A General Zero-Knowledge Protocol for Blockchain Transactions

Mystiko.Network

March 26, 2024[1]

Abstract

Blockchain exposes all users' transaction data to the public, including account balances, asset holdings, trading history, etc. Such data exposure leads to potential security risks that restrict blockchain from broader adoption. Although some existing projects focus on single-chain confidential payment, no existing cross-chain system supports zero-knowledge transactions yet, which is incompatible with regulations such as GDPR. Also, current confidential payment systems require users to pay high extra fees. However, an anonymous protocol encrypting all transaction data raises concerns about malicious and illegal activities since the protocol is difficult to audit. We need to balance anonymous and auditability in blockchain.

We propose an auditable and affordable protocol for cross-chain and single-chain transactions. This protocol leverages zero-knowledge proofs to encrypt transactions and perform validation without disclosing sensitive users' data. To meet regulations, each auditor from an auditing committee will have an encrypted secret share of the transaction data. Auditors may view the zero-knowledge transaction data only if a majority of the committee agrees to decrypt the data. We employ a ZK-rollup scheme by processing multiple transactions in batches, which reduces zero-knowledge transaction costs to 90% lower compared with solutions without ZK-rollup. We implemented the proposed scheme using Zokrates and Solidity and evaluated the protocol on the Ethereum test network, and the total one-to-one zero-knowledge transactions cost only 5 seconds. We also proved the security of the protocol utilizing the standard real/ideal world paradigm.

1 Introduction

Blockchain exposes all user transaction data to the public, including account balances, asset holdings, trade activity, etc., due to its transparency and traceability features. Such data exposure creates possible security problems, which prevent blockchain from being adopted more widely. Additionally, the blockchain ecosystem is built on interoperability (connectivity) across various blockchains and blockchain applications. Hundreds of cross-chain bridges have formed, serving as a conduit for users to transfer assets from one chain to another, accumulating billions of dollars in inter-chain trading volumes. However, no cross-chain bridge currently in existence has allowed zero-knowledge transactions that safeguard users' trade histories.

Although there exist single chain zero-knowledge transaction solutions, they are not particularly user-friendly. In the market, single chain coin mixing systems have already been tested. The user experience of such systems, however, is extremely restricted because customers not only bear high transaction charges but also run into difficulties when transferring assets between chains. In order to conduct secret cross-chain transactions using such solutions, a user must first use single-chain coin mixers on the few supported chains before using a bridge to complete the cross-chain operation. Zero-knowledge transactions typically have expensive gas fee. Users must pay an additional gas fee if they activate zero-knowledge features. Users are facing difficulties to using zero-knowledge services because of the very high gas fee for cross-chain zero-knowledge transactions.

The Office of Foreign Assets Control (OFAC) of the U.S. Department of the Treasury sanctioned Tornado Cash on August 8th, 2022, citing it as "a substantial danger to national security." Regulators' primary worries may stem from the lack of visibility of suspicious transaction flows, or the source and destination of possible "laundered money," as well as from their inability to distinguish between the assets of criminal actors and users who have reasonable expectations of zero-knowledge. There are, nevertheless, a lot of good reasons why someone would desire to employ zero-knowledge features. For instance, a worker who receives cryptocurrency payments from their employer might not want their employer to be aware of all of their financial information. Investors in NFTs might not want to be the subject of robbery or harassment in the future. When donating money, donors sometimes desire to keep their identity a secret.

We offers a zero-knowledge proof based protocol solution to all cross chain bridges, which breaks the linkability between a user's deposit on a source chain and withdrawal on a destination chain. This protocol has an easy-to-plug-in feature that can be integrated into almost all major inter-chain bridges in the blockchain ecosystem. Moreover, this protocol offers a zero-knowledge proof based scaling solution to reduce such gas fees to an affordable level, i.e., ZK-rollup. To balance the right to protect users without illicit purpose and the compliance request of global regulators, we designed a decentralized auditing system for onchain zero-knowledge transactions.

1.1 Motivation

In a cross-chain or single-chain transaction, the source blockchain network will transfer coins to the destination blockchain network. In such a protocol, the sender and receiver addresses are in plain text, and therefore it could be possible to track the transaction graph. Many research works show that this setting cannot provide zero-knowledge [BBB$^+$18, PBF$^+$18]. To address this issue, we employ a similar solution as in Zcash [BSCG$^+$14, HBHW16]: the

transaction is encrypted with the public key of the receiver, and this receiver can then find the transaction and withdraw the coin. However, Zcash is a fork of Bitcoin [Nak08] and maintains a layer-1 blockchain. The network lacks integration with other blockchain networks, e.g., Ethereum [W$^+$14], BSC [Bin20], and Polygon [KNA21], and Zcash can support neither single-chain nor cross-chain transactions across those popular chains. Also, the transaction size in Zcash could be ten times higher than in Bitcoin, which hurts its scalability. The encrypted transaction could provide zero-knowledge; however, it could be abused for criminal purposes. Zcash has viewing keys that allow external viewers to track transactions, but this scheme can only disclose incoming transactions. In particular, auditors cannot view the sender's address and outgoing transactions from an address. We need to provide more complete audit features in which all in and out transactions are auditable.

