Azeroth: Auditable Zero-knowledge Transactions in Smart Contracts

Gweonho Jeong[1], Nuri Lee[2], Jihye Kim[2,3], and Hyunok Oh[1,3]

[1] Hanyang University, Korea {kwonhojeong, hoh}@hanyang.ac.kr
[2] Kookmin University, Korea {nuri, jihyek}@kookmin.ac.kr
[3] Zkrypto, Korea

Abstract. With the rapid growth of the blockchain market, privacy and security issues for digital assets are becoming more important. In the most widely used public blockchains such as Bitcoin and Ethereum, all activities on user accounts are publicly disclosed, which violates privacy regulations such as EU GDPR. Encryption of accounts and transactions may protect privacy, but it also raises issues of validity and transparency: encrypted information alone cannot verify the validity of a transaction and makes it difficult to meet anti-money laundering regulations, i.e. auditability.

In this paper, we propose Azeroth, an auditable zero-knowledge transfer framework. Azeroth connects a zero-knowledge proof to an encrypted transaction, enabling it to check its validation while protecting its privacy. Our proposal also allows authorized auditors to audit transactions, and is designed as a smart contract for flexible deployment on existing blockchains. We implement the Azeroth whole framework, execute it on various platforms including an Ethereum testnet blockchain, and measure the time to show the practicality of our proposal. The end-to-end latency of a privacy-preserving transfer takes about $4.4s$. In particular, the client's transaction generation time with a proof only takes about $0.9s$. The security of Azeroth is proven under the cryptographic assumptions.

1 Introduction

With the widespread adoption of blockchains, various decentralized applications (DApps) and digital assets used in DApps are becoming popular. Unlike traditional banking systems, however, the blockchain creates privacy concerns about digital assets since all transaction information is shared across the network for strong data integrity. Various studies have been attempted to protect transaction privacy by utilizing cryptographic techniques such as mixers [2, 20, 21], ring signatures [26], homomorphic encryption [6], zero-knowledge proofs [4, 6, 17, 25], 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 [4], which operates on the UTXO model, secures transactions, while leveraging zero-knowledge proofs [15] to ensure transactions in a valid set of UTXOs. Zether [6] based on the account model encrypts accounts with homomorphic encryption and provides zero-knowledge proofs [8] to ensure valid modification of encrypted accounts. Zether builds up partial privacy for unlinkability between sender and receiver addresses, similar to Monero [26].

As asset transactions on the blockchain increase, the demand for adequate auditing capabilities is also increasing. Moreover, if transaction privacy is protected without proper regulation, financial transactions can be abused by criminals and terrorists. Without the management of illegal money flows, it would be also difficult to establish a monetary system required to maintain a sound financial system and enforce policies accordingly. Recently, Bittrex[4] delisted dark coins such as Monero [26], Dash [12], and Zerocash [4]. In fact, many global cryptocurrency exchanges are also strengthening their distance from the dark coins as recommended by Financial Action Task Force (FATF) [13]. Thus, we need to find a middle ground of contradiction between privacy preservation and fraudulent practices. This paper focuses on a privacy-preserving transfer that provides auditability for auditors while protecting transaction information from non-auditors.

Auditable private transfer can be designed by using encryption and the ZKP. A sender encrypts a transaction so that only its receiver and the auditor can decrypt it. At the same time, the sender should prove that the ciphertext satisfies all the requirements for the validity of the transaction. In particular, we utilize zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) [15, 16, 24] to prove *arbitrary* functionalities for messages, including encryption. Although the encryption check incurs non-negligible overhead for the prover, it is essential for the validity and auditability of the transaction. Indeed, without a proof for encryption as in Zerocash [4], even if a ciphertext passes all other transaction checks, there always exists a possibility that an incorrectly generated ciphertext, either accidentally or intentionally, will be accepted, resulting in the loss of the validity and auditability of the transaction.

There are two main privacy considerations in a transfer transaction: confidentiality (concealing a balance and a transfer amount) and anonymity (concealing a sender and a recipient). Confidentiality can be provided by utilizing encryption, while anonymity can be provided by utilizing an accumulator. Specifically, we employ dual accounts that constitute an encrypted account in addition to a plain account, similar to Blockmaze [17]. A user may execute deposit/withdrawal transactions between its own plain/encrypted accounts. At the same time, the user may send some encrypted value from its accounts to the accumulator (implemented as an Merkle tree). The owner of the encrypted value may receive it from the accumulator to the user's accounts.

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

In this paper, we propose Azeroth, an auditable zero-knowledge transaction framework based on zk-SNARK. The Azeroth framework provides privacy, verifiability, and auditability for personal digital assets while maintaining the efficiency of transactions. Azeroth preserves the original functionality of the account-based blockchain as well as providing the additional zero-knowledge feature to meet the standard privacy requirements. Azeroth is devised using encryption for two-recipients (i.e., the recipient and the auditor) so that the auditor can audit all transactions. Still, the auditor's capability is limited to auditing and cannot manipulate any transactions. Azeroth enhances the privacy of the transaction by performing multiple functions such as deposit, withdrawal, and transfer in one transaction. For the real-world use, we adopt a SNARK-friendly hash algorithm to instantiate encryption to have an efficient proving time, and execute experiments on various platforms.

The contributions of this paper are summarized as follows.

- Framework: We design a privacy-preserving transfer framework Azeroth on an account-based blockchain model, while including encryption verifiability, and auditability. Moreover, since Azeroth constructed as a smart contract does not require any modifications to the base-ledger, it advocates flexible deployment, which means that any blockchain models supporting smart-contract can utilize our framework.
- Security: We revise and extend the security properties of private transactions: ledger indistinguishability, transaction non-malleability, balance, and auditability, and prove that Azeroth satisfies all required properties under the security of underlying cryptographic primitives.
- Implementation: We implement and test Azeroth on the existing account-based blockchain models, such as Ethereum [7]. According to our experiment, it takes 4.38s to generate a transaction in a client and process it in a smart contract completely on the Ethereum testnet. While Azeroth additionally supports encryption verifiability and auditability, it shows better performance results than the other existing schemes through implementation optimization. To show the practicality of our scheme, we experiment the zk-SNARK performance on various devices including Android/iOS mobile phones. For the details, refer to section 6.

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 in section 5. Section 6 shows the implementation and the experimental results. Finally, we make a conclusion in section 7.

