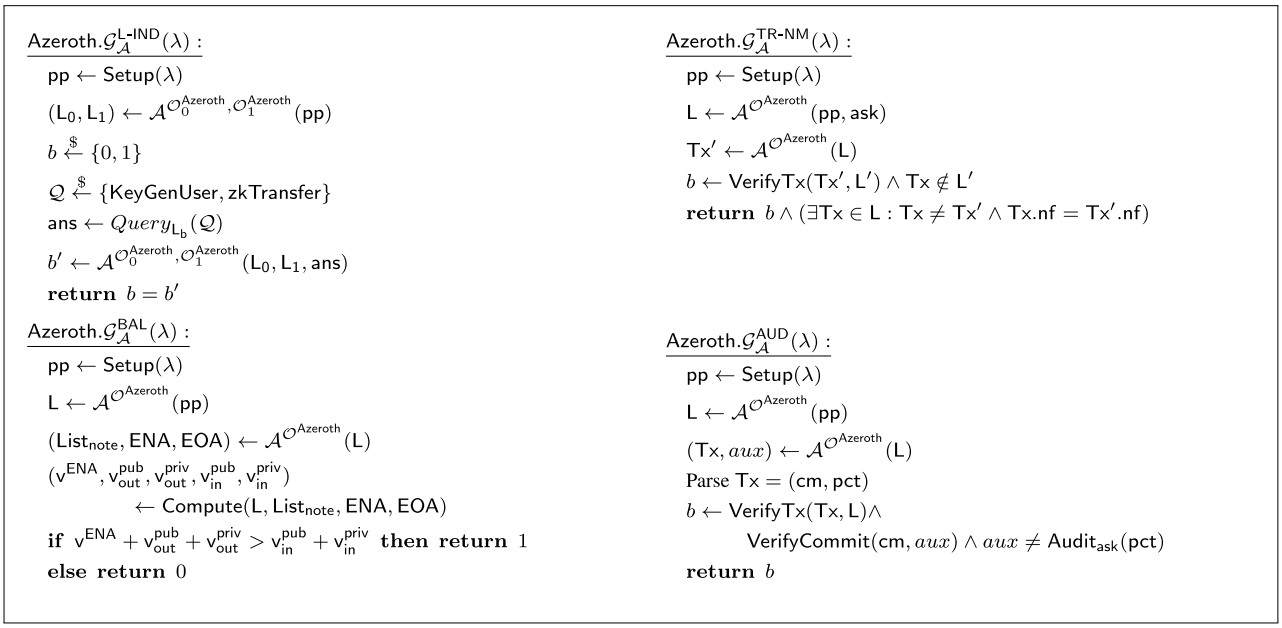


FIGURE 5. The experiments to ledger indistinguishability (L-IND), transaction non-malleability(TR-NM), balance(BAL), and auditability(AUD). In $\mathcal{G}_{\mathcal{A}}^{\text{BAL}}(\lambda)$, we use $\text{List}_{\text{note}}$ to denote a table of note and ENA to denote the new encrypted account balance. Compute is denoted as a function that computes balance values with returned variables from $\mathcal{A}$.

4) AUDITABILITY

If the auditor can always monitor the confidential data of any user, we informally say that the scheme has *auditability*. More precisely, we define that Azeroth is auditable if there is no transaction in which the decrypted plaintext differs from commitment openings. We define an experiment AUD as shown in fig. 5. Let *aux* be the auxiliary input consisting of the committed value, it is opening, and $\mathsf{addr}^{\text{recv}}$, utilized when verifying the commitment. If the commitment is correct, the function VerifyCommit returns 1; otherwise returns 0.

Definition 5.5: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, RetreiveNote, zkTransfer, Audit) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is AUD secure if the following equation holds:

$$\Pr\left[\text{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\text{AUD}}(\lambda) = 1\right] \leq \text{negl}(\lambda)$$

Theorem 5.1: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, RetreiveNote, zkTransfer,Audit) be a Azeroth scheme in fig. 2. Π_{Azeroth} satisfies *ledger indistinguishability*, *transaction non-malleability*, *balance*, and *auditability*.

A. SECURITY PROOFS

We now formally prove Theorem 5.1 by showing that Azeroth construction satisfies ledger indistinguishability, transaction non-malleability, balance, and auditability.

1) LEDGER INDISTINGUISHABILITY

By using a hybrid game, we prove ledger indistinguishability. Thus, we say that it is indistinguishable if the difference between a real game $\text{Game}_{\text{Real}}$ and a simulation game Game_{Sim} is negligible. All Games are executed by the interaction of an adversary $\mathcal{A}$ with a challenger $\mathcal{C}$, as in the L-IND experiment. However, Game_{Sim} is distinct from the others since it runs regardless of a bit b where it means a chosen bit from the L-IND experiment. Thus, for Game_{Sim}, the advantage of $\mathcal{A}$ is 0. Moreover, the zk-SNARK keys are generated as (ek, vk, td) ← $\Pi_{\text{snark}}.\text{Sim}(\mathcal{R})$ to obtain the zero-knowledge trapdoor td. We now show that $\mathbf{Adv}^{\text{L-IND}}_{\Pi_{\text{Azeroth}},\mathcal{A}}$ is at most negligibly different than $\mathbf{Adv}^{\text{Game}_{\text{Sim}}}$. First of all, we define the notations as follows.

TABLE 2. Notations.

Symbol	Meaning
$\text{Game}_{\text{Real}}$	The original L-IND experiment
Game_i	A hybrid game altered from $\text{Game}_{\text{Real}}$
Game_{Sim}	The fake L-IND experiment
q_{KGU}	The total number of KeyGenUser queries by $\mathcal{A}$
q_{ZKT}	The total number of zkTransfer queries by $\mathcal{A}$
$\mathbf{Adv}^{\text{Game}}$	The advantage of $\mathcal{A}$ in Game
$\mathbf{Adv}^{\text{PRF}}$	The advantage of $\mathcal{A}$ in distinguishing PRF from random
$\mathbf{Adv}^{\text{SE}}$	The advantage of $\mathcal{A}$ in SE's IND-CPA
$\mathbf{Adv}^{\text{COM}}$	The advantage of $\mathcal{A}$ against the hiding property of COM

We describe how the challenger $\mathcal{C}$ responds to the answer of each query to provide it with the adversary $\mathcal{A}$ in the simulation game Game_{Sim}. The challenger $\mathcal{C}$ responds to each $\mathcal{A}$'s query as below :

- *Query*(KeyGenUser): $\mathcal{C}$ actions under the $\mathcal{Q}$(KeyGenUser) query, except that it makes the following modifications: $\mathcal{C}$ generates a key pair (upk, usk) from KeyGenUser(pp), supersedes $\mathsf{pk}_{\text{own}}, \mathsf{pk}_{\text{enc}}$ to a random string of the appropriate length, and then computes the user address addr ← $\text{CRH}(\mathsf{pk}_{\text{own}}, \mathsf{pk}_{\text{enc}})$.