Our contribution. We designed a confidential protocol for blockchain transactions supporting popular chains. It is noteworthy that the sender and receiver may be on the same chain, i.e., the user sends a single-chain transaction, and the source chain and the destination chain are on the same chain. Alternatively, the user can send a cross-chain transaction, and the source chain and the destination chain are on different chains. The protocol supports **JoinSplit** and allows internal transfers. We also employed the **ZK-rollups** scheme to increase the throughput of our protocol. We built a confidential protocol while making it auditable for auditors, in fact the protocol will not disclose users' transaction data unless a large enough partition of the auditors agrees so.

1.2 Protocol Overview

Suppose a user u on *Block A* want to send a coin valued v to u_1 on *Block B*, where v belongs to some default values $\mathbb{V}$. Let $PRF_x^{addr}(\cdot)$, $PRF_x^{sn}(\cdot)$ and $PRF_x^{pk}(\cdot)$ denote three pseudorandom functions for a seed x. Each user u_i generates an address key pair $(addr_{pk,i}, addr_{sk,i})$, where $addr_{pk,i} = (a_{pk,i}, pk_{enc,i})$ and $addr_{sk,i} = (a_{sk,i}, sk_{enc,i})$, and a nullifier key nk. $a_{pk,i}$ is generated as $PRF_{a_{sk}}^{addr}(0)$. nk is generated as $PRF_{a_{sk}}^{addr}(1)$.$(pk_{enc,i}, sk_{enc,i})$ are key-private encryption scheme. Here, we outline the protocol in three steps:

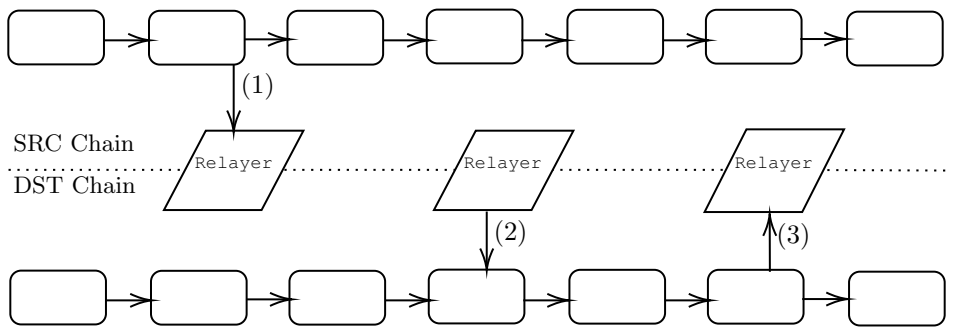
(1) u generates randomness r, s, and ρ, where ρ is the coin's serial number randomness. Let $COMM$ denote a commit scheme and E_{enc} denote a public-key encryption scheme. u commits the serial number in two steps (1) $k = COMM_r(a_{pk,1}||\rho)$ (2) $cm := COMM_s(v||k)$. Then, u computes the ciphertext $Ct = E_{enc}(pk_{enc}, v, \rho, r, s)$. The tuple (v, k, s, cm, Ct) is the new transaction $tx_{deposit}$. The ledger will keep a CRH(collision-resistant hash)-based Merkle tree $CMList$ of all committed serial numbers (cm). If cm is already in the ledger, the transaction will be rejected. Logically, the coin u sends to u_1 is defined as $c := (a_{pk,1}, v, \rho, r, s, cm)$.

(2) u_1 can scan over the public ledger and find the transaction $tx_{deposit}$. The user then decrypts Ct and gets (v, ρ, r, s).

(3) When $u1$ wants to withdraw the coin (or more than one received coins), u_1 will generate two new coins $c_1^{new} c_2^{new}$*and* a *zk-SNARK* proof π_{WITHDRAW} over the following statements:
For each old coins c, given the Merkle root rt, serial number sn, $u1$ knows c and address secret key $a_{sk,1}$ s.t.

