Azeroth: Auditable Zero-knowledge Transactions in Smart Contracts

***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: encrypted information alone cannot verify the validity of a transaction and makes it difficult to meet anti-money laundering regulations, i.e. 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, 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 an 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.**

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 [3], [25], [26], ring signatures [31], homomorphic encryption [8], zero-knowledge proofs [6], [8], [21], [30], etc.

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 [6], which operates on the UTXO model, secures transactions, while leveraging zero-knowledge proofs [19] to ensure transactions in a valid set of UTXOs. Zether [8] based on the account model encrypts accounts with homomorphic encryption and provides zero-knowledge proofs [10] to ensure valid modification of encrypted accounts. Zether builds up partial privacy for unlinkability between sender and receiver addresses, similar to Monero [31].

As asset transactions on the blockchain increase, the demand for adequate auditing capabilities is also increasing. 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 be also 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 [31], Dash [15], and Zerocash [6]. In fact, many global cryptocurrency exchanges are also strengthening their distance from the dark coins as recommended by Financial Action Task Force (FATF) [16]. 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) [19], [20], [29] 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 [6], 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.

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). Confidentiality can be provided by utilizing encryption, while anonymity can be provided by utilizing an accumulator. Specifically, we employ dual accounts that constitute an encrypted account in addition to a plain account, similar

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

Network and Distributed System Security (NDSS) Symposium 2023
28 February - 4 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23xxx
www.ndss-symposium.org

to Blockmaze [21]. 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 an Merkle tree). The owner of the encrypted value may receive it from the accumulator to the user's accounts.

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 recipients encryption so that an authorizer as well as the receiver can decrypt the transaction. The auditor's ability can be distributed through a threshold scheme. In order to have this strong anonymity with auditablity, the most challenging part is how to adopt and implement zk-SNARKs for this complex relation proof.

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 the additional zero-knowledge feature to meet the standard privacy requirements. 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 the real-world use, we adopt a SNARK-friendly hash algorithm to instantiate encryption to have an efficient proving time and execute experiments on various platforms.

The contributions of this paper are summarized as follows.

- Framework: We design a privacy-preserving framework Azeroth on an account-based blockchain model, while including encryption verifiability, and auditability. Moreover, since Azeroth constructed as a 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 unlinkability, 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 the existing account-based blockchain models, such as Ethereum [9]. 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 shows better performance results than the other existing 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 give an explanation of data structures utilized in Azeroth. Afterward, we elucidate the overview and construction, algorithms, construction, and security definitions in section V. Section VI shows the implementation and the experimental results. Finally, we make a conclusion in section VII.

II. Related Work

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

A mixing service(or tumbler) such as CoinJoin [25], Möbius [26], and Tornado Cash [3] offers a service for mixing identifiable cryptocurrency transfer with others, so as to obscure the trail back to the transfer's original source. Thus, the mixer supports personal privacy protection for transactions on the blockchain. However, since the mixers take heed of anonymity, it exists the possibility of a malevolent problem.

Zerocash [6] 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 [30] 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 Zerocash. ZEXE [7], extending Zerocash with scripting capabilities, supports function privacy that nobody knows which computation is executed in off-line. Hawk [24] 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 [21] proposes a dual balance model for account-model blockchains, consisting of a public balance and private balance. For hiding 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 [17] 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 an anonymity set before the transaction is broadcasted. Moreover, this system is a standalone cryptocurrency not supporting deployment on any smart contract platform.

Zether [8] accomplishes the privacy protection in the account-based model using ZKPs (Bulletproofs [10]) 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 limitation to support a high level of anonymity. Diamond [14] 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 [11] is a privacy-preserving protocol for asset transfer in which banks play a role as an arbitrator of mediation. Solidus utilizes ORAM to support update accounts without revealing the values, but it cannot provide a dedicated audit function. Specifically, it can only offer auditing by revealing whole keys to an auditor, and opening transactions.

zkLedger [27] and FabZK [23] 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 the performance issue.

PGC [12] aims for a middle ground between privacy and auditability and then proposes auditable decentralized confidential payment using an additively homomorphic public-key encryption. However, the anonymity set size should be small in the approach. The work [4] 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 that checks if the transaction is valid.

III. Preliminaries

In this section, we describe notations for standard cryptographic primitives. Let λ be the security parameter.

Collision-resistant hash. A hash function $\mathsf{CRH} : \{0,1\}^* \rightarrow \{0,1\}^{poly(\lambda)}$ is a collision-resistant hash function if it is difficult to find two different inputs of which their output is equivalent. That is, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl, $\Pr\left[(x_0, x_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{CRH}) : x_0 \neq x_1 \wedge \mathsf{CRH}(x_0) = \mathsf{CRH}(x_1)\right] \leq \mathsf{negl}(\lambda)$.

Commitment. A commitment scheme COM provides the computational binding property and the statistical hiding property. It also provides the ability to reveal the committed value later. We denote a commitment cm for a value u as $\mathsf{cm} \leftarrow \mathsf{COM}(u; \mathsf{o})$ where o denotes the commitment opening key. Given the values u and o, a commitment cm to a value u can be disclosed; one can check if $\mathsf{cm} \leftarrow \mathsf{COM}(u; \mathsf{o})$.

Pseudorandom function. A pseudorandom function $\mathsf{PRF}_k(x) : \{0,1\}^* \rightarrow \{0,1\}^{poly(\lambda)}$ is a function of a random seed key k and an input x that is indistinguishable from a uniform random function.

Zero-Knowledge Succinct Non-interactive Arguments of Knowledge. As described in [19], [20], given a relation $\mathcal{R}$, a 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.

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.

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

Knowledge soundness. Knowledge soundness says that if the honest prover outputs a proof π, the prover must know a witness and 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 $\Pr\left[\mathsf{Game}^{\mathsf{KS}}_{\mathcal{RG},\mathcal{A},\mathcal{E}} = \mathsf{true}\right] = \mathsf{negl}(\lambda)$, a 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 $\mathsf{res} \leftarrow (\mathsf{VerProof}(\mathsf{vk}, x, \pi) \wedge (x, \pi) \notin \mathcal{R})$

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. Formally, if there exists a simulator such that the following conditions hold for any adversary $\mathcal{A}$, we say that Π_{snark} is zero-knowledge.

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

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

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

Symmetric-key encryption. We use a symmetric-key encryption scheme SE is 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 takes a key k and a plaintext msg as inputs and returns a ciphertext sct.
- $\mathsf{Dec}_\mathsf{k}(\mathsf{sct}) \rightarrow \mathsf{msg}$: The Dec algorithm takes a key k and a ciphertext sct as inputs. 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.

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.

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

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

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

IV. DATA STRUCTURES

This section describes the data structures used in our proposed scheme Azeroth, referring to the notion.

Ledger. All users are allowed to 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 $\mathsf{L}_{\mathsf{T}'}$).

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 blockchain network and interacts with the smart contract, while ENA registration and updates are managed by a smart contract, and users cannot see the value in ENA without its secret key.

Auditor key. An auditor generates a pair of private/public keys $(\mathsf{ask}, \mathsf{apk})$ 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.

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

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

Commitment and Note. To build a privacy-preserving transaction, a commitment is utilized to hide the 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 given 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 stores his own notes in his wallet privately for his convenience.

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 with 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 algorithms appends a new commitment cm, performs hash computation for each tree layer, and returns a new tree root $\mathsf{rt}_{\mathsf{new}}$.

Value Type. A transaction includes several input/output asset values which are publicly visible or privately secured. In our description, pub and priv represent a publicly visible value and an encrypted (or committed) value 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 available in the encrypted account ENA.
- $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$: The digital asset value 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.

V. Azeroth

A. 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 the 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 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 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 is equal to 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 in the form of ciphertexts and whether remittance is between own accounts or between 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, for the transaction validity, zkTransfer proves that all of the above procedures are correctly performed by generating a zk-SNARK proof.

Auditability is achieved by utilizing public-key encryption with two recipients; all pct ciphertexts can be decrypted by an auditor as well as 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 $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$.

B. Algorithms

Azeroth consists of three components: Client, Smart Contract, and Relation. Client generates a transaction which includes a ciphertext and a proof. Smart Contract denotes a smart contract running on a blockchain. Relation represents a zk-SNARK circuit for generating a zkTransfer proof. The our system is a tuple of algorithms defined as follows.

