Azeroth: Auditable Zero-knowledge Transactions in Smart Contracts

Gweonho Jeong
Hanyang University, Korea
kwonhojeong@hanyang.ac.kr

Nuri Lee
Kookmin University, Korea
nuri@kookmin.ac.kr

Jihye Kim
Kookmin University, Korea
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University, Korea
hoh@hanyang.ac.kr

Abstract

With the rapid growth of the blockchain market, privacy and security issues for digital assets are becoming more and more important. In the most widely used public blockchains such as Bitcoin and Ethereum, all activities on user accounts are publicly disclosed and also violate 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, i.e. auditability.

To solve the above problem, we propose an auditable zero-knowledge transfer framework called Azeroth. Azeroth connects a zero-knowledge proof for an encrypted transaction, enabling 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 top of an existing blockchain. According to the result of our experiment, the additional time required to generate a proof is about $901ms$.The security of Azeroth is formally proven under the cryptographic assumptions.

1 Introduction

With the widespread adoption of blockchains, various decentralized applications (DApps) and digital assets used in DApps are becoming popular. There are two classical models of blockchain networks: the unspent-transaction-output (UTXO) model and the account model. The former uses individual UTXOs to represent assets and conducts transactions by destroying and creating UTXOs on the network. The first cryptocurrency based on the UTXO model, Bitcoin [14], adopts a proof of work to implement the mining process. The latter performs transactions on a per-account basis and continuously updates the account status, very similar to the current customary global procedure. A representative example of this model is Ethereum [5], which produces smart contracts that allow users to execute functions on the network. Ethereum provides more programmability than Bitcoin model and supports more complex and diverse smart applications. 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 [7], ring signatures [19], homomorphic encryption [4], zero-knowledge proofs [3, 4, 12, 18], 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 [3], which operates on the UTXO model, secures transactions, while leveraging zero-knowledge proofs [10] to ensure transactions in a valid set of UTXOs. Zether [4] based on the account model encrypts accounts with homomorphic encryption and provides zero-knowledge proofs [6] to ensure valid modification of encrypted accounts. Zether builds up partial privacy for the linkability between sender and receiver addresses, similar to Monero [19]. In the case of Zeth [18] based on the account model, privacy is guaranteed by transferring assets in a form of commitments to accounts along with zero-knowledge proofs similar to the Zerocash approach.

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, it can be abused by criminals and terrorists for financial transactions. 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 [19], Dash [7], and Zerocash [3].

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

Moreover, many global cryptocurrency exchanges are also strengthening their distance from dark coin as recommended by Financial Action Task Force (FATF) [8]. Thus, we need to find a middle ground of contradiction between privacy preservation and fraudulent practices. This paper focuses on private transfer that provides an auditor with auditability, while protecting transaction privacy for 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 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) [10, 11, 16] 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; without a proof for encryption as in Zerocash [3] , even if a ciphertext passes all other transaction checks, there always exists a possibility that the wrongly generated ciphertext, either by mistake or intentionally can be accepted, resulting in the loss of the validity and auditability of the transaction.

In an account based blockchain, since the externally owned account exists in its structure, it is easy to determine whether the account has changed or not through the transaction records. For such reason, it should be carefully designed to maintain private transactions in the account based blockchain. In this paper, we employ dual accounts that constitute an encrypted account in addition to a plain account to resolve this difficulty, similar to Blockmaze [12]. Intuitively, the private transfer function between the sender and receiver performs on encrypted accounts. A legal receiver with a proper private key can decrypt the hidden value and deposit its amount in its owned encrypted account.

A private transfer scheme based on dual accounts needs to support basically deposit/withdrawal functions between plain/encrypted accounts and a transfer function between encrypted accounts. These functions can be either separately implemented, or integrated into one. In terms of anonymity of functions, the latter solution is desired; when devised in separate transactions as in Blockmaze [12], information about which transaction was performed is leaked breaching transaction anonymity, while it is hidden whether the user is making a deposit, a withdrawal, or a remittance in an integrated transaction. In this paper, we propose a smart contract to perform all functions internally in one transaction. Thus, in the proposed scheme, the same transaction is always executed regardless of the transaction type, so privacy protection is strengthened. Moreover, our proposal performs private transfer with fewer number of transactions than separately implemented transactions since the integrated function based on multi-input/output can flexibly support more transfer combinations between plain/encrypted accounts. For example, Blockmaze [12] requires at least 4 transactions for private transfer, but in the proposed scheme, 2 transactions are enough.[2]

In this paper, we propose an auditable zero-knowledge transaction framework called Azeroth based on zk-SNARK to tackle the above problems. The Azeroth framework provides privacy, verifiability, and auditability for personal digital assets while maintaining its 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 in various platforms.

The contributions of this paper are summarized as follows:

- Framework: We design a privacy-preserving framework Azeroth on 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 security properties of private transaction: 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 [5], Hyperledger Besu [13], and Klaytn [20]. 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 faster performance results than other existing schemes through implementation optimization. For the details, refer to section 6.

[2]Blockmaze [12] has four algorithms; Mint, Send, Deposit, and Redeem. If a user intends to transfer money anonymously into another user's account from the own account, it should proceed in the order of Mint, Send, Deposit, and Redeem. On the other hand, our scheme requires only two transactions. For the details refer to 5.1.

Organizations. The paper is comprised of the following contents: First, we provide the related works concerning our proposed scheme in section 2. In section 3, we describe preliminaries on our proposed system. In section 4, we give an explanation of data structures utilized in Azeroth. Afterward, we elucidate the overview and construction , algorithms, security definitions, and construction in section 5. Section 6 shows the implementation and the experimental results. Finally, we make a conclusion in section 7.

2 Related Work

Blockchains can be categorized as an unspent-transaction-output(UTXO) model and an account-based model. In the UTXO model, the inputs to a new transaction are the unspent outputs of previous transactions. The representative examples of UTXO model blockchain include Bitcoin [14], Zerocash [3], Monero [19], etc. In the meanwhile, the account-based model blockchain including Ethereum [5] and Klaytn [20] tracks the state of accounts. Every state of accounts is updated in the blockchain after the transaction execution. In the view of blockchain privacy, various schemes have been proposed in both approaches. Zero-knowledge proof based schemes, ring signature based schemes and mixer based schemes have been researched.