- c is well-formatted.
- The address secret key matches the public key, i.e., $a_{pk,1} = PRF_{a_{sk,1}}^{addr}(0)$.
- The nullifier key matches the address secret key, i.e., $nk = PRF_{a_{sk,1}}^{addr}(1)$.
- The serial number is computed correctly, i.e., $sn = PRF_{nk}^{sn}(\rho)$.
- The coin commitment cm appears as a leaf of Merkle-tree with root rt.
- New coins c_1^{new} and c_2^{new} are well formatted.
- $v_1^{new} + v_2^{new} + v^{pub} = \sum v$.

The withdraw transaction $tx_{withdraw} := ([(rt, sn)], cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}})$ is appended in the ledger, where $ADDR$ is the plain text address, and $[(rt, sn)]$ is a set of the Merkle root and the serial number for each old coins. The relayer will verify the proof and check if all sn do not appear on the ledger. It will send the public coin to $ADDR$ and new coins c_1^{new} and c_2^{new} to anonymous addresses if validated. Furthermore, we employ a MAC scheme to prevent malleability attacks. When withdrawing a coin, the user samples a key pair (pk_{sig}, sk_{sig}) and uses sk_{sig} to sign every value associated with the $tx_{withdraw}$ transaction. The user also computes $h_{sig} := CRH(pk_{sig})$ and $h := PRF_{a_{sk}}^{pk}(h_{sig})$, which acts like a MAC to sign the secret address key. The user then modifies the statement to prove that h is computed correctly. The signature σ along with pk_{sig} are included in the $tx_{withdraw}$ transaction. The overview process is illustrated in Figure 1.



(1) $u1$ locks his coin
(2) $u2$ finds the transaction
(3) $u2$ withdraws the coin

Figure 1: Solution overview when u_1 on the source chain sends coins to u_2 on the destination chain. u_1 submits a deposit transaction and locks the coins. u_2 then scans over the network and finds the deposit transaction. u_2 submits a withdrawal transaction to spend the coins.

As cryptocurrency becomes popular, blockchain protocols must also adapt their auditing capacity. Recent enforcement actions against Tornado Cash[1] and Bittrex[2] alert cryptocurrency exchanges to the risk of failing to implement effective regulations. As U.S. Treasury stated, exchanges should understand the identity and location of the user. To meet regulations, when u_1 withdraws coins, the user has to disclose the commitments of old coins $[c]$ to auditors, and the auditors could then track the transaction link. Suppose there are n auditors, and to audit users' transactions there should be more than t auditors agree. The user divides commitments $[cm]$ into n pieces $[cm_1^a, cm_2^a, \ldots, cm_n^a]$ using $(t,n)-$secret sharing, in which one can recover the commitments only if the user has more than t pieces. The user then encrypts each share with an auditor's public key and sends it to the corresponding auditor. The auditors can decrypt the received messages and jointly recover the commitments. Let $Share^{(t,n)}$ denotes a (t,n)-secret sharing scheme and $Recover^{(t,n)}$ denotes the recovering scheme. $(pk_{enc,i}^a, sk_{enc,i}^a)$ are auditors' elliptic curve key pair. To provide a zero knowledge friendly implementation, we leverage an elliptic curve hybrid encryption scheme [KD04]. Namely, the protocol generates a shared secret key k^a in a symmetric-key encryption scheme $(SEC.Enc_k, SEC.Dec_k)$ from a public key scheme. Let (pk_u^a, sk_u^a) denote an elliptic curve key pair for audit purpose. We then set $k_i^a = sk_u^a \cdot pk_{enc,i}^a = sk_{enc,i}^a \cdot pk_u^a$ and the encrypted message $msg_i^a = SEC.Enc_{k_i^a}(cm_i^a)$. Without loss of generality, we assume there are three auditors. The user then proofs following statements along with other statements in π_{WITHDRAW}:
Let G be the generator in the elliptic curve. Given commitments $[cm]$. encrypted messages $msg_1^a, msg_2^a, msg_3^a$ and public keys $pk_u^a, pk_{enc,1}^a, pk_{enc,2}^a, pk_{enc,3}^a$, I know $[cm_1^a, cm_2^a, cm_3^a]$ and sk_u^a s.t.

- The commitments are well secret shared, i.e., $[cm_1^a, cm_2^a, cm_3^a] = Share^{(t,n)}([cm])$.
- The public key match the private key, i.e., $pk_u^a = sk_u^a G$.
- Each commitments share is well encrypted, i.e., for each $i \in \{1,2,3\}$, $msg_i^a = SEC.Enc_{sk_u^a pk_{enc,i}^a}(cm_i^a)$.