[Azeroth Client]

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

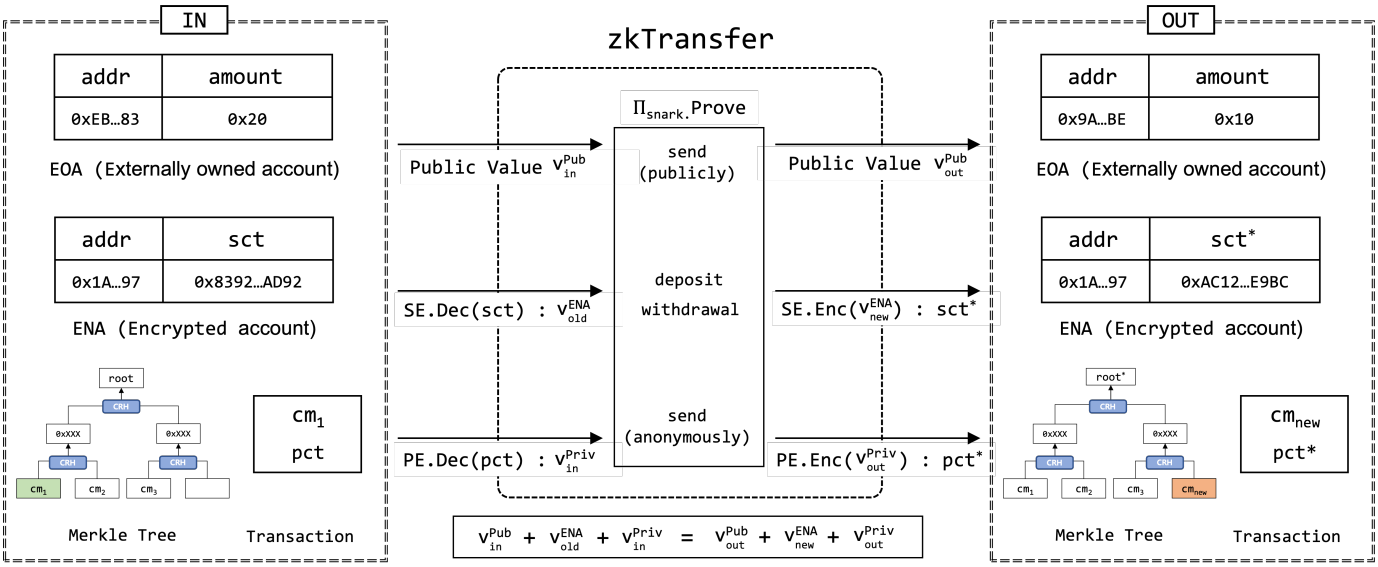


Fig. 1: Overview of zkTransfer

ii) Generating $\mathsf{cm}_{\mathsf{new}}$: By executing $\mathsf{COM}(\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$, a new commitment and its opening key are obtained. Then it encrypts $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}})$ via $\mathsf{PE.Enc}$ and outputs pct.

iii) Processing currency: The sender's ENA balance is updated based on $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}$ (from note), $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$, and $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$ or $\Delta v^{\mathsf{ENA}} = \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} - \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$.

With prepared witnesses and statements, the algorithm generates a zk-SNARK proof and finally outputs a 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.
- Audit: An auditor with a valid ask runs this algorithm to audit a transaction by decrypting the ciphertext pct in the transaction.

[Azeroth Smart Contract]

- $\mathsf{Setup}_{\mathsf{SC}}$: 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 Azeroth's smart contract.
- $\mathsf{RegisterUser}_{\mathsf{SC}}$: This algorithm registers a new encrypted account for address addr. If the address already exists in $\mathsf{List}_{\mathsf{addr}}$, the transaction is reverted. 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; $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$ is acquired from $\mathsf{EOA}^{\mathsf{send}}$, and $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$ is delivered to $\mathsf{EOA}^{\mathsf{recv}}$. If the transaction is invalid, it is reverted and aborted.

[Azeroth Relation] The statement and witness of Relation $\mathcal{R}_{\mathsf{ZKT}}$ are as follows:

$$\vec{x} = (\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{upk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{pct}_{\mathsf{new}})$$
$$\vec{w} = (\mathsf{usk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{aux}_{\mathsf{new}}, \mathsf{Path})$$

where a sender public key $\mathsf{upk}^{\mathsf{send}}$ is $(\mathsf{addr}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{send}}_{\mathsf{own}}, \mathsf{pk}^{\mathsf{send}}_{\mathsf{enc}})$ and a receiver's public key $\mathsf{upk}^{\mathsf{recv}}$ is $(\mathsf{addr}^{\mathsf{recv}}, \mathsf{pk}^{\mathsf{recv}}_{\mathsf{own}}, \mathsf{pk}^{\mathsf{recv}}_{\mathsf{enc}})$.

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

- If $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} > 0$, then $\mathsf{cm}_{\mathsf{old}}$ must exist in MT with given rt and Path.
- $\mathsf{pk}^{\mathsf{send}}_{\mathsf{own}} = \mathsf{CRH}(sk^{\mathsf{send}}_{\mathsf{own}})$.
- 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 $\mathsf{sk}^{\mathsf{send}}_{\mathsf{own}}$.
- $\mathsf{pct}_{\mathsf{new}}$ 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., $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \ldots$) are not negative.

Given the building blocks of a public-key encryption PE, a symmetric-key encryption SE, and a zk-SNARK Π_{snark}, we construct Azeroth as in Figure 2.

C. Security

Following the similar model defined in [6], [21], we define the security properties of Azeroth including *ledger indistinguishability*, *transaction unlinkability*, *transaction non-*

Azeroth **Client**

- $\mathsf{Setup}_{\mathsf{Client}}(1^\lambda, \mathcal{R}_{\mathsf{ZKT}})$:
  $(\mathsf{ek}, \mathsf{vk}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathcal{R}_{\mathsf{ZKT}})$
  $G \xleftarrow{\$} \mathbb{G}$
  Return $\mathsf{pp} = (\mathsf{ek}, \mathsf{vk}, G)$

- $\mathsf{KeyGenAudit}_{\mathsf{Client}}(\mathsf{pp})$:
  $(\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{PE.Gen}(\mathsf{pp})$
  $\mathsf{Tx}_{\mathsf{KGA}} = (\mathsf{apk})$
  Return $(\mathsf{apk}, \mathsf{ask}), \mathsf{Tx}_{\mathsf{KGA}}$

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

- $\mathsf{RetreiveNote}_{\mathsf{Client}}(\mathsf{L}, \mathsf{sk}, \mathsf{pk})$:
  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 $\mathsf{Tx}_{\mathsf{ZKT}}$ on L:
    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 $= (\mathsf{cm}, \mathsf{o}, \mathsf{v})$
  end if

- $\mathsf{zkTransfer}_{\mathsf{Client}}$ $(\mathsf{note}, \mathsf{apk}, \mathsf{usk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{EOA}^{\mathsf{recv}})$:
  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 $\mathsf{upk}_{\mathsf{send}}$ as $(\mathsf{addr}^{\mathsf{send}}, \mathsf{pk}_{\mathsf{own}}^{\mathsf{send}}, \mathsf{pk}_{\mathsf{enc}}^{\mathsf{send}})$
  Parse $\mathsf{upk}_{\mathsf{recv}}$ as $(\mathsf{addr}^{\mathsf{recv}}, \mathsf{pk}_{\mathsf{own}}^{\mathsf{recv}}, \mathsf{pk}_{\mathsf{enc}}^{\mathsf{send}})$
  Parse note as $(\mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}})$ if note $\neq \perp$;
    otherwise $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} \leftarrow 0; \mathsf{o}_{\mathsf{old}} \xleftarrow{\$} \mathbb{F}; \mathsf{cm}_{\mathsf{old}} \leftarrow \mathsf{COM}(\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}}; \mathsf{o}_{\mathsf{old}})$
  $\mathsf{sct}_{\mathsf{old}} \leftarrow \mathsf{ENA}[\mathsf{addr}^{\mathsf{send}}]$
  $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \leftarrow \mathsf{SE.Dec}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{sct}_{\mathsf{old}})$; $\mathsf{nf} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}}(\mathsf{cm}_{\mathsf{old}})$
  $\mathsf{rt} \leftarrow \mathsf{List}_{\mathsf{rt}}.\mathsf{Top}$
  $\mathsf{Path} \leftarrow \mathsf{ComputePath}_{\mathsf{MT}}(\mathsf{cm}_{\mathsf{old}})$ if $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} > 0$
  $\mathsf{cm}_{\mathsf{new}} \leftarrow \mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$
  $\mathsf{pct}_{\mathsf{new}}, \mathsf{aux}_{\mathsf{new}} \leftarrow \mathsf{PE.Enc}_{\mathsf{pk}_{\mathsf{enc}}^{\mathsf{recv}}, \mathsf{apk}}(\mathsf{o}_{\mathsf{new}} || \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} || \mathsf{addr}^{\mathsf{recv}})$
  $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} \leftarrow \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$
  $\mathsf{sct}_{\mathsf{new}} \leftarrow \mathsf{SE.Enc}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}})$

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

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

  $\pi \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{ek}, \vec{x}, \vec{w})$
  $\mathsf{Tx}_{\mathsf{ZKT}} = (\pi, \mathsf{rt}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{EOA}^{\mathsf{recv}})$
  Return $\mathsf{Tx}_{\mathsf{ZKT}}$

- $\mathsf{Audit}(\mathsf{ask}, \mathsf{pct})$:
  $\mathsf{msg} \leftarrow \mathsf{PE.Dec}_{\mathsf{ask}}(\mathsf{pct})$
  Return msg

Azeroth **Smart contract**

- $\mathsf{Setup}_{\mathsf{SC}}(\mathsf{vk})$:
  Store a zk-SNARK verification key vk
  Initialize a Merkle Tree

- $\mathsf{RegisterAuditor}_{\mathsf{SC}}(\mathsf{apk})$:
  $\mathsf{APK} \leftarrow \mathsf{apk}$

