BlockMaze: An Efficient Privacy-Preserving Account-Model Blockchain Based on zk-SNARKs

Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang

***Abstract*—The disruptive blockchain technology is expected to have broad applications in many areas due to its advantages of transparency, fault tolerance, and decentralization, but the open nature of blockchain also introduces severe privacy issues. Since anyone can deduce private information about relevant accounts, different privacy-preserving techniques have been proposed for cryptocurrencies under the UTXO model, e.g., Zerocash and Monero. However, it is more challenging to protect privacy for account-model blockchains (e.g., Ethereum) since it is much easier to link accounts in the account-model blockchain. In this paper, we propose *BlockMaze*, an efficient privacy-preserving account-model blockchain based on zk-SNARKs. Along with dual-balance model, BlockMaze achieves strong privacy guarantees by hiding account balances, transaction amounts, and linkage between senders and recipients. Moreover, we provide formal security definitions and prove the security of *BlockMaze*. Finally, we implement a prototype of *BlockMaze* based on Libsnark and Go-Ethereum, and conduct extensive experiments to evaluate its performance. Our 300-node experiment results show that BlockMaze has high efficiency in computation and transaction throughput: one transaction verification takes about 13.8 ms, one transaction generation takes 4.6-18.2 seconds, and its throughput is around 20 TPS.**

***Index Terms*—Blockchain, Zero-Knowledge Proof, Account-Model, Privacy-Preserving, zk-SNARK**

I. Introduction

With the rise of cryptocurrencies, the blockchain has attracted tremendous interests worldwide, from IT industries, financial institutions, as well as academia. According to the statistics from BitInfoCharts [1], there are thousands of cryptocurrencies based on blockchain in the world. Among them, the market capitalization of Bitcoin [2] has exceeded \$180 billion and Ethereum [3] has exceeded \$29 billion. IT giants (e.g. Amazon, Alibaba, Google, and Facebook) and international financial institutions (e.g., JP Morgan and Goldman Sachs) also start to invest heavily on blockchains. In academia, researchers have started to investigate various issues of blockchain systems, e.g., consensus mechanisms, sharding mechanisms, and privacy protection.

From the perspective of balance representation, there are two popular models in blockchain networks: UTXO (Unspent Transaction Output) model and account model. The former is a directed graph of assets moving among users, in which the balance of a user is represented by all UTXOs related to that user in the blockchain system. The latter maintains a global state of all accounts and account balances that are updated whenever relevant transactions are executed. Bitcoin [2] is the first cryptocurrency based on the UTXO model, which is distributed through a proof of work “mining” process. Ethereum [3] adopts the account model with smart contract functionality to achieve Turing completeness. As shown in Table I, we briefly compare the advantages and disadvantages we have found with these two models. It shows that the primary advantages of the account model are superior programmability (i.e., smart contract) and fungibility of cryptocurrencies.

Both models achieve balance management for blockchains in different ways, and they both suffer from privacy leakage due to the openness of blockchains. Any adversary can obtain all transaction data, which contains too much privacy of the accounts. For example, the current transaction data of the Ethereum system is about 232 GB [1], which includes all transaction records from 30 July 2015 to 20 February 2020. By analyzing the transaction data in the blockchain [8], attackers can analyze the transaction relationship among different accounts. Since transactions are permanently recorded on the blockchain, which may cause an issue: once a historical transaction discloses the real identity of a user, the information of this user in all relevant transaction records will be revealed. Moreover, attackers can also use off-chain auxiliary information to infer the identity of accounts in the blockchain [9].

To address privacy issues in the blockchain, researchers have proposed several effective techniques in the literature, such as mixes, ring signatures, zero-knowledge proofs, and homomorphic encryption. For the UTXO model, there are existing projects such as Zerocoin [10], Zerocash [4], Monero [5], Dash [11], and CoinJoin [12] solving privacy issues. For the account model, however, there are very few proposals such as [13] providing limited privacy protection. Clearly, the account model is more user-friendly than the UTXO model, but it is also more challenging to protect privacy than the UTXO model. Each account is associated with a balance on the account-based blockchain, and it is updated whenever a relevant transaction is confirmed on the blockchain. Indeed, the balance of an account is the accumulated result of all transactions related to this account. In contrast, the UTXO-model blockchain does not calculate the accumulated balance for any account, and just stores the transactions on the blockchain. Hence the account model is closer to our current banking system and more user-friendly for users.

The above features of the account model make it more difficult to preserve privacy for the account-model blockchain.

Corresponding authors: Zhiguo Wan and Yang Yang.

Z. Guan, Z. Wan, and Y. Zhou are with the School of Computer Science and Technology, Shandong University, Qingdao, 266237, China (e-mail:{guanzs@mail.,wanzhiguo@,yanzhousdu@mail.}sdu.edu.cn).

Y. Yang is with the School of Mathematics and Computer Science, Fuzhou University, Fuzhou, 350116, China (e-mail:yang.yang.research@gmail.com).

B. Huang is with Hangzhou Yunphant Network Technology Co. Ltd., Hangzhou, 311121, China (e-mail: hbt@yunphant.com).

TABLE I: Comparison of balance models

Model	Typical projects	Smart contract	Advantages	Disadvantages
UTXO	Bitcoin [2] Zerocash [4] Monero [5]	No	Allow multi-threads for computations; support complete transparency of asset movements; process transactions easily in parallel.	Hard to work with smart contracts; increase the computational and storage burdens.
Account	Ethereum [3] Fabric [6] EOS [7]	Yes	Provide an intuitively clear approach of balances; give a simple implementation of smart contracts; achieve Turing-complete.	Need to store all accounts states; hard to track assets; analyze the states more easily.

In the account model, one needs to not only realize private fund transfer, but also update the accumulated balances for relevant accounts. In the UTXO model, however, one can generate many randomized addresses for his/her account (the actual wallet) without the trouble to accumulate them together. One straightforward way to enhance privacy is for a user to generate many random addresses, and use each of them for only once. However, holding a large number of addresses will be cumbersome for each user as well as smart contracts. Hence we aim to solve the privacy problem for blockchains where each user has only one address/account, which is much more challenging situation than the UTXO model.

Inspired by the design of the UTXO-model blockchain Zerocash, we design BlockMaze, a privacy-preserving account-model blockchain based on zk-SNARKs (zero-knowledge Succinct Non-interactive ARguments of Knowledge) [14]–[16]. To the best of our knowledge, BlockMaze is the first privacy-preserving account-model blockchain that protects both transaction amounts and sender/recipient relationship. More specifically, we propose a *dual-balance* model for account-model blockchains, which is composed of a plaintext balance and a zero-knowledge balance for each account. Along with the zero-knowledge balance, we employ zk-SNARKs to construct privacy-preserving transactions to hide transaction amounts and account balances. To disconnect the linkage between senders and recipients, we design a two-step fund transfer procedure based on the privacy-preserving transactions.

Contributions. In summary, the main contributions of this paper are as follows:

- We present *BlockMaze*, to the best of our knowledge, the first privacy-preserving account-model blockchain that hides both transaction amounts and the sender-recipient linkage. Using zk-SNARKs, we design a *dual-balance* model and a two-step fund transfer procedure for account-model blockchains.
- We provide a formal security model for *BlockMaze*, under which we prove its security. Moreover, we give a comprehensive discussion on practical issues regarding its compatibility and scalability.
- We implement a prototype of *BlockMaze* based on Libsnark [17] and Go-Ethereum [18], and conduct comprehensive experiments on a 300-node testbed to evaluate its performances. The results show that a transaction verification takes about 13.8 ms, a transaction generation takes 4.6-18.2 seconds, and its throughput is around 20 TPS.

A. Related Work

Privacy-preserving blockchains. Quite a few privacy-preserving cryptocurrencies have been proposed in the literature, and they allow a user to hide transaction amounts and/or obscure the linkage between a transaction and its sender and/or recipient. We now survey typical privacy-preserving cryptocurrencies and compare them in Table II. For UTXO-model blockchain systems, we can see that only Verge [20] supports hiding IP address using Tor and I2P [24], and others obscure the linkage between a transaction and the public wallet addresses of relevant parties. Zcash [4] offers privacy guarantees to hide the sender address, recipient address, and transaction amounts using zk-SNARK. Monero [5] achieves the same goal using CryptoNote, which is a protocol based on ring signatures. Grin [21] also solves it using MimbleWimble, which is based on elliptic curve cryptography and derived from confidential transactions based on Perdersen commitments.Moreover, Zcoin [10] utilizes Pedersen commitments and constructs corresponding zero-knowledge proofs to achieve unlinkability and untraceability. Dash [11] and CoinShuffle [19] employ CoinJoin, a method based on Chaum's mix idea, to obscure the linkage between a transaction and its sender and recipient.

Under the account model, DSC [13] provides an efficient NIZK scheme, utilizes homomorphic encryption to hide users' balance and transaction amount, and proves the validity of transactions with the NIZK scheme, but it does not hide the transaction senders and recipients. Moreover, compared with other solutions, the main issues with NIZK include limited functionalities and high computation cost. Zether [22] utilizes a special smart contract to realize privacy-preserving transactions based on homomorphic encryption and zero knowledge proofs. More specifically, it uses homomorphic encryption to hide fund amounts and its internal account balance, utilizes anonymous account sets to hide transaction senders and recipients, and ensures the validity of the transaction using zero-knowledge proofs. The Zether smart contract converts the underlying currency ETH (i.e., Ethereum coin) to new anonymous tokens (i.e., ZTH), and maintains multiple public tables to record the real-time status of its internal account. Anonymous fund transfers are carried out in the form of ZTH within the smart contract, and any internal account can covert its ZTH into ETH.

Similarly, ZETH [23] achieves privacy protection for Ethereum [18] by realizing Zerocash [4] with a smart contract. Just like Zerocash, ZETH creates anonymous coins under the UTXO model within the smart contract, while other operations of ZETH are equivalent to those in Zerocash. Those two

TABLE II: Comparison of privacy-preserving blockchain systems

Model	Cryptocurrencies	Privacy guarantees				Techniques	Pros/Cons
		IP/Sender/Recipient/Amount					
UTXO	Zcash [4]	×	√	√	√	zk-SNARK	Provide privacy protection of transaction amount and sender/recipient, but require a trust setup.
UTXO	Monero [5]	×	√	√	√	Ring Signature	Achieve unlinkability of transactions but with limited privacy protection and large transaction size.
UTXO	Zcoin [10]	×	√	√	×	ZKP	Provide privacy protection of sender/recipient but with large proof size and high verification latency.
UTXO	Dash [11]	×	√	√	×	Mix	Achieve untraceability of transactions, but mixing process is slow.
UTXO	CoinShuffle [19]	×	√	√	×	Mix	Allow users to utilize Bitcoin in a truly anonymous manner without any trusted third party.
UTXO	Verge [20]	√	×	×	×	Tor and I2P	Provide end-user identity obfuscation, but repeatedly suffer from 51% attacks.
UTXO	Grin [21]	×	√	√	√	MimbleWimble	Focus on privacy, fungibility, and scalability, but require a secure communication channel.
Account	DSC [13]	×	×	×	√	HE and ZKP	Provide an efficient NIZK scheme, but fail to disconnect the linkage between senders and recipients.
Account	Zether [22]	×	√	√	√	HE and ZKP	Achieve privacy protection using Zether smart contract without modifying underlying blockchain systems.
Account	ZETH [23]	×	√	√	√	zk-SNARK	Implement Zcash on Ethereum using extra smart contracts, but require complicated operations.

Remark ZKP denotes Zero-Knowledge Proof, HE denotes Homomorphic Encryption, and NIZK denotes Non-Interactive Zero-Knowledge.

proposals mentioned above achieve privacy protection based on smart contracts, their advantage is that they do not need to modify the underlying blockchain. However, compared with the approach of Zether/ZETH, it will be more convenient and efficient to achieve privacy at the underlying blockchain.

Privacy-preserving technologies. As shown in Table II, techniques considered for achieving privacy protection for blockchains and cryptocurrencies are summarized as follows:

- Mix: A specialized mix serves as the intermediary between multiple senders and recipients, hiding the relationship between senders and recipients (e.g., [11], [12], [19]), which is similar to Chaum's mix in the setting of untraceable emails.
- Ring signature: The genuine sender of a transaction can be hidden among several decoys using ring signature (e.g., [5]). Although miners can verify the signature, they cannot identify the genuine sender.
- Homomorphic encryption: Homomorphic encryption (mainly somewhat homomorphic encryption) is a potent tool that can be used in blockchains to preserve transaction amounts and users' balances (e.g., [13], [22], [23]). Currently, fully homomorphic encryption schemes cannot be used in blockchains for their low efficiency.
- Zero-knowledge proof: The aim of zero-knowledge proof technology is for the prover to convince the verifier that he/she does know a secret without disclosing it. There are several zero-knowledge proof systems that are employed frequently in the blockchain. A transaction sender can employ zero-knowledge proofs to prove that the transaction is valid and he/she is entitled to spend some cryptocurrencies without leaking identity information and transaction amounts (e.g., [4], [10], [13], [22], [23]).

Moreover, we refer the reader to [25] for other typical and detailed privacy-preserving mechanisms for blockchains.

Zero knowledge proof has been proved to be a promising solution for blockchains, and it develops very fast in recent years. The most popular zero-knowledge proof system deployed in blockchain is zk-SNARK, which has been successfully applied to preserve privacy for cryptocurrencies such as Zcash [4] and ZETH [23] due to its versatility. Concerning privacy issues in smart contracts, there have been several proposals [26], [27], which achieve privacy protection for smart contracts using zk-SNARKs. Many zk-SNARK constructions [14]–[16] have been proposed in past years. However, these traditional zk-SNARKs rely on a trusted setup, which also creates 'toxic waste' that, if leaked, can be utilized to generate undetectable fake proofs. Note that the setup of traditional zk-SNARKs is a one-time trusted setup for the specific circuit.

To solve above drawbacks, several new SNARKs [28]–[33] are presented in the literature. Among them, Fractal [29], Halo [30] and Supersonic [31] utilize transparent setups, which are public and create a Common Reference String (CRS) without toxic waste. However, these new constructions are still impractical for blockchain applications since their proof sizes (up to 250 KB for Fractal) are larger than traditional constructions. In contrast, Sonic [28], Marlin [32] and PLONK [33] utilize universal setups, which create a Structured Reference String (SRS) with toxic waste. But the reference string is a universal and updatable SRS, that is, it supports for an unlimited number of arbitrary circuits.

Moreover, Bulletproof [34] is an efficient zero knowledge proof and range proof without trusted setup. Monero [5] has used it as the range proof to reduce transaction fee costs and transaction sizes. Zether [22] has implemented a variant of Bulletproofs for ElGamal encryption to hide transaction amount along with its sender and recipient. But its prover's computation cost and verification cost are much more than zk-SNARKs. Compared with others, zk-STARK [35] does not need a trusted setup, and it is much more efficient in computation than zk-SNARK. However, zk-STARK in the current state is not suitable for blockchains because of its large proof size (a few hundred kilobytes). Overall, Groth16 [15] is still unbeatable in terms of proof size and runtime.

B. Organization.

The remainder of the paper is structured as follows: we first provide preliminaries on our proposal, including cryptographic building blocks in Section II. Then, we give an intuition of the idea, data structures used in the system, and an overview of the system architecture in Section III. After that, we describe and construct our BlockMaze scheme with security proof in detail in Section IV. Furthermore, we give a comprehensive discussion and analysis of BlockMaze in Section V. In Section VI, we describe details on the implementation of the prototype of BlockMaze, evaluate its performance, and compare it with Zerocash. Finally, Section VII concludes this paper.

II. Preliminaries

In this section, we recall zk-SNARKs and public key encryption used throughout this paper.

A. zk-SNARK

A zk-SNARK scheme [14]–[16] can be represented by a tuple of polynomial-time algorithms $\Pi_{\mathcal{Z}}$ = (Setup, KeyGen, GenProof, VerProof).

$\mathrm{Setup}(1^\lambda) \rightarrow \mathsf{pp}_{\mathcal{Z}}$. Given a security parameter λ, this algorithm generates a list of public parameters $\mathsf{pp}_{\mathcal{Z}} = (p, e, \mathbb{G}_1, \mathcal{P}_1, \mathbb{G}_2, \mathcal{P}_2, \mathbb{G}_T, \mathbb{F}_p)$, where p is a prime; e is a bilinear map: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$; $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are three cyclic groups of order p; $\mathcal{P}_1$ and $\mathcal{P}_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively; $\mathbb{F}_p$ is a finite field. All algorithms utilize $\mathsf{pp}_{\mathcal{Z}}$ as default input public parameters.

$\mathrm{KeyGen}(C) \rightarrow (\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}})$. Given a circuit C, this algorithm utilizes the public parameters $\mathsf{pp}_{\mathcal{Z}}$ to generate a key pair $(\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}})$. $\mathsf{pk}_{\mathcal{Z}}$ is a proving key for proof generation, while $\mathsf{vk}_{\mathcal{Z}}$ is a verification key for proof verification.

$\mathrm{GenProof}(\mathsf{pk}_{\mathcal{Z}}, \vec{x}, \vec{a}) \rightarrow \pi$. The algorithm generates a zero-knowledge proof π (or $\perp$ if GenProof fails). $\mathsf{pk}_{\mathcal{Z}}$ is a proving key, $\vec{x}$ is a public statement which is an input of circuit C, $\vec{a}$ is a private witness which is an auxiliary input of circuit C, and π is a zero-knowledge proof proving the relation constructed by circuit C between $\vec{x}$ and $\vec{a}$. Note that $\vec{x}$ and π are published and made available to anyone.

$\mathrm{VerProof}(\mathsf{vk}_{\mathcal{Z}}, \vec{x}, \pi) \rightarrow b$. According to this algorithm, anyone can check and verify a zero-knowledge proof. The algorithm outputs $b = 1$ if the check is successful; otherwise it outputs $b = 0$. $\mathsf{vk}_{\mathcal{Z}}$ is a verification key, π is a zero-knowledge proof generated in $\mathrm{GenProof}$, and $\vec{x}$ is public data used to generate π in $\mathrm{GenProof}$.

A zk-SNARK satisfies completeness, succinctness, proof of knowledge, and perfect zero-knowledge properties [36], [37]. Given a security parameter λ and any circuit C with a relation $\mathcal{R}_C$, the honest prover can utilize a proof π to convince the verifier for every pair $(\vec{x}, \vec{a}) \in \mathcal{R}_C$ where $\vec{x}$ is a statement and $\vec{a}$ is a witness.

Completeness. For each pair $(\vec{x}, \vec{a}) \in \mathcal{R}_C$, we have

$$\Pr\left[\mathrm{VerProof}(\mathsf{vk}_{\mathcal{Z}}, \vec{x}, \pi) \rightarrow 1 \,\middle|\, \begin{matrix} \mathrm{KeyGen}(C) \rightarrow (\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}) \\ \mathrm{GenProof}(\mathsf{pk}_{\mathcal{Z}}, \vec{x}, \vec{a}) \rightarrow \pi \end{matrix}\right] = 1.$$

Succinctness. An honestly-generated proof π has $O_\lambda(1)$ bits and $\mathrm{VerProof}(\mathsf{vk}_{\mathcal{Z}}, \vec{x}, \pi)$ runs in time $O_\lambda(|x|)$.

Proof of knowledge (and Soundness). For every probabilistic polynomial-time adversary $\mathcal{A}$, there is a probabilistic polynomial-time witness extractor $\mathcal{E}$, we have

$$\Pr\left[\begin{matrix} (\vec{x}, \vec{a}) \notin \mathcal{R}_C \\ \mathrm{VerProof}(\mathsf{vk}_{\mathcal{Z}}, \vec{x}, \pi) \rightarrow 1 \end{matrix} \,\middle|\, \begin{matrix} \mathrm{KeyGen}(C) \rightarrow (\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}) \\ \mathcal{A}(\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}) \rightarrow (\vec{x}, \pi) \\ \mathcal{E}(\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}) \rightarrow \vec{a} \end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

Perfect zero-knowledge. The proof is perfect zero-knowledge if there is a simulator $\mathcal{S}$ such that, for every pair $(\vec{x}, \vec{a}) \in \mathcal{R}_C$ and all non-uniform polynomial time adversaries $\mathcal{A}$, the following two probabilities are equal:

$$\Pr\left[\mathcal{A}(\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}, \pi) = 1 \,\middle|\, \begin{matrix} \mathrm{KeyGen}(C) \rightarrow (\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}) \\ \mathrm{GenProof}(\mathsf{pk}_{\mathcal{Z}}, \vec{x}, \vec{a}) \rightarrow \pi \end{matrix}\right],$$

$$\Pr\left[\mathcal{A}(\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}, \pi) = 1 \,\middle|\, \begin{matrix} \mathcal{S}(\mathcal{R}_C) \rightarrow (\mathsf{pk}_{\mathcal{Z}}, \mathsf{vk}_{\mathcal{Z}}, \mathsf{trap}) \\ \mathcal{S}(\mathsf{pk}_{\mathcal{Z}}, \vec{x}, \mathsf{trap}) \rightarrow \pi \end{matrix}\right].$$

B. Encryption

A public key encryption scheme can be represented by a tuple of polynomial-time algorithms $\Pi_{\mathcal{E}}$ = (Setup, KeyGen, Enc, Dec).

$\mathrm{Setup}(1^\lambda) \rightarrow \mathsf{pp}_{\mathcal{E}}$. On input a security parameter λ, this algorithm generates a list of public parameters $\mathsf{pp}_{\mathcal{E}}$.

$\mathrm{KeyGen}(\mathsf{pp}_{\mathcal{E}}) \rightarrow (sk_{\mathcal{E}}, pk_{\mathcal{E}})$. On input a list of public parameters $\mathsf{pp}_{\mathcal{E}}$, this algorithm generates a private/public key pair $(sk_{\mathcal{E}}, pk_{\mathcal{E}})$ for an account.

$\mathrm{Enc}_{pk_{\mathcal{E}}}(m) \rightarrow c$. Given a public key $pk_{\mathcal{E}}$, this algorithm encrypts an input plaintext m to output a ciphertext c.

$\mathrm{Dec}_{sk_{\mathcal{E}}}(c) \rightarrow m$. Given a private key $sk_{\mathcal{E}}$, this algorithm decrypts an input ciphertext c to get back the plaintext m.

For the sake of privacy, the pubic encryption scheme $\Pi_{\mathcal{E}}$ used in our scheme needs to satisfy the following two security properties: (i) *key indistinguishability under chosen-ciphertext attack* (IK-CCA security) [38], and (ii) *ciphertext indistinguishability under chosen-ciphertext attack* (IND-CCA security) [39].

III. System Architecture and Models

In this section, we firstly introduce an intuition of the idea, and then give data structures used in the system, and finally describe the system model to show how the proposed BlockMaze works.