1.3 Architecture Overview

In this section, we introduce the proposed architecture as illustrated in Figure 2. We described the overview protocols and algorithms for depositing and withdrawing coins in Section 1.2, and then we implement the algorithms, naming **Mystiko**, in two phases: **Mystiko Deposit** and **Mystiko Withdraw**. During the **Mystiko Deposit** phase, a user sends coins from a source chain to a destination chain via a bridge, and Mystiko locks those coins on the source chain. It is noteworthy that Mystiko employs the bridge as a data bridge instead of an asset bridge, i.e., the bridge actively syncs invokes and events only. Moreover, all notes are encrypted. Only the user with the corresponding private key may decrypt it; therefore, only this user could generate the valid zero-knowledge proof and then withdraw the coin.

[1] `https://home.treasury.gov/news/press-releases/jy0916`
[2] `https://home.treasury.gov/news/press-releases/jy1006`

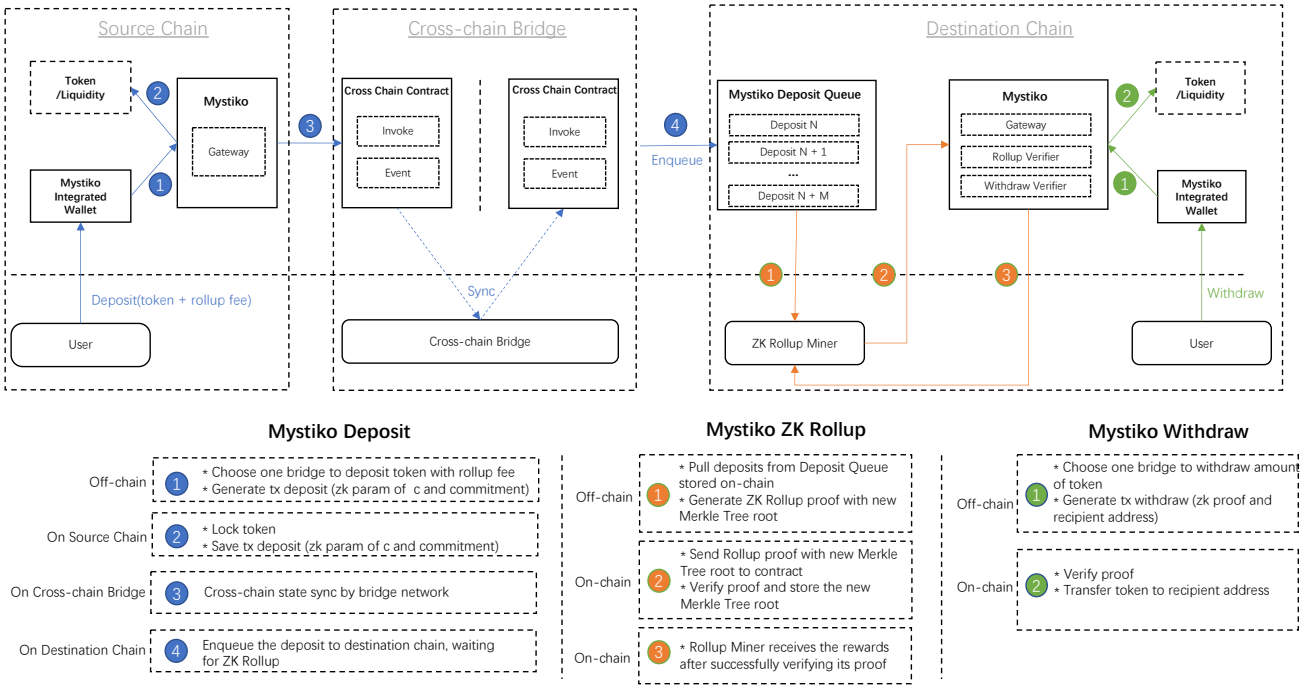


Figure 2: Architecture overview of the implementation. Transactions from the source chain are synced to the destination chain via a cross-chain bridge. The system enables **deposit**, **withdraw**, and **ZK-rollup** by a set of smart contracts.

If the receiver wants to withdraw the coins, the user then generates a withdraw transaction off-chain and verifies it on-chain. As mentioned in Section 1.2, Mystiko keeps a Merkle tree for all deposited coins and updates the tree when adding a new coin. This operation could be expensive if it is executed on-chain. In Mystiko, this problem has been solved by means of **ZK-rollups**. Namely, a ZK-rollup miner will pull on-chain deposits locally and calculate a Merkle tree root. The miner then generates a zero-knowledge proof: the Merkle tree root is correct and validated. The user then sends the proof with the root to the contract, and if the proof is validated, we update the Merkle tree root.