Privacy-preserving UTXO blockchains. Specifically, Zerocash is a well-known privacy-preserving blockchain system using ZKPs. 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. The proof proves that the input commitments are unspent ones, and new commitments are constructed well. Once the proof is verified, the receiver can use the transferred value. For accomplishing the same goal, Monero [19] utilizes CryptoNote, a protocol based on ring signatures [17], to cut off links between the sender and the recipient.

Privacy-preserving account-based blockchains. Zether [4] accomplishes the privacy-protection in the account-based model using ZKPs (Bulletproofs [6]) and the ElGamal encryption scheme. In Zether, when a sender wants to transfer some value to a receiver, the sender makes a ciphertext of the value with the receiver's public key. The ciphertext can be added to the receiver's state, and the sender proves the correctness of the ciphertext and whether the remaining value of the sender's account is positive or not. However, the sender should generate a zero-knowledge proof for the large user set for anonymity. Specifically, the sender generates dummy ciphertexts with a zero value except the receiver's ciphertext. Zeth [18] sorts the privacy problem out by implementing Zerocash into a smart contract. Thus, 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 and it does not support encryption checks like Zerocash. The recent work Blockmaze [12] 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-SNARKs 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 and does support auditability.

3 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 [10, 11], given a relation $\mathcal{R}$, a zk-SNARK is composed of a set of algorithms $\Pi_{\mathsf{snark}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerProof})$ 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.

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.

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

$$\approx$$

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

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

Symmetric-key encryption: We use a symmetric-key encryption scheme SE is a set of algorithms SE = (Gen, Enc, Dec) which operates as follows.

- $\mathsf{Gen}(1^\lambda) \rightarrow k$: The Gen algorithm takes a security parameter 1^λ and returns a key k.
- $\mathsf{Enc}_k(\mathsf{msg}) \rightarrow \mathsf{sct}$: The Enc algorithm takes a key k and a plaintext msg as inputs and returns a ciphertext sct.
- $\mathsf{Dec}_k(\mathsf{sct}) \rightarrow \mathsf{msg}$: 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 PE = (Gen, Enc, Dec) which operates as follows.

- $\mathsf{Gen}(1^\lambda) \rightarrow (sk, pk)$: The Gen algorithm takes a security parameter 1^λ and returns a key pair (sk, pk).
- $\mathsf{Enc}_{pk}(\mathsf{msg}) \rightarrow \mathsf{pct}$: The Enc algorithm takes a public key pk and a message msg as inputs and returns a ciphertext pct.
- $\mathsf{Dec}_{sk}(\mathsf{pct}) \rightarrow \mathsf{msg}$: 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 output aux as a SNARK witness as follows:

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

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

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

4 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). EOA is maintained by blockchain network and interacts with the smart contract. The latter is an account which includes a ciphertext indicating an amount in the account. 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 $(\mathsf{usk} = (k_{\mathsf{ENA}}, sk_{\mathsf{own}}, sk_{\mathsf{enc}}), \mathsf{upk} = (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}}))$.

- k_{ENA} : It indicates a secret key for encrypted account of ENA in a symmetric-key encryption system.
- $(sk_{\mathsf{own}}, pk_{\mathsf{own}})$: 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.
- $(sk_{\mathsf{enc}}, pk_{\mathsf{enc}})$: 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 as follows:

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

The commitment scheme takes v and 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. The commitment is published on a blockchain. A recipient with addr is given the opening key o and the value v from the encrypted transaction and uses them when the value amount needs to be transferred. 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. 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 addition, "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.

5 Azeroth

5.1 Overview

We construct Azeroth by integrating deposit/withdrawal transactions and public/private transfer transactions to a single transaction $\mathsf{zkTransfer}$. Since $\mathsf{zkTransfer}$ 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 $\mathsf{zkTransfer}$. $\mathsf{zkTransfer}$, 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.

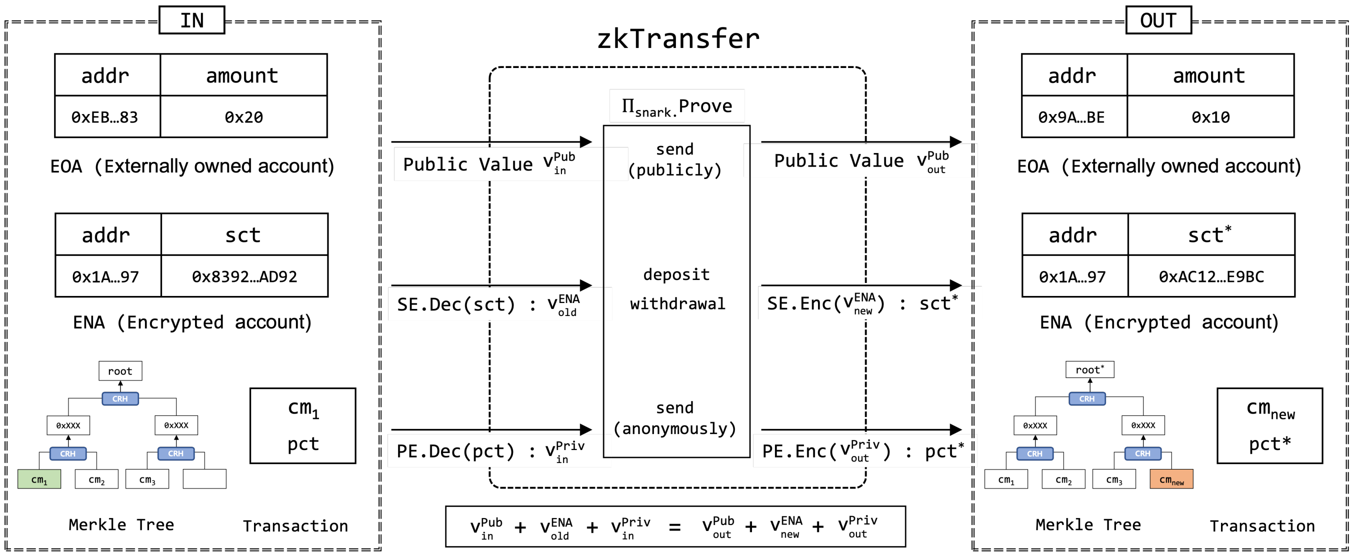

Figure 1: Overview of zkTransfer

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 from double spending, for each spent commitment a nullifier is generated by hashing the commitment and the private key sk_{own} and appended; it is still unlinkable between the commitment and the nullifier without the private key sk_{own}.