2 Related Work

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

A mixing service(or tumbler) such as CoinJoin [20], Möbius [21], and Tornado Cash [2] offers a service for mixing identifiable cryptocurrency transfer with others, so as to obscure the trail back to the transfer's original source. Thus, the mixer supports personal privacy protection for transactions on the blockchain. However, since the mixers take heed of anonymity, it exists the possibility of a malevolent problem.

Zerocash [4] provides a well-known privacy-preserving transfer on UTXO-model blockchains. In Zerocash, a sender makes new commitments that hide the information of the transaction (i.e., value, receiver) which is open only to the receiver. The sender then proves the correctness of the commitments using the zero-knowledge proof. As an extension to a smart contract, Zeth [25] sorts the privacy problem out by implementing Zerocash into a smart contract. Zeth creates the anonymous coin within the smart contract in the form of underlying the UTXO model. Thus, operations and mechanisms in Zeth are almost the same as Zerocash. ZEXE [5], extending Zerocash with scripting capabilities, supports function privacy that nobody knows which computation is executed in off-line. Hawk [19] is a privacy-preserving framework for building arbitrary smart contracts. However, there exists a manager, entrusted with the private data, that generates a zk-SNARK proof to show that the process is executed correctly.

Blockmaze [17] proposes a dual balance model for account-model blockchains, consisting of a public balance and private balance. For hiding the internal confidential information, they employ zk-SNARK when constructing the privacy-preserving transactions. Thus, it performs within the transformation between the public balance and the private balance to disconnect the linkage of users. Blockmaze is implemented by revising the blockchain core algorithm, restricting its deployment to other existing blockchains.

Zether [6] accomplishes the privacy protection in the account-based model using ZKPs (Bulletproofs [8]) and the ElGamal encryption scheme. While Zether is stateful, it does not hide the identities of parties involved in the transaction. Moreover, since the sender should generate a zero-knowledge proof for the large user set for anonymity, it has limitation to support a high level of anonymity. Diamond [11] proposes "many out of many proofs" to enhance proving time for a subset of a fixed list of commitments. Nevertheless, the anonymity corresponding to all system users is not supported.

Among the privacy-preserving payment systems, some approaches allow auditability. Solidus [9] is a privacy-preserving protocol for asset transfer in which banks play a role as an arbitrator of mediation. Solidus utilizes ORAM to support update accounts without revealing the values, but it cannot provide a dedicated audit function. Specifically, it can only offer auditing by revealing whole keys to an auditor, and opening transactions.

zkLedger [22] and FabZK [18] enable anonymous payments via the use of homomorphic commitments and NIZK while supporting auditability. However, since these systems are designed based on organizational units, there is the problem of performance degradation as the number of organizations increases. Thus,

they are practical only when there are a small number of organizations due to the performance issue.

PGC [10] aims for a middle ground between privacy and auditability and then proposes auditable decentralized confidential payment using an additively homomorphic public-key encryption. However, the anonymity set size should be small in the approach. The work [3] proposes a privacy-preserving auditable token management system using NIZK. However, the work is designed for enterprise networks on a permissioned blockchain. Moreover, whenever transferring a token, a user should contact the privileged party called by Certifier that checks if the transaction is valid.

3 Preliminaries

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

zk-SNARK: As described in [15,16], 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.

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

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.

$$\frac{\mathsf{Game}^{\mathsf{KS}}_{\mathcal{RG},\mathcal{A},\mathcal{E}} \rightarrow \mathsf{res}}{\begin{array}{l}(\mathcal{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda); (\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}), \mathsf{td}) \leftarrow \mathsf{Setup}(\mathcal{R}); \\ (x, \pi) \leftarrow \mathcal{A}(\mathcal{R}, \mathsf{aux}_R, \mathsf{crs}); w \leftarrow \mathcal{E}(\mathsf{transcript}_{\mathcal{A}}); \\ \textbf{Return } \mathsf{res} \leftarrow (\mathsf{VerProof}(\mathsf{vk}, x, \pi) \wedge (x, \pi) \notin \mathcal{R})\end{array}}$$

Zero knowledge. Simply, a zero-knowledge means that a proof π for $(x, w) \in \mathcal{R}$ on Π_{snark} only has information about the truth of the statement x. Formally, if there exists a simulator such that the following conditions hold for any adversary $\mathcal{A}$, we say that Π_{snark} is zero-knowledge.

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

$$\approx$$

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

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

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

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

The encryption scheme SE satisfies ciphertext indistinguishability under chosen plaintext attack $\mathsf{IND\text{-}CPA}$ security and key indistinguishability under chosen plaintext attack $\mathsf{IK\text{-}CPA}$ security.

Public-key encryption: A public-key encryption scheme $\mathsf{PE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which operates as follows.

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

The encryption scheme PE satisfies ciphertext indistinguishability under chosen plaintext attack $\mathsf{IND\text{-}CPA}$ security and key indistinguishability under chosen plaintext attack $\mathsf{IK\text{-}CPA}$ security.

Remark. To prove that encryption is performed correctly within a zk-SNARK circuit, we need the 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 zk-SNARK witness as follows:

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

4 Data Structures

This section describes the data structures used in our proposed scheme $\mathsf{Azeroth}$, 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 $\mathsf{Azeroth}$: an externally owned account denoted as EOA, and an encrypted account denoted as ENA. The former is the same one as in other account-based blockchains (e.g., Ethereum) and the latter is an account that includes a ciphertext indicating an amount in the account. $\mathsf{Azeroth}$ smart contract manages the registration, updates, and other interfaces of ENA, and users cannot see the value in hidden account without its secret key.

Auditor key. An auditor generates a pair of private/public keys $(\mathsf{ask}, \mathsf{apk})$ for public-key system; apk is utilized when a user generates an encrypted transaction, and ask is for auditing the ciphertext.

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

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

Commitment and Note. To build a privacy-preserving transaction, a commitment is utilized to hide the confidential information (i.e., amount, address). Our commitment is noted as follows.

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

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

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

Note that each user stores own notes to his wallet for 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. The tree 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. We define three algorithms related with MT.