A. Intuition of the idea

In the context of cryptocurrency, three types of privacy information need to be protected in the account-model blockchain: account balances, transaction amounts, and sender/recipient relationship. To preserve these types of privacy information, we present a *dual-balance* model that divides the balance of an account into two parts: a plaintext balance and a zero knowledge balance. Essentially, the zero-knowledge balance is associated with a commitment over the corresponding value, and it represents one part of the user's balance without disclosing the amount. The two types of balance can be converted to each other, and the zero-knowledge balance can be used to transfer fund to another account using zk-SNARKs.

However, the sender/recipient relationship cannot be hidden using the above approach. This problem does not exist in Zerocash [4] because Zerocash creates new accounts for each fund transfer transaction. Since the account model does not allow a user having multiple accounts, we design a two-step fund transfer procedure in BlockMaze. In the first step, the sender makes the fund transfer commitment with a `Send` transaction. To enforce the zero-knowledge balance update, we utilize its serial number to prevent double-spending issues based on the `Send` transaction. After the `Send` transaction is confirmed on the blockchain, the recipient collates its fund transfer commitment with other fund transfer commitments to form a Merkle tree. Then the recipient generates a zero-knowledge proof to receive the transferred fund without leaking from which transaction he/she receives the fund.

In summary, BlockMaze makes the following changes: (i) account balances and transaction amounts are hidden with a secure commitment scheme; (ii) the linkage between transactions is obscured by a two-step procedure: first, a sender computes a commitment on the transferred amount and generates a zero-knowledge transaction to transfer funds; then, the recipient recovers the transfer commitment from sender and generates a zero-knowledge transaction to deposit funds from the sender; (iii) zk-SNARK is employed to guarantee that a zero-knowledge transaction is valid and account balance can be updated legally without creating new money.

B. Data structures

Ledger. Given any time $\mathtt{T}$, all users can access $\mathtt{Ledger}_{\mathtt{T}}$, which is a sequence of blocks including all transactions. For convenience, we assume that $\mathtt{Ledger}_{\mathtt{T}}$ stores $\mathtt{Ledger}_{\mathtt{T}'}$ for all $\mathtt{T}^{'} \leq \mathtt{T}$. Each block in the ledger includes transactions and a new data set `TCMSet` described below. And the transactions in the ledger include both basic transactions as well as four zero-knowledge transactions. For convenience, we assume that $\mathtt{Ledger}_{\mathtt{T}}$ has stored $\mathtt{block}_{\mathtt{N}}$ before the given time $\mathtt{T}$.

Address key pair. There is an address key pair (sk, pk) instantiated for each account. sk is a private key for decrypting shared parameters and accessing private data, pk is a public key encrypting shared parameters, and $addr := \mathtt{CRH}(pk)$ ($\mathtt{CRH}$ is a collision-resistant hash function) is an account address sending and receiving payments.

Commitments. There are two types of commitments: balance commitments and fund transfer commitments.

A balance commitment is a commitment of the account balance as follows:

$$cmt_A := \mathtt{COMM}_{\mathtt{bc}}(addr_A, value_A, sn_A, r_A),$$

where $\mathtt{COMM}_{\mathtt{bc}}$ is a statistically-hiding non-interactive commitment scheme for account balance, cmt_A is the commitment of current account balance for user A, $addr_A$ is the account address of A, $value_A$ is the plaintext balance corresponding to cmt_A, sn_A is the serial number associated with cmt_A, and r_A is a random number masking sn_A. Moreover, $sn_A := \mathtt{PRF}(sk_A, r_A)$, where $\mathtt{PRF}$ is a pseudorandom function and sk_A is the private key of account A. Note that cmt_A is published and recorded on the blockchain as A's balance commitment.

A fund transfer commitment is a commitment of an amount being transferred between two parties as follows:

$$cmt_v := \mathtt{COMM}_{\mathtt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A),$$

where $\mathtt{COMM}_{\mathtt{tc}}$ is a statistically-hiding non-interactive commitment scheme for fund transfer, cmt_v is a fund transfer commitment from sender A to recipient B, $addr_A$ is the account address of sender A, v is the plaintext amount to be transferred, pk_B is the public key of recipient B, $sn_v := \mathtt{PRF}(sk_A, r_v)$ is the serial number associated with cmt_v, r_v is a random number masking cmt_v, and sn_A is the serial number associated with the balance commitment cmt_A of sender A.

Balance. Account-model blockchains adopt Merkle Patricia Tree (MPT) [3] to record the new account balance (e.g., Ethereum). Using MPT, we design a *dual-balance* model for account balances. There are two forms of account balance: one is a plaintext balance, denoted as `pt_balance`, which is public and made accessible to anyone; the other one is a zero-knowledge balance, denoted as `zk_balance`, which hides the corresponding plaintext value using balance commitments. For example, the balance of account A is as follows:

$$\mathtt{pt\_balance}_A := \text{the amount of plaintext balance,}$$
$$\mathtt{zk\_balance}_A := (cmt_A, addr_A, value_A, sn_A, r_A).$$

The asset of account A is the sum of both $\mathtt{pt\_balance}_A$ and $\mathtt{zk\_balance}_A.value_A$. Note that $\mathtt{zk\_balance}_A$ is only stored secretly by the owner of account A. $value_A$, sn_A and r_A are private data of account A, while $\mathtt{pt\_balance}_A$ and cmt_A are publicly recorded on MPT and made accessible to anyone. As shown in Fig. 1, the MPT realizes both fast search and authentication of an account balance. As each path from the root to a leaf node represents the account address, MPT records account balances at the corresponding leaf nodes for all existing accounts. One can not only quickly find the balance of a given account by searching MPT, but also authenticate the account balance by verifying whether the corresponding leaf node is on the MPT.

Our scheme realizes two-way balance transformation between plaintext balance and zero-knowledge balance. Although the transformed amount is public in the initial transformation transaction, the zero-knowledge balance is no longer visible after a fund transfer transaction.

Data Set. There are two new types of data set to store the published one-time variables. Since that $\mathtt{Ledger}_{\mathtt{T}}$ has stored

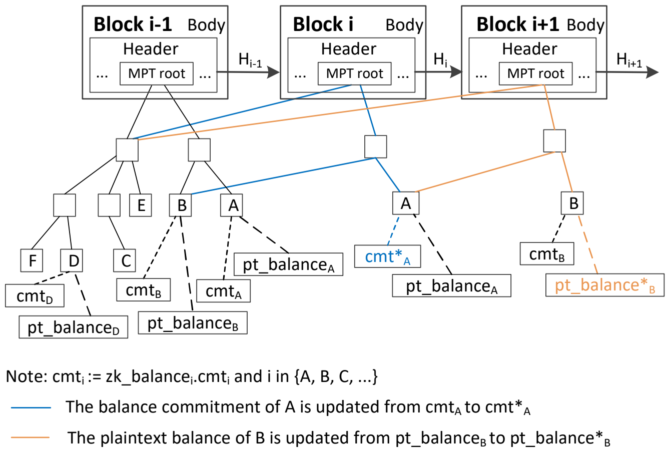


Fig. 1: Merkle Patricia Tree (MPT)

$\mathtt{block_N}$ before the given time $\mathtt{T}$, it is convenient to deduce these sets from $\mathtt{Ledger_T}$.

- $\mathtt{TCMSet}$. For any given block number $\mathtt{N}$, $\mathtt{TCMSet_N}$ denotes the set of all transfer commitments cmt_v appearing in transactions in $\mathtt{block_N}$.
- $\mathtt{SNSet}$. For any given time $\mathtt{T}$, $\mathtt{SNSet_T}$ denotes the set of all serial numbers sn_A and sn_v appearing in transactions in $\mathtt{Ledger_T}$, and these serial numbers are allowed to be used only once.

Merkle tree. Given any time $\mathtt{T}$, a user selects a set of block sequence numbers $\mathtt{SEQ}$ corresponding to randomly selected blocks. Each block contains one or more fund transfer commitments cmt_v. Based on the block whose number belongs to $\mathtt{SEQ}$, all $\mathtt{TCMSet}$ of these blocks are set as leaf nodes t form a Merkle tree. We utilize $\mathtt{MT_T}$ to denote a Merkle tree over $\bigcup_{n \in \mathtt{SEQ}} \mathtt{TCMSet_n}$ and $rt_\mathtt{T}$ to denote its root. Furthermore, $\mathtt{Path_T}(cmt_v)$ is an authentication path from the specified fund transfer commitment cmt_v appearing in $\mathtt{TCMSet_N}$ to $rt_\mathtt{T}$ at any given time $\mathtt{T}$.

On-Chain transactions. Apart from basic transactions, BlockMaze defines four new types of zero-knowledge transactions using zk-SNARK as below. More details about the format of these transactions are described in Section IV-C.

- $\mathtt{Mint}$. This transaction $\mathtt{tx_{Mint}}$ converts a plaintext amount into a zero-knowledge amount and merges the zero-knowledge amount into the current zero-knowledge balance of an account.
- $\mathtt{Redeem}$. This transaction $\mathtt{tx_{Redeem}}$ converts a zero-knowledge amount back into a plaintext amount and merges the plaintext amount into the current plaintext balance of an account.
- $\mathtt{Send}$. This transaction $\mathtt{tx_{Send}}$ is built to send a zero-knowledge amount from a sender to a recipient. And the transaction $\mathtt{tx_{Send}}$ hides the recipient's address and the transaction amount.
- $\mathtt{Deposit}$. This transaction $\mathtt{tx_{Deposit}}$ allows a recipient to deposit a received payment into his/her account.

C. System Model

Definition 1 (Account-Model Blockchain)**.** An account-model blockchain stores and maintains an append-only ledger $\mathtt{LEDGER}$, an account list $\mathtt{ACCOUNT}$ and a stateful tree $\mathtt{BALANCE}$ as below:

$$\mathtt{LEDGER} \overset{\text{def}}{=} List[\mathtt{transaction}]$$
$$\mathtt{ACCOUNT} \overset{\text{def}}{=} List[\mathtt{user} \Rightarrow \mathtt{address}]$$
$$\mathtt{BALANCE} \overset{\text{def}}{=} List[\mathtt{address} \Rightarrow \mathtt{balance}]$$

Given an account-based transaction tx, we can parse it as $tx \overset{\text{def}}{=} \{sender, recipient, value, *\}$ where $sender$ and $recipient$ are the account addresses, $value$ is the amount to be transferred and $*$ denotes other useful fields. Note that each user has only one account address. If tx is a valid transaction, $tx.value$ is deducted from the balance of $tx.sender$ while $tx.value$ is added to the balance of $tx.recipient$.

Given any time $\mathtt{T}$ and a transaction list $\mathtt{TxList}$ containing all transactions which are confirmed during $(\mathtt{T}-1, \mathtt{T}]$, for each $tx \in \mathtt{TxList}$ and $(u, addr)$ in tx, the above data structures change as follows:

$$\mathtt{LEDGER_T} = \mathtt{LEDGER_{T-1}} \cup \mathtt{TxList}$$
$$\mathtt{ACCOUNT_T} = \mathtt{ACCOUNT_{T-1}}.updateAddress(u \Rightarrow addr)$$
$$\mathtt{BALANCE_T} = \mathtt{BALANCE_{T-1}}.updateBalance(addr \Rightarrow (bal \pm tx.value))$$

Definition 2 (Privacy-preserving Account-Model Blockchain)**.** A privacy-preserving account-model blockchain stores and maintains an append-only ledger $\mathtt{LEDGER}$, an account list $\mathtt{ACCOUNT}$ and a modified tree combining $\mathtt{PT\_BALANCE}$ and $\mathtt{ZK\_BALANCE}$ defined as below:

$$\mathtt{LEDGER} \overset{\text{def}}{=} List[\mathtt{zk\_transaction}]$$
$$\mathtt{ACCOUNT} \overset{\text{def}}{=} List[\mathtt{user} \Rightarrow \mathtt{address}]$$
$$\mathtt{PT\_BALANCE} \overset{\text{def}}{=} List[\mathtt{address} \Rightarrow \mathtt{pt\_balance}]$$
$$\mathtt{ZK\_BALANCE} \overset{\text{def}}{=} List[\mathtt{address} \Rightarrow \mathtt{zk\_balance}]$$

Given a zero-knowledge account-based transaction tx, we can parse it as $tx \overset{\text{def}}{=} \{sender, cmt, proof, *\}$ where $sender$ is an account address, cmt is a commitment of value to be transferred, $proof$ is a zero-knowledge proof and $*$ denotes other useful fields. Note that each user has only one account address. $\mathtt{LEDGER}$, $\mathtt{ACCOUNT}$, $\mathtt{PT\_BALANCE}$ are identical to the one defined in Definition 1.

Given any time $\mathtt{T}$ and a transaction list $\mathtt{TxList}$ containing all zero-knowledge transactions which are confirmed during $(\mathtt{T}-1, \mathtt{T}]$, for each $tx \in \mathtt{TxList}$ and $(u, addr)$ in tx, the $\mathtt{ZK\_BALANCE}$ changes as follows:

$$\mathtt{ZK\_BALANCE_T} = \mathtt{ZK\_BALANCE_{T-1}}.updateZKBalance(addr \Rightarrow cmt).$$

As shown in Fig. 2, the system is composed of a ledger and nodes including users (i.e., senders and recipients) and miners. Users can convert plaintext amounts into zero-knowledge balances by generating $\mathtt{Mint}$ transactions, and vice versa (resp., $\mathtt{Redeem}$ transactions). Senders can transfer money to others using $\mathtt{Send}$ transactions, while recipients can deposit received payment from others to their accounts by building $\mathtt{Deposit}$ transactions with a Merkle tree. Miners are responsible for processing and confirming these zero-knowledge transactions, utilize the consensus algorithm to pack the transactions into a block, and maintain the ledger. After confirming the zero-knowledge transactions, miners update the balance of accounts as shown in Fig. 1.

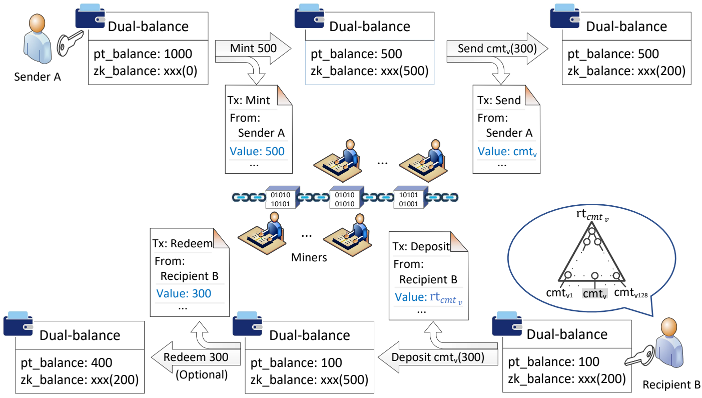


Fig. 2: System architecture and workflow of BlockMaze

IV. BlockMaze Scheme

In this section, we firstly describe a general BlockMaze scheme, and then give its security definition and security proof, and finally give a specific construction of BlockMaze.

For the sake of convenience, we suppose that there are two accounts: sender A (Alice) and recipient B (Bob), their key pairs are (sk_A, pk_A) and (sk_B, pk_B) respectively, their account addresses are $addr_A$ and $addr_B$, their plaintext balances are denoted as $\texttt{pt\_balance}_A$ and $\texttt{pt\_balance}_B$, and their current zero-knowledge balances can be represented as follows:

$$\texttt{zk\_balance}_A := (cmt_A, addr_A, value_A, sn_A, r_A),$$
$$\texttt{zk\_balance}_B := (cmt_B, addr_B, value_B, sn_B, r_B).$$

A. Description of BlockMaze

A BlockMaze scheme Π is composed of polynomial-time algorithms Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**). It has three properties: ledger indistinguishability, transaction unlinkability, and balance.

1) **Setup**$(1^\lambda) \rightarrow \texttt{pp}$. Given a security parameter λ, this algorithm generates a list of public parameters $\texttt{pp}$, which are published and made available to anyone. Note that **Setup** is just executed only once by a trusted party.

2) **CreateAccount**$(\texttt{pp}) \rightarrow \{addr, (sk, pk)\}$. Given public parameters $\texttt{pp}$, this algorithm creates an account address $addr$ and key pair (sk, pk) for a user, where sk is a private key for accessing private data (also decrypting transaction data), pk is a public key for encrypting transaction data, and $addr$ is an account address for sending and receiving payments.

3) **Mint**$(\texttt{pp}, \texttt{zk\_balance}_A, \texttt{pt\_balance}_A, sk_A, v) \rightarrow \{\texttt{zk\_balance}^*_A, \texttt{tx}_{\texttt{Mint}}\}$. This algorithm enables an account (say A) to convert a plaintext amount v into a zero-knowledge amount and merge it with the current zero-knowledge balance. Given public parameters $\texttt{pp}$, the current zero-knowledge balance $\texttt{zk\_balance}_A$, the current plaintext balance $\texttt{pt\_balance}_A$, the account private key sk_A, and a plaintext amount v to be converted into a zero-knowledge amount, account A utilizes this algorithm to mint her new zero-knowledge balance $\texttt{zk\_balance}^*_A$ and generate a transaction $\texttt{tx}_{\texttt{Mint}}$.

Once the $\texttt{tx}_{\texttt{Mint}}$ is recorded on blockchain successfully, the state change of A is as follows:

$$\textit{A's state recorded on MPT before } \textbf{Mint} : \{\texttt{pt\_balance}_A, cmt_A\},$$
$$\textit{A's state recorded on MPT after } \textbf{Mint} : \{\texttt{pt\_balance}_A - v, cmt^*_A\}.$$

4) **Redeem**$(\texttt{pp}, \texttt{zk\_balance}_A, sk_A, v) \rightarrow \{\texttt{zk\_balance}^*_A, \texttt{tx}_{\texttt{Redeem}}\}$. This algorithm enables an account (say A) to convert a zero-knowledge amount back into a plaintext balance. Given public parameters $\texttt{pp}$, the current zero-knowledge balance $\texttt{zk\_balance}_A$, the account private key sk_A, and a plaintext amount v to be converted back from the zero-knowledge balance, account A utilizes this algorithm to redeem a plaintext amount v from her new zero-knowledge balance $\texttt{zk\_balance}^*_A$ and generate a transaction $\texttt{tx}_{\texttt{Redeem}}$.

After that the $\texttt{tx}_{\texttt{Redeem}}$ is recorded on blockchain successfully, the state change of A is as follows:

$$\textit{A's state recorded on MPT before } \textbf{Redeem} : \{\texttt{pt\_balance}_A, cmt_A\},$$
$$\textit{A's state recorded on MPT after } \textbf{Redeem} : \{\texttt{pt\_balance}_A + v, cmt^*_A\}.$$

5) **Send**$(\texttt{pp}, \texttt{zk\_balance}_A, sk_A, pk_B, v) \rightarrow \{\texttt{zk\_balance}^*_A, \texttt{tx}_{\texttt{Send}}\}$. This algorithm enables sender A to send a zero-knowledge amount to recipient B. Given public parameters $\texttt{pp}$, the current zero-knowledge balance $\texttt{zk\_balance}_A$, the account private key sk_A, recipient's public key pk_B, and a plaintext amount v to be transferred, account A calls this algorithm to obtain her new zero-knowledge balance $\texttt{zk\_balance}^*_A$ and generate a transaction $\texttt{tx}_{\texttt{Send}}$.

After building transaction $\texttt{tx}_{\texttt{Send}}$, A should inform B with a hash $h_{\texttt{tx}_{\texttt{Send}}} := \texttt{CRH}(\texttt{tx}_{\texttt{Send}})$ such that B can retrieve and parse $\texttt{tx}_{\texttt{Send}}$ to construct $\texttt{tx}_{\texttt{Deposit}}$. When the $\texttt{tx}_{\texttt{Send}}$ is recorded on blockchain successfully, the state change of A is as follows:

$$\textit{A's state recorded on MPT before } \textbf{Send} : \{\texttt{pt\_balance}_A, cmt_A\},$$
$$\textit{A's state recorded on MPT after } \textbf{Send} : \{\texttt{pt\_balance}_A, cmt^*_A\}.$$

6) **Deposit**$(\texttt{Ledger}_{\texttt{T}}, \texttt{pp}, (sk_B, pk_B), h_{\texttt{tx}_{\texttt{Send}}}, \texttt{zk\_balance}_B) \rightarrow \{\texttt{zk\_balance}^*_B, \texttt{tx}_{\texttt{Deposit}}\}$. This algorithm enables recipient B to check and deposit a received payment into his account. Given the current ledger $\texttt{Ledger}_{\texttt{T}}$, public parameters $\texttt{pp}$, account key pair (sk_B, pk_B), the hash of a $\texttt{Send}$ transaction $h_{\texttt{tx}_{\texttt{Send}}}$ and the current zero-knowledge balance $\texttt{zk\_balance}_B$, recipient B calls **Deposit** to obtain his new zero-knowledge balance $\texttt{zk\_balance}^*_B$ and build a transaction $\texttt{tx}_{\texttt{Deposit}}$.

Based on the hash of $\texttt{tx}_{\texttt{Send}}$ whose sender is A, recipient B can retrieve and parse $\texttt{tx}_{\texttt{Send}}$ to construct $\texttt{tx}_{\texttt{Deposit}}$. Once the $\texttt{tx}_{\texttt{Deposit}}$ is recorded on blockchain successfully, the state change of B is as follows:

$$\textit{B's state recorded on MPT before } \textbf{Deposit} : \{\texttt{pt\_balance}_B, cmt_B\},$$
$$\textit{B's state recorded on MPT after } \textbf{Deposit} : \{\texttt{pt\_balance}_B, cmt^*_B\}.$$

7) **VerTx**$(\texttt{Ledger}_{\texttt{T}}, \texttt{pp}, \texttt{tx}) \rightarrow b$. Given the current ledger $\texttt{Ledger}_{\texttt{T}}$, public parameters $\texttt{pp}$ and a zero-knowledge transaction $\texttt{tx}$, miners call this algorithm to check the validity of all zero-knowledge transactions. The algorithm outputs $b = 1$ if $\texttt{tx}$ is valid, otherwise it outputs $b = 0$. Miners (or nodes maintaining the blockchain) are responsible for verifying all transactions and then update the state of related accounts.