Auditability is achieved by utilizing a public key encryption with two recipients; all sct 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 a symmetric 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 $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$.

Finally, for the transaction validity, zkTransfer proves that all of the above procedures are correctly performed by generating a zk-SNARK proof.

5.2 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 algorithms of each component are as follows:

[Azeroth **Client]**

- $\mathsf{Setup}_{\mathsf{Client}}(1^\lambda, \mathcal{R}_{\mathsf{ZKT}}) \rightarrow \mathsf{pp}$: This algorithm is executed by a trusted party. It takes a security parameter λ and a relation $\mathcal{R}_{\mathsf{ZKT}}$ as input, and returns the public parameter pp, which are and published and available to all parties.

- $\mathsf{KeyGenAudit}_{\mathsf{Client}}(\mathsf{pp}) \rightarrow (\mathsf{ask}, \mathsf{apk}), \mathsf{Tx}_{\mathsf{KGA}}$: This algorithm takes a public parameter pp and outputs an auditor pair $(\mathsf{ask}, \mathsf{apk})$. It also outputs an transaction $\mathsf{Tx}_{\mathsf{KGA}}$ to register the auditor public key.

- $\mathsf{KeyGenUser}_{\mathsf{Client}}(\mathsf{pp}) \rightarrow (\mathsf{usk}, \mathsf{upk}), \mathsf{Tx}_{\mathsf{KGU}}$: This algorithm takes a public parameter pp and outputs a user key pair $(\mathsf{usk}, \mathsf{upk}) = ((k_{\mathsf{ENA}}, sk_{\mathsf{own}}, sk_{\mathsf{enc}}), (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}}))$. It also returns a transaction $\mathsf{Tx}_{\mathsf{KGU}}$ to register the user public key.

- $\mathsf{zkTransfer}_{\mathsf{Client}}(\mathsf{note}, \mathsf{apk}, \mathsf{usk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{EOA}^{\mathsf{recv}}) \rightarrow \mathsf{Tx}_{\mathsf{ZKT}}$: On inputs of a note, an auditor public key apk, a sender's key pair, a receiver's public key, public/private value amounts, and the receiver account $\mathsf{EOA}^{\mathsf{recv}}$ for $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$, this algorithm collects values from a sender's EOA and ENA, and a note (or commitment), and send them to a receiver's

EOA, and a commitment. The remaining balance is stored back to the sender's ENA. The internal procedures are described as follows:

i) Consuming note $= (\mathsf{cm}, \mathsf{o}, \mathsf{v})$: It proves the knowledge of the committed value v using the opening key o and the membership of a commitment cm in MT and derives a nullifier nf from PRF to nullify the used commitment.

ii) Generating $\mathsf{cm}_{\mathsf{new}}$: By executing $\mathsf{COM}(\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$, 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 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 $\mathsf{Tx}_{\mathsf{ZKT}} = (\pi, \mathsf{rt}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{EOA}^{\mathsf{recv}})$.

- $\mathsf{RetreiveNote}_{\mathsf{Client}}(\mathsf{L}, \mathsf{usk}, \mathsf{upk}) \rightarrow \mathsf{note}$: This algorithm is a sub-algorithm computing a note used in $\mathsf{zkTransfer}_{\mathsf{Client}}$. The key pair is parsed as $(\mathsf{usk}, \mathsf{upk}) = ((k_{\mathsf{ENA}}, sk_{\mathsf{own}}, sk_{\mathsf{enc}}), (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}}))$. This algorithm allows a user to find cm transferred to the user along with its opening key and its committed value. The algorithm decrypts using sk_{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.

[Azeroth **Smart Contract]**

- $\mathsf{Setup}_{\mathsf{SC}}(\mathsf{vk})$: 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.

- $\mathsf{RegisterAuditor}_{\mathsf{SC}}(\mathsf{apk})$: This algorithm stores an auditor public key apk in Azeroth's smart contract.

- $\mathsf{RegisterUser}_{\mathsf{SC}}(\mathsf{addr})$: 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.

- $\mathsf{zkTransfer}_{\mathsf{SC}}$ $(\pi, \mathsf{rt}_{\mathsf{old}}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{EOA}^{\mathsf{recv}})$: 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 $(\mathsf{addr}^{\mathsf{send}}, pk^{\mathsf{send}}_{\mathsf{own}}, pk^{\mathsf{send}}_{\mathsf{enc}})$ and a receiver's public key $\mathsf{upk}^{\mathsf{recv}}$ is $(\mathsf{addr}^{\mathsf{recv}}, pk^{\mathsf{recv}}_{\mathsf{own}}, pk^{\mathsf{recv}}_{\mathsf{enc}})$.

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

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

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

iii) The user address $\mathsf{addr}^{\mathsf{send}}$ and $\mathsf{addr}^{\mathsf{recv}}$ are well-formed.

iv) $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{cm}_{\mathsf{new}}$ are valid.

v) nf is derived from $\mathsf{cm}_{\mathsf{old}}$ and $sk^{\mathsf{send}}_{\mathsf{own}}$.

vi) $\mathsf{pct}_{\mathsf{new}}$ is an encryption of $\mathsf{cm}_{\mathsf{new}}$ via $\mathsf{aux}_{\mathsf{new}}$.

vii) $\mathsf{sct}_{\mathsf{new}}$ is an encryption of updated ENA balance.

viii) All amounts (e.g., $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \ldots$) are not negative.