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

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

- $\mathsf{v}^{\mathsf{ENA}}$: The digital asset value available in the encrypted account ENA.
- $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$ and $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$: The digital asset value to be publicly transferred from the sender's EOA and the value to the receiver's public account EOA, respectively.
- $\mathsf{v}^{\mathsf{priv}}_{\mathsf{in}}$ and $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$: 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 into a single transaction $\mathsf{zkTransfer}$. Since the

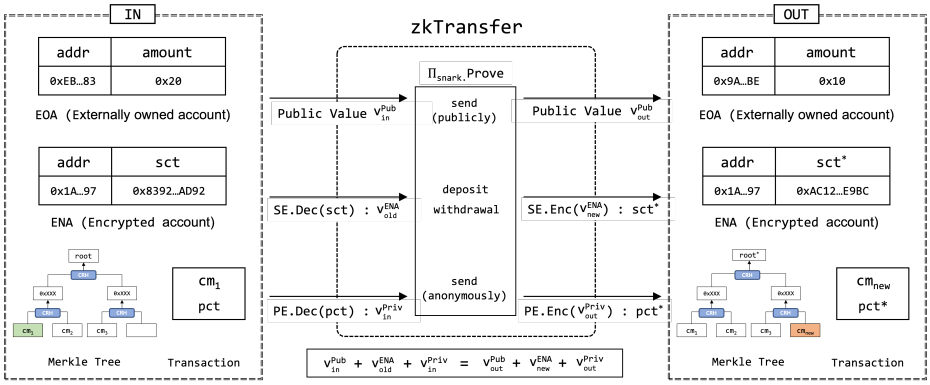


Fig. 1. Overview of zkTransfer

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

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

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

Auditability is achieved by utilizing public-key encryption with two recipients; all pct ciphertexts can be decrypted by an auditor as well as a receiver so that the auditor can monitor all the transactions. We note that ENA exploits symmetric-key encryption only for the performance gain although it 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 $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$.

5.2 Algorithms

Azeroth consists of three components: Client; Smart Contract; and Relation. A $\mathsf{Tx}_{\mathsf{ZKT}}$ including a proof is generated from Client. Smart Contract denotes a smart contract running on a blockchain. Relation represents a zk-SNARK circuit for a zkTransfer proof. See fig. 8 in Appendix for more details.

[Azeroth Client]

- $\mathsf{Setup}_{\mathsf{Client}}$: A trusted party runs this algorithm only once to set up the whole system. It also returns public parameter pp.
- $\mathsf{KeyGenAudit}_{\mathsf{Client}}$: This algorithm generates an auditor key pair $(\mathsf{ask}, \mathsf{apk})$. It also outputs the key pair and a transaction $\mathsf{Tx}_{\mathsf{KGA}}$.
- $\mathsf{KeyGenUser}_{\mathsf{Client}}$: This algorithm generates a user key pair $(\mathsf{usk}, \mathsf{upk})$. It also returns a transaction $\mathsf{Tx}_{\mathsf{KGU}}$ to register the user's public key.
- $\mathsf{zkTransfer}_{\mathsf{Client}}$: A user executes this algorithm for transfer. The internal procedures are described as follows:
  i) Spend $\mathsf{note} = (\mathsf{cm}, \mathsf{o}, \mathsf{v})$: It proves the knowledge of the committed value v using the opening key o and the membership of a commitment cm in MT and derives a nf from PRF to nullify the used commitment.
  ii) Generate $\mathsf{cm}_{\mathsf{new}}$: It gets a new commitment and its opening by executing $\mathsf{COM}(\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}}; \mathsf{o}_{\mathsf{new}})$. Then it encrypts $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}})$ as the message of PE.Enc then outputs pct.
  iii) Update ENA: It computes and updates the ENA balance on $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$ (from note), $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$, and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ or $\Delta v^{\mathsf{ENA}} = \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ - $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$ - $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$.
  To prove that above operations are correctly done, it generates a zk-SNARK proof then outputs $\mathsf{Tx}_{\mathsf{ZKT}}$.
- $\mathsf{RetreiveNote}_{\mathsf{Client}}$: This algorithm is a sub-algorithm computing a note used in $\mathsf{zkTransfer}_{\mathsf{Client}}$. It allows a user to find cm transferred to the user along with its opening key and its committed value. Then, a user decrypts the ciphertext using $\mathsf{sk}_{\mathsf{enc}}$ each transaction $\mathsf{pct} \in \mathsf{L}$ to $(\mathsf{o}, \mathsf{v}, \mathsf{addr}^*)$ and stores $(\mathsf{cm}, \mathsf{o}, \mathsf{v})$ as note in the user's wallet if addr^* matches its address addr.
- Audit: An auditor with a valid ask runs this algorithm to audit a transaction by decrypting the ciphertext pct in the transaction.

[Azeroth Smart Contract]

- $\mathsf{Setup}_{\mathsf{SC}}$: This simply 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.

- $\mathsf{RegisterAuditor}_{\mathsf{SC}}$: An auditor calls this function to register his public key apk.
- $\mathsf{RegisterUser}_{\mathsf{SC}}$: An user calls this function to register 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.
- $\mathsf{zkTransfer}_{\mathsf{SC}}$: To transfer, this checks the validity of the transaction beforehand. A transaction is valid iff: a merkle root $\mathsf{rt}_{\mathsf{old}}$ exists; a nullifier nf appears not yet; $\mathsf{addr}^{\mathsf{send}}$ is registered; $\mathsf{cm}_{\mathsf{new}}$ is fresh; and a proof π is valid. Once the validity check passes, it appends $\mathsf{cm}_{\mathsf{new}}$ to MT and update tree by $\mathsf{TreeUpdate}_{\mathsf{MT}}(\mathsf{cm})$. Then $\mathsf{rt}_{\mathsf{new}}$ and nf are added to each list $\mathsf{List}_{\mathsf{rt}}$, $\mathsf{List}_{\mathsf{nf}}$ respectively. A sender's encrypted account is updated to input $\mathsf{sct}_{\mathsf{new}}$, and finally handle the public values; gain $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ from $\mathsf{EOA}^{\mathsf{send}}$, and send $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ to $\mathsf{EOA}^{\mathsf{recv}}$.