1.4 Paper organization

This paper is organized as follows. Section 2 provides background on the blockchain, zero-knowledge proofs, and previous works. We define the protocol in section 3 and describe construction in section 4. Section 5 provides benchmarks for our protocol implementation. We formally prove completeness and security in section 6. Section 7 summarize our contributions and future works.

2 Background

2.1 Blockchain

A blockchain is a distributed ledger shared among a computer network. The first generation of blockchain is Bitcoin [Nak08], which employs a blockchain to record peer-to-peer transactions. Ethereum [W+14] introduced *smart contracts* into the blockchain. Smart contracts are programs on the block that self-execute when certain conditions meet, which enables decentralized applications on blockchains. As the blockchain grows, all nodes in the network have to agree one the current state and append the same block. To ensure the consistent among nodes, the blockchain defines a set of protocol called *consensus algorithms*.

2.2 Zero-Knowledge Proof

Goldwasser, Micali, and Rackoff [GMR85] introduced the notion of zero-knowledge proof in 1985, which allows a prover to prove the truth of a statement without revealing anything but the statement is true. For example, the prover may prove that two graphs are isomorphic without revealing anything, especially the isomorphism between the two graphs. Blum, Feldman, and Micali [BFM88] introduced the non-interactive zero-knowledge proof, in which the proof is publicly verifiable without interaction with the prover. However, the early zero-knowledge proof protocols are inefficient or redundant in terms of proof size; therefore, those protocols are impractical. The zk-SNARK algorithm [Gro10, BCI$^+$13], which stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, is one of the first practical protocols. In zk-SNARK, the proof size is succinct and independent of the complexity of the statement. In this paper, we employed the Groth16 [Gro16, GM17] construction of zk-SNARK, which provides the best concrete efficient and shortest proof size.

2.3 Previous Works

Zerocash is the first zero-knowledge blockchain using zero-knowledge proof. Zexe [BCG$^+$20], Zkay [SBG$^+$19], and ZeeStar [SBBV22] extend the zero-knowledge framework to arbitrary smart contracts using Groth16 construction. Zexe requires users to compute smart contracts off-chain and then submit a zero-knowledge proof for correctness, but Zexe provides no development tools. Zkay and its following work, Zeestar, proposed a language for zero-knowledge smart contracts. However, those protocols are extremely compute-intensive, e.g., they require many cores (Zkay, 12 cores) or high RAM (Zexe, 256 GB).

Monero [NM16] is another zero-knowledge blockchain based on Bitcoin, but unlike Zerocash, Monero leverages ring signature and range-proofs [BBB$^+$18]. Range-proofs are special forms of zero-knowledge proofs which prove a value is within a range. For example, in Monero, users must prove that the inputs and outputs in their transactions are within valid range to prevent overflow. Zether [BAZB20] is a similar work to Monero but atop Ethereum. Both Monero and Zether are inherently limited in scalability since the size of range-proofs is not constant. Notably, all previously mentioned works provide no auditability and suffer from potential illicit uses.

ZkLedger [NVV18] and Fabzk [KDJL$^+$19] support auditability using homomorphic commitments and non-interactive zero-knowledge proof. However, these systems focus on the cross-organization transactions, and the performance degrades as the user number increases, which makes them impractical for supporting popular chains. ZEBRA [RPX$^+$22] is a zero-knowledge blockchain for anonymous credential scheme supporting auditability. Finally, Azeroth proposed zero-knowledge transactions with auditing, but the framwork does not support cross-chain transactions.

3 Definition of the Protocol

We introduce the notion of the anonymous protocol. This section is similar to the notation of Zerocash [BSCG$^+$14].

3.1 Data Structures

We describe the data structures used in the protocol.

Ledger This protocol is based on a blockchain network. There are two ledgers: the source chain's ledger L^{src} and the destination chain's ledger L^{dst}. At any given time T, all users have access to $L_T^{\{src,dst\}}$. Both ledgers are append-only.

Public parameters[3]. A list of public parameters pp is available to all users in the system. These are generated by a trusted party at the "start of time" and are used by the system's algorithms.

Address[4]. Each user generates at least one address key pair ($addr_{pk}$, $addr_{sk}$) and a nullifier key nk. The public key $addr_{pk}$ is published and enables others to direct payments to the user. The secret key $addr_{sk}$ is used to receive payments sent to $addr_{pk}$. The nullifier key nk is used to generate serial numbers of receiving coins. A user may generate any number of address key pairs.

Auditable keys. Each user generates at least one audit key pair (pk_u^a, sk_u^a). The public key pk_u^a is published and enables auditors to generate the shared secret key with their own private keys. The private key sk_u^a is used to generate the shared secret key with the auditors' public keys.