$\mathcal{C}$ also puts these elements in a table and returns upk to $\mathcal{A}$. $\mathcal{C}$ does the above procedure for $\mathcal{Q}'$.

- *Query*(zkTransfer, note, upk^{send}, upk^{recv}, $\text{v}_{\text{out}}^{\text{priv}}$, $\text{v}_{\text{in}}^{\text{pub}}$, $\text{v}_{\text{out}}^{\text{pub}}$, EOA^{recv}): $\mathcal{C}$ actions under the $\mathcal{Q}$(zkTransfer) query, except that it makes the following modifications: by default, we assume that upk^{send} exists in the table. We abort the queries if upk^{send} does not exist in the table. $\mathcal{C}$ comes up with random strings and replaces nf and cm_{new} to these values, respectively. If upk^{recv} is a public key generated by a previous query to KeyGenUser, then $\mathcal{C}$ sets sct_{new} and pct_{new} to an arbitrary string. Otherwise, $\mathcal{C}$ computes these elements as in the zkTransfer algorithm. Also, $\mathcal{C}$ stores the changed elements in the table.

We now define each game to prove the ledger indistinguishability of Azeroth. Once again, $\mathbf{Adv}_{\mathcal{A}}^{\text{Game}_{\text{Sim}}}$ is 0 since $\mathcal{A}$ is computed independently of the bit b where b is chosen by $\mathcal{C}$ in the experiments.

- Game_1. We now define the Game_1, which equals $\text{Game}_{\text{Real}}$, except that $\mathcal{C}$ simulates the zk-SNARK proof. For zkTransfer, the zk-SNARK key is generated as $(\text{ek}, \text{vk}, \text{td}_{\text{ZKT}}) \leftarrow \Pi_{\text{snark}}.\text{Sim}(\mathcal{R}_{\text{ZKT}})$ instead of $\Pi_{\text{snark}}.\text{Setup}(\mathcal{R}_{\text{ZKT}})$ to procure the trapdoor td_{ZKT}. After obtaining the td_{ZKT}, $\mathcal{C}$ computes the proof π_{sim} without a proper witness. The view of the simulated proof π_{sim} is identical to that of the proof computed in $\text{Game}_{\text{Real}}$. In addition, when $\mathcal{A}$ asks for the KeyGenUser query, we replace the elements of public key upk as a random string. The simulated (usk, upk) distribution is also identical to that of the key pairs computed in $\text{Game}_{\text{Real}}$. In a nutshell, $\mathbf{Adv}^{\text{Game}_1} = 0$.
- Game_2. We define the Game_2 equal to Game_1 except that $\mathcal{C}$ uses a random string r of a suitable length to replace the ciphertext pct_{new}. If the address addr of upk^{send} does not exist in the table, then $\mathcal{C}$ aborts. By Lemma 5.2, $|\mathbf{Adv}^{\text{Game}_2} - \mathbf{Adv}^{\text{Game}_1}| \leq 2 \cdot \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PE}}$.
- Game_3. We define the Game_3 as Game_2 with one modification where $\mathcal{C}$ changes the ciphertext sct_{new} from correct to an acceptable random string r. Specifically, if the address addr of upk^{send} exists in the table, $\mathcal{C}$ replaces sct_{new} as r. Otherwise, $\mathcal{C}$ aborts. By Lemma 5.3, $|\mathbf{Adv}^{\text{Game}_3} - \mathbf{Adv}^{\text{Game}_2}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{SE}}$.
- Game_4. We define the Game_4 as the same as Game_3 except that $\mathcal{C}$ uses a random string to change the nullifier nf created by PRF. By Lemma 5.4, $|\mathbf{Adv}^{\text{Game}_4} - \mathbf{Adv}^{\text{Game}_3}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PRF}}$.
- Game_{Sim}. Game_{Sim} is identical to Game_4, except that $\mathcal{C}$ replaces commitments (e.g., cm_{old}, cm_{new}) computed by COM to an arbitrary string. By Lemma 5.5, $|\mathbf{Adv}^{\text{Game}_{\text{Sim}}} - \mathbf{Adv}^{\text{Game}_4}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{COM}}$.

By summing over all the above $\mathcal{A}$'s advantages in the games, $\mathcal{A}$'s advantage in the L-IND experiment can be computed as follows:

$$\mathbf{Adv}_{\Pi,\mathcal{A}}^{\text{L-IND}}(\lambda) \leq \text{q}_{\text{ZKT}} \cdot (2 \cdot \mathbf{Adv}^{\text{PE}} + \mathbf{Adv}^{\text{SE}} + \mathbf{Adv}^{\text{PRF}} + \mathbf{Adv}^{\text{COM}})$$

Since $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\text{L-IND}}(\lambda) = 2 \cdot \Pr[\text{Azeroth}_{\Pi,\mathcal{A}}^{\text{L-IND}}(\lambda) = 1] - 1$ and $\mathcal{A}$'s advantage in the L-IND experiment is negligible for λ, we can conclude that it provides ledger indistinguishability.

Lemma 5.2: Let $\mathbf{Adv}^{\Pi_{\text{PE}}}$ be $\mathcal{A}$'s advantage in Π_{PE}'s IND-CPA and IK-CPA experiments. If $\mathcal{A}$'s zkTransfer query occurs q_{ZKT}, then $|\mathbf{Adv}^{\text{Game}_2} - \mathbf{Adv}^{\text{Game}_1}| \leq 2 \cdot \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PE}}$.