- $\mathsf{RegisterUser}_{\mathsf{SC}}(\mathsf{addr})$:
  Assert $\mathsf{addr} \notin \mathsf{List}_{\mathsf{addr}}$
  $\mathsf{ENA}[\mathsf{addr}] \leftarrow 0$

- $\mathsf{zkTransfer}_{\mathsf{SC}}$ $(\pi, \mathsf{rt}_{\mathsf{old}}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{EOA}^{\mathsf{recv}})$:
  Assert $\mathsf{rt}_{\mathsf{old}} \in \mathsf{List}_{\mathsf{rt}}$
  Assert $\mathsf{nf} \notin \mathsf{List}_{\mathsf{nf}}$
  Assert $\mathsf{addr}^{\mathsf{send}} \in \mathsf{List}_{\mathsf{addr}}$
  Assert $\mathsf{cm}_{\mathsf{new}} \notin \mathsf{List}_{\mathsf{cm}}$

$$\vec{x} = \left\{ \begin{matrix} \mathsf{APK}, \mathsf{rt}_{\mathsf{old}}, \mathsf{nf}, \mathsf{upk}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \\ \mathsf{ENA}[\mathsf{addr}^{\mathsf{send}}], \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}} \end{matrix} \right\}$$

  Assert $\Pi_{\mathsf{snark}}.\mathsf{VerProof}(\mathsf{vk}, \pi, \vec{x}) = \mathsf{true}$
  $\mathsf{ENA}[\mathsf{addr}^{\mathsf{send}}] \leftarrow \mathsf{sct}_{\mathsf{new}}$
  $\mathsf{rt}_{\mathsf{new}} \leftarrow \mathsf{TreeUpdate}_{\mathsf{MT}}(\mathsf{cm}_{\mathsf{new}})$
  $\mathsf{List}_{\mathsf{rt}}.\mathsf{append}(\mathsf{rt}_{\mathsf{new}})$
  $\mathsf{List}_{\mathsf{nf}}.\mathsf{append}(\mathsf{nf})$
  if $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} > 0$ **then** $\mathsf{TransferFrom}(\mathsf{EOA}_{\mathsf{send}}, this, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}})$
  end if
  if $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}} > 0$ **then** $\mathsf{TransferFrom}(this, \mathsf{EOA}_{\mathsf{recv}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}})$
  end if

Azeroth **Relation**

- Relation $R(\vec{x}; \vec{w})$:

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

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

  Parse $\mathsf{sk}^{\mathsf{send}}$ as $(\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}, \mathsf{sk}_{\mathsf{enc}}^{\mathsf{send}})$
  Parse $\mathsf{upk}_{\mathsf{send}}$ as $(\mathsf{addr}^{\mathsf{send}}, \mathsf{pk}_{\mathsf{own}}^{\mathsf{send}}, \mathsf{pk}_{\mathsf{enc}}^{\mathsf{send}})$
  Parse $\mathsf{upk}_{\mathsf{recv}}$ as $(\mathsf{addr}^{\mathsf{recv}}, \mathsf{pk}_{\mathsf{own}}^{\mathsf{recv}}, \mathsf{pk}_{\mathsf{enc}}^{\mathsf{send}})$
  if $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} > 0$ **then**
    Assert $\mathsf{true} = \mathsf{Membership}_{\mathsf{MT}}(\mathsf{rt}, \mathsf{cm}_{\mathsf{old}}, \mathsf{Path})$
  end if
  Assert $\mathsf{pk}_{\mathsf{own}}^{\mathsf{send}} = \mathsf{CRH}(\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}})$
  Assert $\mathsf{addr}^{\mathsf{send}} = \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}} || \mathsf{pk}_{\mathsf{enc}})$
  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 $\mathsf{addr}^{\mathsf{recv}} = \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}}^{\mathsf{recv}} || \mathsf{pk}_{\mathsf{enc}}^{\mathsf{recv}})$
  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 $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \leftarrow \mathsf{SE.Dec}_{\mathsf{k}_{\mathsf{ENA}}^{\mathsf{send}}}(\mathsf{sct}_{\mathsf{old}})$
  end if
  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 $\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} \geq 0$; $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} \geq 0$

Fig. 2: Azeroth scheme Π_{Azeroth}

malleability, and *balance*, and define *auditability* as a new property.

Ledger Indistinguishability. Informally, we say that the ledger is indistinguishable if it does not disclose new information, even when an adversary $\mathcal{A}$ can see the public information and even adaptively engender honest parties to execute Azeroth functions. Namely, even if there are two ledgers L_0 and L_1, designed by the adversary using queries to the oracle, $\mathcal{A}$ cannot tell the difference between the two ledgers. We design an experiment L-IND as shown in fig. 3.

$\underline{\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{L\text{-}IND}}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{L}_0, \mathsf{L}_1) \leftarrow \mathcal{A}^{\mathcal{O}_0^{\mathsf{Azeroth}}, \mathcal{O}_1^{\mathsf{Azeroth}}}(\mathsf{pp})$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{Q} \xleftarrow{\$} \{\mathsf{KeyGenUser}, \mathsf{zkTransfer}\}$
$\mathsf{ans} \leftarrow \mathbf{Query}_{\mathsf{L}_b}(\mathcal{Q})$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_0^{\mathsf{Azeroth}}, \mathcal{O}_1^{\mathsf{Azeroth}}}(\mathsf{L}_0, \mathsf{L}_1, \mathsf{ans})$
return $b = b'$

Fig. 3: The ledger indistinguishability experiment (L-IND)

We say that Azeroth scheme is *ledger indistinguishable* if $|\mathbf{Adv}^{\mathsf{L\text{-}IND}}_{\mathsf{Azeroth},\mathcal{A}} - 1/2| \leq \mathsf{negl}(\lambda)$ for every $\mathcal{A}$ and adequate security parameter λ.

$\underline{\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{TR\text{-}UN}}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{L} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{pp})$
$\left\{(\mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, aux), (\overline{\mathsf{pk}^{\mathsf{send}}}, \overline{\mathsf{pk}^{\mathsf{recv}}}, \overline{aux})\right\}$
$\leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{L})$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{Tx} \leftarrow \mathsf{zkTransfer}(\mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, aux)$
if $b = 0$
$\mathsf{Tx}' \leftarrow \mathsf{zkTransfer}(\overline{\mathsf{pk}^{\mathsf{send}}}, \overline{\mathsf{pk}^{\mathsf{recv}}}, \overline{aux})$
else if $b = 1$
$\mathsf{Tx}' \leftarrow \mathsf{zkTransfer}(\overline{\mathsf{pk}^{\mathsf{send}}}, \mathsf{pk}^{\mathsf{recv}}, \overline{aux})$
$b' \leftarrow \mathcal{A}(\mathsf{Tx}, \mathsf{Tx}', \mathsf{L})$
return $b = b'$

Fig. 4: The transaction unlinkability experiment (TR-UN)

Transaction Unlinkability. Transaction unlinkability is defined as an indistinguishability problem in which a PPT adversary $\mathcal{A}$ cannot distinguish the receiver's information. Note that a transaction caller (sender) is always identifiable in an existing account-based blockchain model. Still, if the receiver's information is unlinkable, it is guaranteed that the linkability information between the sender and the receiver is hidden. Because the recipient information is hidden, the sender can create a transaction with himself as the recipient privately as well. We say that a transaction is unlinkable if it does not disclose the correlation between its sender and receiver. We describe an experiment TR-UN as shown in fig. 4 where aux denotes the remaining input of zkTransfer. Formally, let $\mathbf{Adv}^{\mathsf{TR\text{-}UN}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game $\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{TR\text{-}UN}}$. Azeroth satisfies the *transaction unlinkability* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{TR\text{-}UN}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda) - 1/2| \leq \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{TR\text{-}NM}}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{L} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{pp}, \mathsf{ask})$
$\mathsf{Tx}' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{L})$
$b \leftarrow \mathsf{VerifyTx}(\mathsf{Tx}', \mathsf{L}') \wedge \mathsf{Tx} \notin \mathsf{L}'$
return $b \wedge (\exists \mathsf{Tx} \in \mathsf{L} : \mathsf{Tx} \neq \mathsf{Tx}' \wedge \mathsf{Tx.nf} = \mathsf{Tx}'.\mathsf{nf})$

Fig. 5: The transaction non-malleability experiment (TR-NM)

Transaction Non-malleability. Intuitively, a transaction is non-malleable if no new transaction is constructed differently from the previous transactions without knowing private data (witness) such as secret keys. Even when an auditor tries to attack or an auditor's secret key is exposed, this property should hold. We show an experiment $\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{TR\text{-}NM}}$ in fig. 5. Let $\mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game TR-NM. The Azeroth satisfies the *transaction non-malleability* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{BAL}}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{L} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{pp})$
$(\mathsf{List}_{\mathsf{cm}}, \mathsf{ENA}, \mathsf{EOA}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{L})$
$(\mathsf{v}^{\mathsf{ENA}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}})$
$\leftarrow \mathsf{Compute}(\mathsf{L}, \mathsf{List}_{\mathsf{cm}}, \mathsf{ENA}, \mathsf{EOA})$
if $\mathsf{v}^{\mathsf{ENA}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} > \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}$ **then return** 1
else return 0

Fig. 6: The balance experiment (BAL)