[3]Taken from [BSCG$^+$14] **3.1 Data structures Public parameters**

[4]Taken from [BSCG$^+$14] **3.1 Data structures Addresses**

Coin. A coin is a data object c. Across this paper, c refers to a logical coin since a user will not mint a new coin when transferring the coin. A coin is associated with *commitment, value, serial number, address.*

- commitment, denoted $cm(c)$: a string that appears on the ledger once c is deposited.
- value, denoted $v(c)$: the denomination of c. We limit the value within some pre-defined default values, denoted $\mathbb{V}$, i.e., $v \in \mathbb{V}$.
- serial number, denoted $sn(c)$: a unique string associated with c, used to prevent double withdrawing.
- address, denoted $addr_{pk}(c)$: an address public key, representing who owns c.

Transaction. We introduce three new types of transactions.

- Deposit transactions. A deposit transaction $tx_{deposit}$ is a tuple $(cm, v, *)$, where cm is the coin commitment, v is the coin value, and $*$ are other information, e.g., randomness. The transaction $tx_{deposit}$ records that a user deposits a coin with commitment cm and value v, which could be withdrawn on other chains.
- Withdraw transactions. A withdraw transaction $tx_{withdraw}$ is a tuple $([(rt, sn)] , cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}, [m$ where $[(rt, sn)]$ is a set of the Merkle root and the serial number for each old coins, cm_1^{new}, cm_2^{new} are commitments of new coins, v^{pub} is the public coin value, $ADDR$ is a plain text address, $[msg_1^a, msg_2^a, \ldots, msg_n^a]$ are encrypted messages for the audit, and $*$ denotes other information. The transaction $tx_{withdraw}$ records that a user withdraws some coins c and sends a coin to a public address and two new coins to anonymous addresses. It also contains messages that auditors may decrypt and then track the transaction.
- ZK-rollup transactions. A ZK-rollup transaction tx_{rollup} is a tuple $(rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{\mathsf{ROLLUP}}, *)$, where rt^{old} is the old Merkle tree root, rt^{new} is the new Merkle tree root after updating with coin commitments $[cm]$, $hash_{[cm]}$ is the hash of $[cm]$, $pathIndices$ is the direction selector of the authentication path of $[cm]$, N^{rollup} is the number of commitments been rolluped, and $*$ denotes other information. The transaction tx_{rollup} records that a user update the commitment tree with a set of deposited coin commitments.

Committed of deposit coins and serial numbers of withdraw coins. For any given time T

- $CMList_T$ denotes the list of all commitments appearing in deposit transactions in L_T^{src}.
- $SNList_T$ denotes the list of all serial numbers appearing in withdraw transactions in L_T^{src}

Merkle tree over commitments. For any given time T, $Tree_T$ denotes a Merkle tree over $CMList_T$ and rt_T is the root. $Path_T(cm)$ denotes the path function which outputs the authentication path given a coin commitment cm.
Queue of commitments. For any given time T, Q_T^{cm} denotes a queue of commitments waiting for rollup.

3.2 Algorithms

The protocol Π is a tuple of polynomial-time algorithms
(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup,
VerifyTransaction, Receive, Audit) with the following syntax and semantics.
System setup. The algorithm Setup generates a list of public parameters:

- Inputs: security parameter λ
- Outputs: public parameters pp

The Setup algorithm is executed once by a trusted party.
Creating payment address. The CreateAddress algorithm generates a new pair of payment address and a nullifier key:

- Inputs: public parameters pp
- Outputs:
  - address key pair $(addr_{pk}, addr_{sk})$

– nullifier key nk

Each user needs to generate at least one address pair. $addr_{pk}$ is public, and $addr_{sk}$ is kept secretly and used to withdraw the coin sent to the address.

Creating audit keys. The CreateAuditKey algorithm generates a new pair of key for the audit:

- Inputs: public parameters pp
- Outputs: address key pair (pk_u^a, sk_u^a)

Each user needs to generate at least one audit key pair. pk_u^a is public, and sk_u^a is kept secretly.

Depositing coins. The Deposit generates a logical coin and a deposit transaction:

- Inputs:
  - public parameters pp
  - coin value $v \in \mathbb{V}$
  - destination address public key $addr_{pk}$
- Outputs:
  - coin c
  - deposit transaction $tx_{deposit}$

The output coin c has value v and coin address $addr_{pk}$; the output deposit transaction $tx_{deposit}$ equals $(cm, v, *)$, where cm is the coin commitment of c.

Withdrawing coins. The Withdraw algorithm transfers value from coins on one chain to coins on another chain.