B. Security of BlockMaze

Following a similar model defined in the extended version of Zerocash [4], we define secure properties of our scheme including ledger indistinguishability, transaction unlinkability, transaction non-malleability, and balance, and extend these four properties in regard to the account model.

```
$\mathsf{BlockMaze}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda)$ :
1. $\mathsf{pp} \leftarrow \mathbf{Setup}(1^\lambda)$
2. $(L_0, L_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}_{\mathbf{0}}, \mathcal{O}^{\mathrm{BM}}_{\mathbf{1}}}(\mathsf{pp})$
3. $b \xleftarrow{\$} \{0, 1\}$
4. $Q \xleftarrow{\$} \{$*CreateAccount, Mint, Redeem, Send, Deposit, Insert*$\}$
5. $a \leftarrow Query_{L_b}(Q)$
6. $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}_{\mathbf{0}}, \mathcal{O}^{\mathrm{BM}}_{\mathbf{1}}}(L_0, L_1, a)$
7. return $b = b'$
```

```
$\mathsf{BlockMaze}^{\mathtt{TR\text{-}UL}}_{\Pi,\mathcal{A}}(\lambda)$ :
1. $\mathsf{pp} \leftarrow \mathbf{Setup}(1^\lambda)$
2. $L \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(\mathsf{pp})$
3. $(\mathtt{tx}, \mathtt{tx}') \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(L)$
4. if $\mathtt{participantOf}(\mathtt{tx}, \mathtt{tx}') = \mathtt{addrOf}(\mathcal{A})$ then return 0
5. if $\mathtt{tx} = \mathtt{tx}_{\mathtt{Send}}$ then
     return $\mathtt{tx} \neq \mathtt{tx}' \wedge \mathtt{senderOf}(\mathtt{tx}) = \mathtt{senderOf}(\mathtt{tx}')$
6. if $\mathtt{tx} = \mathtt{tx}_{\mathtt{Deposit}}$ then
     return $\mathtt{tx} \neq \mathtt{tx}' \wedge \mathtt{recipientOf}(\mathtt{tx}) = \mathtt{recipientOf}(\mathtt{tx}')$
7. return 0
```

```
$\mathsf{BlockMaze}^{\mathtt{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda)$ :
1. $\mathsf{pp} \leftarrow \mathbf{Setup}(1^\lambda)$
2. $L \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(\mathsf{pp})$
3. $\mathtt{tx}' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(L)$
4. $b \leftarrow \mathbf{VerTx}(L, \mathsf{pp}, \mathtt{tx}')$
5. return $b \wedge (\exists \mathtt{tx} \in L : \mathtt{tx} \neq \mathtt{tx}' \wedge \mathtt{tx}.sn = \mathtt{tx}'.sn)$
```

```
$\mathsf{BlockMaze}^{\mathtt{BAL}}_{\Pi,\mathcal{A}}(\lambda)$ :
1. $\mathsf{pp} \leftarrow \mathbf{Setup}(1^\lambda)$
2. $L \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(\mathsf{pp})$
3. $(S_{cmt_v}, \mathtt{zk\_balance}^*, \mathtt{pt\_balance}^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{BM}}}(L)$
4. $(v_{\mathtt{zk\_unspent}}, v_{\mathtt{pt\_unspent}}, v_{\mathsf{zk}:\mathcal{A}\rightarrow\mathtt{ACCOUNT}}, v_{\mathsf{pt}:\mathcal{A}\rightarrow\mathtt{ACCOUNT}},$
         $v_{\mathsf{zk}:\mathtt{ACCOUNT}\rightarrow\mathcal{A}}, v_{\mathsf{pt}:\mathtt{ACCOUNT}\rightarrow\mathcal{A}}) \leftarrow$
             $\mathtt{Compute}(L, S_{cmt_v}, \mathtt{zk\_balance}^*, \mathtt{pt\_balance}^*)$
5. if $(v_{\mathtt{zk\_unspent}} + v_{\mathtt{pt\_unspent}} + v_{\mathsf{zk}:\mathcal{A}\rightarrow\mathtt{ACCOUNT}} + v_{\mathsf{pt}:\mathcal{A}\rightarrow\mathtt{ACCOUNT}}) >$
         $(v_{\mathsf{zk}:\mathtt{ACCOUNT}\rightarrow\mathcal{A}} + v_{\mathsf{pt}:\mathtt{ACCOUNT}\rightarrow\mathcal{A}})$ then return 1
6. else return 0
```

Fig. 3: The ledger indistinguishability, transaction unlinkability, transaction non-malleability, and balance experiment for BlockMaze. In $\mathsf{BlockMaze}^{\mathtt{TR\text{-}UL}}_{\Pi,\mathcal{A}}(\lambda)$, `participantOf` denotes participants including sender and recipient of transactions, `addrOf` denotes the address of an account, `senderOf` denotes the sender of a transaction, and `recipientOf` denotes the recipient of a transaction. In $\mathsf{BlockMaze}^{\mathtt{BAL}}_{\Pi,\mathcal{A}}(\lambda)$, S_{cmt_v} is a table of transfer commitments, $\mathtt{zk\_balance}^*$ and $\mathtt{pt\_balance}^*$ are the new account balance, and `Compute` is a function to compute variables related to $\mathcal{A}$'s account balance.

Definition 3 (Security)**.** A BlockMaze scheme is *secure* if it satisfies ledger indistinguishability, transaction unlinkability, transaction non-malleability, and balance as defined below and its experiments as shown in Fig. 3.

Ledger Indistinguishability. A ledger is indistinguishable if it does not reveal new information to the adversary besides the publicly-revealed information. We say a BlockMaze scheme Π is ledger indistinguishable if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and sufficiently large λ, we have $\Pr\left[\mathsf{BlockMaze}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where $\Pr\left[\mathsf{BlockMaze}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) = 1\right]$ is $\mathcal{A}$'s winning probability in the `L-IND` experiment.

Transaction Unlinkability. A transaction is unlinkable if it does not leak the linkage between its sender and recipient during fund transfers. We say a BlockMaze scheme Π is transaction unlinkable if, for every PPT adversary $\mathcal{A}$ and sufficiently large λ, we have $\Pr\left[\mathsf{BlockMaze}^{\mathtt{TR\text{-}UL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$, where $\Pr\left[\mathsf{BlockMaze}^{\mathtt{TR\text{-}UL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right]$ is $\mathcal{A}$'s winning probability in the `TR-UL` experiment.

Transaction Non-malleability. A transaction is non-malleable if no adversary can produce a new transaction different from any previous transactions but with the same revealed data (i.e., serial number of commitments). We say a BlockMaze scheme Π is transaction non-malleable if, for every PPT adversary $\mathcal{A}$ and sufficiently large λ, we have $\Pr\left[\mathsf{BlockMaze}^{\mathtt{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$, where $\Pr\left[\mathsf{BlockMaze}^{\mathtt{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) = 1\right]$ is $\mathcal{A}$'s winning probability in the `TR-NM` experiment.

Balance. A ledger is balanced if no adversary can spend the money more than that in his account. We say a BlockMaze scheme Π is balanced if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and sufficiently large λ, we have $\Pr\left[\mathsf{BlockMaze}^{\mathtt{BAL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$, where $\Pr\left[\mathsf{BlockMaze}^{\mathtt{BAL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right]$ is $\mathcal{A}$'s winning probability in the `BAL` experiment.

The formal definitions are provided in Appendix A. Ledger indistinguishability, which is defined by the `L-IND` experiment, means that no PPT adversary $\mathcal{A}$ can distinguish between two ledgers L_0 and L_1, which are constructed by $\mathcal{A}$ requesting queries to two separated BlockMaze oracles. Transaction unlinkability, which is formalized by the `TR-UL` experiment, means that given a valid fund transfer transaction, the adversary cannot tell: (i) who is the recipient of the transaction and (ii) who is the sender of the payment in the transaction. Transaction non-malleability, which is formalized by the `TR-NM` experiment, means that the adversary $\mathcal{A}$ cannot produce a new legal transaction using the same serial number revealed in any previous transactions. Balance, which is defined by the `BAL` experiment, means that the adversary cannot obtain more money than what he minted or received via payments from others.

Theorem 1. The tuple Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**), as defined in Section IV-C, is a *secure* BlockMaze scheme.

Proof. Refer to Appendix B. □

C. Construction of BlockMaze

In the following description, we utilize gray background variables to denote private data (or hidden variables), which is combined into a witness. Each witness $\vec{a}$ is taken as an auxiliary input in $\Pi_{\mathcal{Z}}$.GenProof to generate proofs. Note that hidden variables are accessible only to related account owners. For the sake of convenience, we summarize the construction of BlockMaze in Appendix C.

Setup. This algorithm generates a list of public parameters. To satisfy the requirements of zero-knowledge transactions, we build specific circuits C_i to prove the validity of these transactions (e.g., `Mint`, `Redeem`, `Send`, `Deposit`). These circuits are taken to generate a key pair $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i})$ for proof generation and verification. Note that this algorithm is executed only once to output a list of public parameters. The detailed process proceeds as follows:

```
Setup
  • inputs: a security parameter $\lambda$
  • outputs: public parameters pp
  1) Compute $\mathsf{pp}_{\mathcal{E}} := \Pi_{\mathcal{E}}.\text{Setup}(1^\lambda)$.
  2) Compute $\mathsf{pp}_{\mathcal{Z}} := \Pi_{\mathcal{Z}}.\text{Setup}(1^\lambda)$.
  3) For each $i \in \{\texttt{Mint}, \texttt{Redeem}, \texttt{Send}, \texttt{Deposit}\}$
      a) Construct a circuit $C_i$.
      b) Compute $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i}) := \Pi_{\mathcal{Z}}.\text{KeyGen}(C_i)$.
  4) Set $\mathsf{PK}_{\mathcal{Z}} := \bigcup \mathsf{pk}_{\mathcal{Z}_i}$ and $\mathsf{VK}_{\mathcal{Z}} := \bigcup \mathsf{vk}_{\mathcal{Z}_i}$.
  5) Output $\mathsf{pp} := (\mathsf{pp}_{\mathcal{E}}, \mathsf{pp}_{\mathcal{Z}}, \mathsf{PK}_{\mathcal{Z}}, \mathsf{VK}_{\mathcal{Z}})$.
```

CreateAccount. Based on $\Pi_{\mathcal{E}}$.KeyGen and a collision-resistant hash function `CRH`, this algorithm creates an account address and key pair for each user as follows:

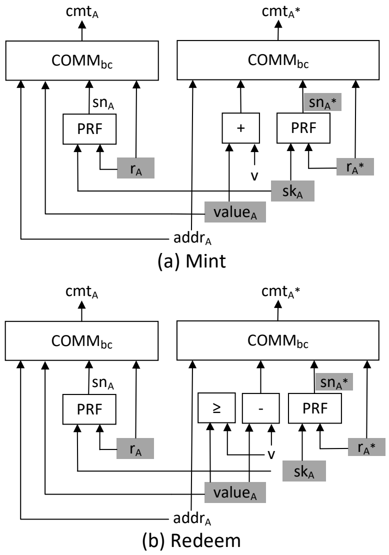
```
CreateAccount
  • inputs: public parameters pp
  • outputs: $(addr, (sk, pk))$
  1) Compute $(sk, pk) := \Pi_{\mathcal{E}}.\text{KeyGen}(\mathsf{pp}_{\mathcal{E}})$.
  2) Compute an account address $addr := \texttt{CRH}(pk)$.
  3) Output $addr$ and an account key pair $(sk, pk)$.
```

Mint. This algorithm builds a `Mint` transaction to convert a plaintext amount into the current zero-knowledge balance of an account (say A). The transaction $\texttt{tx}_{\texttt{Mint}}$ of account A is composed of these variables:

- An account address denoted as $addr_A$: the sender of $\texttt{tx}_{\texttt{Mint}}$.
- A plaintext value denoted as v: a plaintext amount to be converted into a zero-knowledge amount.
- A serial number of the balance commitment denoted as sn_A: a unique string associated with the current balance commitment cmt_A.
- The new balance commitment denoted as cmt^*_A: the new balance commitment to be updated.
- A zero-knowledge proof denoted as prf_m: a proof generated in $\Pi_{\mathcal{Z}}$.GenProof proving that the following equations hold for the circuit of $\texttt{tx}_{\texttt{Mint}}$, as shown in Fig. 4(a):
  - $cmt_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A, sn_A, r_A)$;
  - $sn_A = \texttt{PRF}(sk_A, r_A)$;
  - $cmt^*_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A + v, sn^*_A, r^*_A)$;
  - $sn^*_A = \texttt{PRF}(sk_A, r^*_A)$.

The detailed process proceeds as follows:

```
Mint
  This algorithm merges a plaintext amount with the current zero-
  knowledge balance of an account (say $A$).
  • inputs:
    - public parameters pp
    - the current zero-knowledge balance $\texttt{zk\_balance}_A$
    - the current plaintext balance $\texttt{pt\_balance}_A$
    - account private key $sk_A$
    - a plaintext amount $v$ to be converted into a zero-knowledge
      amount
  • outputs:
    - the new zero-knowledge balance $\texttt{zk\_balance}^*_A$
    - a Mint transaction $\texttt{tx}_{\texttt{Mint}}$
  1) Return fail if $\texttt{pt\_balance}_A < v$ or $v \le 0$.
  2) Parse $\texttt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
  3) Generate a new random number $r^*_A$.
  4) Sample a new serial number $sn^*_A := \texttt{PRF}(sk_A, r^*_A)$.
  5) Compute $cmt^*_A := \texttt{COMM}_{\texttt{bc}}(addr_A, value_A + v, sn^*_A, r^*_A)$.
  6) Set $\vec{x}_1 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
  7) Set $\vec{a}_1 := (value_A, r_A, sk_A, sn^*_A, r^*_A)$.
  8) Compute $prf_m := \Pi_{\mathcal{Z}}.\text{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_1, \vec{a}_1)$.
  9) Set $\texttt{tx}_{\texttt{Mint}} := (addr_A, v, sn_A, cmt^*_A, prf_m)$,
     $\texttt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A + v, sn^*_A, r^*_A)$.
  10) Output $\texttt{zk\_balance}^*_A$ and $\texttt{tx}_{\texttt{Mint}}$.
```



(a) Mint

(b) Redeem

Fig. 4: Mint and Redeem circuits

Redeem. This algorithm generates a `Redeem` transaction to convert a zero-knowledge amount back into the plaintext balance of an account (say A). Some variables are the same as the transaction $\texttt{tx}_{\texttt{Mint}}$ except the proof prf_r, which proves that the following equations hold for the circuit of $\texttt{tx}_{\texttt{Redeem}}$, as shown in Fig. 4(b):

- $cmt_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A, sn_A, r_A)$;
- $sn_A = \texttt{PRF}(sk_A, r_A)$;
- $value_A \ge v$;
- $cmt^*_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A)$;
- $sn^*_A = \texttt{PRF}(sk_A, r^*_A)$.

The detailed process proceeds as follows:

```
Redeem
  This algorithm converts a zero-knowledge amount back into the
  plaintext balance of an account (say $A$).
  • inputs:
    - public parameters pp
    - the current zero-knowledge balance $\texttt{zk\_balance}_A$
    - account private key $sk_A$
    - a plaintext amount $v$ to be converted back from the zero-knowledge balance
  • outputs:
    - the new zero-knowledge balance $\texttt{zk\_balance}^*_A$
    - a Redeem transaction $\texttt{tx}_{\texttt{Redeem}}$
1) Parse $\texttt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
2) Return fail if $value_A < v$ or $v \leq 0$.
3) Generate a new random number $r^*_A$.
4) Sample a new serial number $sn^*_A := \texttt{PRF}(sk_A, r^*_A)$.
5) Compute $cmt^*_A := \texttt{COMM}_{\texttt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A)$.
6) Set $\vec{x}_2 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
7) Set $\vec{a}_2 := (value_A, r_A, sk_A, sn^*_A, r^*_A)$.
8) Compute $prf_r := \Pi_{\mathcal{Z}}.\text{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_2, \vec{a}_2)$.
9) Set $\texttt{tx}_{\texttt{Redeem}} := (addr_A, v, sn_A, cmt^*_A, prf_r)$,
   $\texttt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A - v, sn^*_A, r^*_A)$.
10) Output $\texttt{zk\_balance}^*_A$ and $\texttt{tx}_{\texttt{Redeem}}$.
```

Send. This algorithm sends a zero-knowledge amount from a sender (say A) to a recipient (say B). The transaction $\texttt{tx}_{\texttt{Send}}$ of account A consists of these variables as follows:

- An account address denoted as $addr_A$: the owner of $\texttt{tx}_{\texttt{Send}}$.
- A serial number of the balance commitment denoted as sn_A: a unique string associated with the current balance commitment cmt_A.
- The new balance commitment denoted as cmt^*_A: the new balance commitment to be updated.
- A fund transfer commitment denoted as cmt_v: a commitment of the zero-knowledge amount corresponding to v.
- A ciphertext denoted as aux_A: a ciphertext of sharing parameters using the public key of the recipient.
- An authorization for a ciphertext denoted as $auth_{enc}$: an authorization guarantees the integrity of the ciphertext.
- A zero-knowledge proof denoted as prf_s: a proof generated in $\Pi_{\mathcal{Z}}$.GenProof proving that the following equations hold for the circuit of $\texttt{tx}_{\texttt{Send}}$, as shown in Fig. 5(a):
  - $cmt_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A, sn_A, r_A);$
  - $sn_A = \texttt{PRF}(sk_A, r_A);$
  - $value_A \geq v;$
  - $cmt_v = \texttt{COMM}_{\texttt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A);$
  - $sn_v = \texttt{PRF}\ sk_A, r_v);$
  - $cmt^*_A = \texttt{COMM}_{\texttt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A);$
  - $sn^*_A = \texttt{PRF}(sk_A, r^*_A);$
  - $auth_{enc} = \texttt{PRF}(sk_A, h_{enc}).$

The detailed process proceeds as follows:

```
Send
 This algorithm sends a zero-knowledge amount from sender $A$ to recipient $B$.
  • inputs:
    - public parameters $\mathsf{pp}$
    - the current zero-knowledge balance $\texttt{zk\_balance}_A$
    - account private key $sk_A$
    - recipient's public key $pk_B$
    - a plaintext amount $v$ to be transferred
  • outputs:
    - the new zero-knowledge balance $\texttt{zk\_balance}^*_A$
    - a Send transaction $\texttt{tx}_{\texttt{Send}}$
 1) Return fail if $v \leq 0$.
 2) Parse $\texttt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
 3) Generate a new random number $r_v$.
 4) Sample a new serial number $sn_v := \texttt{PRF}(sk_A, r_v)$.
 5) Return fail if $sn_v$ appears in $\texttt{SNSet}_{\texttt{T}}$.
 6) Compute $cmt_v := \texttt{COMM}_{\texttt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A)$.
 7) Set $aux_A := \Pi_{\mathcal{E}}.\text{Enc}_{pk_B}(\{v, sn_v, r_v, sn_A\})$.
 8) Generate a new random number $r^*_A$.
 9) Sample a new serial number $sn^*_A := \texttt{PRF}(sk_A, r^*_A)$.
10) Compute $cmt^*_A := \texttt{COMM}_{\texttt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A)$.
11) Compute $h_{enc} := \texttt{CRH}(aux_A)$.
12) Compute $auth_{enc} := \texttt{PRF}(sk_A, h_{enc})$.
13) Set $\vec{x}_3 := (cmt_A, addr_A, sn_A, cmt_v, cmt^*_A, h_{enc}, auth_{enc})$.
14) Set $\vec{a}_3 := (value_A, r_A, sk_A, v, pk_B, sn_v, r_v, sn^*_A, r^*_A)$.
15) Compute $prf_s := \Pi_{\mathcal{Z}}.\text{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_3, \vec{a}_3)$.
16) Set $\texttt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A - v, sn^*_A, r^*_A)$
    $\texttt{tx}_{\texttt{Send}} := (addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
17) Output $\texttt{zk\_balance}^*_A$ and $\texttt{tx}_{\texttt{Send}}$.
```

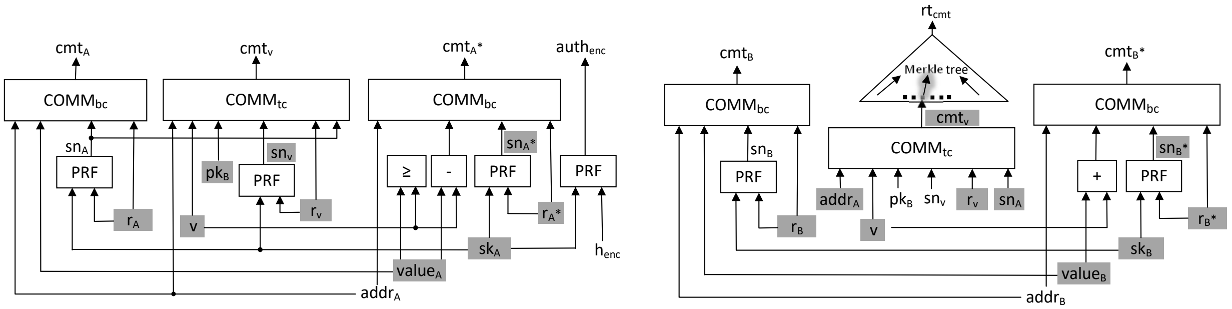
The new plaintext value in cmt^*_A is determined by the plaintext value in both cmt_A and cmt_v. The cmt_v in the above transaction is associated with cmt_A via sn_A, i.e., the same variable sn_A is used in both cmt_v and cmt_A. Note that pk_B is visible only between two negotiating parties during **Send**.

Deposit. This algorithm enables a recipient (say B) to check and deposit a received payment into his account. The transaction $\texttt{tx}_{\texttt{Deposit}}$ of B is composed of these variables:

- A sequence set denoted as seq: a set of block numbers to get transfer commitments and construct a Merkle tree $\texttt{MT}$.
- A Merkle root denoted as rt_{cmt}: the root of $\texttt{MT}$.
- A serial number of the balance commitment denoted as sn_B: a unique string associated with the current balance commitment cmt_B.
- The new balance commitment denoted as cmt^*_B: the new balance commitment to be updated.
- A serial number of the fund transfer commitment denoted as sn_v: a unique string associated with the transfer commitment cmt_v.
- A public key denoted as pk_B: the public key of B.
- A zero-knowledge proof denoted as prf_d: a proof generated in $\Pi_{\mathcal{Z}}$.GenProof proving that the following conditions hold for the circuit of $\texttt{tx}_{\texttt{Deposit}}$, as shown in Fig. 5(b):
  - $cmt_v = \texttt{COMM}_{\texttt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A);$
  - a $path$ from cmt_v to rt_{cmt} recorded on a Merkle tree;
  - $cmt_B = \texttt{COMM}_{\texttt{bc}}(addr_B, value_B, sn_B, r_B);$
  - $sn_B = \texttt{PRF}(sk_B, r_B);$
  - $cmt^*_B = \texttt{COMM}_{\texttt{bc}}(addr_B, value_B + v, sn^*_B, r^*_B);$
  - $sn^*_B = \texttt{PRF}(sk_B, r^*_B).$

The detailed process proceeds as follows:

```
Deposit
 This algorithm enables a recipient (say B) to check and deposit a received payment into his account.
  • inputs:
    - the current ledger $\texttt{Ledger}_{\texttt{T}}$
    - public parameters $\mathsf{pp}$
    - account key pair $(sk_B, pk_B)$
    - the hash of a Send transaction $h_{\texttt{tx}_{\texttt{Send}}}$
    - the current zero-knowledge balance $\texttt{zk\_balance}_B$
  • outputs:
```



(a) Send

(b) Deposit

Fig. 5: Send and Deposit circuits

```
    - the new zero-knowledge balance $\texttt{zk\_balance}^*_B$
    - a Deposit transaction $\texttt{tx}_{\texttt{Deposit}}$
1) Parse $\texttt{zk\_balance}_B$ as $(cmt_B, addr_B, value_B, sn_B, r_B)$.
2) Obtain transaction information of $h_{\texttt{tx}_{\texttt{Send}}}$ from $\texttt{Ledger}_\texttt{T}$
     - the Send transaction is $\texttt{tx}_{\texttt{Send}}$,
     - the block number of $\texttt{tx}_{\texttt{Send}}$ is $N$.
3) Parse a Send transaction $\texttt{tx}_{\texttt{Send}}$ as
   $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
4) Compute $(v, sn_v, r_v, sn_A) := \Pi_{\mathcal{E}}.\text{Dec}_{sk_B}(aux_A)$.
5) Return fail if $sn_v$ appears in $\texttt{SNSet}_\texttt{T}$.
6) Return fail if $cmt_v \neq \texttt{COMM}_{\texttt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A)$
7) Randomly select a set $seq := \{n_1, n_2, ..., N, ..., n_9\}$ from
   existed block numbers.
8) Construct a Merkle tree MT over $\bigcup_{n \in seq} \texttt{TCMSet}_\texttt{n}$.
9) Compute $path := \texttt{Path}(cmt_v)$ and $rt_{cmt}$ over MT.
10) Generate a new random number $r^*_B$.
11) Sample a new serial number $sn^*_B := \texttt{PRF}(sk_B, r^*_B)$.
12) Compute $cmt^*_B := \texttt{COMM}_{\texttt{bc}}(addr_B, value_B + v, sn^*_B, r^*_B)$.
13) Set $\vec{x}_4 := (pk_B, sn_v, rt_{cmt}, cmt_B, addr_B, sn_B, cmt^*_B)$,
14) Set $\vec{a}_4 := (\#, value_B, r_B, sk_B, sn^*_B, r^*_B, path)$
    and # as $(cmt_v, addr_A, v, r_v, sn_A)$.
15) Compute $prf_d := \Pi_{\mathcal{Z}}.\text{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_4, \vec{a}_4)$.
16) Set $\texttt{tx}_{\texttt{Deposit}} := (seq, rt_{cmt}, sn_B, cmt^*_B, sn_v, pk_B, prf_d)$,
    $\texttt{zk\_balance}^*_B := (cmt^*_B, addr_B, value_B + v, sn^*_B, r^*_B)$.
17) Output $\texttt{zk\_balance}^*_B$ and $\texttt{tx}_{\texttt{Deposit}}$.
```

Only after generating a valid proof prf_d with pk_B can B deposit the received payment into his account address $addr_B := \texttt{CRH}(pk_B)$. Moreover, cmt_v associated with pk_B is organized and hidden as a Merkle tree for proof generation, thus outsiders do not know which cmt_v is taken to generate prf_d in a Deposit transaction.