Proof: We utilize a hybrid game Game_H as an intermediate between Game_1 and Game_2. First of all, to prove that $\mathbf{Adv}^{\text{Game}_H}$ is negligibly different from $\mathbf{Adv}^{\text{Game}_1}$, we define a security model of our encryption scheme PE. It performs with the interaction between the adversary $\mathcal{A}$ and the IND-CPA challenger. $\mathcal{A}$ queries the encryption for a random message, and then $\mathcal{C}$ returns the ciphertext of it. After querying, $\mathcal{A}$ sends two messages M_0, M_1 to the challenger $\mathcal{C}$. $\mathcal{C}$ chooses one of the two received messages and returns the ciphertext to the adversary $\mathcal{A}$. If the adversary $\mathcal{A}$ correctly answers which message is encrypted, $\mathcal{A}$ wins. We denote this experiment as $\mathcal{E}_{\text{real}}$. We define another experiment $\mathcal{E}_{\text{sim}}$, which simulates the real one with only the following modification: When encrypting a message, replace SE.Enc's output with a random string. $\mathcal{A}$ cannot distinguish the $\mathcal{E}_{\text{sim}}$ from $\mathcal{E}_{\text{real}}$ but a negligible probability, due to the security of SE. The probability of $\mathcal{A}$ distinguishes the ciphertexts in $\mathcal{E}_{\text{sim}}$ is 1/2; a ciphertext pct from $\mathcal{E}_{\text{sim}}$ is uniformly distributed in $\mathcal{A}$'s view. Overall, the advantage of $\mathcal{A}$ in distinguishing the ciphertexts is negligible, which means that PE is IND-CPA. Finally, the advantage of $\mathbf{Adv}^{\text{Game}_H}$ is equal to $\mathbf{Adv}^{\text{PE}}$, hence $|\mathbf{Adv}^{\text{Game}_H} - \mathbf{Adv}^{\text{Game}_1}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PE}}$.

Like the above, Game_2 is the same as Game_H except that it encrypts plaintext by setting the key to a new public key instead of the one obtained by querying KeyGenUser. After querying KeyGenUser, $\mathcal{A}$ queries the IK-CPA challenger to gain pk_0, whereas pk_1 is obtained from the KeyGenUser query. The IK-CPA challenger encrypts the same plaintext as pct* using pk_b where b is the bit selected by the IK-CPA challenger per zkTransfer query. The challenger sets pct in Tx_{ZKT} to pct* and appends it to L. $\mathcal{A}$ outputs a bit b by guessing b with respect to the IK-CPA experiment. If $b = 0$ then $\mathcal{A}$'s view is equal to Game_2, whereas if $b = 1$ then $\mathcal{A}$'s view is Game_H. If the maximum advantage for IK-CPA experiment is $\mathbf{Adv}^{\text{PE}}$, then we can say that $|\mathbf{Adv}^{\text{Game}_2} - \mathbf{Adv}^{\text{Game}_H}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PE}}$. As a result, the sum of $\mathcal{A}$'s two advantages is $|\mathbf{Adv}^{\text{Game}_2} - \mathbf{Adv}^{\text{Game}_1}| \leq 2 \cdot \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{PE}}$.

Lemma 5.3: Let $\mathbf{Adv}^{\Pi_{\text{SE}}}$ be $\mathcal{A}$'s advantage in Π_{SE}'s IND-CPA experiment. If $\mathcal{A}$'s zkTransfer query occurs q_{ZKT} times, then $|\mathbf{Adv}^{\text{Game}_3} - \mathbf{Adv}^{\text{Game}_2}| \leq \text{q}_{\text{ZKT}} \cdot \mathbf{Adv}^{\text{SE}}$.

Proof: To prove that $\mathbf{Adv}^{\text{Game}_3}$ is negligibly different from $\mathbf{Adv}^{\text{Game}_2}$, we define a security model of our encryption scheme SE. It performs with the interaction between the adversary $\mathcal{A}$ and the IND-CPA challenger. $\mathcal{A}$ queries the encryption for a random message, and then $\mathcal{C}$ returns the ciphertext of it. After querying, $\mathcal{A}$ sends two messages M_0, M_1 to the challenger $\mathcal{C}$. $\mathcal{C}$ chooses one of the two received messages and returns the ciphertext to the adversary $\mathcal{A}$. If the adversary $\mathcal{A}$ correctly answers which

message is encrypted, $\mathcal{A}$ wins. However, since SE is based on PRF, $\mathcal{A}$ cannot distinguish the ciphertexts with all but negligible. the advantage of $\mathbf{Adv}^{\mathsf{Game2}}$ is equal to $\mathbf{Adv}^{\mathsf{SE}}$. Hence, $|\mathbf{Adv}^{\mathsf{Game}_3} - \mathbf{Adv}^{\mathsf{Game}_2}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{SE}}$.

Lemma 5.4: Let $\mathbf{Adv}^{\mathsf{PRF}}$ be $\mathcal{A}$'s advantage in distinguishing PRF from a true random function. If $\mathcal{A}$ makes $\mathsf{q}_{\mathsf{ZKT}}$ queries, then $|\mathbf{Adv}^{\mathsf{Game}_4} - \mathbf{Adv}^{\mathsf{Game}_3}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

Proof: We now describe that the difference between Game_4 and Game_3 is negligibly different. In zkTransfer algorithm, nf is computed by $\mathsf{PRF}_{sk^{\mathsf{send}}_{\mathsf{own}}}(\mathsf{cm}_{\mathsf{old}})$. Thus, the advantage of Game_4 is only related to PRF's advantage. In other words, the advantage $\mathbf{Adv}^{\mathsf{PRF}}$ is negligible and $|\mathbf{Adv}^{\mathsf{Game}_4} - \mathbf{Adv}^{\mathsf{Game}_3}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

Lemma 5.5: Let $\mathbf{Adv}^{\mathsf{COM}}$ be $\mathcal{A}$'s advantage against the hiding property of COM. If $\mathcal{A}$ makes $\mathsf{q}_{\mathsf{ZKT}}$ queries, then $|\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}} - \mathbf{Adv}^{\mathsf{Game}_4}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{COM}}$

Proof: On the zkTransfer query, the challenger $\mathcal{C}$ substitutes the commitment $\mathsf{cm}_{\mathsf{new}}$ as a random string r of an acceptable length. The advantage of adversary $\mathcal{A}$ is at most like that of COM. Thus, since the commitment $\mathsf{cm}_{\mathsf{new}}$ exists only in the zkTransfer query, $\mathcal{C}$ performs one replication of each zkTransfer query. Hence, we conclude that $|\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}} - \mathbf{Adv}^{\mathsf{Game}_4}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{COM}}$.