- Inputs:
  - public parameters pp
  - For each old coins c,
    - the Merkle root rt
    - authentication path $path$ from commitment $cm(c)$ to root rt
    - the address secret key $addr_{sk}$
  - new address $ADDR$
  - public value v^{pub}
  - new values v_1^{new}, v_2^{new}
  - new address public keys $addr_{pk,1}^{new}, addr_{pk,2}^{new}$
  - user's audit key pair (sk_u^a, pk_u^a)
  - auditors' public keys $[pk_{enc}^a]$
- Outputs:
  - withdraw transaction $tx_{withdraw}$
  - new coins c_1^{new}, c_2^{new}

For each coin c, the Withdraw algorithm takes as inputs an coin c and its address secret key $addr_{sk}$. The Withdraw algorithm also takes as inputs the Merkle tree root rt and an authentication path $path$ of the commitment $cm(c)$. $ADDR$ is the new address where the user sends the public coin, which could be on a different chain other than c's. The value v^{pub} specifies the value to be public transferred. (sk_u^a, pk_u^a) and $[pk_{enc,i}^a]$ encrypt commitments for the audit. Moreover, the Withdraw algorithm also generates two new anonymous coins c_1^{new}, c_2^{new} with values v_1^{new}, v_2^{new} and recipients address $addr_{pk,1}^{new}, addr_{pk,2}^{new}$ respectively. $v^{pub} + v_1^{new} + v_2^{new}$ should be equal to c's value. $[msg_1^a, msg_2^a, \ldots, msg_n^a]$ are encrypted commitments.

The Withdraw algorithm outputs a withdraw transaction $tx_{withdraw}$. The transaction $tx_{withdraw}$ equals ($[(rt, sn)]$, $cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}$, $[msg_1^a, msg_2^a, \ldots, msg_n^a]$). This transaction will not reveal the payment address of the old coin.

ZK-rollup. The algorithm Rollup generates a new Merkle tree root and a ZK-rollup transaction:

- Inputs:
  - public parameters pp
  - rollup size N^{rollup}
  - a queue of deposited commitments Q^{cm}
  - an old Merkle tree root rt^{old}
  - an authentication path $path$
- Outputs:
  - a set of deposited commitments $[cm]$
  - ZK-rollup transaction tx_{rollup}

The Rollup algorithm takes as inputs an old Merkle root rt^{old}, an authentication path $path$, a rollup size N^{rollup}, and a queue of deposited commitments Q^{cm}. The Rollup algorithm outputs a set of deposite commitments $[cm]$ by dequeuing N^{rollup} commitments from Q^{cm}. It also generates a new Merkle root rt^{new} by updating leaves in the old Merkle tree with new leaves $[cm]$. There is an authentication path $path$ toward the ancestor node of new leaves, which is equal to the root of a CRH-based Merkle tree over $[cm]$. The algorithm then generates a zk-SNARK π_{ROLLUP} to prove that all calculations are valid and correct. The output ZK-rollup transaction tx_{rollup} equals $(rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{\mathsf{ROLLUP}}, *)$, where $hash_{[cm]}$ is the hash of $[cm]$, $pathIndices$ is the direction selector of $path$.

Verifying transactions. The algorithm VerifyTransaction checks the validity of a transaction:

- Inputs:
  - public parameters pp
  - a (withdraw, deposit or ZK-rollup) transaction tx
  - the current source and destination ledgers L_{src} and L_{dst}
- Outputs: bit $\mathcal{B}$, equals 1 iff the transaction is valid

Deposit, withdraw, and ZK-rollup transactions must be verified before being executed.

Receiving coins.[5] The algorithm Receive scans the ledger and retrieves unwithdrawn coins paid to a particular user address:

- Inputs:
  - recipient address key pair $(addr_{pk}, addr_{sk})$
  - recipient nullifier address nk
  - the current source and destination ledgers L_{src} and L_{dst}
- Outputs: set of (unwithdrawn) received coins

When a user with address key pair $(addr_{pk}, addr_{sk})$ wishes to receive payments sent to $addr_{pk}$, the user uses the Receive algorithm to scan the ledger. For each payment to $addr_{pk}$ appearing in the ledger, Receive outputs the corresponding coins whose serial numbers do not appear on the ledger $L_{src,dst}$. Coins received in this way may be withdrawn by using Withdraw algorithm.

Audit. The algorithm Audit audits user transactions:

- Inputs:
  - Encrypted commitments sharings $[msg_1^a, msg_2^a, \ldots, msg_n^a]$
  - User's public key pk_u^a
  - Auditors' private keys $[sk_{enc,1}^a, sk_{enc,2}^a, \ldots, sk_{enc,n}^a]$
- Outputs: A set of commitments $[cm]$

The auditors decrypt each message msg_i^a with the shared secret key $sk_{enc,}^a, pk_u^a$ and jointly recover the commitments $[cm]$. The auditors can recover the transaction link with those commitments.