Balance. We say that Azeroth is balanced if and only if no attacker can spend more than what she has or receives. Let $\mathbf{Adv}^{\mathsf{BAL}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game BAL as described in fig. 6. For a negligible function $\mathsf{negl}(\lambda)$, the Azeroth is *balanced* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{BAL}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

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 is different from commitment openings. Let $\mathbf{Adv}^{\mathsf{AUD}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game AUD as described in fig. 7. For a negligible function $\mathsf{negl}(\lambda)$, the Azeroth is *auditable* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{AUD}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

Theorem 5.1: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) be a Azeroth scheme in Figure 2. Π_{Azeroth} satisfies *ledger indistinguishability*, *transaction un-*

$\underline{\mathsf{Azeroth}.\mathcal{G}_{\mathcal{A}}^{\mathsf{AUD}}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{L} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{pp})$
$(\mathsf{Tx}, aux) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Azeroth}}}(\mathsf{L})$
Parse $\mathsf{Tx} = (\mathsf{cm}, \mathsf{pct})$
$b \leftarrow \mathsf{VerifyTx}(\mathsf{Tx}, \mathsf{L}) \wedge$
$\quad \mathsf{VerifyCommit}(\mathsf{cm}, aux) \wedge aux \neq \mathsf{Audit}_{\mathsf{ask}}(\mathsf{pct})$
return b

Fig. 7: The auditability experiment (AUD)

linkability, *transaction non-malleability*, *balance*, and *auditability*.

Proof. The proofs for the above theorem are described in appendix A.1 A.2.

D. Extension to threshold auditing

In a blockchain, the presence of a privileged entity(e.g., an auditor) who can infringe on the privacy of all participants is able to induce another baleful problem. One solution to mitigate this problem is to use a threshold scheme. In a threshold public-key encryption [13], the message is encrypted using a public key, and the private key is shared among the participants. Several participants more than some threshold number t must cooperate in the decryption procedure to decrypt an encrypted message.

Our audit function can be extended to a threshold public-key encryption using a standard secret sharing scheme. Briefly, let the secret key k be shared secret key among $P_1, \ldots, P_n$ using a polynomial f of degree t where t is a threshold, meaning that the encryption tolerates the passive corruption of t parties. In key generation phase, a trusted manager chooses $t+1$ random coefficients $a_0, a_1, a_2, \ldots, a_{t+1}$ s.t. $a_0 = k$ and computes $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t+1} x^{t+1} \mod q$ where q is a field prime number. The dealer sends a split secret key ($sk_j = f(j)$) to each P_j for all j. In the encryption phase, all processes are the same as the standard ElGamal encryption with $pk = g^k$. Thus, the ciphertext consists of $(c_0, c_1) = (G^r, pk^r \cdot m)$ for the message m. For threshold decryption, an auditor sends a decryption request to other auditors. Upon receiving a decryption request on (c_0, c_1), an auditor A_i computes a decryption share $c_0^{sk_i}$ and sends it back to the requestor (or other auditors in case of group auditing). Upon receiving shares from a set of $t+1$ parties, the auditors can compute the message m.

The role of the trusted manager can be also distributed [32] so that the participants jointly choose shares corresponding to a (t, n)-secret sharing of a random value for a system without the trusted manager. Each participant P_i arbitrary selects a polynomial $f_i(x)$ of degree t over $\mathbb{Z}_q$ and sends $f_i(j)$ to every participant P_j. Next, P_j obtains its share $a_j = f(j)$ where $f(j) = \sum_{k=1}^{n} f_k(j)$. Note that the values $(a_1, \ldots, a_n)$ make up a (t, n)-threshold secret sharing. Thus, in order to audit a transaction, the mutual cooperation of auditors exceeding a certain quorum is unconditionally required.

VI. Implementation and Experiment

A. Implementation

Our Azeroth implementation coded in Python, C++, and Solidity languages consists of two parts; the client and the smart contract. The client interacts with blockchain network using Web3.py[2]. To generate a zk-SNARK proof in Azeroth, we use libsnark[3] with Groth16 [19] and BN254 curve.

PRF, COM, and CRH: We instantiate 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 MiMC7 [2] and Poseidon [18] as well as a standard hash function SHA256 [28].

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 MiMC7 and Poseidon as follows.

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

$$\frac{\mathsf{SE.Dec}_k(r, \mathsf{sct}) \rightarrow \mathsf{msg}}{\mathsf{msg} \leftarrow \mathsf{sct} - \mathsf{PRF}_k(r)}$$
$$\textbf{Return } \mathsf{msg}$$

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

Public-key encryption: We implement public-key encryption with two recipients of a receiver and an auditor by extending ElGamal encryption system such that the randomness can be re-used. (Refer to the general treatment of randomness re-use in multi-recipient encryption in [5].) For the performance gain, we also utilize the standard hybrid encryption; a random key is encrypted by the public key encryption and the key is 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, and the symmetric-key encryption SE, the resulting implementation of the public key encryption is as follows.

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

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

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

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

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

Return $\mathsf{pct}, \mathsf{aux}$

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

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

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

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 [2] and its Merkle tree depth is 32. Table I(a) illustrates our system environments. For the overall performance evaluation, we execute all experiments on the machine Server described in Table I(a) as a default machine. And the default blockchain is the Ethereum testnet. The used proof system is Gro16 [19].

Overall performance. We show that the performance and the gas consumption in Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7,32}}$ as shown in Table I (b).[4] The execution time $4.04s$ of Setup is composed of the zk-SNARK key generation time $2.2s$ and the deployment time $1.84s$ of the Azeroth's smart contract to the blockchain. Setup consumes a considerable amount of gas due to the initialization of Merkle Tree. In zkTransfer, the executed time is $4.38s$ including both the Client part and the Smart Contract part. The gas is mainly consumed to verify the SNARK proof and update the Merkle hash tree. Varying the hash function, the further analysis of zkTransfer is described in the following experiments.

zk-SNARK performance. We evaluate the performance of zk-SNARK used to execute zkTransfer on various systems $\mathsf{Server}, \mathsf{System}_1, \cdots, \mathsf{System}_4$ as described in Table I (b). Table I (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.

Hash type and tree depth. We evaluate Azeroth performance depending on hash tree depths and hash types as shown in Figure 8 and Figure 9.

Figure 8 illustrates the execution time of zk-SNARK for MiMC7 [2], Poseidon [18][5], and SHA256 [28] 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 20x and 40x 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 8 (d), which is similar to the gas consumption trend shown in Figure 8 (e).

Figure 9 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 9 (c) shows that MiMC7 and Poseidon hashes consume relatively more gas than SHA256 since not only is SHA256 natively supported in Ethereum [9], but also it shows better native performance as shown in Figure 8 (e).

The number of constraints. We measure the number of constraints for each algorithm component as shown in Table II. 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.

Performance analysis of smart contract.

We analyze the execution time and the gas consumption in smart contract $\mathsf{zkTransfer}_{\mathsf{SC}}$. Table III shows the gas consumption for each function in smart contract $\mathsf{zkTransfer}_{\mathsf{SC}}$. In the smart contract, the most time-consuming functions are the proof verification Π_{snark}.Verify and the Merkle hash tree update TreeUpdate. The gas consumption for TreeUpdate depends on hash tree depth and hash type while Π_{snark}.Verify remains constant.

Figure 10 represents the execution time of Π_{snark}.Verify and TreeUpdate. The verification time is $11.9ms$, and the execution time of TreeUpdate is $12.8ms$, $17.4ms$, and $7.3ms$ on MiMC7, Poseidon, and SHA256 respectively.

Deployment to various blockchains. We execute Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7,8}}$ on various blockchains such as Ethereum [9], Hyperledger Besu [22], and Klaytn [1] which support a smart contract. The gas consumption result is shown in table IV.

Comparison with the other existing schemes. We compare the proposed scheme Azeroth with the other privacy-preserving transfer schemes such as Zeth [30], Blockmaze [21], Zether [8], and PGC [12] in Table V. Our proposal shows better performance than the existing schemes, even if Azeroth provides an additional function of auditability. Zeth and Blockmaze are implemented with $\mathsf{cfg}_{\mathsf{MiMC7,32}}$ and $\mathsf{cfg}_{\mathsf{SHA256,8}}$ respectively and the same configuration is applied to the proposed scheme for a fair comparison. The experiment

[4]The result includes the execution time until the task of receiving the receipt of the transaction.