VerTx. This algorithm checks all zero-knowledge transactions. Once these transactions are packed into a candidate block, each transaction must be rechecked to confirm that its related account information (e.g., serial number and new transfer commitment) has not become published, and its Merkle root is valid. If all the checks above are satisfied, then miners will: (i) replace the balance commitment of an account with the new balance commitment (i.e., changing from cmt_A to cmt^*_A); (ii) append a published serial number (e.g., sn_A, sn_v and sn_B) into $\texttt{SNSet}_\texttt{T}$; and (iii) add a transfer commitment (e.g., cmt_v) into $\texttt{TCMSet}_\texttt{N}$ stored in the new block $\texttt{block}_\texttt{N}$. The detailed process proceeds as follows:

```
VerTx
The algorithm checks all zero-knowledge transactions.
• inputs:
    - the current ledger $\texttt{Ledger}_\texttt{T}$
    - public parameters pp
    - a zero-knowledge transaction tx
• outputs: bit $b$
1) If given a transaction tx is $\texttt{tx}_{\texttt{Mint}}$
   a) Parse $\texttt{tx}_{\texttt{Mint}}$ as $(addr_A, v, sn_A, cmt^*_A, prf_m)$.
   b) Obtain related information of $addr_A$ from $\texttt{Ledger}_\texttt{T}$
        - the current plaintext balance is $\texttt{pt\_balance}_A$,
        - the current balance commitment is $cmt_A$.
   c) Return 0 if $\texttt{pt\_balance}_A < v$ or $v \leq 0$.
   d) Return 0 if $sn_A$ appears in $\texttt{SNSet}_\texttt{T}$.
   e) Set $\vec{x}_1 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
   f) Output $b := \Pi_{\mathcal{Z}}.\text{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_1, prf_m)$.
2) If given a transaction tx is $\texttt{tx}_{\texttt{Redeem}}$
   a) Parse $\texttt{tx}_{\texttt{Redeem}}$ as $(addr_A, v, sn_A, cmt^*_A, prf_r)$.
   b) Return 0 if $v \leq 0$.
   c) Obtain related information of $addr_A$ from $\texttt{Ledger}_\texttt{T}$
        - the current balance commitment is $cmt_A$.
   d) Return 0 if $sn_A$ appears in $\texttt{SNSet}_\texttt{T}$.
   e) Set $\vec{x}_2 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
   f) Output $b := \Pi_{\mathcal{Z}}.\text{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_2, prf_r)$.
3) If given a transaction tx is $\texttt{tx}_{\texttt{Send}}$
   a) Parse $\texttt{tx}_{\texttt{Send}}$ as
      $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
   b) Obtain related information of $addr_A$ from $\texttt{Ledger}_\texttt{T}$
        - the current balance commitment is $cmt_A$.
   c) Return 0 if $sn_A$ appears in $\texttt{SNSet}_\texttt{T}$.
   d) Compute $h_{enc} := \texttt{CRH}(aux_A)$.
   e) Set $\vec{x}_3 := (cmt_A, addr_A, sn_A, cmt_v, cmt^*_A, h_{enc}, auth_{enc})$.
   f) Output $b := \Pi_{\mathcal{Z}}.\text{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_3, prf_s)$.
4) If given a transaction tx is $\texttt{tx}_{\texttt{Deposit}}$
   a) Parse $\texttt{tx}_{\texttt{Deposit}}$ as $(seq, rt_{cmt}, sn_B, cmt^*_B, sn_v, pk_B, prf_d)$.
   b) Compute $addr_B := \texttt{CRH}(pk_B)$.
   c) Obtain related information of $addr_B$ from $\texttt{Ledger}_\texttt{T}$
        - the current balance commitment is $cmt_B$.
   d) Return 0 if $sn_B$ or $sn_v$ appears in $\texttt{SNSet}_\texttt{T}$.
   e) Return 0 if $rt_{cmt}$ is not the root over $\bigcup_{n \in seq} \texttt{TCMSet}_\texttt{n}$.
   f) Set $\vec{x}_4 := (pk_B, sn_v, rt_{cmt}, cmt_B, addr_B, sn_B, cmt^*_B)$.
   g) Output $b := \Pi_{\mathcal{Z}}.\text{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_4, prf_d)$.
```

Note that confirming a Deposit transaction, miners first compute $addr_B := \texttt{CRH}(pk_B)$ and then update the new balance commitment cmt^*_B for $addr_B$, in other words, only the owner of pk_B can deposit the received payment.

V. Discussion and Analysis

In this section, we first discuss details of BlockMaze, then analyze its privacy protection, and finally consider attacks on BlockMaze.

A. Discussion

Serial number. In BlockMaze, we utilize a unique serial number to prevent the same balance from being spent more than once, which is to be discussed in the next subsection. A unique serial number sn_A associated with the balance commitment cmt_A is generated by hashing related data such as the account private key, balance, current time, random number, etc. Once cmt_A is spent and changed, sn_A is published and deemed as used. It should be replaced by a new serial number sn_A^* to unlock the account. Each account stores its sn_A^* privately until it is used. Then miners append sn_A into $\texttt{SNSet}$ after confirming corresponding transactions. In the two-step fund transfer procedure, sn_v associated with the fund transfer commitment cmt_v is processed in the same manner.

Off-Chain communication. Like most blockchain systems, the public key of recipient pk_B and the hash of Send transaction $h_{\texttt{tx}_{\texttt{Send}}}$ are transmitted off-chain between both parties. Additionally, the network communication used to broadcast zero-knowledge transactions should be anonymous, to avoid leaking identity information. For instance, these issues can be addressed with I2P and Tor [24].

Sharing parameters. In BlockMaze, both parties share parameters to construct a same fund transfer commitment. When a sender A wants to share parameters with a recipient B, A can utilize B's public key pk_B to encrypt parameters (e.g., v, sn_v, r_v, and sn_A mentioned in $\mathbf{Send}$), and put the ciphertext into $\texttt{tx}_{\texttt{Send}}$. Then B can retrieve and parse $\texttt{tx}_{\texttt{Send}}$ with its hash $h_{\texttt{tx}_{\texttt{Send}}}$ obtained from A, and decrypt to get back the relevant parameters with his private key sk_B. Of course, B can also utilize sk_B to scan the ledger decrypting the ciphertext in $\texttt{tx}_{\texttt{Send}}$ paid to him. Finally, both parties use the same parameters to compute the fund transfer commitment.

Merkle tree. Since a Merkle proof in zk-SNARK needs a Merkle tree with a fixed depth, it is particularly important to set an appropriate depth of Merkle tree. There are two ways to maintain a Merkle tree. One is organized by all cmt_v which are published in $\texttt{tx}_{\texttt{Send}}$ recorded on $\texttt{LEDGER}_{\texttt{T}}$. The other method is based on a random block number sequence set including $\texttt{block}_{\texttt{N}}$, which includes a transfer commitment cmt_v to be proved. The former must set a large depth (e.g., 32) in advance, which may bring about a long time because of proof generation. Considering efficiency and scalability, the latter applies a flexible method to construct a Merkle tree with a smaller depth (e.g., 8) using cmt_v contained in randomly selected block mentioned in Section III.

Transaction fee. Since we design a dual-balance model for BlockMaze, each account has a plaintext balance and a zero-knowledge balance. When a user generates a zero-knowledge transaction, he can utilize an amount from the plaintext balance to pay for the zero-knowledge transaction fee, which is as a reward for the miners checking the validity of the zero-knowledge transaction during the "mining" process.

Batch transfer. One drawback with BlockMaze is that it cannot transfer funds to multiple recipients in one shot. Instead, BlockMaze needs to do it sequentially, and this would be quite inefficient in some applications, e.g. payroll processing. To tackle this problem, we can modify $\mathbf{Send}$ and $\mathbf{Deposit}$ algorithms to support multiple payments within a single transaction. Then one sender can call $\mathbf{Send}$ algorithm to output n cmt_vs stored in $\texttt{tx}_{\texttt{Send}}$ for n different recipients. Similarly, one recipient can utilize the $\mathbf{Deposit}$ algorithm to deposit m cmt_vs from m different senders simultaneously.

B. Privacy protection

In BlockMaze, the privacy of transaction amount and zero-knowledge balance amount is preserved. Meanwhile, the attacker cannot get the linkage of transactions.

Transaction amount. There are two types of transaction amounts: one is a fund transfer commitment in Send (and Deposit) transactions, and the other is a plaintext amount in Mint (and Redeem) transactions. Obviously, the transaction amount in Send (and Deposit) is an unknown value to outsiders due to that the transfer commitment satisfies the hiding property (see Lemma 3). Although an adversary can monitor an amount in Mint (and Redeem), he cannot utilize it to infer the specific amount associated with the zero-knowledge balance of an account since he does not know other secret data. Moreover, since the users pay with a zero-knowledge commitment in Send (and Deposit) transactions, an adversary cannot deduce the transaction amount from the transfer commitment.

Zero-knowledge balance amount. Based on a dual-balance model, each account has a plaintext balance and a zero-knowledge balance. The zero-knowledge balance is only stored secretly by its owner, while its balance commitment is publicly recorded on MPT and made accessible to anyone. After publishing its serial number, the zero-knowledge balance is spent and replaced by a new one; then its balance commitment is also replaced by the new one. Since the balance commitment satisfies the hiding property (see Lemma 3, an adversary cannot infer the specific amount from both the balance commitment and published serial numbers.

The linkage of transactions. In contrast to traditional transactions in account-model blockchains, each zero-knowledge transaction (e.g., Mint, Redeem, Send and Deposit) has a sender and no recipients. Moreover, we hide the linkage of a transaction sender and recipient with the two-step fund transfer procedure: a sender executes the $\mathbf{Send}$ algorithm to publish a fund transfer commitment cmt_v, and then a recipient runs $\mathbf{Deposit}$ algorithm to spend the corresponding cmt_v in a Merkle tree and deposit it into his account. Therefore, an adversary cannot obtain the linkage between a transaction sender and its recipient (see Appendix B).

C. Attacks

Here, we neglect the common attacks such as selfish mining attack, Sybil attack, etc. That is because these attacks are for consensus algorithms, which are not suitable for BlockMaze. In section IV-B, we define properties of BlockMaze, including *ledger indistinguishability*, *transaction unlinkability*, and *balance*, where *transaction unlinkability* resists linkability attack, and *balance* resists both double-spending attack and over-spending attack. These attacks are discussed as follows.

Double-spending attack. Double-spending attack means that the same balance is spent more than once. In traditional account-model based blockchains, a nonce (i.e., a transaction counter in each account) is set in each transaction to resist this attack. In BlockMaze, we utilize a unique serial number (instead of a nonce) to address this attack. More specifically, we associate each zero-knowledge balance with a unique serial number. When each zero-knowledge balance is spent, its serial number must be published in a corresponding transaction and replaced by a new one. Then the miners confirm this transaction and insert the old serial number into the used serial number set (i.e., `SNSet`) to resist double-spending attack.

Over-spending attack. Over-spending attack means that an adversary spends more money than what he owns. In BlockMaze, each zero-knowledge balance must be spent with a valid zero-knowledge proof. If the adversary attempts to spend a specific amount of money that does not belong to him, he has to forge a zero-knowledge proof from the public data. The security of zk-SNARK guarantees that the success probability of such attack is negligible.

Linkability attack. Linkability attack means that an adversary identifies the linkage between a `Send` transaction and its corresponding `Deposit` transaction. To resist this attack, we design a two-step fund transfer procedure, which we have discussed in Section V-B.

Duplicated serial numbers. In BlockMaze, the serial number is used to indicate whether a zero-knowledge fund has been used or a user's zero-knowledge balance is fresh. The serial numbers of a fresh zero-knowledge balance and an unused zero-knowledge fund are hidden from the public. As a result, a malicious sender can produce the same serial number for both his new zero-knowledge balance and a zero-knowledge fund (to be sent to the recipient) without being detected. For example, the sender A produces the serial number as $sn_v = sn_A^*$. If A publishes sn_A^* before the recipient deposits the zero-knowledge fund into his account, the recipient will not be able to deposit the fund anymore because the same serial number has been published.

There are several ways to prevent such attacks. The simplest way is for the recipient to deposit the received fund as soon as it receives it from the sender. However, this approach exposes the timing information which can be used to infer the sender-recipient relationship. A better way is to let $r_A^* = \texttt{CRH}(r_v)$ such that $sn_A^* = \texttt{PRF}(sk_A, r_A^*)$ and $sn_v = \texttt{PRF}(sk_A, r_v)$ are equal for negligible probability. At the end, the sender needs to provide a zero-knowledge proof that proves that $r_A^* = \texttt{CRH}(r_v)$. This modification only adds a new constraint to the zero-knowledge proof, which only slightly increases computation complexity of BlockMaze.

VI. Implementation and Performance Evaluation

In this section, we first implement a prototype of BlockMaze, and then introduce how to conduct comprehensive experiments evaluating its performance.

TABLE III: Overview of zk-SNARKs used in BlockMaze

Proof system	Setup time$^+$	Public parameters$^\#$	Proof*
PGHR13 [14]	126.714s	334MB	287B
Groth16 [15]	96.458s	266MB	128B
GM17 [16]	161.927s	471MB	128B

$^+$ It denotes the time executing **Setup** using different proof systems.
$^\#$ It denotes the size of pp produced from **Setup** algorithm.
* It denotes the size of a proof fixed in a zero-knowledge transaction.

A. Implementation

We implement our BlockMaze scheme based on Libsnark [17] and Ethereum [3], where Libsnark is a library that implements zk-SNARK schemes [14]–[16] in C++. The source code of BlockMaze is available at our GitHub repository[1], which is composed of the following two components.

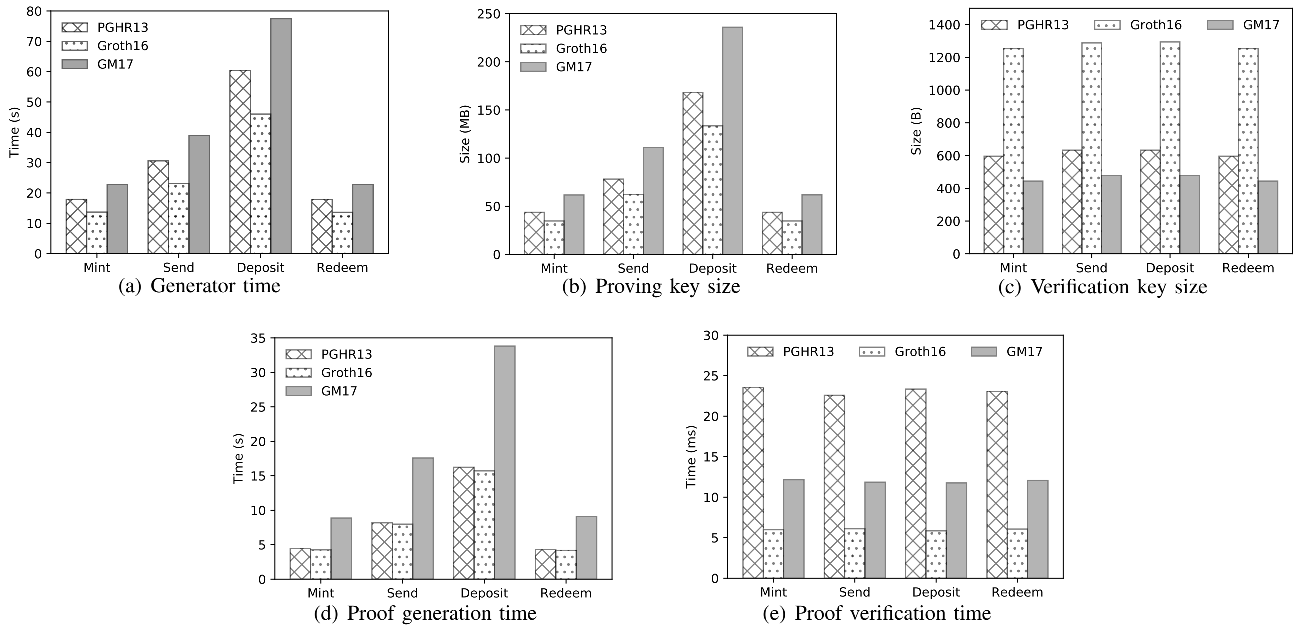
zk-SNARKs for BlockMaze Transactions. For zero-knowledge transactions in BlockMaze (i.e., `Mint`, `Redeem`, `Send` and `Deposit`), we use zk-SNARKs to construct a zero-knowledge proof based on their respective circuits, as shown in Fig. 4 and Fig. 5. These circuits are composed by combining addition, subtraction, less-than, hash, merkle_tree components according to their specifications. The key pair $(\mathsf{PK}_{\mathcal{Z}}, \mathsf{VK}_{\mathcal{Z}})$ for zk-SNARK proof generation/verification is generated and pre-installed at each node. For zk-SNARKs, we take ALT_BN128 as the default elliptic curve and instantiate the `CRH` function with SHA-256 hash function, same as the one used in the Merkle tree. Regarding the Merkle tree used in `Deposit`, it is constructed by taking 256 cmt_v's from the set of blocks (i.e., $block_n$ and $n \in seq$) as tree leaves.

The Underlying Account-Model Blockchain. Our implementation is based on Go-Ethereum [18] v1.8.12 and we have made the following revisions to it: (i) add four new types of transactions (i.e., `Mint`, `Redeem`, `Send` and `Deposit`) besides the original Ethereum transactions; (ii) add functions to generate and verify different zero-knowledge transactions using the Libsnark module; (iii) implement two global tables (e.g., `TCMSet` and `SNSet`) for double spending prevention. All zero-knowledge transactions are verified with the **VerTx** algorithm when they are received to full nodes, while Ethereum's original transactions are processed as usual. Moreover, we adopt ECIES, a hybrid encryption scheme providing semantic security against chosen plaintext attack (CPA) and chosen ciphertext attack (CCA), to instantiate public key encryption $\Pi_{\mathcal{E}}$, and instantiate `PRF`, $\texttt{COMM}_{\texttt{bc}}$, $\texttt{COMM}_{\texttt{tc}}$ and `CRH` with SHA-256 hash function.

B. Experiment Setup and Results

We design comprehensive experiments to evaluate the performance of the proposed scheme. First of all, we evaluate the performance of different zk-SNARK proof systems for zero knowledge transactions on a desktop. Then we compare BlockMaze with Zerocash in terms of computation and storage costs. At last, we establish a 270-node test network for BlockMaze, and evaluate the performance in a practical scenario.

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



(a) Generator time

(b) Proving key size

(c) Verification key size

(d) Proof generation time

(e) Proof verification time

Fig. 6: The performance of zk-SNARKs used in BlockMaze.

Performance of zk-SNARKs. We embed three zk-SNARK proof systems in BlockMaze and evaluate their performance in terms of computation and storage costs on a desktop equipped with Intel Core i5-8500 CPU @3.00GHz and 16-GB RAM running 64bit Ubuntu 16.04 LTS. We first give an overview of performance of these proof systems in Table III. Among them, PGHR13 has been used in Zerocash with a 288-byte zk-SNARK proof, while Groth16 and GM17 reduce the size of proof (i.e., 128B). More importantly, Groth16 has advantages in terms of setup time and the size of public parameters compared with the other two proof systems. To find the more efficient and suitable zk-SNARKs embedded in BlockMaze, we obtain detailed data of these proof systems and compare their different characteristics in Fig. 6. Note that the generator time means the time executing $\Pi_{\mathcal{Z}}.\mathrm{Setup}$ and $\Pi_{\mathcal{Z}}.\mathrm{KeyGen}$ for each of zk-SNARKs.

For each proof system, it is clear that the generator time is determined by the complexity of its circuit (e.g., `Send` circuit contains 7 SHA-256 gates while `Deposit` circuit contains 5 SHA-256 gates and 1 Merkle tree gate), and the majority of the time is to generate the key pair $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i})$. Correspondingly, the generator time is approximately linear to the size of the proving key $\mathsf{pk}_{\mathcal{Z}_i}$, which accounts for the vast majority size of public parameters pp. However, since the **Setup** algorithm is executed only once, it will not impact BlockMaze's performance afterwards. Similarly, the time of proof generation also depends on the complexity of these circuits, especially that `Deposit` takes the vast majority of the time to generate a Merkle proof. In contrast, the size of verification key and the time of proof generation are maintained at a stable value respectively, which is not affected by the circuits' complexity.