5.3 Security

Following the similar model defined in [3, 12], we define the security properties of Azeroth including *ledger indistinguishability*, *transaction unlinkability*, *transaction non-malleability*, and *balance*, and define *auditability* as a new property.

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

Figure 2: The ledger indistinguishability experiment (L-IND)

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. 2. 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.

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

Figure 3: 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 describe an experiment TR-UN as shown in fig. 3 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 $\mathsf{Azeroth}.\mathcal{G}^{\mathsf{TR\text{-}UN}}_{\mathcal{A}}$. Azeroth satisfies the *transaction unlinkability* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{TR\text{-}UN}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda) - 1/2| \leq \mathsf{negl}(\lambda)$.

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 $\mathsf{Azeroth}.\mathcal{G}^{\mathsf{TR\text{-}NM}}_{\mathcal{A}}$ in fig. 4. 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 $|\mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

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

Figure 4: The transaction non-malleability experiment (TR-NM)

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

Figure 5: 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. 5. 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. 6. For a negligible function $\mathsf{negl}(\lambda)$, the Azeroth is *auditable* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{AUD}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

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

Figure 6: The auditability experiment (AUD)

5.4 Construction

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 7. The proofs for the following theorem appear in appendix.

Theorem 1. Let Π_{Azeroth} = (Setup, KeyGenAudit, KeyGenUser, zkTransfer) be a Azeroth scheme in Figure 7. Π_{Azeroth} satisfies *ledger indistinguishability*, *transaction unlinkability*, *transaction non-malleability*, *balance*, and *auditability*.

6 Implementation and Experiment

6.1 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 [3]. To generate a zk-SNARK proof in Azeroth, we use libsnark[4] with Groth16 [10] 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 [1] and Poseidon [9] as well as a standard hash function SHA256 [15].

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

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:

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

$\underline{\mathsf{SE.Dec}_k(r, \mathsf{sct}) \rightarrow \mathsf{msg}}$
$\mathsf{msg} \leftarrow \mathsf{sct} - \mathsf{PRF}_k(r)$
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 [2].) 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 encryption scheme. Given two key pairs (pk_1, sk_1) and (pk_2, sk_2) for ElGamal encryption, and a symmetric key encryption SE, the resulting implementation of the public key encryption is as follows:

```
$\underline{\mathsf{PE.Enc}_{pk_1,pk_2}(\mathsf{msg}) \rightarrow \mathsf{pct}, \mathsf{aux}}$
$k \xleftarrow{\$} \mathbb{F}; r \xleftarrow{\$} \mathbb{F}$
$c_0 \leftarrow G^r; c_1 \leftarrow k \cdot pk_1^r; c_2 \leftarrow k \cdot pk_2^r; c_3 \leftarrow \mathsf{SE.Enc}_k(\mathsf{msg})$
$\mathsf{pct} \leftarrow (c_0, c_1, c_2, c_3); \mathsf{aux} \leftarrow (k, r)$
Return $\mathsf{pct}, \mathsf{aux}$

$\underline{\mathsf{PE.Dec}_{sk_i}(\mathsf{pct}) \rightarrow \mathsf{msg}}$
$(c_0, c_1, c_2, c_3) \leftarrow \mathsf{pct}$
$k \leftarrow c_{i+1}/c_0^{sk_i}; \mathsf{msg} \leftarrow \mathsf{SE.Dec}_k(c_3)$
Return $\mathsf{msg}$
```

6.2 Experiment

In our experiment, the term $\mathsf{cfg}_{\mathsf{Hash},\mathsf{Depth}}$ denotes a configuration of Merkle hash tree depth and hash type in Azeroth. For instance, $\mathsf{cfg}_{\mathsf{MiMC7},32}$ means that we run Azeroth with MiMC7 [1] and its Merkle tree depth is 32. Table 1 illustrates our system environments. For the overall performance evaluation, we execute all experiments on the machine Server described in table 1 as a default machine. Without specification of the blockchain, the default blockchain is selected as the Ethereum testnet.

Overall performance. We show that the performance and the gas consumption in Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7},32}$ as shown in