[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 I: Benchmark of Azeroth

(a) System specification

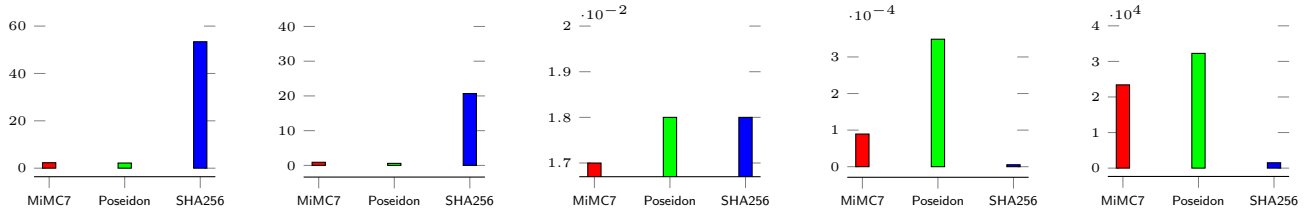
Machine	OS	CPU	RAM
Server	Ubuntu 20.04	Intel(R) Xeon Gold 6264R@3.10GHz	256GB
System_1	macOS 11.2	M1@3.2GHz	8GB
System_2	macOS 11.6	Intel(R) i7-8850H CPU @ 2.60GHz	32GB
System_3	android 11	Exynos9820	8GB
System_4	iOS 15.1	A12 Bionic	4GB

(b) Execution time and gas consumption of Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7},32}$

	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

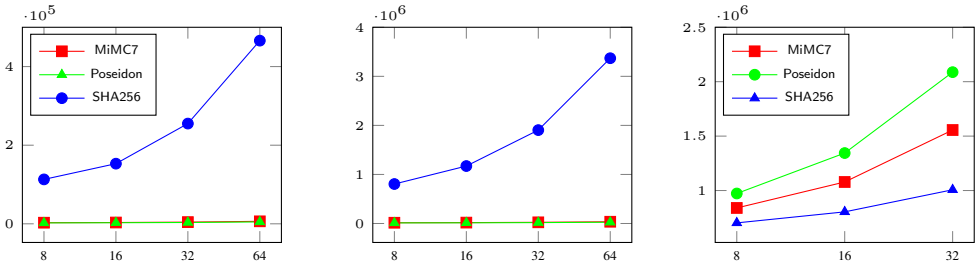
(c) Execution time of zk-SNARK in zkTransfer

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



(a) $\Pi_{\mathsf{snark}}.\mathsf{Setup}$ (b) $\Pi_{\mathsf{snark}}.\mathsf{Prove}$ (c) $\Pi_{\mathsf{snark}}.\mathsf{Verify}$ (d) Native (e) Gas consumption

Fig. 8: 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

Fig. 9: Key size, constraints in circuits and gas consumption by varying hash tree depth and hash type

TABLE II: The number of constraints for each algorithm component

	$\mathsf{Membership}_{32}$	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 III: Gas cost for each function

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

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 significantly long in Blockmaze.

TABLE IV: Gas consumption on various blockchains with $\mathsf{cfg}_{\mathsf{MiMC7},32}$

	Ethereum	Besu	Klaytn
Setup	3,778,604	4,382,410	4,845,674
RegisterAuditor	63,179	66,115	68,891
RegisterUser	45,543	55,215	56,967
zkTransfer	854,110	1,773,406	951,220

On the other hand, PGC [12] and Zether [8] use standard ElGamal encryption and NIZK to provide confidentiality instead of utilizing Merkle Tree. The performance results in these works exclude anonymity, which means that 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 4x smaller and the size of pp is 22x 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. Note that it takes $20s$ to load the proving key in Blockmaze while it is only $1s$ in Azeroth. Hence Azeroth provides much better performance than Blockmaze in practice.

Zether [8] and PGC [12] 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.

[7]https://github.com/clearmatics/zeth
[8]https://github.com/trufflesuite/ganache
[9]https://github.com/Agzs/BlockMaze
[10]The meaning is that the account is renewed immediately through one transaction.

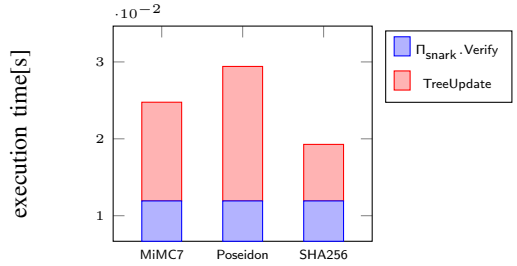


Fig. 10: The execution time of functions in $\mathsf{zkTransfer}_{\mathsf{SC}}$

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 while the transferring correctness is guaranteed by a zero-knowledge proof. In addition, the proposed Azeroth supports an auditing functionality in which an authorized auditor can trace transactions to comply an anti-money laundry law. Its security is proven formally and it is implemented in various platforms including an Ethereum testnet blockchain. The experimental results show that the proposed Azeroth is efficient enough to be practically deployed.

References

[1] Klaytn position paper v2.1.0. https://www.klaytn.com/Klaytn_PositionPaper_V2.1.0.pdf.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, pages 191–219, 2016.

[3] Roman Storm Alexey Pertsev, Roman Semenov. Tornado cash privacy solution.

[4] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. page 255–267. Association for Computing Machinery, 2020.

[5] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Trans. Inf. Theory*, 53(11):3927–3943, 2007.

[6] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[7] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. pages 947–964, 2020.

[8] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference*, pages 423–443, 2020.

[9] V. Buterin. Ethereum white paper: a next generation smart contract-decentralized application platform. 2013.

[10] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066, 2017. https://ia.cr/2017/1066.

[11] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via pvorm. CCS '17, page 701–717. Association for Computing Machinery, 2017.

[12] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. Pgc: Decentralized confidential payment system with auditability. In *Computer Security – ESORICS 2020*, pages 591–610, 2020.

[13] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[14] Benjamin E. Diamond. Many-out-of-many proofs and applications to anonymous zether. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1800–1817, 2021.

[15] E. Duffield and D. Diaz. "dash: A privacycentric cryptocurrency. https://github.com/dashpay/dash/wiki/Whitepaper, 2015.

[16] FATF. "virtual assets and virtual asset service providers". 2021.

[17] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. *Quisquis: A New Design for Anonymous Cryptocurrencies*, pages 649–678. 11 2019.

[18] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

TABLE V: Comparison between our proposed scheme and existing work

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

Azeroth	Setup		zkTransfer	
	time	pp	time	$\mathsf{tx}_{\mathsf{size}}$
	2.846s	4.32MB	0.983s	1,186B
Zeth	10.646s	94.24MB	10.47s	1,380B
	time	pp	time	$\mathsf{tx}_{\mathsf{size}}$
	Setup		Mix	

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

Azeroth	Setup		zkTransfer							
	time	pp	time				$\mathsf{tx}_{\mathsf{size}}$			
	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	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$
	Setup		Mint		Redeem		Send		Deposit	

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

	$\mathsf{Azeroth}_{\mathsf{cfg}_{\mathsf{MiMC7,32}}}$	Zether [8]	PGC [12]
gas cost	1,555,957	7,188,000	8,282,000
transaction size (bytes)	1,186	1,472	1,310

[19] Jens Groth. On the size of Pairing-Based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 305–326, 2016.

[20] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from Simulation-Extractable SNARKs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference*, pages 581–612, 2017.

[21] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[22] hyperledger.org. Besu. https://github.com/hyperledger/besu, 2018.

[23] Hui Kang, Ting Dai, Nerla Jean-Louis, Shu Tao, and Xiaohui Gu. Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 543–555, 2019.

[24] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[25] G. Maxwell. Coinjoin: Bitcoin privacy for the real world. 2013.

[26] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. volume 2018, pages 105–121, 2018.

[27] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving auditing for distributed ledgers. USENIX Association, April 2018.

[28] National Institute of Standards and Technology (NIST). Fips180-2: Secure hash standard. 2002.

[29] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[30] Antoine Rondelet and Michal Zajac. ZETH: on integrating zerocash on ethereum. *CoRR*, abs/1904.00905, 2019.

[31] N. Nicolas Saberhagen. Cryptonote v2.0. https://cryptonote.org/whitepaper.pdf, 2013.

[32] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

APPENDIX

A. *Security of* Azeroth

In section V-C, we briefly explain the security, such as the model and the probability. From now on, we describe the formal security of Azeroth. Once restating Theorem 5.1, a Azeroth scheme Π_{Azeroth} is $secure$ if it satisfies ledger indistinguishability, transaction unlinkability, transaction non-malleability, balance, and auditability.

In the elucidation of the security for each property and its experiment, we assume that there exists a (stateful) Azeroth oracle $\mathcal{O}^{\mathsf{Azeroth}}$ answering queries from an adversary $\mathcal{A}$ utilizing a challenger $\mathcal{C}$ which is the role of the performer about the experiment sanity checks. We first recount how $\mathcal{O}^{\mathsf{Azeroth}}$ works as below.

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] CMT, a set of a commitment; [IV] ENA, a set of an encrypted account which includes the user account address in itself; [V] EOA, a set of an external-owned account being similar with ENA excepting for whether its encryption. In the beginning, all of the elements are empty. Additionally, we denote $*$ as renewal.

Now, we present each type of query $\mathcal{Q}$ and how it works.

- $\mathcal{Q}(\mathsf{KeyGenUser})$: $\mathcal{C}$ generates a user key pair $(\mathsf{pk}, \mathsf{sk})$ using KeyGenUser algorithm, where $\mathsf{pk} = (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}})$. $\mathcal{C}$ then registers public key to the smart contract and initializes ENA[addr]. The generated key pair and ENA[addr] are stored into Acct and ENA, respectively. Finally $\mathcal{C}$ outputs pk.
- $\mathcal{Q}(\mathsf{zkTransfer}, \mathsf{apk}, \mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{EOA})$: $\mathcal{C}$ generates a transaction $\mathsf{Tx}_{\mathsf{ZKT}}$ for corresponding inputs using zkTransfer algorithm. Once $\mathsf{Tx}_{\mathsf{ZKT}}$ is submitted to ledger, the EOA and ENA stores updated amount, and the commitment is stored into CMT as well.
- $\mathcal{Q}(\mathsf{Insert}, \mathsf{Tx})$: $\mathcal{C}$ simply submits the received transaction (i.e., $\mathsf{Tx}_{\mathsf{ZKT}}$, $\mathsf{Tx}_{\mathsf{KGU}}$). If Tx is invalid, the smart contract

reverts and nothing happens.