According to these bar charts, we compare them and obtain that: (i) Groth16 saves the more time in **Setup** and proof generation/verification, and reduces storage costs on the size of proving key; (ii) GM17 has the lowest size of verification key; (iii) PGHR13 has advantages on proof generation. We can conclude that Groth16 is the more efficient and suitable zk-SNARKs embedded in BlockMaze, whose time cost of proof generation is between about 4 and 16 seconds while proof verification takes about 5.85 milliseconds. Note that the efficient proof verification enables miners to process transactions faster, which is of paramount importance for transaction throughput. Therefore, we only evaluate the performance of BlockMaze equipped with Groth16 to compare with Zerocash later.

TABLE IV: Comparison with Zerocash

This work (Account-model)			Zerocash$^{\#}$ (UTXO-model)		
Setup	time	96.458s	271.243s	time	**Setup**
	pp	266MB	1.87GB	pp	
Create Account	time	1ms	743ms	time	**Create Address**
	pk	64B	343B	$\mathsf{addr}_{\mathsf{pk}}$	
	sk	32B	319B	$\mathsf{addr}_{\mathsf{sk}}$	
Mint	time	4.611s	1µs	time	**Mint**
	$\mathtt{tx}_{\mathtt{Mint}}$	357B	72B	$\mathtt{tx}_{\mathtt{Mint}}$	
Redeem	time	4.636s	102.692s	time	**Pour**
	$\mathtt{tx}_{\mathtt{Redeem}}$	357B			
Send	time	8.436s	1004B	$\mathtt{tx}_{\mathtt{Pour}}$	
	$\mathtt{tx}_{\mathtt{Send}}$	509B			
Deposit	time	18.247s	1.92ms	time	**Receive**
	$\mathtt{tx}_{\mathtt{Deposit}}$	433B			
VerTx	time^{+}	13.833ms	10.9µs	mint	**Verify Transaction**
			28.637ms	pour	

Remark A common transaction except payload in Ethereum is about 198B in size. We exclude the storage cost of seq in `Deposit` transactions (as this cost depends on the number of cmt_v). Each zero-knowledge transaction is extended with extra fields (e.g., $code$, $price$, $gasLimit$ and $hash$).

$^{+}$ It denotes average time cost for verifying a zero-knowledge transaction.

Comparison with Zerocash. To compare with Zerocash, we deploy a private network consisting of 4 desktops which are all equipped with Intel Core i5-8500 CPU @3.00GHz and

8-GB RAM running 64bit Ubuntu 16.04 LTS. Note that each desktop runs one miner node and connects with other nodes to form the test blockchain network. In a duration of 1 hour, we run this test blockchain as follows: each miner node creates all types of transactions to transfer funds every 10 seconds, while it checks these transactions and mines blocks. Then we run and evaluate Zerocash v0.11.2.z0[2], which invokes a libzerocash[3] library, in the same manner.

We provide the experiment results of two schemes in Table IV. Although these two schemes are based on different blockchain model, there are some similar features. From the functional perspective, combining our $\mathbf{Redeem}$ and $\mathbf{Send}$ algorithms are equivalent to $\mathbf{Pour}$ algorithm of Zerocash. For the size of public parameters pp, we neglect basic parameters of $\mathsf{pp}_{\mathcal{E}}$ since the size of proving key dominates the size of pp in both schemes. Obviously, we can see that BlockMaze has clearly advantages on both time costs and storage sizes of $\mathbf{Setup}$ and $\mathbf{CreateAccount}$ algorithms comparing with Zerocash. Since each zero-knowledge transaction contains extra information to prevent double spending and be proved by zero-knowledge proof, its size is larger than a basic transaction in Ethereum. Combining with Fig. 6(d), we can obtain that the time of zero-knowledge transactions generation is dominated by the running time of $\Pi_{\mathcal{Z}}.\mathrm{GenProof}$.

Processing Performance of BlockMaze. To evaluate the performance of BlockMaze, we deploy a test blockchain network consisting of 100 Aliyun ECS g6.xlarge instances which are equipped with 4 vCPU and 16 GB memory running 64bit Ubuntu 16.04 LTS. On each instance, we instantiate 3 independent docker containers as miner nodes. Note that each miner node connects with other 25 randomly selected miner nodes to form the test blockchain network. We run this 300-node test network as follows: each miner node first starts mining blocks, then it creates basic Ethereum transactions and zero-knowledge transactions periodically (every 10 seconds) with variable percentage $\alpha \in \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$ which denotes the proportion of zero-knowledge transactions in all transactions. In a 1-hour experiment, each miner node confirms these transactions and mines them into blocks with PoW consensus algorithm. Although the test network is relatively small in size, it can still provide meaningful insights about the performance of BlockMaze.

Fig. 7 shows the transactions per second (TPS), transaction latency, block size, and block generation and verification time of BlockMaze with different number of miner nodes and zero-knowledge transaction proportion. TPS denotes the number of transactions which each miner node can process per second, transaction latency denotes the time of a transaction from its creation to being recorded on the blockchain, and block generation/verification time denotes the average time required for each miner node mining/validating a block.

As shown in Fig. 7(a)(b), when the number of miner nodes increases, TPS sees a slight decline while transaction latency shows the opposite trend. This is mainly caused by communication between nodes and PoW consensus algorithm. About the former, each node connects with other nodes and requires to broadcast transactions and synchronize blocks. With PoW, it requires that miners compete against each other to confirm transactions on the test network. When the test blockchain has more miner nodes, its mining would be much more competitive and it would show a higher communication load, increasing the time cost of processing transactions. Similarly, increasing the proportion of zero-knowledge transactions has an effect on TPS and transaction latency. The reason is that the size of zero-knowledge transactions is larger than that of basic Ethereum transactions (see Table IV), and each zero-knowledge transaction requires more additional checks including proof verification, serial number uniqueness validation, and merkle root computation. Overall, compared with the case of all basic Ethereum transactions (α = 0%), TPS decreases by 6 transactions and transaction latency increases by 25 seconds in the case of all zero-knowledge transactions (α = 100%).

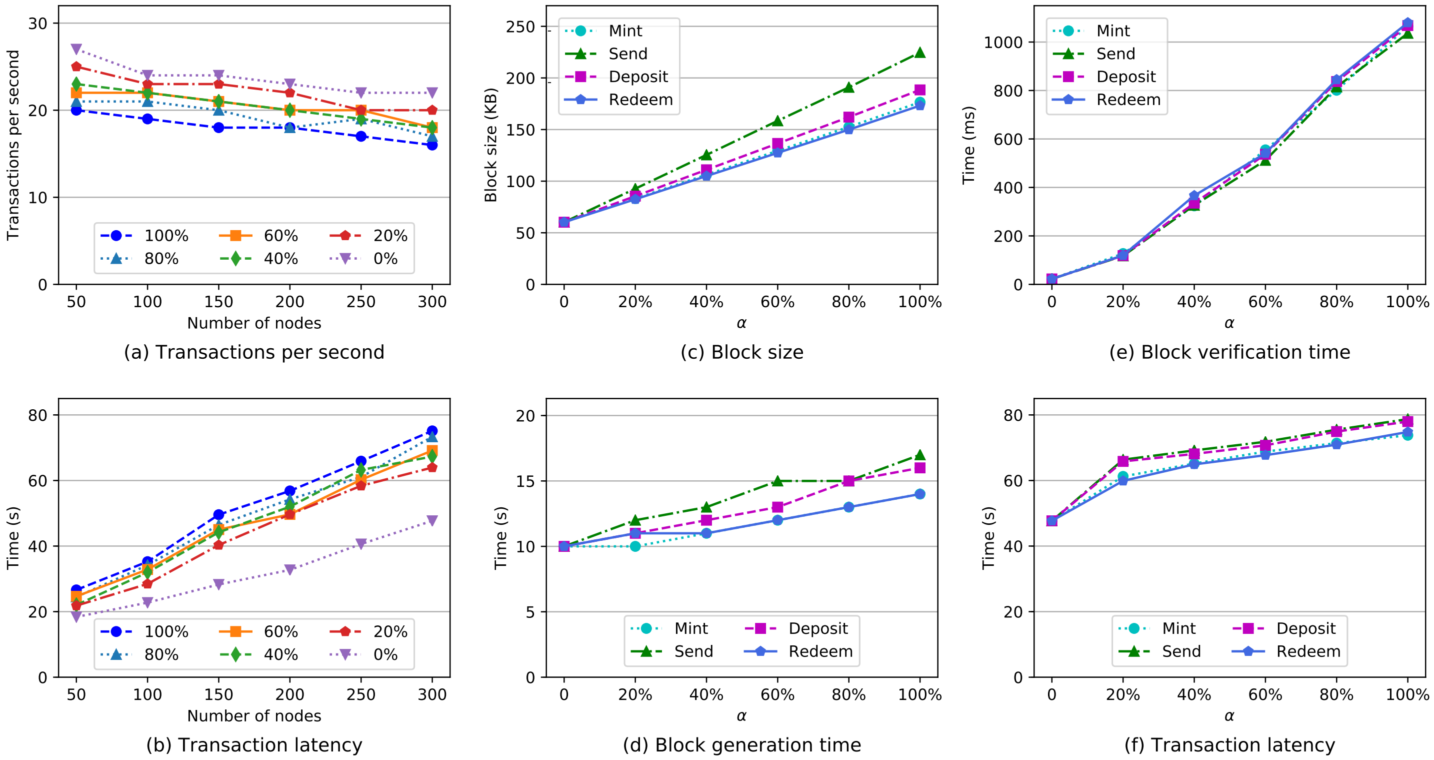
When we set the number of miner nodes to 300, as shown in Fig. 7(c)(d)(e)(f), increasing the proportion of different zero-knowledge affects these metrics: block size, block generation time, block verification time, and transaction latency. Obviously, from α = 0% to α = 100%, the block size sees a steady increase from 61KB ot 225KB, block generation time fluctuates between 10 and 17 seconds, block verification time shows an upward trend reaching 1.08 seconds, and transaction latency is in the range of 48-78 seconds, which means users must wait for approximately 78 seconds before spending received payments. Among four types of zero-knowledge transactions, it is clear that `Send` transaction requires more storage and time cost, which is caused by the $\mathbf{Send}$ algorithm. As mentioned above, adding additional fields related to zero-knowledge proof increases the size of zero-knowledge transactions, which results in that block size gradually rises with the increase in the number of zero-knowledge transactions. Moreover, each miner node requires to validate proof, serial number and merkle root during confirming each zero-knowledge transaction, and update the state of the dual-balance model, which all increase the time cost of block generation and verification. These are also the reasons for a marked increase in the figure for transaction latency. Overall, BlockMaze is very efficient in achieving fund transfer without leaking identity information and transaction amounts, although it requires more storage and time cost compared with basic Ethereum transactions.

VII. Conclusion

In this paper, we have proposed BlockMaze, a privacy-preserving account-model blockchain that allows users to pay each other with strong privacy guarantees. It shows the practicability to resolve two privacy issues: transaction amounts and the linkage between a transaction sender and its recipient. To solve the above issues, we present a *dual-balance* model and design a two-step fund transfer procedure combining zk-SNARK. More importantly, we provide a concrete construction of BlockMaze to show the compatibility to account-model blockchains, and give a formal security proof. Finally, we implement BlockMaze based on Go-Ethereum and Libsnark and design comprehensive experiments to evaluate its performance.

[2] https://github.com/Agzs/zcash/tree/v0.11.2.z0

[3] https://github.com/Agzs/libzerocash



(a) Transactions per second

(c) Block size

(e) Block verification time

(b) Transaction latency

(d) Block generation time

(f) Transaction latency

Fig. 7: The performance of BlockMaze.

A future direction is to utilize a new non-interactive zero-knowledge scheme to achieve higher efficiency over account-model blockchains without the trusted setup.

References

[1] BitInfoCharts, "Cryptocurrency statistics," https://bitinfocharts.com/, Accessed 02/20/2020.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2008.

[3] V. Buterin, "Ethereum white paper: a next generation smart contract & decentralized application platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[4] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *IEEE Symposium on Security and Privacy (SP)*, 2014, pp. 459–474.

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

[6] E. Androulaki *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proc. 13th European Conference on Computer Systems*. ACM, 2018.

[7] I. Grigg, "EOS: an introduction," https://eos.io/documents/EOS_An_Introduction.pdf, 2017.

[8] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," *arXiv preprint*, 2015.

[9] J. DuPont and A. C. Squicciarini, "Toward De-Anonymizing Bitcoin by Mapping Users Location," in *Proc. 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 139–141.

[10] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," in *IEEE Symposium on Security and Privacy (SP)*, 2013, pp. 397–411.

[11] E. Duffield and D. Diaz, "Dash: A PrivacyCentric CryptoCurrency," https://github.com/dashpay/dash/wiki/Whitepaper, 2015.

[12] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," https://bitcointalk.org/index.php?topic=279249.0, 2013.

[13] S. Ma, Y. Deng, D. He, J. Zhang, and X. Xie, "An Efficient NIZK Scheme for Privacy-Preserving Transactions over Account-Model Blockchain." *IACR Cryptology ePrint Archive*, 2017.

[14] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," in *IEEE Symposium on Security and Privacy (SP)*, 2013, pp. 238–252.

[15] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proc. Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 305–326.

[16] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual International Cryptology Conference*. Springer, 2017, pp. 581–612.

[17] SCIPR Lab, "Libsnark: a C++ library for zkSNARK proofs," https://github.com/scipr-lab/libsnark, Accessed 02/20/2020.

[18] Ethereum, "Official Go implementation of the Ethereum protocol," https://github.com/ethereum/go-ethereum, Accessed 02/20/2020.

[19] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Proc. European Symposium on Research in Computer Security*. Springer, 2014, pp. 345–364.

[20] CryptoRekt, "Blackpaper: Verge Currency," https://vergecurrency.com/static/blackpaper/verge-blackpaper-v5.0.pdf, Accessed 02/20/2020.

[21] I. Peverell, "Introduction to MimbleWimble and Grin," https://github.com/mimblewimble/grin/blob/master/doc/intro.md, Accessed 02/20/2020.

[22] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards Privacy in a Smart Contract World," *IACR Cryptology ePrint Archive*, 2019.

[23] A. Rondelet and M. Zajac, "ZETH: On Integrating Zerocash on Ethereum," *arXiv preprint arXiv:1904.00905*, 2019.

[24] B. Conrad and F. Shirazi, "A Survey on Tor and I2P," in *Proc. 9th International Conference on Internet Monitoring and Protection*, 2014.

[25] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *Journal of Network and Computer Applications*, 2018.

[26] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 839–858.

[27] J. Eberhardt and S. Tai, "ZoKrates: Scalable Privacy-Preserving Off-Chain Computations," in *Proc. IEEE International Congress on Cybermatics*, 2018, pp. 1084–1091.

[28] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured

reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2111–2128.

[29] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," *Cryptology ePrint Archive*, 2019.

[30] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive Proof Composition without a Trusted Setup," *Cryptology ePrint Archive*, 2019.

[31] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from dark compilers," *Cryptology ePrint Archive*, 2019.

[32] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," *Cryptology ePrint Archive*, 2019.

[33] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge," *Cryptology ePrint Archive*, 2019.

[34] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334.

[35] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *IACR Cryptology ePrint Archive*, 2018.

[36] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *Proc. Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 626–645.

[37] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture," in *Proc. USENIX Security Symposium*, 2014, pp. 781–796.

[38] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 566–582.

[39] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes," in *Proc. Annual International Cryptology Conference*. Springer, 1998, pp. 26–45.

Appendix A
Security of BlockMaze schemes

A BlockMaze scheme Π = ($\mathbf{Setup}$, $\mathbf{CreateAccount}$, $\mathbf{Mint}$, $\mathbf{Redeem}$, $\mathbf{Send}$, $\mathbf{Deposit}$, $\mathbf{VerTx}$) is *secure* if it satisfies ledger indistinguishability, transaction unlinkability, transaction non-malleability, and balance. For each property, let λ be a security parameter, Π be a BlockMaze scheme, and $\mathcal{A}$ be an adversary, which is formally just a (stateful) algorithm. We now formally define them here.

Following a similar model defined in Zerocash [4], we design an experiment as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Each experiment is employed depending on a stateful BlockMaze oracle $\mathcal{O}^{\mathrm{BM}}$, which provides an interface for executing the algorithms defined in a BlockMaze scheme Π. The oracle $\mathcal{O}^{\mathrm{BM}}$ stores and maintains a ledger L, a set of accounts ACCOUNT, a set of plaintext balance PT_BALANCE, and a set of zero-knowledge balance ZK_BALANCE, where the sets are initialized to be empty at the beginning. The oracle $\mathcal{O}^{\mathrm{BM}}$ responses to queries for different algorithms.

- $Query$(***CreateAccount***). Receiving ***CreateAccount*** query, $\mathcal{C}$ computes an account address $addr$ and a key pair (sk, pk) by calling $\mathbf{CreateAccount}$ algorithm, which are added to ACCOUNT. Then, $\mathcal{C}$ outputs $(addr, pk)$.
- $Query$(***Mint***$, addr_A, v)$. Receiving ***Mint*** query, $\mathcal{C}$ computes $(\mathtt{zk\_balance}^*_A, \mathtt{tx}_{\mathtt{Mint}})$ for user A by calling $\mathbf{Mint}$ algorithm. Add $\mathtt{zk\_balance}^*_A$ to ZK_BALANCE, $\mathtt{pt\_balance}^*_A$ to PT_BALANCE, and $\mathtt{tx}_{\mathtt{Mint}}$ to L, where $\mathtt{zk\_balance}^*_A.value = \mathtt{zk\_balance}_A.value + v$ and $\mathtt{pt\_balance}^*_A = \mathtt{pt\_balance}_A - v$.
- $Query$(***Redeem***$, addr_A, v)$. Receiving ***Redeem*** query, $\mathcal{C}$ computes $(\mathtt{zk\_balance}^*_A, \mathtt{tx}_{\mathtt{Redeem}})$ for user A by calling $\mathbf{Redeem}$ algorithm. Add $\mathtt{zk\_balance}^*_A$ to ZK_BALANCE, $\mathtt{pt\_balance}^*_A$ to PT_BALANCE, and $\mathtt{tx}_{\mathtt{Redeem}}$ to L, where $\mathtt{zk\_balance}^*_A.value = \mathtt{zk\_balance}_A.value - v$ and $\mathtt{pt\_balance}^*_A = \mathtt{pt\_balance}_A + v$.
- $Query$(***Send***$, addr_A, addr_B, v)$. Receiving ***Send*** query, $\mathcal{C}$ computes $(\mathtt{zk\_balance}^*_A, \mathtt{tx}_{\mathtt{Send}})$ by calling $\mathbf{Send}$ algorithm. Then, add $\mathtt{zk\_balance}^*_A$ to ZK_BALANCE, and $\mathtt{tx}_{\mathtt{Send}}$ to L, where $\mathtt{zk\_balance}^*_A.value$ is equal to $(\mathtt{zk\_balance}_A.value - v)$.
- $Query$(***Deposit***$, addr_B, h_{\mathtt{tx}_{\mathtt{Send}}})$. Receiving ***Deposit*** query, $\mathcal{C}$ computes $\mathtt{zk\_balance}^*_B$ and generates $\mathtt{tx}_{\mathtt{Deposit}}$ for user B by executing $\mathbf{Deposit}$ algorithm. Then, add $\mathtt{tx}_{\mathtt{Deposit}}$ to L, and $\mathtt{zk\_balance}^*_B$ to ZK_BALANCE, where $\mathtt{zk\_balance}^*_B.value$ is equal to $(\mathtt{zk\_balance}_B.value{+}v)$.
- $Query$(***Insert***$, \mathtt{tx})$. Receiving ***Insert*** query, $\mathcal{C}$ verifies the output of $\mathbf{VerTx}$ algorithm: if the output is 1, add the Mint/Redeem/Send/Deposit transaction tx to L; otherwise, it aborts.

B.1 Ledger Indistinguishability

We now describe a ledger indistinguishability experiment L-IND, which involves a probabilistic polynomial-time adversary $\mathcal{A}$ trying to attack a BlockMaze scheme. Before giving a formal experiment, we first define public consistency for a pair of queries.

Definition 4 (Public consistency)**.** A pair of queries (Q, Q') is *publicly consistent* if both queries are of the *same* type and consistent in $\mathcal{A}$'s view. The public information contained in (Q, Q') must be equal including: (i) the value to be transformed; (ii) the account address; (iii) the balance commitment; (iv) the transfer commitment; and (v) published serial number. Moreover, both queries must satisfy the following restrictions for different query types:

For ***CreateAccount*** type, (Q, Q') are always publicly consistent since that the ledgers remain unchanged during the queries. Moreover, we require that both oracles generate the same account to reply to both queries.

For ***Mint*** and ***Redeem*** type, (Q, Q') must be mutually independent and meet the following restrictions:

- the balance commitment cmt_A in Q must correspond to $\mathtt{zk\_balance}_A$ that appears in ZK_BALANCE;
- the zero-knowledge balance $\mathtt{zk\_balance}_A$ must be valid, i.e., its serial number must never be published before;
- the account address $addr_A$ contained in Q must match with that in $\mathtt{zk\_balance}_A$ and $\mathtt{zk\_balance}^*_A$;
- $(\mathtt{zk\_balance}_A.value + \mathtt{pt\_balance}_A)$ is equal to $(\mathtt{zk\_balance}^*_A.value + \mathtt{pt\_balance}^*_A)$.

For ***Send*** type, (Q, Q') must be mutually independent and meet the following restrictions:

- the balance commitment cmt_A in Q must correspond to $\mathtt{zk\_balance}_A$ that appears in ZK_BALANCE;
- the zero-knowledge balance $\mathtt{zk\_balance}_A$ must be valid, i.e., its serial number must never be published before;

- the account address $addr_A$ specified in Q must match with that in $\texttt{zk\_balance}_A$, $\texttt{zk\_balance}^*_A$ and cmt_v;
- $\texttt{zk\_balance}_A.value - v = \texttt{zk\_balance}^*_A.value$.

For ***Deposit*** type, (Q, Q') must be mutually independent and meet the following restrictions:

- the balance commitment cmt_B in Q must correspond to $\texttt{zk\_balance}_B$ that appears in ZK_BALANCE;
- the zero-knowledge balance $\texttt{zk\_balance}_B$ must be valid, i.e., its serial number must never be published before;
- the transfer commitment cmt_v must be valid, i.e., it must appear in a valid transaction $\texttt{tx}_{\texttt{Send}}$ on the corresponding oracle's ledger, and its serial number must never be published before;
- the account address $addr_B$ in Q must match with that in $\texttt{zk\_balance}_B$ and $\texttt{zk\_balance}^*_B$;
- $\texttt{zk\_balance}_B.value + v = \texttt{zk\_balance}^*_B.value$.