Azeroth **Client**

```
○ $\mathsf{Setup}_{\mathsf{Client}}(1^\lambda, \mathcal{R}_{\mathsf{ZKT}})$ :
    Compute $(\mathsf{ek}, \mathsf{vk}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathcal{R}_{\mathsf{ZKT}})$
    Select a generator $G \xleftarrow{\$} \mathbb{G}$
    Set a public parameter $\mathsf{pp} = (\mathsf{ek}, \mathsf{vk}, G)$
    **Return** $\mathsf{pp}$

○ $\mathsf{KeyGenAudit}_{\mathsf{Client}}(\mathsf{pp})$:
    Set a new auditor key pair $(\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{PE.Gen}(\mathsf{pp})$
    Set $\mathsf{Tx}_{\mathsf{KGA}} = (\mathsf{apk})$
    **Return** $(\mathsf{apk}, \mathsf{ask})$, $\mathsf{Tx}_{\mathsf{KGA}}$

○ $\mathsf{KeyGenUser}_{\mathsf{Client}}(\mathsf{pp})$:
    Get a new user key pair $(sk_{\mathsf{enc}}, pk_{\mathsf{enc}}) \xleftarrow{\$} \mathsf{PE.Gen}(\mathsf{pp})$
    Select a random $k_{\mathsf{ENA}} \xleftarrow{\$} \mathsf{SE.Gen}(\mathsf{pp})$
    Select a random $sk_{\mathsf{own}} \xleftarrow{\$} \mathbb{F}$
    Compute a user binding key for ownership $pk_{\mathsf{own}} \leftarrow \mathsf{CRH}(sk_{\mathsf{own}})$
    Compute a user address $\mathsf{addr} \leftarrow \mathsf{CRH}(pk_{\mathsf{own}} || pk_{\mathsf{enc}})$
    Set a user secret key $\mathsf{usk} = (k_{\mathsf{ENA}}, sk_{\mathsf{own}}, sk_{\mathsf{enc}})$
    Set a user public key $\mathsf{upk} = (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}})$
    Set $\mathsf{Tx}_{\mathsf{KGU}} = (\mathsf{addr})$
    **Return** $(\mathsf{usk}, \mathsf{upk})$, $\mathsf{Tx}_{\mathsf{KGU}}$

○ $\mathsf{RetreiveNote}_{\mathsf{Client}}(\mathsf{L}, \mathsf{usk}, \mathsf{upk})$:
    Parse usk as $(k_{\mathsf{ENA}}, sk_{\mathsf{own}}, sk_{\mathsf{enc}})$
    Parse upk as $(\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}})$
    For each $\mathsf{Tx}_{\mathsf{ZKT}}$ on $\mathsf{L}$:
        Parse $\mathsf{Tx}_{\mathsf{ZKT}}$ as $(\mathsf{cm}, \mathsf{pct}, \dots)$
        Compute $(\mathsf{o}, \mathsf{v}, \mathsf{addr}^*) \leftarrow \mathsf{PE.Dec}_{sk_{\mathsf{enc}}}(\mathsf{pct})$
        **if** $\mathsf{addr} = \mathsf{addr}^*$ **then**
            **Return** $\mathsf{note} = (\mathsf{cm}, \mathsf{o}, \mathsf{v})$
        **end if**

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

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

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

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

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** $\mathsf{addr} \notin \mathsf{List}_{\mathsf{addr}}$
    $\mathsf{ENA}[\mathsf{addr}] \leftarrow 0$
○ $\mathsf{zkTransfer}_{\mathsf{SC}}$ $(\pi, \mathsf{rt}_{\mathsf{old}}, \mathsf{nf}, \mathsf{addr}^{\mathsf{send}}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{pct}_{\mathsf{new}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{EOA}_{\mathsf{recv}})$:
    **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** $\mathsf{cm}_{\mathsf{new}} \notin \mathsf{List}_{\mathsf{cm}}$

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

    **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** $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} > 0$ **then** $\mathsf{TransferFrom}(\mathsf{EOA}_{\mathsf{send}}, this, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}})$
    **end if**
    **if** $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} > 0$ **then** $\mathsf{TransferFrom}(this, \mathsf{EOA}_{\mathsf{recv}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}})$
    **end if**
```