[Azeroth Relation]

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

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

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

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

- If $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} > 0$, then $\mathsf{cm}_{\mathsf{old}}$ must exist in MT with given rt and Path.
- $\mathsf{pk}_{\mathsf{own}}^{\mathsf{send}} = \mathsf{CRH}(sk_{\mathsf{own}}^{\mathsf{send}})$.
- The user address $\mathsf{addr}^{\mathsf{send}}$ and $\mathsf{addr}^{\mathsf{recv}}$ are well-formed.
- $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{cm}_{\mathsf{new}}$ are valid.
- nf is derived from $\mathsf{cm}_{\mathsf{old}}$ and $\mathsf{sk}_{\mathsf{own}}^{\mathsf{send}}$.
- $\mathsf{pct}_{\mathsf{new}}$ is an valid encryption with $\mathsf{aux}_{\mathsf{new}}$.
- $\mathsf{sct}_{\mathsf{new}}$ is an valid encryption of updated ENA balance.
- All values (e.g., $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}, \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}, ...$) are not negative.

Given the building blocks of a public-key encryption PE, a symmetric-key encryption SE, and a zk-SNARK Π_{snark}, we construct $\mathsf{Azeroth}$ as in Figure 8.

6 Experiment

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

Table 1. Benchmark of Azeroth

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

(a) System specification

	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

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

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

(c) Execution time of zk-SNARK in zkTransfer

And the default blockchain is the Ethereum local testnet[5]. We utilize Gro16 [15] as our zk-SNARK.

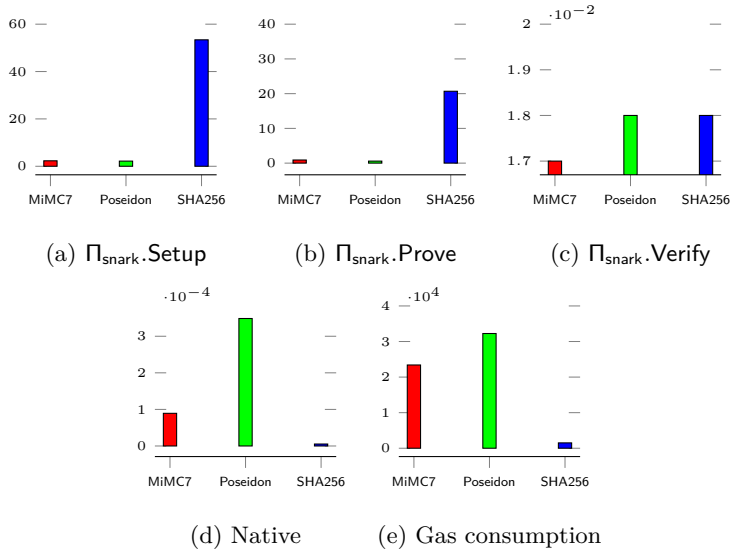
Overall performance. We show that the performance and the gas consumption in Azeroth with $\mathsf{cfg}_{\mathsf{MiMC7},32}$ as shown in Table 1 (b).[6] The execution time $4.04s$ of Setup is composed of the zk-SNARK key generation time $2.2s$ and the deployment time $1.84s$ of the Azeroth's smart contract to the blockchain. Setup consumes a considerable amount of gas due to the initialization of Merkle Tree. In zkTransfer, the executed time is $4.38s$ including both the Client part and the Smart Contract part. The gas is mainly consumed to verify the SNARK proof and update the Merkle hash tree. Varying the hash function, the further analysis of zkTransfer is described in the following experiments.

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

Hash type and tree depth. We evaluate Azeroth performance depending on hash tree depths and hash types as shown in Figure 2 and Figure 3.

[5] Ganache - Truffle Suite: https://trufflesuite.com/ganache

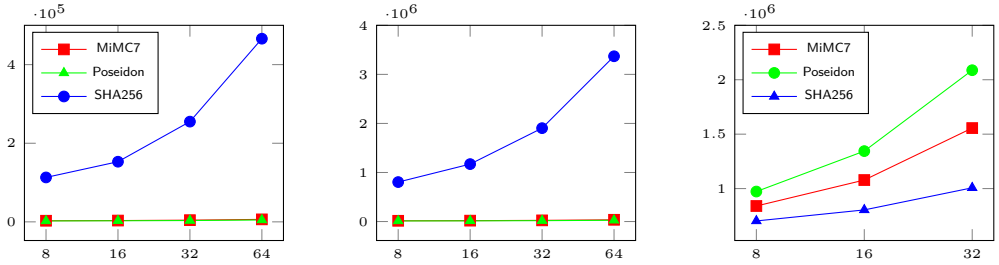
[6] The result includes the execution time until the task of receiving the receipt of the transaction.



(a) Π_{snark}.Setup (b) Π_{snark}.Prove (c) Π_{snark}.Verify

(d) Native (e) Gas consumption

Fig. 2. Performance with 32 hash tree depth. (a)-(c): The execution time of zk-SNARK's algorithms where the y axis is time(s). (d): The native execution time of each hash algorithm written in C++ where the y axis is time(s). (e): The gas consumption where the y axis denotes the gas consumption.



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

Fig. 3. Key size, constraints in circuits and gas consumption by varying hash tree depth and hash type

Figure 2 shows the zk-SNARK execution time for MiMC7 [1], Poseidon [14][7], and SHA256 [23] where the tree depth is fixed to 32. The SNARK key generation times are $2.311s$, $2.182s$, and $53.393s$ respectively. The proving times for MiMC7 and Poseidon are $0.901s$ and $0.582s$ respectively, while it takes 20.69 seconds with SHA256; SHA256 is about 20x and 40x slower than MiMC7 and Poseidon

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

in zk-SNARK. The verification time is almost independent of the hash type. However, when each hash function is executed natively, SHA256 shows the best performance as shown in Figure 2 (d), which is similar to the gas consumption trend shown in Figure 2 (e).

Figure 3 shows the key size. The proving key (ek) size is proportional to the tree depth, whereas the verification key size remains 1KB.[8] Poseidon has the smallest sizes of ek and constraints. Specifically, in depth 32, the ek sizes in Poseidon, MiMC7, and SHA256 are 3,341KB, 4,339KB, and 255,000KB respectively. The circuit size of SHA256 is enormous due to numerous bit operations, and Poseidon has 30% smaller size than MiMC7's. Figure 3 (c) shows that MiMC7 and Poseidon hashes consume relatively more gas than SHA256 since not only is SHA256 natively supported in Ethereum [7], but also it shows better native performance as shown in Figure 2 (e).