1) Ledger Indistinguishability: We describe ledger indistinguishability using an experiment L-IND including a PPT adversary $\mathcal{A}$ struggling to find a crack of a given Azeroth scheme.

Intuitively, the meaning of the above equation is that $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{L\text{-}IND}}$, the advantage of $\mathcal{A}$ in the L-IND experiment, is at most $\mathsf{negl}(\lambda)$.

We now describe *public consistency* for two queries $(\mathcal{Q}, \mathcal{Q}')$ which must be the same type and publicly consistent in $\mathcal{A}$'s viewpoint. First of all, it must be equal to the public information in the pair of queries such as the following: $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$, the value to be transferred; $\mathsf{addr}^{\mathsf{send}}$, the receiver address. Moreover, the commitment, a nullifier, and the ciphertexts are valid in both queries. In addition, each of the different query types has to fulfill the following restriction respectively.

For KeyGenUser, the independent pair of queries $(\mathcal{Q}, \mathcal{Q}')$ must satisfy the following restriction :

- The user account address addr in the query must correspond to the address existing in Acct
- Both oracles must generate and return the same address

For zkTransfer, the independent pair of queries $(\mathcal{Q}, \mathcal{Q}')$ must satisfy the following restrictions :

- The commitment $\mathsf{cm}_{\mathsf{old}}$ in $\mathcal{Q}$ must correspond to commitment that appears in CMT
- The $\mathsf{v}^{\mathsf{ENA}}{}_{\mathsf{old}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ is equal to $\mathsf{v}^{\mathsf{ENA}}{}_{\mathsf{new}} + \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$.
- The account address $\mathsf{addr}^{\mathsf{send}}$ should match that in ENA

We now give a formal definition of an experiment L-IND as follows:

i) Compute a public parameter pp, provide it to an adversary $\mathcal{A}$, and initialize two distinct oracles $\mathcal{O}_0^{\mathsf{Azeroth}}, \mathcal{O}_1^{\mathsf{Azeroth}}$.
ii) $\mathcal{C}$ chooses a random bit $b \in \{0, 1\}$.
iii) $\mathcal{A}$ sends *public consistency* queries $(\mathcal{Q}, \mathcal{Q}')$ to $\mathcal{C}$ as many times she wants, and $\mathcal{C}$ answers the queries as following:
   a) Set $(\mathsf{L}_b, \mathsf{L}_{1-b})$ as a ledger tuple. These ledgers are corresponding to $\mathcal{O}_b^{\mathsf{Azeroth}}, \mathcal{O}_{1-b}^{\mathsf{Azeroth}}$ respectively.
   b) Give a ledger tuple to $\mathcal{A}$ in each stage, and send $\mathcal{Q}$ to $\mathcal{O}_b^{\mathsf{Azeroth}}$ and $\mathcal{Q}'$ to $\mathcal{O}_{1-b}^{\mathsf{Azeroth}}$.
   c) Obtain two oracle answers (a_b, a_{1-b}) and return it to $\mathcal{A}$.
   d) Repeat the process b), c) until $\mathcal{A}$ outputs a bit b'.
iv) If $b = b'$ then the experiment L-IND returns 1; otherwise, 0.

Definition A.1: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) 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 Unlinkability: We denote $\mathcal{T}$ as the table of zkTransfer transactions generated by $\mathcal{O}^{\mathsf{Azeroth}}$ in response to the zkTransfer query. We formally describe the transaction unlinkability experiment TR-UN as follows:

i) Compute a public parameter pp, and provide it to an adversary $\mathcal{A}$.
ii) $\mathcal{A}$ makes a query (zkTransfer) to $\mathcal{O}^{\mathsf{Azeroth}}$ and receives its answer along with the ledger L.
iii) Repeat the above procedure (Step ii) until $\mathcal{A}$ outputs a pair of tuples: $(\mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, aux)$, $(\overline{\mathsf{pk}^{\mathsf{send}}}, \overline{\mathsf{pk}^{\mathsf{recv}}}, \overline{aux})$, satisfied with the following conditions: (a) $\mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, \overline{\mathsf{pk}^{\mathsf{send}}}, \overline{\mathsf{pk}^{\mathsf{recv}}}$ are public keys from accounts in Acct; (b) both queries must generate valid transactions.
iv) Set $\mathsf{Tx}_{\mathsf{ZKT}}$ as a transaction output from $\mathcal{Q}(\mathsf{pk}^{\mathsf{send}}, \mathsf{pk}^{\mathsf{recv}}, aux)$
v) If $b = 0$, make $\mathsf{Tx}_{\mathsf{ZKT}}{}'$ with query $\mathcal{Q}(\overline{\mathsf{pk}^{\mathsf{send}}}, \overline{\mathsf{pk}^{\mathsf{recv}}}, \overline{aux})$, else make $\mathsf{Tx}_{\mathsf{ZKT}}{}'$ with $\mathcal{Q}(\overline{\mathsf{pk}^{\mathsf{send}}}, \mathsf{pk}^{\mathsf{recv}}, \overline{aux})$.
vi) $\mathcal{A}$ receives a tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, \mathsf{Tx}_{\mathsf{ZKT}}{}')$ and returns b'.
vii) If $b = b'$ then the experiment TR-UN returns 1; otherwise, 0.

Definition A.2: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is TR-UN secure if the following equation holds:

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

3) Transaction Non-malleability: Let T be the $\mathsf{Tx}_{\mathsf{ZKT}}$ set generated by $\mathcal{O}^{\mathsf{Azeroth}}$ in response to zkTransfer queries. We define an experiment TR-NM with PPT adversary $\mathcal{A}$ trying to break a given Azeroth scheme. Note that $\mathcal{A}$ includes an AUD trying to attack our scheme. Now we illustrate the TR-NM experiment in detail.

i) Compute a public parameter pp, and provide it to an adversary $\mathcal{A}$.
ii) $\mathcal{A}$ makes a query (zkTransfer) to $\mathcal{O}^{\mathsf{Azeroth}}$ and receives its answer along with the ledger L.
iii) Repeat the above procedure (Step ii) until $\mathcal{A}$ sends a transaction Tx', satisfied with the following conditions:
   a) There exists a transaction $\mathsf{Tx} \in \mathsf{L}$
   b) $\mathsf{Tx}' \neq \mathsf{Tx}$
   c) A nullifier nf appeared in Tx' is the same nullifier revealed in Tx.
   d) $\mathsf{VerifyTx}(\mathsf{Tx}', \mathsf{L}') = 1$, where L' is the snapshot of the previous ledger state which does not contain Tx yet;
iv) If the transaction satisfying all conditions exists, then the experiment TR-NM returns 1; otherwise, 0.

Definition A.3: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) 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:

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

4) Balance: We define an experiment BAL with PPT adversary $\mathcal{A}$ trying to break a given Azeroth scheme. Now we characterize an experiment BAL as follows:

i) Compute a public parameter pp, and provide it to an adversary $\mathcal{A}$.

ii) $\mathcal{A}$ makes a query (zkTransfer) to $\mathcal{O}^{\mathsf{Azeroth}}$ and receives its answer along with the ledger L.
iii) Repeat the above procedure (Step ii) until $\mathcal{A}$ sends CMT.
iv) $\mathcal{C}$ with key pair calculates each value mentioned above, and checks if the following equation holds:

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

v) If the values satisfy the equation, then the experiment BAL returns 1; otherwise, 0.

Definition A.4: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) 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:

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

5) Auditability: Let aux be the auxiliary input consisting of the committed value and its opening, utilized when verifying the commitment. If the commitment is correct then the function returns 1, otherwise returns 0. We now define precisely the experiment AUD as follows:

i) Compute a public parameter pp, and provide it to an adversary $\mathcal{A}$.
ii) $\mathcal{A}$ requests a zkTransfer query to $\mathcal{O}^{\mathsf{Azeroth}}$ and receives a response.
iii) Repeat the above procedure (Step ii) until $\mathcal{A}$ sends a tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, aux)$, satisfied with the following conditions:
   a) $\mathsf{Tx}_{\mathsf{ZKT}}$ is valid.
   b) aux and the commitment $\mathsf{cm}_{\mathsf{new}}$ in $\mathsf{Tx}_{\mathsf{ZKT}}$ is valid.
   c) The decrypted message of ciphertext $\mathsf{pct}_{\mathsf{new}}$ is not equal to $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{send}})$.
iv) If the tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, aux)$ satisfies with all conditions above, then the experiment AUD returns 1; otherwise, 0.