Formally, we define the ledger indistinguishability experiment $\mathsf{BlockMaze}^{\texttt{L-IND}}_{\Pi,\mathcal{A}}(\lambda)$ as follows:

1) The public parameters $\mathsf{pp} := \mathbf{Setup}(1^\lambda)$ is computed and given to $\mathcal{A}$. Two *independent* BlockMaze oracles $\mathcal{O}^{\mathrm{BM}}_0$ and $\mathcal{O}^{\mathrm{BM}}_1$ are initialized. A uniform bit $b \in \{0, 1\}$ is chosen.
2) Whenever $\mathcal{A}$ sends a pair of *publicly consistent* queries (Q, Q'), answer the queries in the following way:
   a) Provide two separate ledgers (L_b, L_{1-b}) to $\mathcal{A}$ in each step. L_b is the current ledger in $\mathcal{O}^{\mathrm{BM}}_b$, and L_{1-b} is the one in $\mathcal{O}^{\mathrm{BM}}_{1-b}$.
   b) Send Q to $\mathcal{O}^{\mathrm{BM}}_b$ and Q' to $\mathcal{O}^{\mathrm{BM}}_{1-b}$ to obtain two oracle answers (a_b, a_{1-b}).
   c) Return (a_b, a_{1-b}) to $\mathcal{A}$.
3) Continue answering *publicly consistent* queries of $\mathcal{A}$ until $\mathcal{A}$ outputs a bit b'.
4) The game outputs 1 if $b' = b$, and 0 otherwise. If $\mathsf{BlockMaze}^{\texttt{L-IND}}_{\Pi,\mathcal{A}}(\lambda) = 1$, we say that $\mathcal{A}$ succeeds.

Definition 5 (L-IND Security)**.** A BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) is L-IND secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that, for security parameter λ,

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

B.2 Transaction Unlinkability

Let $\mathcal{T}$ be the table of zero-knowledge transactions (i.e., $\texttt{tx}_{\texttt{Send}}$ and $\texttt{tx}_{\texttt{Depoist}}$) generated by $\mathcal{O}^{\mathrm{BM}}$ in response to ***Send*** and ***Deposit*** queries. We define the transaction unlinkability experiment $\mathsf{BlockMaze}^{\texttt{TR-UL}}_{\Pi,\mathcal{A}}(\lambda)$ as follows:

1) The public parameters $\mathsf{pp} := \mathbf{Setup}(1^\lambda)$ is computed and given to $\mathcal{A}$. A BlockMaze oracle $\mathcal{O}^{\mathrm{BM}}$ is initialized.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathrm{BM}}$, answer this query along with the ledger L at each step.
3) Continue answering queries of $\mathcal{A}$ until $\mathcal{A}$ sends a pair of zero-knowledge transactions $(\texttt{tx}, \texttt{tx}')$ with the requirements: (i) $(\texttt{tx}, \texttt{tx}') \in \mathcal{T}$ are of the *same* type; (ii) $\texttt{tx} \neq \texttt{tx}'$; (iii) the senders of $(\texttt{tx}, \texttt{tx}')$ are not $\mathcal{A}$ if $\texttt{tx} = \texttt{tx}_{\texttt{Send}}$; (iv) the recipients of $(\texttt{tx}, \texttt{tx}')$ are not $\mathcal{A}$ if $\texttt{tx} = \texttt{tx}_{\texttt{Deposit}}$.
4) The experiment outputs 1 (indicating $\mathcal{A}$ wins the game) if one of the following conditions holds: (i) the recipients of payments contained in $(\texttt{tx}, \texttt{tx}')$ are the same if $\texttt{tx} = \texttt{tx}_{\texttt{Send}}$; and (ii) the senders of payments contained in $(\texttt{tx}, \texttt{tx}')$ are the same if $\texttt{tx} = \texttt{tx}_{\texttt{Deposit}}$. Otherwise, it outputs 0.

Definition 6 (TR-UL Security)**.** A BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) is TR-UL secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that, for security parameter λ,

$$\Pr\left[\mathsf{BlockMaze}^{\texttt{TR-UL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

B.3 Transaction Non-malleability

Let $\mathcal{T}$ be the table of four types of zero-knowledge transactions generated by $\mathcal{O}^{\mathrm{BM}}$ in response to corresponding queries. We define the transaction non-malleability experiment $\mathsf{BlockMaze}^{\texttt{TR-NM}}_{\Pi,\mathcal{A}}(\lambda)$ as follows:

1) The public parameters $\mathsf{pp} := \mathbf{Setup}(1^\lambda)$ is computed and given to $\mathcal{A}$. A BlockMaze oracle $\mathcal{O}^{\mathrm{BM}}$ is initialized.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathrm{BM}}$, answer this query along with the ledger L at each step.
3) Continue answering queries of $\mathcal{A}$ until $\mathcal{A}$ sends a zero-knowledge transaction $\texttt{tx}' \in \mathcal{T}$.
4) The experiment outputs 1 if there exists $\texttt{tx} \in \mathcal{T}$ satisfying all following conditions: (i) $\texttt{tx}' \neq \texttt{tx}$; (ii) both $\texttt{tx}'$ and $\texttt{tx}$ published the same serial number; and (iii) $\mathbf{VerTx}(L, \mathsf{pp}, \texttt{tx}') = 1$, where L is the current ledger containing $\texttt{tx}$. Otherwise, it outputs 0.

Definition 7 (TR-NM Security)**.** A BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) is TR-NM secure if, for every polynomial-size adversary $\mathcal{A}$ and sufficiently large λ,

$$\Pr\left[\mathsf{BlockMaze}^{\texttt{TR-NM}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

B.4 Balance

We employ an experiment BAL, which involves a probabilistic polynomial-time adversary $\mathcal{A}$ trying to attack a given BlockMaze scheme, where a similar definition is given in [4]. Firstly, we define eight variables for the security model of balance.

- $v_{\texttt{zk\_unspent}}$, the spendable amount in $\texttt{zk\_balance}^*$, i.e., $\texttt{zk\_balance}^*.value$. The challenger $\mathcal{C}$ can check that $\texttt{zk\_balance}^*$ is valid by accessing to $\mathcal{A}$'s balance commitment recorded on MPT on L.
- $v_{\texttt{pt\_unspent}}$, the spendable amount of plaintext balance $\texttt{pt\_balance}^*$. The challenger $\mathcal{C}$ can check that $\texttt{pt\_balance}^*$ is valid by accessing to $\mathcal{A}$'s account plaintext balance recorded on MPT on L.
- $v_{\texttt{Mint}}$, the total value of all plaintext amount minted by $\mathcal{A}$. To compute $v_{\texttt{Mint}}$, the challenger $\mathcal{C}$ looks up all Mint transactions placed on L via ***Mint*** queries and sums up the values that were transformed to $\mathcal{A}$.

- $v_{\texttt{Redeem}}$, the total value of all zero-knowledge amount redeemed by $\mathcal{A}$. To compute $v_{\texttt{Redeem}}$, the challenger $\mathcal{C}$ looks up all Redeem transactions placed on L via ***Redeem*** queries and sums up the values that were transformed to $\mathcal{A}$.
- $v_{\mathsf{zk}:\texttt{ACCOUNT}\to\mathcal{A}}$, the total value of payments received by $\mathcal{A}$ from account addresses in ACCOUNT. To compute $v_{\mathsf{zk}:\texttt{ACCOUNT}\to\mathcal{A}}$, the challenger $\mathcal{C}$ looks up all Deposit transactions placed on L via ***Deposit*** queries and sums up the values in S_{cmt_v} whose recipient is $\mathcal{A}$.
- $v_{\mathsf{pt}:\texttt{ACCOUNT}\to\mathcal{A}}$, the total value of payments received by $\mathcal{A}$ from account addresses in ACCOUNT. To compute $v_{\mathsf{pt}:\texttt{ACCOUNT}\to\mathcal{A}}$, the challenger $\mathcal{C}$ looks up all plaintext transactions placed on L and sums up the values that were transferred to $\mathcal{A}$.
- $v_{\mathsf{zk}:\mathcal{A}\to\texttt{ACCOUNT}}$, the total value of payments sent by $\mathcal{A}$ to account addresses in ACCOUNT. To compute $v_{\mathsf{zk}:\mathcal{A}\to\texttt{ACCOUNT}}$, the challenger $\mathcal{C}$ looks up all Send transactions placed on L via ***Send*** queries and sums up the values in S_{cmt_v} whose sender is $\mathcal{A}$.
- $v_{\mathsf{pt}:\mathcal{A}\to\texttt{ACCOUNT}}$, the total value of payments sent by $\mathcal{A}$ to account addresses in ACCOUNT. To compute $v_{\mathsf{pt}:\mathcal{A}\to\texttt{ACCOUNT}}$, the challenger $\mathcal{C}$ looks up all plaintext transactions placed on L and sums up the values whose sender is $\mathcal{A}$.

For an honest account u, the following equations hold:

$$v_{\texttt{zk\_unspent}} + v_{\texttt{Redeem}} + v_{\mathsf{zk}:u\to\texttt{ACCOUNT}} = v_{\texttt{Mint}} + v_{\mathsf{zk}:\texttt{ACCOUNT}\to u},$$

$$v_{\texttt{pt\_unspent}} + v_{\texttt{Mint}} + v_{\mathsf{pt}:u\to\texttt{ACCOUNT}} = v_{\texttt{Redeem}} + v_{\mathsf{pt}:\texttt{ACCOUNT}\to u}.$$

Add these two equations together, and we can obtain that

$$v_{\texttt{zk\_unspent}} + v_{\texttt{pt\_unspent}} + v_{\mathsf{zk}:u\to\texttt{ACCOUNT}} + v_{\mathsf{pt}:u\to\texttt{ACCOUNT}} = v_{\mathsf{zk}:\texttt{ACCOUNT}\to u} + v_{\mathsf{pt}:\texttt{ACCOUNT}\to u}.$$

Formally, we use $\mathsf{BlockMaze}^{\texttt{BAL}}_{\Pi,\mathcal{A}}(\lambda)$ to represent the balance experiment as follows:

1) The public parameters $\mathsf{pp} := \mathbf{Setup}(1^\lambda)$ is computed and given to $\mathcal{A}$. A BlockMaze oracle $\mathcal{O}^{\mathrm{BM}}$ is initialized.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathrm{BM}}$, answer this query along with the ledger L in each step.
3) Continue answering queries of $\mathcal{A}$ until $\mathcal{A}$ sends a table of transfer commitments S_{cmt_v}, the new account balance $\texttt{zk\_balance}^*$ and $\texttt{pt\_balance}^*$.
4) Compute the eight variables mentioned above.
5) The experiment outputs 1 if $(v_{\texttt{zk\_unspent}} + v_{\texttt{pt\_unspent}} + v_{\mathsf{zk}:\mathcal{A}\to\texttt{ACCOUNT}} + v_{\mathsf{pt}:\mathcal{A}\to\texttt{ACCOUNT}})$ is greater than $(v_{\mathsf{zk}:\texttt{ACCOUNT}\to\mathcal{A}} + v_{\mathsf{pt}:\texttt{ACCOUNT}\to\mathcal{A}})$. Otherwise, it outputs 0.

Definition 8 (BAL Security)**.** A BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) is BAL secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that, for security parameter λ,

$$\Pr\left[\mathsf{BlockMaze}^{\texttt{BAL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \le \mathsf{negl}(\lambda).$$

Appendix B
Proof of Security

A BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) is *secure* if it satisfies ledger indistinguishability, transaction unlinkability, transaction non-malleability, and balance.

C.1 Proof of Ledger Indistinguishability

We now give a formal proof to prove Theorem 1, which is proved using game-based frameworks. The notations used in this proof are listed below. The adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ as in the L-IND experiment. After receiving a pair of *publicly consistent* queries (Q, Q') from $\mathcal{A}$, $\mathcal{C}$ answers (Q, Q') as in the simulation $\Game_{\mathsf{sim}}$. Thus, $\mathcal{A}$'s advantage in $\Game_{\mathsf{sim}}$ (represented by $\mathsf{Adv}^{\Game_{\mathsf{sim}}}$) is 0. We now prove that $\mathsf{Adv}^{\texttt{L-IND}}_{\Pi,\mathcal{A}}(\lambda)$ (i.e., $\mathcal{A}$'s advantage in the L-IND experiment) is at most negligibly different than $\mathsf{Adv}^{\Game_{\mathsf{sim}}}$.

TABLE V: Notations

$\Game_{\mathsf{real}}$	The original L-IND experiment
$\Game_i$	A hybrid game with a modification of the $\Game_{\mathsf{real}}$
q_{CA}	The total number of ***CreateAccount*** queries issued by $\mathcal{A}$
q_{M}	The total number of ***Mint*** queries issued by $\mathcal{A}$
q_{R}	The total number of ***Redeem*** queries issued by $\mathcal{A}$
q_{S}	The total number of ***Send*** queries issued by $\mathcal{A}$
q_{D}	The total number of ***Deposit*** queries issued by $\mathcal{A}$
$\mathsf{Adv}^{\Game 4}$	$\mathcal{A}$'s advantage in game $\Game$
$\mathsf{Adv}^{\Pi_{\mathcal{E}}}$	$\mathcal{A}$'s advantage in $\Pi_{\mathcal{E}}$'s IND-CCA and IK-CCA experiments
$\mathsf{Adv}^{\texttt{PRF}}$	$\mathcal{A}$'s advantage in distinguishing PRF from a random one
$\mathsf{Adv}^{\texttt{COMM}}$	$\mathcal{A}$'s advantage against the hiding property of COMM

Firstly, we describe a simulation $\Game_{\mathsf{sim}}$ in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ as in the queries defined in Appendix A, with the following modification: for each $i \in \{\texttt{Mint}, \texttt{Redeem}, \texttt{Send}, \texttt{Deposit}\}$, the zk-SNARK keys are generated as $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i}, \mathtt{trap}_i) := \mathcal{S}(C_i)$, to obtain the zero-knowledge trapdoor $\mathtt{trap}_i$. After checking each query from $\mathcal{A}$, the challenger $\mathcal{C}$ answers the queries as below.

- To answer ***CreateAccount*** queries, $\mathcal{C}$ does the same as in $Query$(***CreateAccount***) with the following differences: $\mathcal{C}$ computes $(sk, pk) := \Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp})$; then, $\mathcal{C}$ utilizes a random string to replace pk, and computes an account address $addr := \texttt{CRH}(pk)$; finally, $\mathcal{C}$ stores (sk, pk) in a table and returns $(addr, pk)$ to $\mathcal{A}$.
- To answer ***Mint*** queries, $\mathcal{C}$ makes the following modifications in $Query$(***Mint***$, addr_A, v)$: $\mathcal{C}$ samples a uniformly random sn_A; if $addr_A$ is an account address created by a previous query to ***CreateAccount***, then $\mathcal{C}$ samples a balance commitment cmt^*_A on a random input, otherwise, $\mathcal{C}$ computes cmt^*_A as in the **Mint** algorithm; moreover, $\mathcal{C}$ computes and obtains all remaining values as in the **Mint** algorithm; finally, $\mathcal{C}$ constructs a statement $\vec{x_1}$ and computes the Mint proof $prf_m := \mathcal{S}(\mathsf{pk}_{\mathcal{Z}_1}, \vec{x_1}, \mathtt{trap}_{\texttt{Mint}})$.
- To answer ***Redeem*** queries, $\mathcal{C}$ makes the modifications in $Query$(***Redeem***$, addr_A, v)$ as answering ***Mint*** queries, except for the following modification: $\mathcal{C}$ computes the Redeem proof $prf_r := \mathcal{S}(\mathsf{pk}_{\mathcal{Z}_2}, \vec{x_2}, \mathtt{trap}_{\texttt{Redeem}})$, where $\vec{x_2}$ is a statement.
- To answer ***Send*** queries, $\mathcal{C}$ makes the following modifications in $Query$(***Send***$, addr_A, addr_B, v)$: $\mathcal{C}$ first samples a uniformly random sn_A; if $addr_B$ is an account address created by a previous query to ***CreateAccount***, then $\mathcal{C}$

[4] We abuse $\mathsf{Adv}^{\Game}$ to denote the absolute value of the difference between (i) the L-IND advantage of $\mathcal{A}$ in $\Game$ and (ii) the L-IND advantage of $\mathcal{A}$ in $\Game_{\mathsf{real}}$.

does as follows: (i) sample a transfer commitment cmt_v and a balance commitment cmt^*_A on random inputs, (ii) compute $(sk'_B, pk'_B) := \Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp}_{\mathcal{E}})$; (iii) encrypt $aux_A := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk'_B}(r)$, where r is a random string of appropriate length in the plaintext space of the encryption scheme; otherwise, $\mathcal{C}$ computes (cmt_v, cmt^*_A, aux_A) as in the **Send** algorithm; moreover, $\mathcal{C}$ computes and obtains all remaining values as in the **Send** algorithm; finally, $\mathcal{C}$ constructs a statement $\vec{x_3}$ and computes the Send proof $prf_s := \mathcal{S}(\mathsf{pk}_{\mathcal{Z}_3}, \vec{x_3}, \mathtt{trap}_{\mathtt{Send}})$.

- To answer ***Deposit*** queries, $\mathcal{C}$ makes the following modifications in $Query(\textbf{\textit{Deposit}}, addr_B, h_{\mathtt{tx}_{\mathtt{Send}}})$: $\mathcal{C}$ first samples a uniformly random sn_B; if $addr_B$ is an account address created by a previous query to ***CreateAccount***, then $\mathcal{C}$ samples a balance commitment cmt^*_B on a random input, otherwise, $\mathcal{C}$ computes cmt^*_B as in the **Deposit** algorithm; moreover, $\mathcal{C}$ computes and obtains all remaining values as in the **Deposit** algorithm; finally, $\mathcal{C}$ computes the Deposit proof $prf_d := \mathcal{S}(\mathsf{pk}_{\mathcal{Z}_4}, \vec{x_4}, \mathtt{trap}_{\mathtt{Deposit}})$, where $\vec{x_4}$ is a statement.
- To answer ***Insert*** queries, $\mathcal{C}$ does the same as in $Query(\textbf{\textit{Insert}}, \mathsf{tx})$.

Note that the answer to $\mathcal{A}$ is computed independently of the bit b for each of the above cases. Thus, when $\mathcal{A}$ outputs a guess b', it must be the case that $\Pr[b = b'] = 1/2$, i.e., $\mathcal{A}$'s advantage in $\Game_{\mathsf{sim}}$ is 0.

Game $\Game_1$**.** This is the same as $\Game_{\mathsf{real}}$, except for one modification: $\mathcal{C}$ simulates each zk-SNARK proof. For each $i \in \{\mathtt{Mint}, \mathtt{Redeem}, \mathtt{Send}, \mathtt{Deposit}\}$, the zk-SNARK keys are generated as $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i}, \mathtt{trap}_i) := \mathcal{S}(C_i)$ instead of $\Pi_{\mathcal{Z}}.\mathrm{KeyGen}$, to obtain the zero-knowledge trapdoor $\mathtt{trap}_i$. Then $\mathcal{C}$ computes $prf_i := \mathcal{S}(\mathsf{pk}_{\mathcal{Z}_i}, \vec{x_i}, \mathtt{trap_i})$, without using any witnesses $\vec{a_i}$, instead of using $\Pi_{\mathcal{Z}}.\mathrm{GenProof}$ in the **Mint**, **Redeem**, **Send**, **Deposit** algorithms. Since the zk-SNARK is perfect zero-knowledge, the distribution of the simulated prf_i is identical to that of the proofs computed in $\Game_{\mathsf{real}}$. Moreover, we also modify $\Game_{\mathsf{real}}$ such that: each time $\mathcal{A}$ issues a ***CreateAccount*** query, the value pk associated with the returned $addr$ is substituted with a random string of the same length. Since the $(sk, pk) := \Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp})$, the distribution of the simulated (sk, pk) is identical to that of the key pairs computed in $\Game_{\mathsf{real}}$. Hence $\mathsf{Adv}^{\Game_1} = 0$.

Game $\Game_2$**.** $\Game_2$ is the same as $\Game_1$ with one modification: $\mathcal{C}$ utilizes a random string of the suitable length to replace the ciphertext in a Send transaction. If $\mathcal{A}$ sends a ***Send*** query where the address $addr_B$ is an account address created by a previous query to ***CreateAccount***, then $\mathcal{C}$ invokes $\Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp}_{\mathcal{E}})$ to compute $(sk^{'}_B, pk^{'}_B)$ and obtains $aux_A := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk^{'}_B}(r)$ for a random string r of suitable length; otherwise, $\mathcal{C}$ computes aux_A as in the **Send** algorithm. By Lemma 1 (see below), $|\mathsf{Adv}^{\Game_2} - \mathsf{Adv}^{\Game_1}| \leq 2 \cdot q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}}$.

Game $\Game_3$**.** $\Game_3$ is the same as $\Game_2$ with one modification: $\mathcal{C}$ utilizes random strings of the suitable length to replace all serial numbers generated by PRF. As the subsequent results of the ***Mint***, ***Redeem***, ***Send***, ***Deposit*** queries, these serial numbers (e.g., sn_A and sn_v) are respectively placed in $\mathtt{tx}_{\mathtt{Mint}}$, $\mathtt{tx}_{\mathtt{Redeem}}$, $\mathtt{tx}_{\mathtt{Send}}$, $\mathtt{tx}_{\mathtt{Deposit}}$. By Lemma 2 (see below), $|\mathsf{Adv}^{\Game_3} - \mathsf{Adv}^{\Game_2}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{PRF}}$.

Game $\Game_{\mathsf{sim}}$**.** $\Game_{\mathsf{sim}}$ defined above is the same as $\Game_3$ with one modification: $\mathcal{C}$ replaces all balance commitments generated by COMM with commitments to random inputs. As the subsequent results of the ***Mint***, ***Redeem***, ***Send***, ***Deposit*** queries, these balance commitments (e.g., cmt_A, cmt^*_A and cmt_v) are respectively placed in $\mathtt{tx}_{\mathtt{Mint}}$, $\mathtt{tx}_{\mathtt{Redeem}}$, $\mathtt{tx}_{\mathtt{Send}}$, $\mathtt{tx}_{\mathtt{Deposit}}$. By Lemma 3 (see below), $|\mathsf{Adv}^{\Game_{\mathsf{sim}}} - \mathsf{Adv}^{\Game_3}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{COMM}}$.

As mentioned above, we can obtain $\mathcal{A}$'s advantage in the L-IND experiment (i.e., $\Game_{\mathsf{real}}$) by summing over $\mathcal{A}$'s advantages in all games as follows:

$$\mathsf{Adv}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) \leq 2 \cdot q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}} + (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot (\mathsf{Adv}^{\mathtt{PRF}} + \mathsf{Adv}^{\mathtt{COMM}}).$$

Since $\mathsf{Adv}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr\left[\mathsf{BlockMaze}^{\mathtt{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - 1$ and $\mathcal{A}$'s advantage in the L-IND experiment is negligible in λ, we can conclude that the proof of ledger indistinguishability.

Lemma 1. *Let* $\mathsf{Adv}^{\Pi_{\mathcal{E}}}$ *be* $\mathcal{A}$*'s advantage in* $\Pi_{\mathcal{E}}$*'s* IND-CCA *and* IK-CCA *experiments. If* $\mathcal{A}$ *issues* q_{S} ***Send*** *queries, then* $|\mathsf{Adv}^{\Game_2} - \mathsf{Adv}^{\Game_1}| \leq 2 \cdot q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}}$*.*

Proof. We utilize a hybrid $\Game_{\mathsf{H}}$ as an intermediation between $\Game_1$ and $\Game_2$ to prove that $\mathsf{Adv}^{\Game_2}$ is at most negligibly different than $\mathsf{Adv}^{\Game_1}$.