Comparison with the other existing schemes.

Table 2. Comparison between our proposed scheme and existing work

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

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

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

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

	$\mathsf{Azeroth}_{\mathsf{cfg}_{\mathsf{MiMC7},32}}$	Zether [6]	PGC [10]
gas cost	1,555,957	7,188,000	8,282,000
transaction size (bytes)	1,186	1,472	1,310

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

We compare the proposed scheme Azeroth with the other privacy-preserving transfer schemes such as Zeth [25], Blockmaze [17], Zether [6], and PGC [10] in Table 2. Our proposal shows better performance than the existing schemes,

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

even if Azeroth provides an additional function of auditability. Zeth and Blockmaze are implemented with $\mathsf{cfg}_{\mathsf{MiMC7},32}$ and $\mathsf{cfg}_{\mathsf{SHA256},8}$ respectively and the same configuration is applied to the proposed scheme for a fair comparison. The experiment is conducted on Server. Note that the proof generation time in the table excludes the circuit loading time for a fair comparison, and the loading time is significantly long in Blockmaze. On the other hand, PGC [10] and Zether [6] use standard ElGamal encryption and NIZK to provide confidentiality instead of utilizing Merkle Tree. The performance results in these works exclude anonymity, which means that the anonymity set size is 2. Note that the performance degrades as the anonymity set size increases in PGC and Zether, since the number of Elliptic curve operations in the smart contract increases proportionally to the anonymity set size.

In comparison with Zeth[9], we utilize ganache-cli[10] as our test network. Due to the circuit optimization of Azeroth, the resulting circuit size is 4x smaller and the size of pp is 22x smaller than Zeth. In zkTransfer, Azeroth reduces the execution time by 90% compared with Zeth's Mix function.

BlockMaze[11] has four transaction type: **Mint**, **Redeem**, **Send**, **Deposit**, Azeroth only provides a single transaction zkTransfer which serves the equivalent functionality of Blockmaze's. Note that it takes $20s$ to load the proving key in Blockmaze while it is only $1s$ in Azeroth. Hence Azeroth provides much better performance than Blockmaze in practice.

Zether [6] and PGC [10] are stateful[12] schemes using ElGamal encryption and NIZK. Due to the large difference in structure, the comparison experiment compares the gas cost and transaction size generated per transfer. Zether and PGC require 4.6 times and 5.3 times more gas than Azeroth respectively due to the Elliptic curve operations in the smart contract. In terms of transaction size, Azeroth generates a smaller transaction than Zether and PGC although Azeroth provides higher anonymity than them.

7 Conclusion

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

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

References

1. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: ASIACRYPT. pp. 191–219 (2016)
2. Alexey Pertsev, Roman Semenov, R.S.: Tornado cash privacy solution 1.4 (2019), https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf
3. Androulaki, E., Camenisch, J., Caro, A.D., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B.: Privacy-preserving auditable token payments in a permissioned blockchain system. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. p. 255–267. AFT '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3419614.3423259, https://doi.org/10.1145/3419614.3423259
4. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014)
5. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 947–964 (2020). https://doi.org/10.1109/SP40000.2020.00050
6. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) Financial Cryptography and Data Security - 24th International Conference. pp. 423–443 (2020)
7. Buterin, V.: Ethereum white paper: a next generation smart contract-decentralized application platform (2013)
8. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066 (2017), https://ia.cr/2017/1066
9. Cecchetti, E., Zhang, F., Ji, Y., Kosba, A., Juels, A., Shi, E.: Solidus: Confidential distributed ledger transactions via pvorm. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 701–717. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134010, https://doi.org/10.1145/3133956.3134010
10. Chen, Y., Ma, X., Tang, C., Au, M.H.: Pgc: Decentralized confidential payment system with auditability. In: Computer Security – ESORICS 2020. pp. 591–610 (2020)
11. Diamond, B.E.: Many-out-of-many proofs and applications to anonymous zether. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1800–1817 (2021)
12. Duffield, E., Diaz, D.: "dash: A privacycentric cryptocurrency. https://github.com/dashpay/dash/wiki/Whitepaper (2015)
13. FATF: "virtual assets and virtual asset service providers". https://www.fatf-gafi.org/media/fatf/documents/recommendations/Updated-Guidance-VA-VASP.pdf (2021)
14. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for Zero-Knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association (Aug 2021)
15. Groth, J.: 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. pp. 305–326 (2016)

16. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from Simulation-Extractable SNARKs. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference. pp. 581–612 (2017)
17. Guan, Z., Wan, Z., Yang, Y., Zhou, Y., Huang, B.: Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks. IEEE Transactions on Dependable and Secure Computing (2020)
18. Kang, H., Dai, T., Jean-Louis, N., Tao, S., Gu, X.: Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 543–555 (2019)
19. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
20. Maxwell, G.: Coinjoin: Bitcoin privacy for the real world (2013), `https://bitcointalk.org/index.php?topic=279249`
21. Meiklejohn, S., Mercer, R.: Möbius: Trustless tumbling for transaction privacy. Proceedings on Privacy Enhancing Technologies **2018**, 105–121 (2017)
22. Narula, N., Vasquez, W., Virza, M.: zkLedger: Privacy-Preserving auditing for distributed ledgers. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). pp. 65–80. USENIX Association, Renton, WA (Apr 2018), `https://www.usenix.org/conference/nsdi18/presentation/narula`
23. National Institute of Standards and Technology (NIST): Fips180-2: Secure hash standard (2002), `https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf`
24. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252 (2013). https://doi.org/10.1109/SP.2013.47
25. Rondelet, A., Zajac, M.: ZETH: on integrating zerocash on ethereum. CoRR **abs/1904.00905** (2019), `http://arxiv.org/abs/1904.00905`
26. Saberhagen, N.N.: Cryptonote v2.0. `https://cryptonote.org/whitepaper.pdf` (2013)

A Security of Azeroth

Following the similar model defined in [4], we define the security properties of Azeroth including *ledger indistinguishability*, *transaction non-malleability*, and *balance*; and a new property *auditability*.

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

Given a list of public parameters pp, the oracle $\mathcal{O}^{\mathsf{Azeroth}}$, and auditor public key apk is initialized and retains its state of which it has the elements internally : [I] L, a ledger; [II] Acct, a set of account key pairs; [III] NOTE, a set of notes. In the beginning, all of the elements are empty. Additionally, we denote $*$ as renewal. We now present how $\mathcal{O}^{\mathsf{Azeroth}}$ handles each type of query $\mathcal{Q}$ as follows.