Definition A.5: Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) 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:

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

B. Proofs of Security

We now formally prove 5.1 by showing that Azeroth construction satisfies ledger indistinguishability, transaction unlinkability, 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 $\mathsf{Game}_{\mathsf{Real}}$ and a simulation game $\mathsf{Game}_{\mathsf{Sim}}$ is negligible. All Games are executed by interaction of an adversary $\mathcal{A}$ with a challenger $\mathcal{C}$, as in the L-IND experiment. However, $\mathsf{Game}_{\mathsf{Sim}}$ has a distinctness from the others since it runs regardless of a bit b where b means b of the L-IND experiment. Thus, for $\mathsf{Game}_{\mathsf{Sim}}$, the advantage of $\mathcal{A}$ is 0. Moreover, the zk-SNARK keys are generated as $(\mathsf{ek}, \mathsf{vk}, \mathsf{td}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Sim}(\mathcal{R})$ to obtain the zero-knowledge trapdoor td. We now show that $\mathbf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi_{\mathsf{Azeroth}},\mathcal{A}}$ is at most negligibly different than $\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}}$. First of all, we define the notations as follows.

TABLE VI: Notations

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

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

- **Query**(KeyGenUser) (i.e., $(\mathcal{Q}, \mathcal{Q}')$ = KeyGenUser): $\mathcal{C}$ actions under the $\mathcal{Q}$(KeyGenUser) query, except that it does the following modifications: $\mathcal{C}$ generates user key pair $(\mathsf{pk} = (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}}), \mathsf{sk}) \leftarrow \mathsf{KeyGenUser}(\mathsf{pp})$, supersedes $pk_{\mathsf{own}}, pk_{\mathsf{enc}}$ to a random string of the appropriate length, and then computes the user address $\mathsf{addr} \leftarrow \mathsf{CRH}(pk_{\mathsf{own}}, pk_{\mathsf{enc}})$. $\mathcal{C}$ also puts these elements in a table and returns pk to $\mathcal{A}$. $\mathcal{C}$ does the above procedure for $\mathcal{Q}'$.
- **Query**(zkTransfer,$\mathsf{pk}^{\mathsf{send}}$, $\mathsf{pk}^{\mathsf{recv}}$, $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$, $\mathsf{EOA}_{\mathsf{recv}}$) (i.e., $(\mathcal{Q}, \mathcal{Q}')$ = zkTransfer): $\mathcal{C}$ actions under the $\mathcal{Q}$(zkTransfer) query, except that it does the following modifications: by default, we assume that $\mathsf{pk}^{\mathsf{send}}$ exists in the table. If $\mathsf{pk}^{\mathsf{send}}$ does not exist in the table, we abort the queries. $\mathcal{C}$ comes up with random strings and replaces nf and $\mathsf{cm}_{\mathsf{new}}$ to these values, respectively. If $\mathsf{pk}^{\mathsf{recv}}$ is a public key generated by a previous query to KeyGenUser, then $\mathcal{C}$ sets $\mathsf{sct}_{\mathsf{new}}$ and $\mathsf{pct}_{\mathsf{new}}$ to an arbitrary string. Otherwise, $\mathcal{C}$ computes these elements as in the zkTransfer algorithm. Also, $\mathcal{C}$ stores the changed elements to the table.

We now define each of games to prove the ledger indistinguishability of Azeroth. Once again, $\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}}_{\mathcal{A}}$ 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 is equal to $\mathsf{Game}_{\mathsf{Real}}$ except that $\mathcal{C}$ simulates the zk-SNARK proof. For zkTransfer, the zk-SNARK key is generated as $(\mathsf{ek}, \mathsf{vk}, \mathsf{td}_{\mathsf{ZKT}}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Sim}(\mathcal{R}_{\mathsf{ZKT}})$ instead of $\Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathcal{R}_{\mathsf{ZKT}})$ to procure the trapdoor $\mathsf{td}_{\mathsf{ZKT}}$. After obtaining the $\mathsf{td}_{\mathsf{ZKT}}$, $\mathcal{C}$ 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 $\mathsf{Game}_{\mathsf{Real}}$. In addition, when $\mathcal{A}$ asks for the KeyGenUser query, we replace the elements of public key pk as a random string. The simulated $(\mathsf{sk}, \mathsf{pk})$ distribution is also identical to that of the key pairs computed in $\mathsf{Game}_{\mathsf{Real}}$. In a nutshell, $\mathbf{Adv}^{\mathsf{Game}_1} = 0$.

• Game_2. We define the Game_2 which is equal to Game_1 except that $\mathcal{C}$ uses a random string r of a suitable length to replace the ciphertext $\mathsf{pct}_{\mathsf{new}}$. If the address addr of $\mathsf{pk}^{\mathsf{send}}$ in $\mathcal{A}$'s zkTransfer query exists in the $\mathcal{C}$'s table, $\mathcal{C}$ computes $\mathsf{sct}_{\mathsf{old}}$ as r. Otherwise, $\mathcal{C}$ aborts. By Lemma A.1, $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

• Game_3. We define the Game_3, which is the same as Game_2 with one modification where $\mathcal{C}$ changes the ciphertext $\mathsf{sct}_{\mathsf{new}}$ from correct to an acceptable random string r. Specifically, if the address addr of $\mathsf{pk}^{\mathsf{send}}$ exists in the table, $\mathcal{C}$ computes $\mathsf{sct}_{\mathsf{new}}$ as r. Otherwise, $\mathcal{C}$ aborts. By Lemma A.2, $|\mathbf{Adv}^{\mathsf{Game}_3} - \mathbf{Adv}^{\mathsf{Game}_2}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{SE}}$.

• Game_4. We define the Game_4 which is same as Game_3 except that $\mathcal{C}$ uses a random string to change the nullifier nf created by PRF. By Lemma A.3, $|\mathbf{Adv}^{\mathsf{Game}_4} - \mathbf{Adv}^{\mathsf{Game}_3}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

• $\mathsf{Game}_{\mathsf{Sim}}$. $\mathsf{Game}_{\mathsf{Sim}}$ is identical to Game_4, except that $\mathcal{C}$ replaces commitments (e.g., $\mathsf{cm}_{\mathsf{old}}$, $\mathsf{cm}_{\mathsf{new}}$) computed by COM to an arbitrary string. By Lemma A.4, $|\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}} - \mathbf{Adv}^{\mathsf{Game}_4}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{COM}}$.

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:

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

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

Lemma A.1: Let $\mathbf{Adv}^{\Pi_{\mathsf{PE}}}$ be $\mathcal{A}$'s advantage in Π_{PE}'s IND-CPA and IK-CPA experiments. If $\mathcal{A}$'s zkTransfer query occurs $\mathsf{q}_{\mathsf{ZKT}}$, then $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq 2 \cdot \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

Proof. To prove that $\mathbf{Adv}^{\mathsf{Game}_H}$ is negligibly different from $\mathbf{Adv}^{\mathsf{Game}_1}$, 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 $\mathsf{M}_0, \mathsf{M}_1$ 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 $\mathcal{E}_{\mathsf{real}}$. We define another experiment $\mathcal{E}_{\mathsf{sim}}$ 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 $\mathcal{E}_{\mathsf{sim}}$ from $\mathcal{E}_{\mathsf{real}}$ but a negligible probability, due to the security of SE. The probability of $\mathcal{A}$ distinguishes the ciphertexts in $\mathcal{E}_{\mathsf{sim}}$ is 1/2; a ciphertext $\mathsf{pct}_{\mathsf{old}}$ from $\mathcal{E}_{\mathsf{sim}}$ 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 $\mathbf{Adv}^{\mathsf{Game}_H}$ is equal to $\mathbf{Adv}^{\mathsf{PE}}$, hence $|\mathbf{Adv}^{\mathsf{Game}_H} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

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 public key 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 $\mathsf{Tx}_{\mathsf{ZKT}}$ 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 $\mathbf{Adv}^{\mathsf{PE}}$, then we can say that $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_H}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

As a result, the sum of $\mathcal{A}$'s two advantages is $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq 2 \cdot \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

Lemma A.2: Let $\mathbf{Adv}^{\Pi_{\mathsf{SE}}}$ be $\mathcal{A}$'s advantage in Π_{SE}'s IND-CPA experiment. If $\mathcal{A}$'s zkTransfer query occurs $\mathsf{q}_{\mathsf{ZKT}}$ times, then $|\mathbf{Adv}^{\mathsf{Game}_3} - \mathbf{Adv}^{\mathsf{Game}_2}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{SE}}$.