Azeroth **Relation**

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

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

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

    Parse $\mathsf{usk}^{\mathsf{send}}$ as $(k^{\mathsf{send}}_{\mathsf{ENA}}, sk^{\mathsf{send}}_{\mathsf{own}}, sk^{\mathsf{send}}_{\mathsf{enc}})$
    Parse $\mathsf{upk}_{\mathsf{send}}$ as $(\mathsf{addr}^{\mathsf{send}}, pk^{\mathsf{send}}_{\mathsf{own}}, pk^{\mathsf{send}}_{\mathsf{enc}})$
    Parse $\mathsf{upk}_{\mathsf{recv}}$ as $(\mathsf{addr}^{\mathsf{recv}}, pk^{\mathsf{recv}}_{\mathsf{own}}, pk^{\mathsf{send}}_{\mathsf{enc}})$
    **if** $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} > 0$ **then**
        **Assert** $\mathsf{true} = \mathsf{Membership}_{\mathsf{MT}}(\mathsf{rt}, \mathsf{cm}_{\mathsf{old}}, \mathsf{Path})$
    **end if**
    **Assert** $pk^{\mathsf{send}}_{\mathsf{own}} = \mathsf{CRH}(sk^{\mathsf{send}}_{\mathsf{own}})$
    **Assert** $\mathsf{addr}^{\mathsf{send}} = \mathsf{CRH}(pk_{\mathsf{own}} || pk_{\mathsf{enc}})$
    **Assert** $\mathsf{cm}_{\mathsf{old}} = \mathsf{COM}(\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}, \mathsf{addr}^{\mathsf{send}}; \mathsf{o}_{\mathsf{old}})$
    **Assert** $\mathsf{nf} = \mathsf{PRF}_{sk^{\mathsf{send}}_{\mathsf{own}}}(\mathsf{cm}_{\mathsf{old}})$
    **Assert** $\mathsf{pct}_{\mathsf{new}}, \mathsf{aux}_{\mathsf{new}} = \mathsf{PE.Enc}_{pk^{\mathsf{recv}}_{\mathsf{enc}}, \mathsf{apk}}$ $(\mathsf{o}_{\mathsf{new}} || \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} || \mathsf{addr}^{\mathsf{recv}})$
    **Assert** $\mathsf{addr}^{\mathsf{recv}} = \mathsf{CRH}(pk^{\mathsf{recv}}_{\mathsf{own}} || pk^{\mathsf{recv}}_{\mathsf{enc}})$
    **Assert** $\mathsf{cm}_{\mathsf{new}} = \mathsf{COM}(\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$
    **if** $\mathsf{sct}_{\mathsf{old}} = 0$ **then** $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} \leftarrow 0$
    **else** $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} \leftarrow \mathsf{SE.Dec}_{k^{\mathsf{send}}_{\mathsf{ENA}}}(\mathsf{sct}_{\mathsf{old}})$
    **end if**
    **Assert** $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{new}} \leftarrow \mathsf{SE.Dec}_{k^{\mathsf{send}}_{\mathsf{ENA}}}(\mathsf{sct}_{\mathsf{new}})$
    **Assert** $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{new}} = \mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} - \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$
    **Assert** $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}} \geq 0$; $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} \geq 0$; $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}} \geq 0$; $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}} \geq 0$
    **Assert** $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{new}} \geq 0$; $\mathsf{v}^{\mathsf{ENA}}_{\mathsf{old}} \geq 0$
```

Figure 7: Azeroth scheme Π_{Azeroth}

Table 1: 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

table 2.[5] 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 available in section 6.2.

Table 2: Execution time and gas consumption of Azeroth with $\text{cfg}_{\text{MiMC7},32}$

	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

zk-SNARK performance. We evaluate the performance of zk-SNARK used to execute zkTransfer on various systems $\text{Server}, \text{System}_1, \cdots, \text{System}_4$ as described in table 1. Table 3 shows the setup time, the proving time, and the verification time respectively on each system with $\text{cfg}_{\text{MiMC7},32}$. Although System_3 has the lowest performance, still its proving time of 4.56s is practically acceptable.

Table 3: Execution time of zk-SNARK in zkTransfer

	Server	System_1	System_2	System_3	System_4
Π_{snark}.Setup (s)	2.311	4.19	4.13	8.529	5.529
Π_{snark}.Prove (s)	0.901	2.581	2.77	4.557	3.15
Π_{snark}.Verify (s)	0.057	0.041	0.079	0.062	0.054

Hash type and tree depth. We evaluate Azeroth performance depending on hash tree depths and hash types as shown in fig. 8, fig. 9, and fig. 10.

Figure 8 illustrates the execution time of zk-SNARK for MiMC7 [1], Poseidon [9][6], and SHA256 [15] 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 1.04s 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 fig. 8(d), which is similar to the gas consumption trend shown in fig. 8(e).

Figure 9 shows the key size. The proving key size is proportional to the tree depth, whereas the verification key size remains 1KB.[7] Poseidon has the smallest sizes of ek and constraints. Specifically, in depth 32, the ek sizes of 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. The circuit size of Poseidon is 30% smaller than MiMC7's.

Figure 10 shows that MiMC7 and Poseidon hashes consume relatively more gas than SHA256 since not only is SHA256 natively supported in Ethereum [5], but also it shows better native performance as shown in fig. 8(e).

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

Table 4: Number of constraints for circuits

	$\text{Membership}_{\text{MT}}(32)$	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

Performance analysis of smart contract. We analyze the execution time and the gas consumption in smart contract $\text{zkTransfer}_{\text{SC}}$. Table 5 shows the gas consumption for each function in smart contract $\text{zkTransfer}_{\text{SC}}$. 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 11 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.

Table 5: Gas consumption in smart contract $\text{zkTransfer}_{\text{SC}}$

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

[5] The result includes the execution time of our smart contract on the Ethereum testnet.

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

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

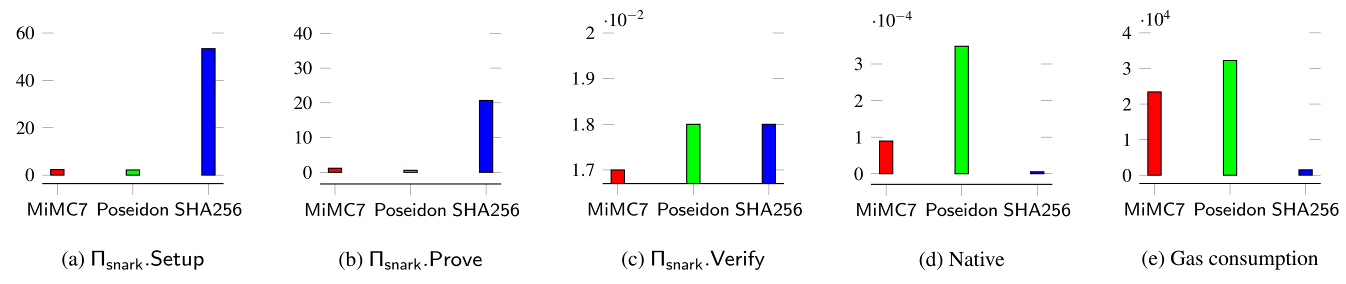

Figure 8: Performance with 32 hash tree depth. (a)-(c): The execution time of zk-SNARK's algorithms where the y axis is time(s). (d): The native execution time of each hash algorithm written in C++ where the *y* axis is time(s). (e): The gas consumption where the *y* axis denotes the gas consumption.

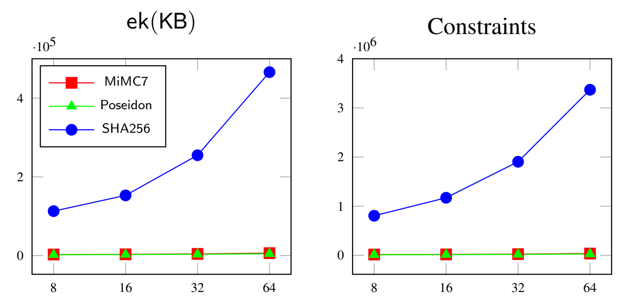

Figure 9: Key size and the number of constraints in circuits by varying hash tree depth and hash type

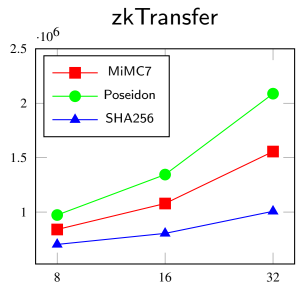

Figure 10: Gas consumption of zkTransfer according to hash tree depth and hash type

Deployment to various blockchains. We execute Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7},8}$ on various blockchains such as Ethereum [5], Hyperledger Besu [13], and Klaytn [20], in which the gas consumption is shown in table 6.

Comparison to other existing schemes. We compare the proposed scheme Azeroth with other privacy-preserving transfer schemes such as Zeth [18], and Blockmaze [12] in table 7. Our proposal shows even 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 config-

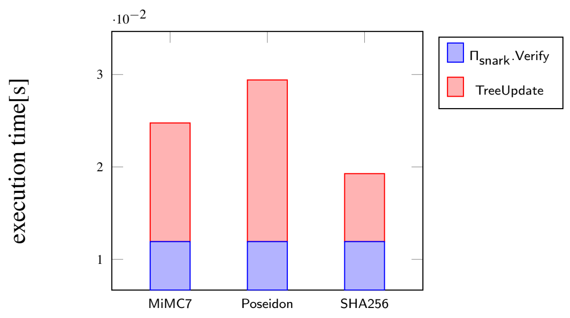

Figure 11: The execution time of functions in $\mathsf{zkTransfer}_{\mathsf{SC}}$

Table 6: 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