- $\mathcal{Q}(\mathsf{KeyGenUser})$

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

- $\mathcal{Q}(\mathsf{RetreiveNote}, \mathsf{upk})$

i) Find usk in Acct. If no such key usk, then $\mathcal{O}^{\mathsf{Azeroth}}$ aborts.
ii) Parse usk as $(\mathsf{k_{ENA}}, \mathsf{sk_{own}}, \mathsf{sk_{enc}})$.
iii) Compute a set of note := $\mathsf{RetreiveNote}(\mathsf{L}, \mathsf{usk}, \mathsf{upk})$.
iv) Add each note to NOTE.
v) Output the set of note.

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

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

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

Public consistency. We now define *public consistency*. Two queries $(\mathcal{Q}, \mathcal{Q}')$ must be the same type and publicly consistent in $\mathcal{A}$'s viewpoint.

- If $(\mathcal{Q}, \mathcal{Q}')$ are both of type $\mathsf{KeyGenUser}$, then they are always publicly consistent. In special case of $\mathsf{KeyGenUser}$, the same key can be generated.
- If $(\mathcal{Q}, \mathcal{Q}')$ are both of type $\mathsf{RetreiveNote}$, then they are always publicly consistent.
- If $(\mathcal{Q}, \mathcal{Q}')$ are both of type $\mathsf{zkTransfer}$, then $\mathcal{Q}, \mathcal{Q}'$ must be well-formed respectively and jointly consistent with respect to public information and $\mathcal{A}$'s view as follows.
  a) note in $\mathcal{Q}$ and $\mathcal{Q}'$ must be same and appear in the ledger oracles' NOTE table.
  b) The notes in two queries are unspent, which means their serial number must not appear in a valid $\mathsf{Tx_{ZKT}}$ transaction on the corresponding oracle's ledger.

c) The sender addresses $\mathsf{addr}^{\mathsf{send}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must match the addresses of their note.
d) The balance equation must hold.

$$\mathsf{v}_{\mathsf{new}}^{\mathsf{ENA}} = \mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}} > 0$$

e) The public values $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$ and $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
f) The receiver's external addresses $\mathsf{EOA}^{\mathsf{recv}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
g) The transaction strings in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal.
h) If the recipient's public key $\mathsf{upk}^{\mathsf{recv}}$ in $\mathcal{Q}$ is not in Acct, then $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$ in $\mathcal{Q}$ and $\mathcal{Q}'$ must be equal(and vice versa for $\mathcal{Q}'$). The fact that $\mathsf{upk}^{\mathsf{recv}}$ is not in Acct is owned by $\mathcal{A}$, so the value $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$ must be set the same.
i) If any of note in $(\mathcal{Q}, \mathcal{Q}')$ was generated from an Insert query, both note in $(\mathcal{Q}, \mathcal{Q}')$ must have been generated from an Insert.

A.1 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 $\mathsf{Azeroth}$ 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 $\mathsf{L\text{-}IND}$ as shown in fig. 4.

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

Fig. 4. The ledger indistinguishability experiment ($\mathsf{L\text{-}IND}$)

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 $\mathsf{Azeroth}$ scheme. Intuitively, $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{L\text{-}IND}}$, the advantage of $\mathcal{A}$ in the $\mathsf{L\text{-}IND}$ experiment, is at most $\mathsf{negl}(\lambda)$. We now give a formal definition of an experiment $\mathsf{L\text{-}IND}$ that consists of an interaction between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

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

Definition 1. *Let* $\Pi_{\mathsf{Azeroth}} = (\mathsf{Setup}, \mathsf{KeyGenAudit}, \mathsf{KeyGenUser}, \mathsf{RetreiveNote}$ $\mathsf{zkTransfer})$ *be a* $\mathsf{Azeroth}$ *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:*

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

A.2 Transaction Non-malleability

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

Fig. 5. The transaction non-malleability experiment ($\mathsf{TR\text{-}NM}$)

Intuitively, a transaction is non-malleable if no transaction could be constructed with incorrect personal data (i.e., secret key). We define an experiment $\mathsf{TR\text{-}NM}$ with PPT adversary $\mathcal{A}$ trying to break a given $\mathsf{Azeroth}$ scheme. Note that $\mathcal{A}$ could be an auditor trying to attack our scheme with his private key ask. We describe the $\mathsf{TR\text{-}NM}$ experiment in detail as follows.

i) $\mathcal{C}$ computes a public parameter pp, provides it to an adversary $\mathcal{A}$, and initializes the oracle $\mathcal{O}^{\mathsf{Azeroth}}$.
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 $\mathcal{A}$ sends a transaction Tx', satisfied with the following conditions.
   a) There exists a Tx which satisfies:
   b) A nullifier in Tx' is the same as the Tx's.
   c) $\mathsf{VerifyTx}(\mathsf{Tx}', \mathsf{L}') = 1$, where L' denotes the snapshot of previous ledger state which does not contain Tx yet.

iv) If the transaction Tx' satisfying all conditions exists, then the experiment TR-NM returns 1; otherwise, 0.

Definition 2. *Let* $\Pi_{\mathsf{Azeroth}} = (\mathsf{Setup}, \mathsf{KeyGenAudit}, \mathsf{KeyGenUser}, \mathsf{RetreiveNote}, \mathsf{zkTransfer})$ *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.3 Balance

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

Fig. 6. The balance experiment (BAL). $\mathsf{List}_{\mathsf{note}}$ denotes a table of note, ENA denotes the new encrypted account balance, **Compute** is a function to compute variables related to $\mathcal{A}$'s account balance

We say that Azeroth has *balance* property if and only if attacker should not spend more than she has or receives. Let $\mathbf{Adv}_{\mathsf{Azeroth},\mathcal{A}}^{\mathsf{BAL}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game BAL as described in fig. 6.