Proof. To prove that $\mathbf{Adv}^{\mathsf{Game}_3}$ is negligibly different from $\mathbf{Adv}^{\mathsf{Game}_2}$, 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 $\mathsf{M}_0, \mathsf{M}_1$ 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 A.3: 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 A.4: 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 zkTransfer query, the challenger $\mathcal{C}$ substitutes the commitment $\mathsf{cm}_{\mathsf{new}} \leftarrow \mathsf{COM}(\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$ 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 Unlinkability: Let T be the transaction set that contains $\mathsf{Tx}_{\mathsf{ZKT}}$ produced by $\mathcal{O}^{\mathsf{Azeroth}}$ in response to $\mathcal{Q}(\mathsf{zkTransfer})$. $\mathcal{A}$ wins the TR-UN experiment on any occasion $\mathcal{A}$ outputs b' which is the same as b chosen by $\mathcal{O}^{\mathsf{Azeroth}}$. Suppose $\mathcal{A}$ is given a pair of transactions $(\mathsf{Tx}, \mathsf{Tx}')$ as follows.

$$\mathsf{Tx} = (\pi, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{pct}_{\mathsf{new}}, aux)$$
$$\mathsf{cm}_{\mathsf{old}} = \mathsf{COM}(\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}}; \mathsf{o}_{\mathsf{old}})$$
$$\mathsf{cm}_{\mathsf{new}} = \mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$$
$$\mathsf{pct}_{\mathsf{new}} = \mathsf{PE.Enc}_{pk_{\mathsf{enc}}^{\mathsf{recv}}, \mathsf{apk}}(\mathsf{o}_{\mathsf{new}} || \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} || \mathsf{addr}^{\mathsf{recv}})$$

$$\mathsf{Tx}' = (\overline{\pi}, \overline{\mathsf{addr}^{\mathsf{send}}}, \overline{\mathsf{cm}_{\mathsf{new}}}, \overline{\mathsf{pct}_{\mathsf{new}}}, \overline{aux})$$
$$\overline{\mathsf{cm}_{\mathsf{old}}} = \mathsf{COM}(\overline{\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}}, \overline{\mathsf{addr}^{\mathsf{send}}}; \overline{\mathsf{o}_{\mathsf{old}}})$$
$$\overline{\mathsf{cm}_{\mathsf{new}}} = \mathsf{COM}(\overline{\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}}, \overline{\mathsf{addr}^{\mathsf{recv}}}; \overline{\mathsf{o}_{\mathsf{new}}})$$
$$\overline{\mathsf{pct}_{\mathsf{new}}} = \mathsf{PE.Enc}_{\overline{pk_{\mathsf{enc}}^{\mathsf{recv}}}, \mathsf{apk}}(\overline{\mathsf{o}_{\mathsf{new}}} || \overline{\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}} || \overline{\mathsf{addr}^{\mathsf{recv}}})$$

We analyze the case in which $\mathcal{A}$ finds out whether the transaction recipients are the same or different. There are three chances for $\mathcal{A}$ to get knowledge of whether recipients of two transactions are the same.

(i) distinguish the address from zk-SNARK proofs $(\pi, \overline{\pi})$.
(ii) distinguish the address from ciphertexts $(\mathsf{pct}_{\mathsf{new}}, \overline{\mathsf{pct}_{\mathsf{new}}})$.
(iii) distinguish the address from generated commitments $(\mathsf{cm}_{\mathsf{new}}, \overline{\mathsf{cm}_{\mathsf{new}}})$.

For condition (i), $\mathcal{A}$ must distinguish $(\mathsf{addr}^{\mathsf{recv}}, \overline{\mathsf{addr}^{\mathsf{recv}}})$ from two different zk-SNARK proofs $(\pi, \overline{\pi})$, which imply that $\mathcal{A}$ should find a crack for *zero knowledge* property of the zk-SNARK. For condition (ii), if $\mathcal{A}$ can distinguish the address from ciphertexts $(\mathsf{pct}_{\mathsf{new}}, \overline{\mathsf{pct}_{\mathsf{new}}})$, it suggests that $\mathcal{A}$ wins the IND-CPA experiment in **Lemma** A.2 so meets contradiction. For condition (iii), if $\mathcal{A}$ can distinguish the address from the commitments without its opening key and inputs (e.g., $\mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}}$), it means that $\mathcal{A}$ breaks the $hiding$ property of the COM scheme in **Lemma** A.4. By the security of COM, zk-SNARK and IND-CPA, a PPT-adversary $\mathcal{A}$ cannot distinguish the recipient from the commitments, ciphertexts, and the zk-SNARK proof.

Remark. $\mathcal{A}$ can try extracting $\overline{\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}}$ from $\overline{\pi}, \overline{\mathsf{pct}_{\mathsf{new}}}, \overline{\mathsf{cm}_{\mathsf{new}}}$, and tracking the recipient. For example, if $\overline{\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}}$ is a specific value that is easy to distinguish from other values, $\mathcal{A}$ pulls out $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$ from later transactions and grasps the recipient. However, by the security of COM, zk-SNARK, and IND-CPA mentioned above, a PPT-adversary $\mathcal{A}$ cannot get such knowledge and distinguish the receiver.

3) Transaction Non-malleability: Let T be the transaction set that contains $\mathsf{Tx}_{\mathsf{ZKT}}$ produced by $\mathcal{O}^{\mathsf{Azeroth}}$ in response to $\mathcal{Q}(\mathsf{zkTransfer})$. $\mathcal{A}$ wins the TR-NM experiment whenever $\mathcal{A}$ outputs transaction Tx' which holds following conditions:

i) There is a transaction $\mathsf{Tx} \in \mathsf{T}$.
ii) $\mathsf{Tx}' \neq \mathsf{Tx}$
iii) $\mathsf{VerifyTx}(\mathsf{Tx}', \mathsf{L}') = 1$, where L' is the snapshot of the previous ledger state which not contains Tx yet.
iv) A nullifier nf appeared in Tx' is the same nullifier revealed in Tx.

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

$$\mathsf{Tx}' = (\pi, \mathsf{rt}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{EOA})$$

Define $\epsilon_1 := \mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$, and let $\mathcal{Q}_{\mathsf{RegisterUser}} = \{\mathsf{sk}_1, ..., \mathsf{sk}_{\mathsf{q_{KGU}}}\}$ be the set of internal address keys created by $\mathcal{C}$ in response to $\mathcal{A}$'s KeyGenUser queries. Now we argue that the probability of ϵ_1 is negligible in λ. For formal proof, we utilize zk-SNARK witness extractor denoted as $\mathcal{E}$ for $\mathcal{A}$. Since $\mathcal{A}$ wins only if Tx' is verified successfully, a proof π for Tx' has valid witnesses $(sk_{\mathsf{own}}, \mathsf{cm}_{\mathsf{old}})$ which satisfies $\mathsf{PRF}_{sk_{\mathsf{own}}}(\mathsf{cm}_{\mathsf{old}}) = \mathsf{nf}$. We use this fact to construct an algorithm $\mathcal{B}$ finding collision for PRF with non-negligible probability 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) Set $\vec{x} = (\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{pk}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}})$ and check whether $\vec{w}$ is a valid witness for $\vec{x}$ or not. If verification fails, then $\mathcal{B}$ aborts and outputs 0.
iv) Parse $\vec{w}$ as $(\mathsf{sk}, \mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{pk}^{\mathsf{recv}}, \mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{aux}, \mathsf{Path}_c,)$.
v) Find a transaction $\mathsf{Tx} \in \mathsf{T}$ that contains nf.
vi) If a transaction Tx is found, let $(sk'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}})$ be the corresponding witness in Tx attained from $\mathcal{E}$. If $sk_{\mathsf{own}} \neq sk'_{\mathsf{own}}$, then output $((sk_{\mathsf{own}}, \mathsf{cm}_{\mathsf{old}}), (sk'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}}))$. 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, $Pr[sk_{\mathsf{own}} = sk'_{\mathsf{own}}] = \frac{1}{2^l}$ where l is the bit length of sk_{own}. Thus, its probability is negl. Putting all the above probabilities together, we conclude that $\mathcal{B}$ finds a collision for PRF with probability $\epsilon_1 - \mathsf{negl}(\lambda)$.

4) Balance: In this section, we show 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.

Balanced ledger. We say that an augmented ledger L is $balanced$ if the following conditions hold.

- **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 $\mathsf{Tx}_{\mathsf{ZKT}}$ before depositing the value form it on L.
- **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 $\mathsf{v}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} = \mathsf{v}^{\mathsf{ENA}*}$ 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 $\mathsf{addr}^{\mathsf{send}}$ 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 infringing 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, it means that there is a violation of condition 1 if its commitment corresponding to nf does not exist in L. The meaning of the violation is equal to break the binding property of COM; Hence $\Pr[\mathcal{A}(\mathcal{C}_1) = 1]$ is negligible.

An infringing 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 transaction $(\mathsf{Tx}_{\mathsf{ZKT}}, \mathsf{Tx}_{\mathsf{ZKT}}{}')$ 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 infringing 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 $\mathsf{v}^{\mathsf{ENA}}$, v, and v^*_{ENA}, satisfying the following equation: $\mathsf{v}^{\mathsf{ENA}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} = \mathsf{v}^{\mathsf{ENA}^*}$. $\Pr[\mathcal{A}(\mathcal{C}_3) = 1]$ is a probability that its equation does not hold. However, this is a violation of the proof knowledge property of the zk-SNARK; It is negligible.

An infringing 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, and 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 zkTransfer transaction. However, since it is contradictory to the binding property of COM, it happens negligibly.

An infringing 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 zkTransfer transaction inserted by $\mathcal{A}$ issues $\mathsf{addr}^{\mathsf{recv}}$, the output of a previous 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 A.4 holds since it is at most negligible that the opposite happens, as mentioned above.

5) 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 there exists an extractor χ which 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 *proof of knowledge* property of the zk-SNARK. Consequently, since the $binding$ property of COM and the *proof of knowledge* property of zk-SNARK, the auditor with an authorized key (i.e., ask) can always observe the correct plaintext and surveil illegal acts in transactions.