2) TRANSACTION NON-MALLEABILITY

Suppose that $\mathcal{A}$ outputs a transaction Tx' as follows:

$$\mathsf{Tx}' = (\pi, \mathsf{nf}, \cdots)$$

Recall that $\mathcal{A}$ wins TR-NM experiment only if Tx' contains a nullifier already revealed and a valid proof. We show that such a transaction cannot be constructed with all but negligible probability under the properties of zk-SNARK. For formal proof, let $\epsilon := \mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$, and utilize zk-SNARK witness extractor denoted as $\mathcal{E}$ for $\mathcal{A}$. We can build an algorithm $\mathcal{B}$ finding collision for PRF with an advantage negligibly close to ϵ, which suffices the proof. Algorithm $\mathcal{B}$ should work as follows:

i) Run $\mathcal{A}$ (simulating its interaction with the challenger $\mathcal{C}$ and obtain Tx').
ii) Run $\mathcal{E}$ to extract a witness $\vec{w}$ for a zk-SNARK proof π for Tx'.
iii) Get $\mathsf{apk}, \mathsf{sct}_{\mathsf{old}}$ from L and parse Tx' to construct a statement $\vec{x}$ for π.
iv) Check whether $\vec{w}$ is a valid witness for $\vec{x}$ or not. If it fails, it aborts and then outputs 0.
v) Parse $\vec{w}$ then get $\mathsf{sk}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{old}}$.
vi) Find a transaction $\mathsf{Tx} \in \mathsf{L}$ that contains nf.
vii) If Tx is found, let $(\mathsf{sk}'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}})$ be the corresponding witness to Tx attained from $\mathcal{E}$. If $\mathsf{sk}_{\mathsf{own}} \neq \mathsf{sk}'_{\mathsf{own}}$, then output $((\mathsf{sk}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{old}}), (\mathsf{sk}'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}}))$. Otherwise, output 0.

Seeing that the proof π for a transaction Tx is valid, with all but negligible probability, the extracted witness $\vec{w}$ is valid. Moreover, $Pr[\mathsf{sk}_{\mathsf{own}} = \mathsf{sk}'_{\mathsf{own}}] = \frac{1}{2^l}$ where l is the bit length of $\mathsf{sk}_{\mathsf{own}}$. Thus, its probability is negl. Putting probabilities together, we conclude that $\mathcal{B}$ finds a collision for PRF with probability $\epsilon - \mathsf{negl}(\lambda)$.

3) BALANCE

This part shows that $\mathbf{Adv}^{\mathsf{BAL}}$ is at most negligible. For each zkTransfer transaction on the ledger L, the challenger $\mathcal{C}$ computes a witness $\vec{w}$ for the zk-SNARK instance $\vec{x}$ corresponding to the transaction $\mathsf{Tx}_{\mathsf{ZKT}}$ in the BAL experiment. It does not affect $\mathcal{A}$'s view. For such a way, $\mathcal{C}$ obtains an augmented ledger $(\mathsf{L}, \vec{W})$ in which $\vec{w}_i$ means a witness for the zk-SNARK instance $\vec{x}_i$ of i-th zkTransfer transaction in L. Note that we can parse an augmented ledger as a list of matched pairs $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}_i)$ where $\mathsf{Tx}_{\mathsf{ZKT}}$ is a zkTransfer transaction and $\vec{w}_i$ is its corresponding witness.

a: BALANCED LEDGER

We say that an augmented ledger L is *balanced* if the following conditions hold as defined in [5] and [19].

- **Condition 1**: In each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$, the opening of unique commitment $\mathsf{cm}_{\mathsf{new}}$ exists, and the commitment $\mathsf{cm}_{\mathsf{new}}$ is also a result of previous $\mathsf{Tx}_{\mathsf{ZKT}}$.
- **Condition 2**: The two different openings in $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$ and $(\mathsf{Tx}_{\mathsf{ZKT}}{}^*, \vec{w}^*)$ are not openings of a single commitment.
- **Condition 3**: Each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$ contains openings of $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{cm}_{\mathsf{new}}$, and values, satisfying that $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} = \mathsf{v}^{\mathsf{ENA}*}$ where we denote an updating of the value as $*$.
- **Condition 4**: The values used to compute $\mathsf{cm}_{\mathsf{old}}$ are the same as the value for $\mathsf{cm}^*_{\mathsf{new}}$, if $\mathsf{cm}_{\mathsf{old}} = \mathsf{cm}^*_{\mathsf{new}}$ where $\mathsf{cm}_{\mathsf{old}}$ is the commitment employed in $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$, and $\mathsf{cm}^*_{\mathsf{new}}$ is the output of a previous transaction before $\mathsf{Tx}_{\mathsf{ZKT}}$.
- **Condition 5**: If $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$ was inserted by $\mathcal{A}$, and $\mathsf{cm}_{\mathsf{new}}$ contained in $\mathsf{Tx}_{\mathsf{ZKT}}$ is the result of an earlier zkTransfer transaction Tx′, then the recipient's account address $\mathsf{addr}^{\mathsf{recv}}$ does not exist in Acct.

We say that $(\mathsf{L}, \vec{w})$ is balanced, if the following equation holds:

$$\mathsf{v}^{\mathsf{ENA}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} = \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}$$

For each of the above conditions, we use a contraction to prove that the probability of each case is, at most, negligible. Note that, for better legibility, we denote the $\mathcal{A}$'s win probability of each case as $\Pr[\mathcal{A}(\mathcal{C}_i) = 1]$, which means $\mathcal{A}$ wins but violates Condition i.

An infringement on condition 1. Each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$, not inserted by $\mathcal{A}$, always satisfies condition 1; The probability $\Pr[\mathcal{A}(\mathcal{C}_1) = 1]$ is that $\mathcal{A}$ inserts $\mathsf{Tx}_{\mathsf{ZKT}}$ to build a pair $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w})$ where $\mathsf{cm}_{\mathsf{old}}$ in $\vec{w}$ is not the output of all previous transactions before receiving the value by zkTransfer. However, each $\mathsf{Tx}_{\mathsf{ZKT}}$ utilizes the witness $\vec{w}$, containing the commitment $\mathsf{cm}_{\mathsf{old}}$ taken as input for making a nullifier nf, to generate the proof by proving the validity of $\mathsf{Tx}_{\mathsf{ZKT}}$. Namely, there is a violation of condition 1 if its commitment

corresponding to nf does not exist in L. The violation's meaning is equal to breaking the binding property of COM; Hence $\Pr[\mathcal{A}(\mathcal{C}_1) = 1]$ is negligible.