uration is applied to the proposed scheme for fair comparison. The experiment is conducted on Server. Note that the proof generation time in the table excludes the circuit loading time for fair comparison.

In comparison with Zeth[8], we utilize ganache-cli[9] as our test network. Note that since the open source code for Zeth generates a zk-SNARK proof in a docker container, it has low performance. Due to the circuit optimization of Azeroth, the resulting circuit size is 4x smaller and the size of pp is 22x smaller than Zeth. Azeroth has two registration functions (e.g., RegisterUser, RegisterAuditor) which correspond to **CreateAccount** of Zeth. In zkTransfer, Azeroth reduces the execution time by 90% compared with Zeth's Mix function.

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

Table 7: Comparison between our proposed scheme and existing work

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

Azeroth	Setup		RegisterAuditor			RegisterUser			zkTransfer	
	time	pp	time	apk	ask	time	upk	usk	time	$\mathsf{tx}_{\mathsf{size}}$
	2.846s	4.32MB	16ms	32B	32B	17ms	32B	32B	0.983s	1,186B

Zeth	10.646s	94.24MB	44ms	64B	64B	10.47s	1,380B
	time	pp	time	pk	sk	time	$\mathsf{tx}_{\mathsf{size}}$
	Setup		CreateAccount			Mix	

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

Azeroth	Setup		RegisterAuditor			RegisterUser			zkTransfer	
	time	pp	time	apk	ask	time	upk	usk	time	$\mathsf{tx}_{\mathsf{size}}$
	26.765s	113MB	16ms	32B	32B	17ms	32B	32B	10.0166s	1,186B

BlockMaze	125.063s	323MB	904ms	64B	32B	6.689s	817B	6.948s	815B	9.224s	899B	18.609s	815B
	time	pp	time	pk	sk	time	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$	time	$\mathsf{tx}_{\mathsf{size}}$
	Setup		CreateAccount			Mint		Redeem		Send		Deposit	

While BlockMaze[10] has four transactions of **Mint**, **Redeem**, **Send**, **Deposit**, Azeroth provides the equivalent functionality using a single transaction zkTransfer. Note that Blockmaze requires 20 seconds to load its proving key. Hence if it is considered, the performance improvement of Azeroth increases.

7 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 the auditing functionality which is required by an anti-money laundry law for an authorized auditor to trace transactions. 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.

Acknowledgements

This work is supported by the Institute of Information Communications Technology Planning Evaluation (IITP) funded by the Korean government (MSIT) (No.2021-0-00518, High performance blockchain privacy preserving techniques based on commitment, encryption, and zero-knowledge proofs, 50%, No.2021-0-00532, Blockchain scalability solutions supporting high performance/capacity transactions, 50%. Other support was provided by a grant from Klaytn foundation.

[10] https://github.com/Agzs/BlockMaze

References

[1] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT (1)*, pages 191–219. Springer, 2016.

[2] 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.

[3] 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, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[4] 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, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 423–443. Springer, 2020.

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

[6] 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.

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

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

[9] 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)*, pages 519–535. USENIX Association, August 2021.

[10] 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, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[11] 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, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

[12] 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*, pages 1–1, 2020.

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

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

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

[16] 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.

[17] Ronald L. Rivest, Adi Shamir, and Yael Tauman Kalai. How to leak a secret. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.

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

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

[20] Ground X. "klaytn position paper v2.1.0". https://www.klaytn.com/Klaytn_PositionPaper_V2.1.0.pdf.

A Security of Azeroth

In section 5.3, 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 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 $\mathcal{O}^{\mathsf{Azeroth}}$ 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 $(\mathsf{upk}, \mathsf{usk})$ using $\mathsf{KeyGenUser}$ algorithm, where $\mathsf{upk} = (\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}})$. $\mathcal{C}$ then registers public key to the smart contract and initializes $\mathsf{ENA}[\mathsf{addr}]$. The generated key pair and $\mathsf{ENA}[\mathsf{addr}]$ are stored into Acct and ENA, respectively. Finally $\mathcal{C}$ outputs upk.

- $\mathcal{Q}(\mathsf{zkTransfer}, \mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}, \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}, \mathsf{EOA})$: $\mathcal{C}$ generates a transaction $\mathsf{Tx}_{\mathsf{ZKT}}$ for corresponding inputs using $\mathsf{zkTransfer}$ 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.

A.1 Ledger Indistinguishability

We describe ledger indistinguishability using an experiment $\mathsf{L\text{-}IND}$ 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 $\mathbf{Adv}^{\mathsf{L\text{-}IND}}_{\mathcal{A}}$, the advantage of $\mathcal{A}$ in the $\mathsf{L\text{-}IND}$ 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: $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, 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 $\mathsf{KeyGenUser}$, 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 $\mathsf{zkTransfer}$, 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 $\mathsf{v^{ENA}}_{\mathsf{old}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$ is equal to $\mathsf{v^{ENA}}_{\mathsf{new}} + \mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$.
- The account address $\mathsf{addr}^{\mathsf{send}}$ should match that in ENA

We now give a formal definition of an experiment $\mathsf{L\text{-}IND}$ as follows:

i) Compute a public parameter pp, provide it to an adversary $\mathcal{A}$, and initialize two distinct oracles $\mathcal{O}^{\mathsf{Azeroth}}_0, \mathcal{O}^{\mathsf{Azeroth}}_1$.

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 $\mathcal{O}^{\mathsf{Azeroth}}_b, \mathcal{O}^{\mathsf{Azeroth}}_{1-b}$ respectively.

   b) Give a ledger tuple to $\mathcal{A}$ in each stage, and send $\mathcal{Q}$ to $\mathcal{O}^{\mathsf{Azeroth}}_b$ and $\mathcal{Q}'$ to $\mathcal{O}^{\mathsf{Azeroth}}_{1-b}$.

   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 $\mathsf{L\text{-}IND}$ returns 1; otherwise, 0.

Definition 1. Let Π_{Azeroth} = (Setup, $\mathsf{KeyGenAudit}$, $\mathsf{KeyGenUser}$, $\mathsf{zkTransfer}$) be a Azeroth scheme. We say that, for every $\mathcal{A}$ and adequate security parameter λ, Π_{Azeroth} is $\mathsf{L\text{-}IND}$ secure if the following equation holds:

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