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) $\mathcal{C}$ computes a public parameter pp, provides it to an adversary $\mathcal{A}$, and initializes the oracle $\mathcal{O}^{\mathsf{Azeroth}}$.
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 $\mathsf{List}_{\mathsf{note}}$ and ENA.
iv) $\mathcal{C}$ computes each value mentioned above using $\mathsf{Compute}(\mathsf{L}, \mathsf{List}_{\mathsf{note}}, \mathsf{ENA})$, 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 3. *Let* $\Pi_{\mathsf{Azeroth}} = (\mathsf{Setup}, \mathsf{KeyGenAudit}, \mathsf{KeyGenUser}, \mathsf{RetreiveNote}, \mathsf{zkTransfer})$ *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.4 Auditability

If the auditor can always monitor the confidential data of any user, we informally say that the scheme has *auditability*. More precisely, we define that Azeroth is auditable if there is no transaction in which the decrypted plaintext is different from the commitment openings. Let $\mathbf{Adv}^{\mathsf{AUD}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the game AUD as described in fig. 7. For a negligible function $\mathsf{negl}(\lambda)$, the Azeroth is *auditable* if for any PPT adversary $\mathcal{A}$, we have that $|\mathbf{Adv}^{\mathsf{AUD}}_{\mathsf{Azeroth},\mathcal{A}}(\lambda)| \leq \mathsf{negl}(\lambda)$.

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

Fig. 7. The auditability experiment (AUD). aux is defined as an auxiliary value not included in Tx. (s.t., o)

Let aux be the auxiliary input consisting of the committed value, its opening, and $\mathsf{addr}^{\mathsf{recv}}$, utilized when verifying the commitment. If the commitment is correct then the function VerifyCommit returns 1, otherwise returns 0. We now define precisely the experiment AUD as follows:

i) Compute a public parameter pp, provides it to an adversary $\mathcal{A}$, and intializes the oracle $\mathcal{O}^{\mathsf{Azeroth}}$.
ii) $\mathcal{A}$ requests a $\mathsf{zkTransfer}$ 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 decryption of $\mathsf{pct}_{\mathsf{new}}$ is not equal to $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}, \mathsf{addr}^{\mathsf{recv}})$.
iv) If the tuple $(\mathsf{Tx}_{\mathsf{ZKT}}, aux)$ satisfies with all conditions above, then the experiment AUD returns 1; otherwise, 0.

Definition 4. *Let* $\Pi_{\mathsf{Azeroth}} = (\mathsf{Setup}, \mathsf{KeyGenAudit}, \mathsf{KeyGenUser}, \mathsf{RetreiveNote}, \mathsf{zkTransfer})$ *be a* $\mathsf{Azeroth}$ *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 the $\mathsf{Azeroth}$ satisfies *ledger indistinguishability*, *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 $\mathsf{L\text{-}IND}$ experiment. However, $\mathsf{Game}_{\mathsf{Sim}}$ has a distinctness from the others since it runs regardless of a bit b where it means a chosen bit from the $\mathsf{L\text{-}IND}$ experiment. Thus, for $\mathsf{Game}_{\mathsf{Sim}}$, the advantage of $\mathcal{A}$ is 0. Moreover, the zk-SNARK keys are generated as $(\mathsf{ek}, \mathsf{vk}, \mathsf{td}) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Sim}(\mathcal{R})$ to obtain the zero-knowledge trapdoor td. We now show that $\mathbf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi_{\mathsf{Azeroth}},\mathcal{A}}$ is at most negligibly different than $\mathbf{Adv}^{\mathsf{Game}_{\mathsf{Sim}}}$. First of all, we define the notations as follows.

Table 3. Notations

Symbol	Meaning
$\mathsf{Game}_{\mathsf{Real}}$	The original $\mathsf{L\text{-}IND}$ experiment
Game_i	A hybrid game altered from $\mathsf{Game}_{\mathsf{Real}}$
$\mathsf{Game}_{\mathsf{Sim}}$	The fake $\mathsf{L\text{-}IND}$ experiment
$\mathsf{q}_{\mathsf{KGU}}$	The total number of $\mathsf{KeyGenUser}$ queries by $\mathcal{A}$
$\mathsf{q}_{\mathsf{ZKT}}$	The total number of $\mathsf{zkTransfer}$ 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 $\mathsf{IND\text{-}CPA}$
$\mathbf{Adv}^{\mathsf{COM}}$	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): $\mathcal{C}$ actions under the $\mathcal{Q}$(KeyGenUser) query, except that it does the following modifications: $\mathcal{C}$ generates a key pair $(\mathsf{upk}, \mathsf{usk})$ from $\mathsf{KeyGenUser}(\mathsf{pp})$, supersedes $\mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}}$ to a random string of the appropriate length, and then computes the user address $\mathsf{addr} \leftarrow \mathsf{CRH}(\mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}})$. $\mathcal{C}$ also puts these elements in a table and returns upk to $\mathcal{A}$. $\mathcal{C}$ does the above procedure for $\mathcal{Q}'$.
- **Query**(zkTransfer, note, $\mathsf{upk}^{\mathsf{send}}$, $\mathsf{upk}^{\mathsf{recv}}$, $\mathsf{v}^{\mathsf{priv}}_{\mathsf{out}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{in}}$, $\mathsf{v}^{\mathsf{pub}}_{\mathsf{out}}$, $\mathsf{EOA}^{\mathsf{recv}}$): $\mathcal{C}$ 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 $(\mathsf{usk}, \mathsf{upk})$ 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 $\mathsf{upk}^{\mathsf{send}}$ does not exist in the table, then $\mathcal{C}$ aborts. By Lemma 1, $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq 2 \cdot \mathsf{q}_{\mathsf{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

• 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, $\mathcal{C}$ replaces $\mathsf{sct}_{\mathsf{new}}$ 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 the 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:

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

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* $\mathsf{q_{ZKT}}$*, then* $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq 2 \cdot \mathsf{q_{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$*.*

Proof. We utilize a hybrid game Game_H as an intermediate between Game_1 and Game_2. First of all, to prove that $\mathbf{Adv}^{\mathsf{Game}_H}$ 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 pct 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 $|\mathbf{Adv}^{\mathsf{Game}_H} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq \mathsf{q_{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

Like the above, Game_2 is the same as Game_H except that it encrypts plaintext by setting the key to a new public key instead of the public key obtained by querying KeyGenUser. After querying KeyGenUser, $\mathcal{A}$ queries the IK-CPA challenger to gain pk_0, whereas pk_1 is obtained from the KeyGenUser query. The IK-CPA challenger encrypts the same plaintext as pct^* using pk_b where b is the bit selected by the IK-CPA challenger per zkTransfer query. The challenger sets pct in $\mathsf{Tx_{ZKT}}$ 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 $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_H}| \leq \mathsf{q_{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$. As a result, the sum of $\mathcal{A}$'s two advantages is $|\mathbf{Adv}^{\mathsf{Game}_2} - \mathbf{Adv}^{\mathsf{Game}_1}| \leq 2 \cdot \mathsf{q_{ZKT}} \cdot \mathbf{Adv}^{\mathsf{PE}}$.