An infringement on condition 2. Each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$, not inserted by $\mathcal{A}$, always satisfies condition 2; The probability $\Pr[\mathcal{A}(\mathcal{C}_2) = 1]$ is that there are two transactions $(\mathsf{Tx}_{\mathsf{ZKT}}, \mathsf{Tx}_{\mathsf{ZKT}}{}')$ in which their commitment is the same but has different two nullifiers nf and nf'. However, it contradicts the binding property of COM; Thus, $\Pr[\mathcal{A}(\mathcal{C}_2) = 1]$ is negligible.

An infringement on condition 3. In each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$, there exists a zk-SNARK proof, which can guarantee each of values $\mathsf{v}^{\mathsf{ENA}}$, $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}$, $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$, and $\mathsf{v}^{\mathsf{ENA}*}$, satisfying the following equation: $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} = \mathsf{v}^{\mathsf{ENA}*}$. $\Pr[\mathcal{A}(\mathcal{C}_3) = 1]$ is a probability that its equation does not hold. However, this violates the proof knowledge property of the zk-SNARK; It is negligible.

An infringement on condition 4. Each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$ encompasses the values taken as the commitment (e.g., $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{addr}^{\mathsf{recv}}$, and $\mathsf{o}_{\mathsf{new}}$). $\Pr[\mathcal{A}(\mathcal{C}_4) = 1]$ is a probability that the commitments are equal. All values related to commitment inputs in two transactions $(\mathsf{Tx}_{\mathsf{ZKT}}, \mathsf{Tx}_{\mathsf{ZKT}}{}^*)$ are equivalent except for the amount (i.e., $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} \neq \mathsf{v}^{\mathsf{priv}*}_{\mathsf{out}}$) where $\mathsf{Tx}_{\mathsf{ZKT}}{}^*$ a pre-existing $\mathsf{zkTransfer}$ transaction. However, since it contradicts the binding property of COM, it happens negligibly.

An infringement on condition 5. Each $(\mathsf{Tx}_{\mathsf{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$ publishes the recipient's address of a commitment $\mathsf{cm}_{\mathsf{new}}$. If the $\mathsf{zkTransfer}$ transaction inserted by $\mathcal{A}$ issues $\mathsf{addr}^{\mathsf{recv}}$, the output of a previous $\mathsf{zkTransfer}$ transaction $\mathsf{Tx}_{\mathsf{ZKT}}{}'$ whose recipient's account address is in Acct, it is the violation of the condition 5; Thus, $\Pr[\mathcal{A}(\mathcal{C}_5) = 1]$. However, this contradicts the collision resistance of CRH.

To sum up, we prove the Definition 5.4 holds since it is at most negligible that the opposite happens, as mentioned above.

4) AUDITABILITY

In the AUD experiment, $\mathcal{A}$ wins if the tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, aux)$ holds the following conditions where aux consists of $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}})$:

i) $\mathsf{Tx}_{\mathsf{ZKT}}$ passes the transaction verification.

$$\mathsf{VerifyTx}(\mathsf{Tx}_{\mathsf{ZKT}}, \mathsf{L}) = \mathsf{true}$$

ii) aux and the commitment $\mathsf{cm}_{\mathsf{new}}$ in $\mathsf{Tx}_{\mathsf{ZKT}}$ are verified.

$$\mathsf{VerCommit}(\mathsf{cm}_{\mathsf{new}}, aux) = \mathsf{true}$$

iii) The decrypted message of $\mathsf{pct}_{\mathsf{new}}$ and the values $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}})$ in aux are not the same.

$$(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}}) \neq \mathsf{Audit}_{\mathsf{ask}}(\mathsf{pct}_{\mathsf{new}})$$

If $\mathcal{A}$ wins in the experiment, when the auditor decrypts $\mathsf{pct}_{\mathsf{new}}$, it implies that the auditor obtains an arbitrary string, not a correct plaintext. However, $\mathcal{A}$'s winning probability is negligible since it breaks the *binding* property of COM. Also, assume that an extractor χ can extract the witness. When obtaining the witness using χ, it is obvious that aux is equal to $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}})$. Thus, $\mathcal{A}$'s winning should also break the *knowledge soundness* property of the zk-SNARK. Consequently, since the properties of COM and zk-SNARK, the auditor with an authorized key (i.e., ask) can always observe the correct plaintext and surveil illegal acts in transactions.

VI. IMPLEMENTATION AND EXPERIMENT

A. IMPLEMENTATION

Our $\mathsf{Azeroth}$ implementation coded in Python, C++, and Solidity languages consists of two parts; the client and the smart contract. The client interacts with the blockchain network using Web3.py.[2] To generate a zk-SNARK proof in $\mathsf{Azeroth}$, we use libsnark[3] with Groth16 [17] and BN254 curve.

1) PRF, COM, AND CRH

We instantiate the pseudorandom function $\mathsf{PRF}_k(x)$ where k is the seed, and x is the input, using a collision-resistant hash function as follows.

$$\mathsf{PRF}_k(x) := \mathsf{CRH}(k||x)$$

A commitment is also realized using a hash function CRH as follows.

$$\mathsf{COM}(\mathsf{v}, \mathsf{addr}; \mathsf{o}) := \mathsf{CRH}(\mathsf{v}||\mathsf{addr}||\mathsf{o})$$

A collision-resistant hash function CRH is implemented using zk-SNARK friendly hash algorithms such as $\mathsf{MiMC7}$ [1] and $\mathsf{Poseidon}$ [16] as well as a standard hash function $\mathsf{SHA256}$ [25].

2) SYMMETRIC-KEY ENCRYPTION

The symmetric-key encryption needs to be efficient in the proof generation for encryption. Therefore, we implement an efficient stream cipher based on PRF using zk-SNARK friendly hash algorithms such as $\mathsf{MiMC7}$ and $\mathsf{Poseidon}$ as follows.

$$\underline{\mathsf{SE.Enc}_k(\mathsf{msg}) \rightarrow (r, \mathsf{sct})}$$
$$r \xleftarrow{\$} \mathbb{F};\ \mathsf{sct} \leftarrow \mathsf{msg} + \mathsf{PRF}_k(r)$$
$$\textbf{return}\ (r, \mathsf{sct})$$
$$\underline{\mathsf{SE.Dec}_k(r, \mathsf{sct}) \rightarrow \mathsf{msg}}$$
$$\mathsf{msg} \leftarrow \mathsf{sct} - \mathsf{PRF}_k(r)$$
$$\textbf{return}\ \mathsf{msg}$$

We also employ the CTR mode if the message size is longer than the range of PRF.