A.2 Transaction Unlinkability

We formally describe the transaction unlinkability experiment $\mathsf{TR\text{-}UN}$ as follows.

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

ii) $\mathcal{A}$ makes a query ($\mathsf{zkTransfer}$) 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: $(\mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, aux)$, $(\overline{\mathsf{upk}^{\mathsf{send}}}, \overline{\mathsf{upk}^{\mathsf{recv}}}, \overline{aux})$, satisfied with the following conditions: (a) $\mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, \overline{\mathsf{upk}^{\mathsf{send}}}, \overline{\mathsf{upk}^{\mathsf{recv}}}$ 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 $\mathcal{Q}(\mathsf{upk}^{\mathsf{send}}, \mathsf{upk}^{\mathsf{recv}}, aux)$

v) if $b = 0$, make $\mathsf{Tx}_{\mathsf{ZKT}}{}'$ with query $\mathcal{Q}(\overline{\mathsf{upk}^{\mathsf{send}}}, \overline{\mathsf{upk}^{\mathsf{recv}}}, \overline{aux})$, else make $\mathsf{Tx}_{\mathsf{ZKT}}{}'$ with $\mathcal{Q}(\overline{\mathsf{upk}^{\mathsf{send}}}, \mathsf{upk}^{\mathsf{recv}}, \overline{aux})$.

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 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:

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

A.3 Transaction Non-malleability

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 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:

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

A.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) $\mathcal{C}$ calculates each value mentioned above, and checks if the following equation holds:

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

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

Definition 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:

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

A.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 $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{send}})$.

iv) If the tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, aux)$ satisfies with all conditions above, then the experiment AUD returns 1; otherwise, 0.

Definition 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 theorem 1 by showing that Azeroth construction satisfies ledger indistinguishability, transaction unlinkability, transaction non-malleability, balance, and auditability.

B.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 (ek, vk, 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 8: 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., $(\mathcal{Q},\mathcal{Q}') = \mathsf{KeyGenUser}$): $\mathcal{C}$ actions under the $\mathcal{Q}$(KeyGenUser) query, except that it does the following modifications: $\mathcal{C}$ generates user key pair (upk = $(\mathsf{addr}, pk_{\mathsf{own}}, pk_{\mathsf{enc}})$, usk) $\leftarrow$ KeyGenUser(pp), supersedes $pk_{\mathsf{own}}, pk_{\mathsf{enc}}$ to a random string of the appropriate length, and then computes the user address addr $\leftarrow \mathsf{CRH}(pk_{\mathsf{own}}, pk_{\mathsf{enc}})$. $\mathcal{C}$ also puts these elements in a table and returns upk to $\mathcal{A}$. $\mathcal{C}$ does the above procedure for $\mathcal{Q}'$.
- **Query**(zkTransfer,$\mathsf{upk}^{\mathsf{send}}$, $\mathsf{upk}^{\mathsf{recv}}$, $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$, $\mathsf{EOA}_{\mathsf{recv}}$) (i.e., $(\mathcal{Q},\mathcal{Q}') = \mathsf{zkTransfer}$): $\mathcal{C}$ actions under the $\mathcal{Q}$(zkTransfer) query, except that it does the following modifications: by default, we assume that $\mathsf{upk}^{\mathsf{send}}$ exists in the table. If $\mathsf{upk}^{\mathsf{send}}$ 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 $\mathsf{upk}^{\mathsf{recv}}$ 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 upk as a random string . The simulated (usk, upk) 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 $\mathsf{upk}^{\mathsf{send}}$ 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 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 $\mathsf{upk}^{\mathsf{send}}$ exists in the table, $\mathcal{C}$ computes $\mathsf{sct}_{\mathsf{new}}$ as r. Otherwise, $\mathcal{C}$ aborts. By Lemma 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 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 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 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 upk_0, whereas upk_1 is obtained from the KeyGenUser query. The IK-CPA challenger encrypts the same plaintext as pct^* using upk_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 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 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 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 $|\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}} - \mathbf{Adv}^{\mathsf{Game}_4}| \leq \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{COM}}$.

B.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:

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

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

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 $\mathsf{addr}^{\mathsf{recv}} = \overline{\mathsf{addr}^{\mathsf{recv}}}$:

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** 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** 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.

B.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 following:

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

Define $\varepsilon_1 := \mathbf{Adv}^{\mathsf{TR-NM}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$, and let $\mathcal{Q}_{\mathsf{RegisterUser}} = \{\mathsf{usk}_1, ..., \mathsf{usk}_{q_{\mathsf{KGU}}}\}$ 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 $\vec{x} = (\mathsf{apk}, \mathsf{rt}, \mathsf{nf}, \mathsf{upk}, \mathsf{cm}_{\mathsf{new}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}, \mathsf{pct}_{\mathsf{new}})$ 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 $(\mathsf{usk}, \mathsf{cm}_{\mathsf{old}}, \mathsf{o}_{\mathsf{old}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{upk}^{\mathsf{recv}}, \mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{aux}, \mathsf{Path}_c,)$.

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 $\varepsilon_1 - \mathsf{negl}(\lambda)$.

B.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 $\mathsf{v}^{\mathsf{ENA}} + \mathsf{v}^{\mathsf{priv}}_{\mathsf{in}} = \mathsf{v}^{\mathsf{ENA}*}$ where we denote an updating of the value as $*$.

- **Condition 4**: The values used to compute $\mathsf{cm}_{\mathsf{new}}$ are the same as the value for $\mathsf{cm}^{*}_{\mathsf{new}}$, if the value of $\mathsf{cm}_{\mathsf{old}}$ is equal to that of $\mathsf{cm}_{\mathsf{new}}$.

- **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 A's win probability of each case as $\Pr[\mathcal{A}(\mathcal{C}_1) = 1]$, which means 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 exists 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 its commitment 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 4 hold since it is at most negligible that the opposite happens, as mentioned above.

B.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.