More precisely, $\Game_{\mathsf{H}}$ is the same as $\Game_1$ with one modification: $\mathcal{C}$ utilizes a new public key generated by the $\Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp}_{\mathcal{E}})$, instead of the public key created by a previous query to ***CreateAccount***, to encrypt the same plaintext. After q_{CA} ***CreateAccount*** queries, $\mathcal{A}$ queries the IK-CCA challenger to obtain $pk_{\mathcal{E}} := pk_{\mathcal{E},0}$ where $pk_{\mathcal{E},0}$ is the public key in $(pk_{\mathcal{E},0}, pk_{\mathcal{E},1})$ provided by the IK-CCA challenger. At each ***Send*** query, the IK-CCA challenger encrypts the corresponding plaintext pt as $\mathsf{ct}^* := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk_{\mathcal{E},\bar{b}}}(\mathsf{pt})$, where $\bar{b}$ is the bit chosen by the IK-CCA challenger, in response to $\mathcal{A}$. Then $\mathcal{C}$ sets $\mathsf{ct} := \mathsf{ct}^*$ and adds $\mathtt{tx}_{\mathtt{Send}}$ (whose aux_A is set by ct) to the ledger L. Finally, $\mathcal{A}$ outputs a guess bit b', which is regarded as the guess in the IK-CCA experiment. Thus, if $\bar{b} = 0$ then $\mathcal{A}$'s view represents $\Game_1$, while $\mathcal{A}$'s view represents $\Game_{\mathsf{H}}$ if $\bar{b} = 1$. Note that $\mathcal{A}$ issues q_{S} ***Send*** queries, then he obtains the q_{S} ciphertexts at most. If the maximum adversarial advantage against the IK-CCA experiment is $\mathsf{Adv}^{\Pi_{\mathcal{E}}}$, then we can conclude that $|\mathsf{Adv}^{\Game_{\mathsf{H}}} - \mathsf{Adv}^{\Game_1}| \leq q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}}$.

In a similar vein, $\Game_2$ is the same as $\Game_{\mathsf{H}}$ with one modification: $\mathcal{C}$ utilizes a random string of the appropriate length in the plaintext space to replace the plaintext computed in the ***Send*** query. For simplicity, we omit the formal description for IND-CCA experiment, which has a similar pattern above. If the maximum adversarial advantage against the IND-CCA experiment is $\mathsf{Adv}^{\Pi_{\mathcal{E}}}$, then we can conclude that $|\mathsf{Adv}^{\Game_2} - \mathsf{Adv}^{\Game_{\mathsf{H}}}| \leq q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}}$. Thus, we can sum the above $\mathcal{A}$'s advantages to obtain $|\mathsf{Adv}^{\Game_2} - \mathsf{Adv}^{\Game_1}| \leq 2 \cdot q_{\mathsf{S}} \cdot \mathsf{Adv}^{\Pi_{\mathcal{E}}}$. □

Lemma 2. *Let* $\mathsf{Adv}^{\mathtt{PRF}}$ *be* $\mathcal{A}$*'s advantage in distinguishing* PRF *from a random function. If* $\mathcal{A}$ *issues* q_{M} ***Mint*** *queries,* q_{R} ***Redeem*** *queries,* q_{S} ***Send*** *queries and* q_{D} ***Deposit*** *queries, then* $|\mathsf{Adv}^{\Game_3} - \mathsf{Adv}^{\Game_2}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{PRF}}$*.*

Proof. We utilize a hybrid $\Game_{\mathsf{H}}$ as an intermediation between $\Game_2$ and $\Game_3$ to prove that $\mathsf{Adv}^{\Game_3}$ is at most negligibly different than $\mathsf{Adv}^{\Game_2}$.

More precisely, $\Game_{\mathsf{H}}$ is the same as $\Game_2$ with one modification: $\mathcal{C}$ utilizes a random string of the appropriate length to replace the public key pk associated with the returned $addr$ for $\mathcal{A}$'s first ***CreateAccount*** query; then, on each subsequent ***Mint***, ***Redeem***, ***Send***, ***Deposit*** queries, $\mathcal{C}$ replaces sn_A respectively with a random string of appropriate length and simulates the respective zk-SNARK proof (e.g., $prf_m, prf_r, prf_s, prf_d$) with the help of a trapdoor by the simulator $\mathcal{S}$ for $\mathtt{tx}_{\mathtt{Mint}}$, $\mathtt{tx}_{\mathtt{Redeem}}$, $\mathtt{tx}_{\mathtt{Send}}$, $\mathtt{tx}_{\mathtt{Deposit}}$.

Let sk be the random, secret key created by the first ***CreateAccount*** query and employed in $\mathtt{PRF}$ to compute $sn^*_A := \mathtt{PRF}(sk_A, r^*_A)$ and $sn_v := \mathtt{PRF}(sk_A, r_v)$ in **Mint**, **Redeem**, **Send**, **Deposit** algorithm. Note that $\mathtt{PRF}$ takes different random number r to publish old serial number sn for different transactions (generated by the same account). Moreover, let $\mathcal{O}$ be an oracle that implements either $\mathtt{PRF}$ or a random function. Then, we utilize $\mathcal{O}$ to generate all random strings (i.e., sn) for the two cases of $\mathcal{O}$ in response to a distinguisher (as an experiment). If $\mathcal{O}$ implements a random function, then it represents $\Game_{\mathsf{H}}$, while the experiment represents $\Game_2$ if $\mathcal{O}$ implements $\mathtt{PRF}$. Thus, $\mathcal{A}$'s advantage in distinguishing $\mathtt{PRF}$ from a random one is at most $\mathsf{Adv}^{\mathtt{PRF}}$.

In a similar vein, we extend the above pattern to all $q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}$ oracle-generated serial numbers (corresponding to what happens in $\Game_3$). We can obtain that $\mathcal{A}$'s advantage in distinguishing $\mathtt{PRF}$ from a random one is at most $(q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{PRF}}$. Finally, we deduce that $|\mathsf{Adv}^{\Game_3} - \mathsf{Adv}^{\Game_2}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{PRF}}$. □

Lemma 3. Let $\mathsf{Adv}^{\mathtt{COMM}}$ be $\mathcal{A}$'s advantage against the hiding property of $\mathtt{COMM}$. If $\mathcal{A}$ issues q_{M} ***Mint*** queries, q_{R} ***Redeem*** queries, q_{S} ***Send*** queries and q_{D} ***Deposit*** queries, then $|\mathsf{Adv}^{\Game_{\mathsf{sim}}} - \mathsf{Adv}^{\Game_3}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{COMM}}$.

Proof. This proof can be proved with the similar pattern used in Lemma 2 above. On each ***Mint***, ***Redeem***, ***Send***, ***Deposit*** query, $\mathcal{C}$ replaces the balance commitment $cmt^* := \mathtt{COMM}_{\mathtt{bc}}(addr, value, sn^*_A, r^*)$ and the transfer commitment $cmt_v := \mathtt{COMM}_{\mathtt{tc}}(addr, v, pk, sn_v, r_v, sn_A)$ with random strings of the appropriate length. Thus $\mathcal{A}$'s advantage in distinguishing this modified experiment from $\Game_3$ is at most $\mathsf{Adv}^{\mathtt{COMM}}$. If we extend it to all q_{M} ***Mint*** queries, all q_{R} ***Redeem*** queries, all q_{S} ***Send*** queries and all q_{D} ***Deposit*** queries, and utilize random strings of the suitable length to replace $q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}$ commitments (i.e., cmt_A and cmt_v), then we obtain $\Game_{\mathsf{sim}}$, and conclude that $|\mathsf{Adv}^{\Game_{\mathsf{sim}}} - \mathsf{Adv}^{\Game_3}| \leq (q_{\mathsf{M}} + q_{\mathsf{R}} + 2 \cdot q_{\mathsf{S}} + q_{\mathsf{D}}) \cdot \mathsf{Adv}^{\mathtt{COMM}}$. □

C.2 Proof of Transaction Unlinkability

Letting $\mathcal{T}$ be the table of zero-knowledge transactions (i.e., $\mathtt{tx}_{\mathtt{Send}}$ and $\mathtt{tx}_{\mathtt{Depoist}}$) generated by $\mathcal{O}^{\mathsf{BM}}$ in response to ***Send*** and ***Deposit*** queries. $\mathcal{A}$ wins the $\mathtt{TR\text{-}UL}$ experiment whenever it outputs a pair of zero-knowledge transactions $(\mathtt{tx}, \mathtt{tx}')$ if one of the following conditions holds: (i) the recipients of payments contained in $(\mathtt{tx}, \mathtt{tx}')$ are the same if $\mathtt{tx} = \mathtt{tx}_{\mathtt{Send}}$; and (ii) the senders of payments contained in $(\mathtt{tx}, \mathtt{tx}')$ are the same if $\mathtt{tx} = \mathtt{tx}_{\mathtt{Deposit}}$.

Suppose $\mathcal{A}$ outputs a pair of $\mathtt{Send}$ transactions $(\mathtt{tx}_{\mathtt{Send}}, \mathtt{tx}'_{\mathtt{Send}})$, where $\mathtt{tx}_{\mathtt{Send}}$ satisfies the following equations:

$$\begin{aligned}
&\mathtt{tx}_{\mathtt{Send}} := (addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)\\
&cmt_v = \mathtt{COMM}_{\mathtt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A)\\
&aux_A := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk_B}(\{v, sn_v, r_v, sn_A\}),
\end{aligned}$$

and $\mathtt{tx}'_{\mathtt{Send}}$ satisfies the following equations:

$$\begin{aligned}
&\mathtt{tx}'_{\mathtt{Send}} := (addr'_A, sn'_A, {cmt^*_A}', cmt'_v, aux'_A, auth'_{enc}, prf'_s)\\
&cmt'_v = \mathtt{COMM}_{\mathtt{tc}}(addr'_A, v', pk'_B, sn'_v, r'_v, sn'_A)\\
&aux'_A := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk'_B}(\{v', sn'_v, r'_v, sn'_A\}).
\end{aligned}$$

$\mathcal{A}$ wins the $\mathtt{TR\text{-}UL}$ experiment if the recipients of payments contained in $(\mathtt{tx}_{\mathtt{Send}}, \mathtt{tx}'_{\mathtt{Send}})$ are the same, i.e., $pk_B = pk'_B$. There are three ways for $\mathcal{A}$ to distinguish whether $pk_B \overset{?}{=} pk'_B$: (i) distinguish the public keys from the ciphertexts; (ii) distinguish the public keys from the transfer commitments; (iii) distinguish the public keys from the zero-knowledge proofs.

For condition (i), $\mathcal{A}$ must distinguish (pk_B, pk'_B) based on the different ciphertexts (aux_A, aux'_A), which indicates that $\mathcal{A}$ should win the $\mathsf{IK\text{-}CCA}$ experiment in Lemma 1. For condition (ii), $\mathcal{A}$ must distinguish (pk_B, pk'_B) from the different commitments (cmt_v, cmt'_v) without knowing other secret values (i.e., hidden variables), which indicates that $\mathcal{A}$ should break the *hiding* property of $\mathtt{COMM}$ in Lemma 3. For condition (iii), $\mathcal{A}$ must distinguish (pk_B, pk'_B) from different zero-knowledge proof (prf_s, prf'_s), which indicates that $\mathcal{A}$ should break the *proof of knowledge* property of the zk-SNARK. However, since the security of Lemma 1, Lemma 3 and zk-SNARK, $\mathcal{A}$ cannot distinguish the two recipients (i.e., public keys) from (aux_A, aux'_A) and (cmt_v, cmt'_v).

Suppose $\mathcal{A}$ outputs a pair of $\mathtt{Deposit}$ transactions $(\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'_{\mathtt{Deposit}})$, where $\mathtt{tx}_{\mathtt{Deposit}}$ satisfies the following equations:

$$\begin{aligned}
&\mathtt{tx}_{\mathtt{Deposit}} := (seq, rt_{cmt}, sn_B, cmt^*_B, sn_v, pk_B, prf_d)\\
&cmt_v = \mathtt{COMM}_{\mathtt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A),
\end{aligned}$$

and $\mathtt{tx}'_{\mathtt{Deposit}}$ satisfies the following equations:

$$\begin{aligned}
&\mathtt{tx}'_{\mathtt{Deposit}} := (seq', rt'_{cmt}, sn'_B, {cmt^*_B}', sn'_v, pk'_B, prf'_d)\\
&cmt'_v = \mathtt{COMM}_{\mathtt{tc}}(addr'_A, v', pk'_B, sn'_v, r'_v, sn'_A)
\end{aligned}$$

$\mathcal{A}$ wins the $\mathtt{TR\text{-}UL}$ experiment if the senders of payments contained in $(\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'_{\mathtt{Deposit}})$ are the same, i.e., $addr_A = addr'_A$. There are two ways for $\mathcal{A}$ to distinguish whether $addr_A \overset{?}{=} addr'_A$: (i) distinguish the address of senders from the zero-knowledge proof; (ii) obtain the transfer commitments used in $\mathtt{Deposit}$ transactions from $\mathcal{A}$'s view, and distinguish the address of senders combining with previous $\mathtt{Send}$ transactions.

For condition (i), $\mathcal{A}$ must distinguish $(addr_A, addr'_A)$ from different zero-knowledge proof (prf_d, prf'_d), which indicates that $\mathcal{A}$ should break the *proof of knowledge* property of the zk-SNARK. For condition (ii), if $\mathcal{A}$ can analyze (cmt_v, cmt'_v) without knowing other secret values (i.e., hidden variables), then, he can distinguish $(addr_A, addr'_A)$ by seeking senders of the previous transactions $(\mathtt{tx}_{\mathtt{Send}}, \mathtt{tx}'_{\mathtt{Send}})$ containing (cmt_v, cmt'_v) respectively. Note that (cmt_v, cmt'_v) are not visible for anyone during the generation of $(\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'_{\mathtt{Deposit}})$. There are two ways for $\mathcal{A}$ to obtain different transfer commitments: (a) the Merkle roots; (b) the zero-knowledge proofs. For condition (a), $\mathcal{A}$ must analyze (cmt_v, cmt'_v) from different Merkle roots (rt_{cmt}, rt'_{cmt}), which indicates that $\mathcal{A}$ should break the collision resistance property of CRH. For condition (b), $\mathcal{A}$ must analyze (cmt_v, cmt'_v) from different zero-knowledge proofs (prf_d, prf'_d), which indicates that $\mathcal{A}$ should break the *proof of knowledge* property of the zk-SNARK. However, since the security of zk-SNARK and CRH, $\mathcal{A}$ cannot neither distinguish the two senders from (prf_d, prf'_d) nor analyze (cmt_v, cmt'_v) from (rt_{cmt}, rt'_{cmt}) and (prf_d, prf'_d).

To conclude, due to the security of zk-SNARK, CRH, COMM and encryption schemes, the adversary $\mathcal{A}$ cannot distinguish the unlinkability from $(\mathtt{tx}_{\mathtt{Send}}, \mathtt{tx}'_{\mathtt{Send}})$ nor $(\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'_{\mathtt{Deposit}})$.

C.3 Proof of Transaction Non-malleability

Letting $\mathcal{T}$ be the table of four types of zero-knowledge transactions generated by $\mathcal{O}^{\mathrm{BM}}$ in response to corresponding queries. $\mathcal{A}$ wins the TR-NM experiment whenever it outputs a zero-knowledge transactions $\mathtt{tx}'$ if there exists $\mathtt{tx} \in \mathcal{T}$ satisfying all following conditions: (i) $\mathtt{tx}' \neq \mathtt{tx}$; (ii) both $\mathtt{tx}'$ and $\mathtt{tx}$ published the same serial number; and (iii) $\mathbf{VerTx}(L, \mathsf{pp}, \mathtt{tx}') = 1$, where L is the current ledger containing $\mathtt{tx}$.

We first consider the case that an adversary sends a Send transaction $\mathtt{tx}$, which can be represented in the form $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$; set $h_{enc} := \mathtt{CRH}(aux_A)$. Let aux'_A be the corresponding ciphertext in $\mathtt{tx}'$ and set $h'_{enc} := \mathtt{CRH}(aux'_A)$. Intuitively, transaction non-malleability of our scheme can be guaranteed by considering IND-CCA experiment in Lemma 1 and the non-malleability of the underlying zk-SNARK. If $\mathcal{A}$ can produce $\mathtt{tx}'$ using the same serial number revealed in $\mathtt{tx}$, then $\mathtt{tx}'$ and $\mathtt{tx}$ would have either different proof or ciphertext. For the ciphertext aux_A, $\mathcal{A}$ cannot modify it without changing its underlying plaintext since the encryption scheme used has IND-CCA-security. Moreover, the plaintext modification means that the statements for the zero-knowledge proof are changed, which result in a new proof prf'_s that must be provided. That is because any adversary who provides a proof for an instance $\vec{x}_3$, such that a pair $(\vec{x}_3, prf'_s)$ has not been computed before, has to know a corresponding witness $\vec{a}_3$ because of the simulation-extractability of the zk-SNARK.

Formally, define $\mathcal{Q}_{\mathsf{CA}} := \{sk_1, \cdots, sk_{q_{\mathsf{CA}}}\}$, $\mathcal{Q}_{\mathsf{S}} = \{aux_{A,1}, \cdots, aux_{A,q_{\mathsf{S}}}\}$, and $\epsilon := \mathsf{Adv}^{\mathrm{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda)$. $\mathcal{Q}_{\mathsf{CA}}$ is the set of account private keys in response to $\mathcal{A}$'s ***CreateAccount*** queries, and $\mathcal{Q}_{\mathsf{S}}$ is the set of the ciphertexts in response to $\mathcal{A}$'s ***Send*** queries for Send transactions. Then we talk about the cases in which $\mathcal{A}$ wins in the following:

- $case$-1: there is $aux''_A \in \mathcal{Q}_{\mathsf{S}}$ such that $h_{enc} := \mathtt{CRH}(aux''_A)$.
- $case$-2: the above case does not occur, and $auth_{enc} \neq \mathtt{PRF}(sk, h_{enc})$ for all $sk \in \mathcal{Q}_{\mathsf{CA}}$.

Define $\epsilon_1 := \Pr[case\text{-}1]$ and $\epsilon_2 := \Pr[case\text{-}2]$. Obviously, $\epsilon = \epsilon_1 + \epsilon_2$.

If $case$-1 occurs, then $\mathcal{A}$ can find a Send transaction $\mathtt{tx}$ that contains a ciphertext aux from $\mathcal{T}$, and output another transaction $\mathtt{tx}'$ containing a ciphertext aux' which requires that (i) $aux \neq aux''$, but (ii) $\mathtt{CRH}(aux) = \mathtt{CRH}(aux'')$. This means, in particular, that $\mathcal{A}$ finds collisions for CRH with probability ϵ_1. However, since the CRH used is a collision-resistant hash function, ϵ_1 must be negligible in λ.

To argue that ϵ_2 is negligible in λ, we let $\mathcal{E}$ be the zk-SNARK witness extractor for $\mathcal{A}$. Based on the fact that PRF is collision resistant, we want to construct an algorithm $\mathcal{B}$ to contradict this fact like Zerocash [4]. The algorithm $\mathcal{B}$ is set to find collisions for PRF with non-negligible probability as follows:

1) $\mathcal{A}$ finds transactions from $\mathcal{T}$ and outputs $\mathtt{tx}'$.
2) Parse $\mathtt{tx}'$ as $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
3) Run $\mathcal{E}(\mathsf{pk}_{\mathcal{Z}_{\mathtt{Send}}}, \mathsf{vk}_{\mathcal{Z}_{\mathtt{Send}}})$ to extract a witness $\vec{a_3}$ for prf_s.
4) Set $\vec{x_3} := (cmt_A, addr_A, sn_A, cmt_v, cmt^*_A, h_{enc}, auth_{enc})$, and judge whether $\vec{a_3}$ is a valid witness for $\vec{x_3}$ or not. If $\vec{a_3}$ is invalid, then abort and output 0.
5) Parse $\vec{a_3}$ as $(value_A, r_A, sk_A, v, pk_B, sn_v, r_v, sn^*_A, r^*_A)$. Since $\vec{a_3}$ is a valid witness, $sn_A = \mathtt{PRF}(sk_A, r_A)$.
6) If there exists $\mathtt{tx} \in \mathcal{T}$ containing sn_A, then let $\bar{sk}$ and $\bar{r}$ be the values used to compute sn_A in $\mathtt{tx}$ (i.e., $sn_A = \mathtt{PRF}(\bar{sk}, \bar{r})$). If $sk_A \neq \bar{sk}$, output $((sk_A, r_A), (\bar{sk}, \bar{r}))$ as a collision for PRF. Otherwise, output 0.

Thus, when $case$-2 occurs, the following conditions hold:

- the serial number sn_A appears in a previous Send transaction $\mathtt{tx} \in \mathcal{T}$;
- the proof prf_s is valid and, with all but negligible probability, the extractable witness $\vec{a_3}$ is valid;
- since the witness $\vec{a_3}$ is valid, this means that $auth_{enc} = \mathtt{PRF}(sk_A, h_{enc})$. As the $case$-2 occurs, however, it cannot be that $sk_A = \bar{sk}$.

Finally, in a similar vein, we can utilize the similar structure of the argument to discuss the $case$-2 for the other three transactions (i.e., Mint, Redeem, and Deposit transaction) generated by $\mathcal{O}^{\mathrm{BM}}$ in response to ***Mint***, ***Redeem*** and ***Deposit*** queries. Overall, we conclude that $\mathcal{B}$ finds a collision for PRF with probability $\epsilon_2 - \mathrm{negl}(\lambda)$.

C.4 Proof of Balance

The BAL experiment is changed without affecting $\mathcal{A}$'s view as follows: for each Deposit transaction $\mathtt{tx}_{\mathtt{Deposit}}$ on the ledger L, $\mathcal{C}$ computes the zk-SNARK proof $prf_d := \Pi_{\mathcal{Z}}.\mathrm{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_4, \vec{a}_4)$ where $\vec{x}_4$ is a statement corresponding to $\mathtt{tx}_{\mathtt{Deposit}}$ and $\vec{a}_4$ is a witness for the zk-SNARK instance $\vec{x}_4$, then $\mathcal{C}$ organizes all $(tx, \vec{a})$ as an augmented ledger $(L, \boldsymbol{A})$, which is a list of matched pairs $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a}_{4j})$ where $\mathtt{tx}_{\mathtt{Deposit}}$ is the j-th Deposit transaction in L and $\vec{a}_{4j} \in \boldsymbol{A}$ is the witness of $\mathtt{tx}_{\mathtt{Deposit}}$ for $\vec{x}_{4j}$.