3) PUBLIC-KEY ENCRYPTION

To enable two recipients, a receiver and an auditor, to use public-key encryption, we extend the ElGamal encryption

[2] https://github.com/ethereum/web3.py
[3] https://github.com/scipr-lab/libsnark

system to allow for the re-use of randomness, as described in the treatment of multi-recipient encryption in [4]. We also use standard hybrid encryption to improve performance, where a random key is encrypted using public key encryption. This key is then used to encrypt the message using a symmetric-key encryption scheme. Given two key pairs (pk_1, sk_1) and (pk_2, sk_2) for ElGamal encryption, as well as the symmetric-key encryption scheme SE, the resulting implementation of the public key encryption is as follows:

$\mathsf{PE.Enc}_{pk_1,pk_2}(\mathsf{msg}) \rightarrow \mathsf{pct}, \mathsf{aux}$

$k \xleftarrow{\$} \mathbb{G}; r \xleftarrow{\$} \mathbb{F}$

$c_0 \leftarrow G^r; c_1 \leftarrow k \cdot pk_1^r; c_2 \leftarrow k \cdot pk_2^r; c_3 \leftarrow \mathsf{SE.Enc}_k(\mathsf{msg})$

$\mathsf{pct} \leftarrow (c_0, c_1, c_2, c_3); \mathsf{aux} \leftarrow (k, r)$

return pct, aux

$\mathsf{PE.Dec}_{sk_i}(\mathsf{pct}) \rightarrow \mathsf{msg}$

$(c_0, c_1, c_2, c_3) \leftarrow \mathsf{pct}$

$k \leftarrow c_i / c_0^{sk_i}; \mathsf{msg} \leftarrow \mathsf{SE.Dec}_k(c_3)$

return msg

B. EXPERIMENT

In our experiment, the term $\mathsf{cfg}_{\mathsf{Hash,Depth}}$ denotes a configuration of Merkle hash tree depth and hash type in Azeroth. For instance, $\mathsf{cfg}_{\mathsf{MiMC7},32}$ means that we run Azeroth with MiMC7 [1] and its Merkle tree depth is 32. Table 3(a) illustrates our system environments. We execute all experiments on the machine Server described in Table 3(a) for the overall performance evaluation as a default machine. The default blockchain is the Ethereum testnet. The used proof system is Gro16 [17].

1) OVERALL PERFORMANCE

We show that the performance and gas consumption of Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7},32}$ are presented in Table 3 (b).[4] The execution time of Setup is 4.04s, comprising the zk-SNARK key generation time of 2.2s and the deployment time of Azeroth's smart contract to the blockchain of 1.84s. Setup consumes a significant amount of gas due to the initialization of the Merkle tree. In zkTransfer, the total execution time is 4.38s, including both the Client and Smart Contract parts. The gas is primarily used to verify the SNARK proof and update the Merkle hash tree. Further analysis of zkTransfer by varying the hash function is described in the following experiments.

2) zk-SNARK PERFORMANCE

We evaluate the performance of zk-SNARK used to execute zkTransfer on various systems (Server, $\mathsf{System}_1, \cdots, \mathsf{System}_4$) as described in Table 3 (b). Table 3 (c) shows the setup time, the proving time, and the verification time, respectively, on each system with $\mathsf{cfg}_{\mathsf{MiMC7},32}$. Although System_3 has the lowest performance, still its proving time of 4.56s is practically acceptable.

3) HASH TYPE AND TREE DEPTH

We evaluate Azeroth performance depending on hash tree depths and hash types as shown in Figure 6 and Figure 7.

Figure 6 illustrates the execution time of zk-SNARK for MiMC7 [1], Poseidon [16],[5] and SHA256 [25] where the hash tree depth is 32. The SNARK key generation times are 2.311s, 2.182s, and 53.393s, respectively. The proving times for MiMC7 and Poseidon are 0.901s and 0.582s respectively, while it takes 20.69 seconds with SHA256; SHA256 is about 20× and 40× slower than MiMC7 and Poseidon in zk-SNARK. The verification time is almost independent of the hash type. However, when each hash function is executed natively, SHA256 shows the best performance as shown in Figure 6 (d), which is similar to the gas consumption trend shown in Figure 6 (e).

Figure 7 shows the key size. The proving key (ek) size is proportional to the tree depth, whereas the verification key size remains 1KB.[6] Poseidon has the smallest sizes of ek and constraints. Specifically, in depth 32, the ek sizes in Poseidon, MiMC7, and SHA256 are 3,341KB, 4,339KB, and 255,000KB respectively. The circuit size of SHA256 is enormous due to numerous bit operations, and Poseidon has 30% smaller size than MiMC7's. Figure 7 (c) shows that MiMC7 and Poseidon hashes consume relatively more gas than SHA256 since not only is SHA256 natively supported in Ethereum [8], but also it shows better native performance as shown in Figure 6 (e).

4) THE NUMBER OF CONSTRAINTS

We measure the number of constraints for each algorithm component as shown in Table 4. The membership proof algorithm performs the hash execution as many as the tree depth. SE.Dec conducts the one hash computation, and PE.Enc executes the group operation (i.e., exponentiation) three times internally. Note that in the table, the number of constraints for CRH is obtained when a single input block is provided to the hash, and the number of constraints is proportional to the number of input blocks.

5) PERFORMANCE ANALYSIS OF SMART CONTRACT

We analyze the execution time and the gas consumption in smart contract $\mathsf{zkTransfer}_{\mathsf{SC}}$. Table 5 shows the gas consumption for each function in smart contract $\mathsf{zkTransfer}_{\mathsf{SC}}$. In particular, the Π_{snark}.Verify and TreeUpdate functions are the most time-consuming in the smart contract, with the gas consumption of the latter depending on the hash tree depth and hash type.

Figure 8 represents the execution time of Π_{snark}.Verify and TreeUpdate. The verification time is 11.9ms, and the

[4]The results include the execution time up to the point of receiving the transaction receipt.

[5]We utilize a well-optimized Poseidon smart contract from circomlib(https://github.com/iden3/circomlib/tree/feature/extend-poseidon)

[6]We omit the graph of vk, since it is constant.

TABLE 3. Benchmark of Azeroth.