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* $\mathsf{q_{ZKT}}$ *times, then* $|\mathbf{Adv}^{\mathsf{Game}_3} - \mathbf{Adv}^{\mathsf{Game}_2}| \leq \mathsf{q_{ZKT}} \cdot \mathbf{Adv}^{\mathsf{SE}}$*.*

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 $\mathsf{cm}_{\mathsf{new}}$ 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 Non-malleability

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

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

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

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

vi) Find a transaction $\mathsf{Tx} \in \mathsf{L}$ that contains nf.
vii) If Tx is found, let $(\mathsf{sk}'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}})$ be the corresponding witness to Tx attained from $\mathcal{E}$. If $\mathsf{sk}_{\mathsf{own}} \neq \mathsf{sk}'_{\mathsf{own}}$, then output $((\mathsf{sk}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{old}}), (\mathsf{sk}'_{\mathsf{own}}, \mathsf{cm}'_{\mathsf{old}}))$. Otherwise, output 0.

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

B.3 Balance

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

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

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

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

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

An infringing on condition 2. Each $(\mathsf{Tx_{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$, not inserted by $\mathcal{A}$, always satisfies condition 2; The probability $\Pr[\mathcal{A}(\mathcal{C}_2) = 1]$ is that there are two transaction $(\mathsf{Tx_{ZKT}}, \mathsf{Tx_{ZKT}}')$ 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 $(\mathsf{Tx_{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$, there exists a zk-SNARK proof, which can guarantee each of values $\mathsf{v}^{\mathsf{ENA}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{priv}}$, $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$, $\mathsf{v}_{\mathsf{in}}^{\mathsf{pub}}$, $\mathsf{v}_{\mathsf{out}}^{\mathsf{pub}}$, and $\mathsf{v}^{\mathsf{ENA}^*}$, satisfying the following equation: $\mathsf{v}_{\mathsf{old}}^{\mathsf{ENA}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{priv}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} + \mathsf{v}_{\mathsf{in}}^{\mathsf{pub}} - \mathsf{v}_{\mathsf{out}}^{\mathsf{pub}} = \mathsf{v}^{\mathsf{ENA}^*}$. $\Pr[\mathcal{A}(\mathcal{C}_3) = 1]$ 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 $(\mathsf{Tx_{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$ encompasses the values taken as the commitment (e.g., $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}$, $\mathsf{addr}^{\mathsf{recv}}$, and $\mathsf{o_{new}}$). $\Pr[\mathcal{A}(\mathcal{C}_4) = 1]$ is a probability that the commitments are equal, and all values related to commitment inputs in two transactions $(\mathsf{Tx_{ZKT}}, \mathsf{Tx_{ZKT}}^*)$ are equivalent except for the amount (i.e., $\mathsf{v}_{\mathsf{out}}^{\mathsf{priv}} \neq \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}^*}$) where $\mathsf{Tx_{ZKT}}^*$ a pre-existing $\mathsf{zkTransfer}$ transaction. However, since it is contradictory to the binding property of COM, it happens negligibly.

An infringing on condition 5. Each $(\mathsf{Tx_{ZKT}}, \vec{w}) \in (\mathsf{L}, \vec{W})$ publishes the recipient's address of a commitment $\mathsf{cm_{new}}$. If the $\mathsf{zkTransfer}$ transaction inserted by $\mathcal{A}$ issues $\mathsf{addr}^{\mathsf{recv}}$, the output of a previous $\mathsf{zkTransfer}$ transaction $\mathsf{Tx_{ZKT}}'$ 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 3 holds since it is at most negligible that the opposite happens, as mentioned above.

B.4 Auditability

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

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

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

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 $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}})$ in aux are not the same.

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

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 $(\mathsf{o}_{\mathsf{new}}, \mathsf{v}_{\mathsf{out}}^{\mathsf{priv}}, \mathsf{addr}^{\mathsf{recv}})$. Thus, $\mathcal{A}$'s winning should also break the *knowledge soundness* property of the zk-SNARK. Consequently, since the properties of COM and zk-SNARK, the auditor with an authorized key (i.e., ask) can always observe the correct plaintext and surveil illegal acts in transactions.

Azeroth **Client**

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

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

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

- $\mathsf{RetreiveNote}_{\mathsf{Client}}(\mathsf{L}, \mathsf{usk}, \mathsf{upk})$:
  $(\mathsf{k}_{\mathsf{ENA}}, \mathsf{sk}_{\mathsf{own}}, \mathsf{sk}_{\mathsf{enc}}) \Leftarrow \mathsf{usk}$
  $(\mathsf{addr}, \mathsf{pk}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{enc}}) \Leftarrow \mathsf{upk}$
  for $\mathsf{Tx}_{\mathsf{ZKT}} \in \mathsf{L}$ **do**
  $(\mathsf{cm}, \mathsf{pct}, \ldots) \Leftarrow \mathsf{Tx}_{\mathsf{ZKT}}$
  $(\mathsf{o}, \mathsf{v}, \mathsf{addr}^*) \leftarrow \mathsf{PE.Dec}_{\mathsf{sk}_{\mathsf{enc}}}(\mathsf{pct})$
  if $\mathsf{addr} = \mathsf{addr}^*$ **then**
  return $\mathsf{note} = (\mathsf{cm}, \mathsf{o}, \mathsf{v})$
  end if
  end for

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

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

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

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

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

Azeroth **Smart Contract**

- $\mathsf{Setup}_{\mathsf{SC}}(\mathsf{vk})$:
  $\mathsf{vk}_{\mathsf{SC}} \leftarrow \mathsf{vk}$
  $\mathsf{Init}_{\mathsf{MT}}$

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

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

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

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

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

Azeroth **Relation**

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

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

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

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

Fig. 8. Azeroth scheme Π_{Azeroth}