Definition 9 (Balanced ledger)**.** An augmented ledger $(L, \boldsymbol{A})$ is balanced if the following conditions hold:

- *Condition I.* Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ publishes openings (e.g., sn_B) of a unique balance commitment

cmt_B, which is the output of a previous zero-knowledge transaction before $\mathtt{tx}_{\mathtt{Deposit}}$ on L.

- *Condition II.* Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ publishes openings (e.g., sn_v and pk_B) of a unique transfer commitment cmt_v, which is the output of a previous $\mathtt{Send}$ transaction before $\mathtt{tx}_{\mathtt{Deposit}}$ on L.
- *Condition III.* For all $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ publish distinct and unique openings (e.g., sn_v, sn_B and pk_B) of the commitment (i.e., cmt_v and cmt_B).
- *Condition IV.* Each cmt_B, cmt_v, cmt_B^* to values $value_B$, v, $value_B^*$ (respectively) contained in $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ satisfies the condition that $value_B + v = value_B^*$.
- *Condition V.* The values (i.e., $addr_A$, v, pk_B, sn_v, r_v, sn_A) used to compute cmt_v are respectively equal to the values for cmt'_v, if $cmt_v = cmt'_v$ where cmt_v is employed in $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$, and cmt'_v is the output of a previous $\mathtt{Send}$ transaction before $\mathtt{tx}_{\mathtt{Deposit}}$.
- *Condition VI.* If $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ was inserted by $\mathcal{A}$, and cmt_v used in $\mathtt{tx}_{\mathtt{Deposit}}$ is the output of a previous $\mathtt{Send}$ transaction $\mathtt{tx}'$, then $addr \notin \mathtt{ACCOUNT}$ where $addr := \mathtt{CRH}(pk_B)$ is the recipient's account address of $\mathtt{tx}'$.

One can prove that $(L, \boldsymbol{A})$ is balanced if the following equation holds: $v_{\mathtt{zk\_unspent}} + v_{\mathtt{pt\_unspent}} + v_{\mathtt{zk}:\mathcal{A}\to\mathtt{ACCOUNT}} + v_{\mathtt{pt}:\mathcal{A}\to\mathtt{ACCOUNT}} = v_{\mathtt{zk}:\mathtt{ACCOUNT}\to\mathcal{A}} + v_{\mathtt{pt}:\mathtt{ACCOUNT}\to\mathcal{A}}$.

For each of the above conditions, we utilize a contraction to prove that each case is in a negligible probability. Note that we suppose $\Pr\left[\mathcal{A}(\textit{Con-i}) = 1\right]$ to denote a non-negligible probability that $\mathcal{A}$ wins but violates *Condition i*.

Violating *Condition I*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$, where $\mathtt{tx}_{\mathtt{Deposit}}$ was not inserted by $\mathcal{A}$, must satisfy *Condition I* in $\mathcal{O}^{\mathrm{BM}}$; thus, $\Pr\left[\mathcal{A}(\overline{\textit{Con-I}}) = 1\right]$ is a probability that $\mathcal{A}$ inserts $\mathtt{tx}_{\mathtt{Deposit}}$ to construct a pair $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ where cmt_B used in $\mathtt{tx}_{\mathtt{Deposit}}$ is not the output of any previous zero-knowledge transaction before $\mathtt{tx}_{\mathtt{Deposit}}$. However, each $\mathtt{tx}_{\mathtt{Deposit}}$ utilizes the witness $\vec{a_4}$, which must contain a balance commitment cmt_B for the serial number sn_B, to compute the proof proving the validity of $\mathtt{tx}_{\mathtt{Deposit}}$. Obviously, cmt_B is the output of an earlier zero-knowledge transaction and recorded on MPT in L. Therefore, if cmt_B to sn_B does not previously appear in L, then it means that this violation contradicts the binding property of $\mathtt{COMM}$ in Lemma 3.

Violating *Condition II*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$, where $\mathtt{tx}_{\mathtt{Deposit}}$ was not inserted by $\mathcal{A}$, must satisfy *Condition II* in $\mathcal{O}^{\mathrm{BM}}$; thus, $\Pr\left[\mathcal{A}(\overline{\textit{Con-II}}) = 1\right]$ is a probability that $\mathcal{A}$ inserts $\mathtt{tx}_{\mathtt{Deposit}}$ to construct a pair $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ where cmt_v used in $\mathtt{tx}_{\mathtt{Deposit}}$ is not the output of any previous $\mathtt{Send}$ transaction before $\mathtt{tx}_{\mathtt{Deposit}}$. However, each $\mathtt{tx}_{\mathtt{Deposit}}$ utilizes the witness $\vec{a_4}$, which must contain a authentication path $path$ for a Merkle tree, to compute the proof proving the validity of $\mathtt{tx}_{\mathtt{Deposit}}$. Obviously, the Merkle tree, whose root rt_{cmt} is published, is constructed using the transfer commitment cmt_v of any previous $\mathtt{Send}$ transactions on L. More precisely, if cmt_v does not previously appear in L, then it means that $path$ is invalid but with a valid rt_{cmt}. Therefore, this violation contradicts the collision resistance of $\mathtt{CRH}$.

Violating *Condition III*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ publishes a unique and distinct sn_v, $(sn_B,\ pk_B)$ for cmt_v, cmt_B (respectively). Obviously, $\Pr\left[\mathcal{A}(\overline{\textit{Con-III}}) = 1\right]$ is a probability that $\mathcal{A}$ wins in the following two situations: (i) spending the same balance commitment cmt_B in two zero-knowledge transactions; or (ii) receiving the same transfer commitment cmt_v. In (i), it means that two $\mathtt{Deposit}$ transactions $\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'$ recorded on L spend the same balance commitment cmt_B, but publish two different serial numbers sn_B and sn'_B, furthermore, their corresponding witnesses $\vec{a_4}, \vec{a_4}'$ must contain different openings of cmt_B since both transactions are valid $\mathtt{Deposit}$ transactions on L. Therefore, this violation contradicts the binding property of $\mathtt{COMM}$. In a similar vein, for (ii), it means that two $\mathtt{Deposit}$ transactions $\mathtt{tx}_{\mathtt{Deposit}}, \mathtt{tx}'$ receive the same transfer commitment cmt_v but publish two different serial numbers sn_v and sn'_v. Therefore, this violation also contradicts the binding property of $\mathtt{COMM}$ in Lemma 3.

Violating *Condition IV*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ contains a proof ensuring that cmt_B, cmt_v, cmt_B^* to values $value_B$, v, $value_B^*$ (respectively) satisfy the equation $value_B + v = value_B^*$. Obviously, $\Pr\left[\mathcal{A}(\overline{\textit{Con-IV}}) = 1\right]$ is a probability that the equation $value_B + v \neq value_B^*$ holds. Therefore, this violation contradicts the *proof of knowledge* property of the zk-SNARK.

Violating *Condition V*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ contains values (i.e., $addr_A$, v, pk_B, sn_v, r_v, sn_A) of cmt_v, and cmt_v is a also transfer commitment to values (i.e., $addr_A$, v', pk_B, sn_v, r_v, sn_A) in a previous $\mathtt{Send}$ transaction. Obviously, $\Pr\left[\mathcal{A}(\overline{\textit{Con-V}}) = 1\right]$ is a probability that the equation $v \neq v'$ holds. Therefore, this violation also contradicts the binding property of $\mathtt{COMM}$ in Lemma 3.

Violating *Condition VI*. Each $(\mathtt{tx}_{\mathtt{Deposit}}, \vec{a_4}) \in (L, \boldsymbol{A})$ publishes a recipient's account address $addr := \mathtt{CRH}(pk_B)$ of a transfer commitment cmt_v. Obviously, $\Pr\left[\mathcal{A}(\overline{\textit{Con-VI}}) = 1\right]$ is a probability that an inserted $\mathtt{Deposit}$ transaction $\mathtt{tx}_{\mathtt{Deposit}}$ publishes $addr$ of cmt_v which is the output of a previous $\mathtt{Send}$ transaction $\mathtt{tx}'$ whose recipient's account address $addr$ lies in $\mathtt{ACCOUNT}$; moreover, the witness associated to $\mathtt{tx}'$ contains pk such that $addr = \mathtt{CRH}(pk)$. Therefore, this violation contradicts the collision resistance of $\mathtt{CRH}$.

Finally, we utilize the similar structure of the argument to prove balance for the other three transactions (i.e., $\mathtt{Mint}$, $\mathtt{Redeem}$ and $\mathtt{Send}$) generated by $\mathcal{O}^{\mathrm{BM}}$ in response to ***Mint***, ***Redeem*** and ***Send*** queries, and obtain that $\Pr\left[\mathsf{BlockMaze}^{\mathtt{BAL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right]$ is negligible in λ.

Appendix C
Algorithms of BlockMaze schemes

For convenience, we summarize algorithms employed in a BlockMaze scheme Π = (**Setup**, **CreateAccount**, **Mint**, **Redeem**, **Send**, **Deposit**, **VerTx**) in Fig. 8.

Setup

The algorithm generates a list of public parameters.

- inputs: a security parameter λ
- outputs: public parameters pp

1) Compute $\mathsf{pp}_{\mathcal{E}} := \Pi_{\mathcal{E}}.\mathrm{Setup}(1^{\lambda})$.
2) Compute $\mathsf{pp}_{\mathcal{Z}} := \Pi_{\mathcal{Z}}.\mathrm{Setup}(1^{\lambda})$.
3) For each $i \in \{\mathtt{Mint}, \mathtt{Redeem}, \mathtt{Send}, \mathtt{Deposit}\}$
    a) Construct a circuit C_i.
    b) Compute $(\mathsf{pk}_{\mathcal{Z}_i}, \mathsf{vk}_{\mathcal{Z}_i}) := \Pi_{\mathcal{Z}}.\mathrm{KeyGen}(C_i)$.
4) Set $\mathsf{PK}_{\mathcal{Z}} := \bigcup \mathsf{pk}_{\mathcal{Z}_i}$ and $\mathsf{VK}_{\mathcal{Z}} := \bigcup \mathsf{vk}_{\mathcal{Z}_i}$.
5) Output $\mathsf{pp} := (\mathsf{pp}_{\mathcal{E}}, \mathsf{pp}_{\mathcal{Z}}, \mathsf{PK}_{\mathcal{Z}}, \mathsf{VK}_{\mathcal{Z}})$.

CreateAccount

The algorithm creates an account address and key pair for a user.

- inputs: public parameters pp
- outputs: $(addr, (sk, pk))$

1) Compute an account key pair $(sk, pk) := \Pi_{\mathcal{E}}.\mathrm{KeyGen}(\mathsf{pp}_{\mathcal{E}})$.
2) Compute an account address $addr := \mathtt{CRH}(pk)$.
3) Output the $(addr, (sk, pk))$.

Mint

This algorithm merges a plaintext amount with the current zero-knowledge balance of an account (say A).

- inputs:
    - public parameters pp
    - the current zero-knowledge balance $\mathtt{zk\_balance}_A$
    - the current plaintext balance $\mathtt{pt\_balance}_A$
    - account private key sk_A
    - a plaintext amount v to be converted into a zero-knowledge amount
- outputs:
    - the new zero-knowledge balance $\mathtt{zk\_balance}^*_A$
    - a $\mathtt{Mint}$ transaction $\mathtt{tx}_{\mathtt{Mint}}$

1) Return fail if $\mathtt{pt\_balance}_A < v$ or $v \leq 0$.
2) Parse $\mathtt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
3) Generate a new random number r^*_A.
4) Sample a new serial number $sn^*_A := \mathtt{PRF}(sk_A, r^*_A)$.
5) Compute $cmt^*_A := \mathtt{COMM}_{\mathtt{bc}}(addr_A, value_A + v, sn^*_A, r^*_A)$.
6) Set $\vec{x}_1 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
7) Set $\vec{a}_1 := (value_A, r_A, sk_A, sn^*_A, r^*_A)$.
8) Compute $prf_m := \Pi_{\mathcal{Z}}.\mathrm{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_1, \vec{a}_1)$.
9) Set $\mathtt{tx}_{\mathtt{Mint}} := (addr_A, v, sn_A, cmt^*_A, prf_m)$,
   $\mathtt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A + v, sn^*_A, r^*_A)$.
10) Output $\mathtt{zk\_balance}^*_A$ and $\mathtt{tx}_{\mathtt{Mint}}$.

Redeem

This algorithm converts a zero-knowledge amount back into the plaintext balance of an account (say A).

- inputs:
    - public parameters pp
    - the current zero-knowledge balance $\mathtt{zk\_balance}_A$
    - account private key sk_A
    - a plaintext amount v to be converted back from zero-knowledge balance
- outputs:
    - the new zero-knowledge balance $\mathtt{zk\_balance}^*_A$
    - a $\mathtt{Redeem}$ transaction $\mathtt{tx}_{\mathtt{Redeem}}$

1) Parse $\mathtt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
2) Return fail if $value_A < v$ or $v \leq 0$.
3) Generate a new random number r^*_A.
4) Sample a new serial number $sn^*_A := \mathtt{PRF}(sk_A, r^*_A)$.
5) Compute $cmt^*_A := \mathtt{COMM}_{\mathtt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A)$.
6) Set $\vec{x}_2 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
7) Set $\vec{a}_2 := (value_A, r_A, sk_A, sn^*_A, r^*_A)$.
8) Compute $prf_r := \Pi_{\mathcal{Z}}.\mathrm{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_2, \vec{a}_2)$.
9) Set $\mathtt{tx}_{\mathtt{Redeem}} := (addr_A, v, sn_A, cmt^*_A, prf_r)$,
   $\mathtt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A - v, sn^*_A, r^*_A)$.
10) Output $\mathtt{zk\_balance}^*_A$ and $\mathtt{tx}_{\mathtt{Redeem}}$.

Send

This algorithm sends a zero-knowledge amount from sender A to recipient B.

- inputs:
    - public parameters pp
    - the current zero-knowledge balance $\mathtt{zk\_balance}_A$
    - account private key sk_A
    - recipient's public key pk_B
    - a plaintext amount v to be transferred
- outputs:
    - the new zero-knowledge balance $\mathtt{zk\_balance}^*_A$
    - a $\mathtt{Send}$ transaction $\mathtt{tx}_{\mathtt{Send}}$

1) Return fail if $v \leq 0$.
2) Parse $\mathtt{zk\_balance}_A$ as $(cmt_A, addr_A, value_A, sn_A, r_A)$.
3) Generate a new random number r_v.
4) Sample a new serial number $sn_v := \mathtt{PRF}(sk_A, r_v)$.
5) Return fail if sn_v appears in $\mathtt{SNSet}_{\mathtt{T}}$.
6) Compute $cmt_v := \mathtt{COMM}_{\mathtt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A)$.
7) Set $aux_A := \Pi_{\mathcal{E}}.\mathrm{Enc}_{pk_B}(\{v, sn_v, r_v, sn_A\})$.
8) Generate a new random number r^*_A.
9) Sample a new serial number $sn^*_A := \mathtt{PRF}(sk_A, r^*_A)$.
10) Compute $cmt^*_A := \mathtt{COMM}_{\mathtt{bc}}(addr_A, value_A - v, sn^*_A, r^*_A)$.
11) Compute $h_{enc} := \mathtt{CRH}(aux_A)$.
12) Compute $auth_{enc} := \mathtt{PRF}(sk_A, h_{enc})$.
13) Set $\vec{x}_3 := (cmt_A, addr_A, sn_A, cmt_v, cmt^*_A, h_{enc}, auth_{enc})$.
14) Set $\vec{a}_3 := (value_A, r_A, sk_A, v, pk_B, sn_v, r_v, sn^*_A, r^*_A)$.
15) Compute $prf_s := \Pi_{\mathcal{Z}}.\mathrm{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_3, \vec{a}_3)$.
16) Set $\mathtt{tx}_{\mathtt{Send}} := (addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$,
    $\mathtt{zk\_balance}^*_A := (cmt^*_A, addr_A, value_A - v, sn^*_A, r^*_A)$.
17) Output $\mathtt{zk\_balance}^*_A$ and $\mathtt{tx}_{\mathtt{Send}}$.

Deposit

This algorithm makes a recipient (say B) check and deposit a received payment into his account.

- inputs:
    - the current ledger $\mathtt{Ledger}_{\mathtt{T}}$
    - public parameters pp
    - account key pair (sk_B, pk_B)
    - the hash of a send transaction $h_{\mathtt{tx}_{\mathtt{Send}}}$
    - the current zero-knowledge balance $\mathtt{zk\_balance}_B$
- outputs:
    - the new zero-knowledge balance $\mathtt{zk\_balance}^*_B$
    - a $\mathtt{Deposit}$ transaction $\mathtt{tx}_{\mathtt{Deposit}}$

1) Parse $\mathtt{zk\_balance}_B$ as $(cmt_B, addr_B, value_B, sn_B, r_B)$.
2) Obtain transaction information of $h_{\mathtt{tx}_{\mathtt{Send}}}$ from $\mathtt{Ledger}_{\mathtt{T}}$
    - the send transaction is $\mathtt{tx}_{\mathtt{Send}}$,
    - the block number of $\mathtt{tx}_{\mathtt{Send}}$ is N.
3) Parse $\mathtt{tx}_{\mathtt{Send}}$ as $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
4) Decrypt shared parameters $(v, sn_v, r_v, sn_A) := \Pi_{\mathcal{E}}.\mathrm{Dec}_{sk_B}(aux_A)$.
5) Return fail if sn_v appears in $\mathtt{SNSet}_{\mathtt{T}}$.
6) Return fail if $cmt_v \neq \mathtt{COMM}_{\mathtt{tc}}(addr_A, v, pk_B, sn_v, r_v, sn_A)$.
7) Randomly select a set $seq := \{n_1, n_2, ..., N, ..., n_9\}$ from existed block numbers.
8) Construct a Merkle tree $\mathtt{MT}$ over $\bigcup_{n \in seq} \mathtt{TCMSet}_{\mathtt{n}}$.
9) Compute $path := \mathtt{Path}(cmt_v)$ and rt_{cmt} over $\mathtt{MT}$.
10) Generate a new random number r^*_B.
11) Sample a new serial number $sn^*_B := \mathtt{PRF}(sk_B, r^*_B)$.
12) Compute $cmt^*_B := \mathtt{COMM}_{\mathtt{bc}}(addr_B, value_B + v, sn^*_B, r^*_B)$.
13) Set $\vec{x}_4 := (pk_B, sn_v, rt_{cmt}, cmt_B, addr_B, sn_B, cmt^*_B)$,
    $\vec{a}_4 := (cmt_v, addr_A, v, r_v, sn_A, value_B, r_B, sk_B, sn^*_B, r^*_B, path)$.
14) Compute $prf_d := \Pi_{\mathcal{Z}}.\mathrm{GenProof}(\mathsf{PK}_{\mathcal{Z}}, \vec{x}_4, \vec{a}_4)$.
15) Set $\mathtt{tx}_{\mathtt{Deposit}} := (seq, rt_{cmt}, sn_B, cmt^*_B, sn_v, pk_B, prf_d)$,
    $\mathtt{zk\_balance}^*_B := (cmt^*_B, addr_B, value_B + v, sn^*_B, r^*_B)$.
16) Output $\mathtt{zk\_balance}^*_B$ and $\mathtt{tx}_{\mathtt{Deposit}}$.

VerTx

This algorithm checks the validity of all zero-knowledge transactions.

- inputs:
    - the current ledger $\mathtt{Ledger}_{\mathtt{T}}$
    - public parameters pp
    - a zero-knowledge transaction $\mathtt{tx}$
- outputs: bit b

1) If given a transaction $\mathtt{tx}$ is $\mathtt{tx}_{\mathtt{Mint}}$
    a) Parse $\mathtt{tx}_{\mathtt{Mint}}$ as $(addr_A, v, sn_A, cmt^*_A, prf_m)$.
    b) Obtain related information of $addr_A$ from $\mathtt{Ledger}_{\mathtt{T}}$
        - the current plaintext balance is $\mathtt{pt\_balance}_A$,
        - the current balance commitment is cmt_A.
    c) Return 0 if $\mathtt{pt\_balance}_A < v$ or $v \leq 0$.
    d) Return 0 if sn_A appears in $\mathtt{SNSet}_{\mathtt{T}}$.
    e) Set $\vec{x}_1 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
    f) Output $b := \Pi_{\mathcal{Z}}.\mathrm{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_1, prf_m)$.
2) If given a transaction $\mathtt{tx}$ is $\mathtt{tx}_{\mathtt{Redeem}}$
    a) Parse $\mathtt{tx}_{\mathtt{Redeem}}$ as $(addr_A, v, sn_A, cmt^*_A, prf_r)$.
    b) Return 0 if $v \leq 0$.
    c) Obtain related information of $addr_A$ from $\mathtt{Ledger}_{\mathtt{T}}$
        - the current balance commitment is cmt_A.
    d) Return 0 if sn_A appears in $\mathtt{SNSet}_{\mathtt{T}}$.
    e) Set $\vec{x}_2 := (cmt_A, addr_A, sn_A, cmt^*_A, v)$.
    f) Output $b := \Pi_{\mathcal{Z}}.\mathrm{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_2, prf_r)$.
3) If given a transaction $\mathtt{tx}$ is $\mathtt{tx}_{\mathtt{Send}}$
    a) Parse $\mathtt{tx}_{\mathtt{Send}}$ as $(addr_A, sn_A, cmt^*_A, cmt_v, aux_A, auth_{enc}, prf_s)$.
    b) Obtain related information of $addr_A$ from $\mathtt{Ledger}_{\mathtt{T}}$
        - the current balance commitment is cmt_A.
    c) Return 0 if sn_A appears in $\mathtt{SNSet}_{\mathtt{T}}$.
    d) Compute $h_{enc} := \mathtt{CRH}(aux_A)$.
    e) Set $\vec{x}_3 := (cmt_A, addr_A, sn_A, cmt_v, cmt^*_A, h_{enc}, auth_{enc})$.
    f) Output $b := \Pi_{\mathcal{Z}}.\mathrm{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_3, prf_s)$.
4) If given a transaction $\mathtt{tx}$ is $\mathtt{tx}_{\mathtt{Deposit}}$
    a) Parse $\mathtt{tx}_{\mathtt{Deposit}}$ as $(seq, rt_{cmt}, sn_B, cmt^*_B, sn_v, pk_B, prf_d)$.
    b) Compute $addr_B := \mathtt{CRH}(pk_B)$.
    c) Obtain related information of $addr_B$ from $\mathtt{Ledger}_{\mathtt{T}}$
        - the current balance commitment is cmt_B.
    d) Return 0 if sn_B or sn_v appears in $\mathtt{SNSet}_{\mathtt{T}}$.
    e) Return 0 if rt_{cmt} is not the Merkle root over $\bigcup_{n \in seq} \mathtt{TCMSet}_{\mathtt{n}}$.
    f) Set $\vec{x}_4 := (pk_B, sn_v, rt_{cmt}, cmt_B, addr_B, sn_B, cmt^*_B)$.
    g) Output $b := \Pi_{\mathcal{Z}}.\mathrm{VerProof}(\mathsf{VK}_{\mathcal{Z}}, \vec{x}_4, prf_d)$.

Fig. 8: BlockMaze main algorithms