(a) System specification

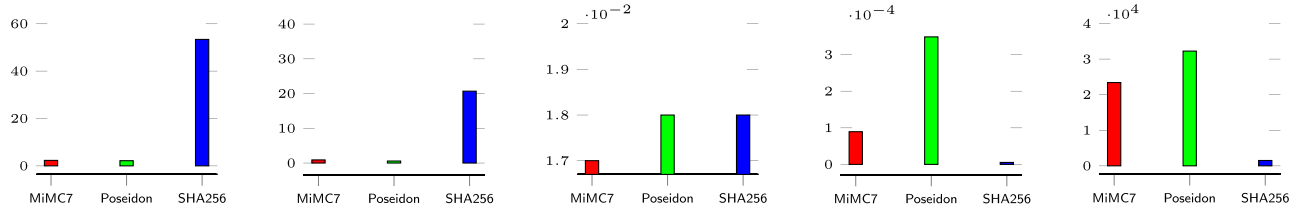
Machine	OS	CPU	RAM
Server	Ubuntu 20.04	Intel(R) Xeon Gold 6264R@3.10GHz	256GB
$System_1$	macOS 11.2	M1@3.2GHz	8GB
$System_2$	macOS 11.6	Intel(R) i7-8850H CPU @ 2.60GHz	32GB
$System_3$	android 11	Exynos9820	8GB
$System_4$	iOS 15.1	A12 Bionic	4GB

(b) Execution time and gas consumption of Azeroth with $cfg_{MiMC7,32}$

	Azeroth				
	Setup	RegisterAuditor	RegisterUser	zkTransfer	Audit
Time (s)	4.04	0.02	0.017	4.38	0.03
Gas	5,790,800	63,179	45,543	1,555,957	N/A

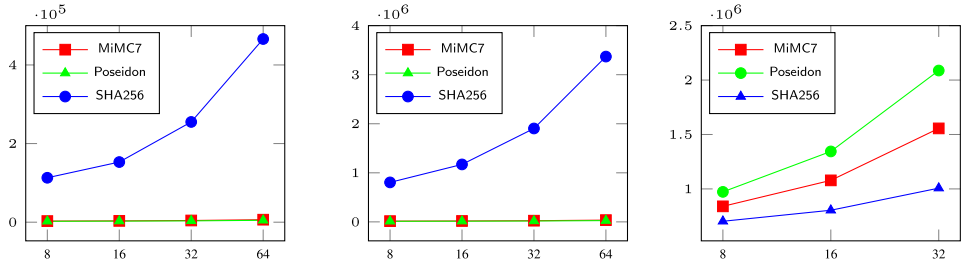
(c) Execution time of zk-SNARK in zkTransfer

	Server	$System_1$	$System_2$	$System_3$	$System_4$
Π_{snark}.Setup (s)	2.311	4.19	4.13	8.529	5.529
Π_{snark}.Prove (s)	0.901	2.581	2.77	4.557	3.15
Π_{snark}.Verify (s)	0.017	0.041	0.079	0.062	0.054



(a) Π_{snark}.Setup (b) Π_{snark}.Prove (c) Π_{snark}.Verify (d) Native (e) Gas consumption

FIGURE 6. Performance with 32 hash tree depth. (a)-(c): The execution time of zk-SNARK's algorithms where the y axis is time(s). (d): The native execution time of each hash algorithm written in C++ where the *y* axis is time(s). (e): The gas consumption where the *y* axis denotes the gas consumption.



(a) The proving key size(KB) (b) The number of constraints (c) Gas consumption

FIGURE 7. Key size, constraints in circuits and gas consumption by varying hash tree depth and hash type.

TABLE 4. The number of constraints for each algorithm component.

	$Membership_{32}$	CRH(1)	SE.Dec	PE.Enc	GroupOp
MiMC7	11,713	364	364	7,211	2,035
Poseidon	7,745	213	240	6,758	2,035
SHA256	1,465,473	25,725	45,794	83,294	2,035

TABLE 5. Gas cost for each function.

$\mathcal{F}$	Π_{snark}.Verify	MiMC7	SHA256	Poseidon	Etc
gas	402,922	23,405	1,506	32,252	42,143

execution time of TreeUpdate is 12.8*ms*, 17.4*ms*, and 7.3*ms* on MiMC7, Poseidon, and SHA256 respectively.

6) COMPARISON WITH THE OTHER EXISTING SCHEMES

We compare the proposed scheme Azeroth with other privacy-preserving transfer schemes such as Zeth [27], Blockmaze [19], Zether [7], and PGC [11] in Table 6. Our proposal performs better than the existing schemes, even if Azeroth provides an additional function of auditability. Zeth and Blockmaze are implemented with $cfg_{MiMC7,32}$ and $cfg_{SHA256,8}$ respectively, and the same configuration is applied to the proposed scheme for a fair comparison.

The experiment is conducted on Server. Note that the proof generation time in the table excludes the circuit loading time for a fair comparison, and the loading time is

TABLE 6. Comparison between our proposed scheme and existing work.

(a) Comparison between Azeroth and Zeth with $\text{cfg}_{\text{MiMC7,32}}$

Azeroth	Setup		zkTransfer	
	time	pp	time	tx_{size}
	2.846s	4.32MB	0.983s	1,186B
Zeth	10.646s	94.24MB	10.47s	1,380B
	time	pp	time	tx_{size}
	Setup		Mix	

(b) Comparison between Azeroth and BlockMaze with $\text{cfg}_{\text{SHA256,8}}$

Azeroth	Setup		zkTransfer							
	time	pp	time				tx_{size}			
	26.765s	113MB	10.0166s				1,186B			
BlockMaze	125.063s	323MB	6.689s	817B	6.948s	815B	9.224s	899B	18.609s	815B
	time	pp	time	tx_{size}	time	tx_{size}	time	tx_{size}	time	tx_{size}
	Setup		Mint		Redeem		Send		Deposit	

(c) Gas cost and transaction size between Azeroth, Zether and PGC

	$\text{Azeroth}_{\text{cfg}_{\text{MiMC7,32}}}$	Zether [7]	PGC [11]
gas cost	1,555,957	7,188,000	8,282,000
transaction size (bytes)	1,186	1,472	1,310

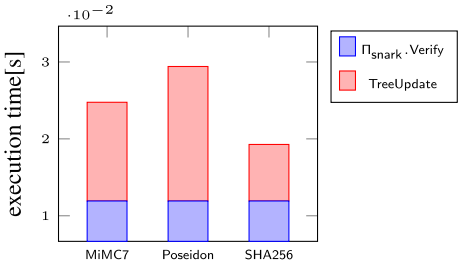


FIGURE 8. The execution time of functions in $\text{zkTransfer}_{\text{SC}}$.

significantly longer in Blockmaze. On the other hand, PGC [11] and Zether [7] use standard ElGamal encryption and NIZK to provide confidentiality instead of utilizing Merkle Tree. The performance results in these works exclude anonymity, meaning the anonymity set size is 2. Note that the performance degrades as the anonymity set size increases in PGC and Zether since the number of Elliptic curve operations in the smart contract increases proportionally to the anonymity set size.

In comparison with Zeth,[7] we utilize ganache-cli[8] as our test network. Due to the circuit optimization of Azeroth, the resulting circuit size is 4× smaller, and the size of pp is 22× smaller than Zeth. In zkTransfer, Azeroth reduces the execution time by 90% compared with Zeth's Mix function.

While BlockMaze[9] has four transactions of *Mint*, *Redeem*, *Send*, *Deposit*, Azeroth provides the equivalent functionality using a single transaction zkTransfer. It takes 20*s* to load the proving key in Blockmaze while it is only 1*s* in Azeroth. Hence Azeroth provides much better performance than Blockmaze in practice.

[7]https://github.com/clearmatics/zeth
[8]https://github.com/trufflesuite/ganache
[9]https://github.com/Agzs/BlockMaze

Zether [7] and PGC [11] are stateful[10] schemes using ElGamal encryption and NIZK. Due to the large difference in structure, the comparison experiment compares the gas cost and transaction size generated per transfer. Zether and PGC require 4.6 times and 5.3 times more gas than Azeroth, respectively, due to the Elliptic curve operations in the smart contract. In terms of transaction size, Azeroth generates a smaller transaction than Zether and PGC, although Azeroth provides higher anonymity than them.

VII. CONCLUSION

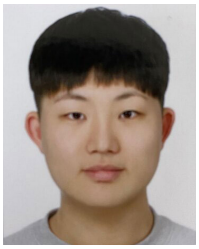
In this paper, we propose an auditable privacy-preserving digital asset-transferring system called Azeroth, which hides the receiver, and the amount value to be transferred. At the same time, a zero-knowledge proof guarantees the transferring correctness. In addition, the proposed Azeroth supports an auditing functionality in which an authorized auditor can trace transactions to comply with an anti-money laundry regulations. Its security is proven formally and implemented onto various platforms, including the Ethereum testnet blockchain. The experimental results show that the proposed Azeroth is efficient enough to be practically deployed.

REFERENCES

[1] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MIMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *Proc. ASIACRYPT*, 2016, pp. 191–219.
[2] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution," Tech. Rep., 2019.
[3] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, Oct. 2020, pp. 255–267.

[10]The meaning is that the account is renewed immediately through one transaction.

[4] M. Bellare, A. Boldyreva, K. Kurosawa, and J. Staddon, "Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security," *IEEE Trans. Inf. Theory*, vol. 53, no. 11, pp. 3927–3943, Nov. 2007.
[5] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 459–474.
[6] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: Enabling decentralized private computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 947–964.
[7] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2020, pp. 423–443.
[8] V. Buterin, "Ethereum white paper: A next generation smart contract-decentralized application platform," Tech. Rep., 2013.
[9] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 315–334.
[10] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via PVORM," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 701–717.
[11] Y. C. X. M. C. Tang and M. H. Au, "PGC: Decentralized confidential payment system with auditability," in *Computer Security—ESORICS 2020*, 2020, pp. 591–610.
[12] B. E. Diamond, "Many-out-of-many proofs and applications to anonymous Zether," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1800–1817.
[13] E. Duffield and D. Diaz. (2015). *Dash: A Privacycentric Cryptocurrency*. [Online]. Available: https://github.com/dashpay/dash/wiki/Whitepaper
[14] *Virtual Assets and Virtual Asset Service Providers*, FATF, Paris, France, 2021.
[15] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2019, pp. 649–678.
[16] D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," in *Proc. 30th USENIX Secur. Symp.*, Aug. 2021, pp. 1–16.
[17] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proc. 35th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2016, pp. 305–326.
[18] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs," in *Proc. 37th Annu. Int. Cryptol. Conf.*, 2017, pp. 581–612.
[19] Z. Guan, Z. Wan, Y. Yang, Y. Zhou, and B. Huang, "BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1446–1463, May 2022.
[20] H. Kang, T. Dai, N. Jean-Louis, S. Tao, and X. Gu, "FabZK: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 543–555.
[21] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 839–858.
[22] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," Tech. Rep., 2013.
[23] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 2, pp. 105–121, Apr. 2018.
[24] N. N. W. Vasquez and M. Virza, "zkLedger: Privacy-preserving auditing for distributed ledgers," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement.*, Apr. 2018, pp. 65–80.
[25] *Secure Hash Standard*, Standard Fips180-2, National Institute of Standards and Technology (NIST), 2002.
[26] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 238–252.
[27] A. Rondelet and M. Zajac, "ZETH: On integrating zerocash on Ethereum," 2019, *arXiv:1904.00905*.
[28] N. N. Saberhagen. (2013). *Cryptonote V2.0*. [Online]. Available: https://cryptonote.org/whitepaper.pdf



GWEONHO JEONG received the B.S. degree in information systems engineering from Hanyang University, Seoul, South Korea, where he is currently pursuing the Ph.D. degree in information systems engineering. His current research interests include applied cryptography, including protocols for applied cryptography, verifiable computation, and zero-knowledge proofs.

NURI LEE received the B.S. and M.S. degrees in electronics engineering from Kookmin University, Seoul, South Korea. His research interests include applied cryptography, zero-knowledge proofs, and their applications to blockchain technology.

JIHYE KIM (Member, IEEE) received the B.S. and M.S. degrees from the School of Computer Science and Engineering, Seoul National University, South Korea, in 1999 and 2003, respectively, and the Ph.D. degree in computer science from the University of California at Irvine, Irvine, in 2008. She is currently a Professor with the Department of Electrical Engineering, Kookmin University. Her research interests include applied cryptography, verifiable computation, and zero-knowledge proofs.

HYUNOK OH (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1996, 1998, and 2003, respectively. He is currently a Full Professor with the Department of Information Systems, Hanyang University, Seoul. His research interests include applied cryptography, zero-knowledge proofs, verifiable computation, non-volatile memory, and embedded systems.