[5] Taken from [BSCG+14] 3.2 **Receiving coins**

3.3 Completeness

Completeness of a protocol requires that unwithdrawn coins can be withdrawn. Suppose a ledger sampler $\mathcal{S}$ outputs a ledger $L_{src,dst}$. If c is a coin whose commitment appears in a valid transaction on $L_{src,dst}$, but its serial number does not appear in L, then c can be withdrawn using Withdraw transaction. Informally, if Withdraw outputs a $tx_{withdraw}$ transaction that VerifyTransaction accepts, the coin could be received by the intended recipient. This property is formalized via an *incompleteness experiment* $INCOMP$.

Definition 1 *A protocol* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *is complete if no polynomial-size ledger sample* $\mathcal{S}$ *wins* $INCOMP$ *with more than negligible probability.*

3.4 Security

Security of the protocol is characterized by four properties, which we call ledger *indistinguishability*, *transaction non-malleability*, *balance*, and *auditability*.

Definition 2 *A protocol* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *is secure if it satisfies ledger indistinguishability, transaction non-malleability, balance, and auditability.*

We describe the informal definition below.
Ledger indistinguishability. This property captures the requirement that the ledger reveals no new information to the adversary beyond the publicly-revealed information (e.g. plain text address, coin's public value).
Transaction non-malleability. This property means no bounded adversary may modify the data stored in a valid withdraw transaction.
Balance. This property requires no bounded adversary could withdraw more coins than what the user received from the deposit transaction.
Auditability. This property requires the auditor can always monitor the confidential data of any user.

4 Construction of the Protocol

In this section, we describe how to construct the protocol with zk-snark and other cryptography building blocks at first. Then we give the concrete design.

4.1 Cryptographic building blocks

We introduce the formal notation of the cryptography building blocks we use. λ denotes the security parameter. This part is similar to [BSCG+14] **section 4.1**.
Collision-resistant hashing. We use a collision-resistant hash function $CRH : \{0,1\}^* \to \{0,1\}^{O(\lambda)}$.
Pseudorandom functions. We use a pseudorandom function family $\mathbb{PRF} = \{PRF_x : \{0,1\}^* \leftarrow \{0,1\}^{O(\lambda)}\}_x$. We then instance three pseudorandom functions from the same $PRF_x s \xleftarrow{\$} \mathbb{PRF}$ and add different prefix to the input. Namely, $PRF_x^{addr}(z) := PRF_x(00||z)$,
$PRF_x^{sn}(z :) = PRF_x(01||z), PRF_x^{pk}(z) := PRF_x(10||z)$. Moreover, we require PRF^{sn} to be collision resistant, i.e. one cannot find $(x,z) \neq (x',z')$ s.t. $PRF_x^{sn}(z) = PRF_{x'}^{sn}(z')$.
Statistically-hiding commitments. We use a computationally binding and statistically hiding commitment scheme $COMM$. Namely, $\{COMM_x : \{0,1\}^* \to \{0,1\}^{O(\lambda)}\}_x$ where x denotes the trapdoor parameter.
One-time strongly-unforgeable digital signatures. We use a digital signature scheme $Sig = (G_{sig}, K_{sig}, S_{sig}, V_{sig})$.

- $G_{sig}(1^\lambda) \to pp_{sig}$. Given a security parameters λ, G_{sig} samples public parameters pp_{sig} for the signature scheme.
- $K_{sig}(pp_{sig}) \to (pk_{sig}, sk_{sig})$. Given public parameters pp_{sig}, K_{sig} samples a public key and a secret key for a single user.
- $S_{sig}(sk_{sig}, m) \to \sigma$. Given a secret key sk_{sig} and a message m, S_{sig} signs m to obtain a signature σ.
- $V_{sig}(pk_{sig}, m, \sigma) \to b$. Given a public key pk_{sig}, message m, and the signature σ, V_{sig} outputs $b = 1$ if validated or otherwise $b = 0$.

We require Sig to be one-time strong unforgeable against chosen-message attacks (SUF-1CMA security).
Key-private public-key encryption. We use a public-key encryption scheme $Enc = (G_{enc}, K_{enc}, E_{enc}, D_{enc})$.

- $G_{enc}(1^\lambda) \to pp_{enc}$. Given a security parameter λ, G_{enc} samples public parameters pp_{enc} for the encryption scheme.
- $K_{enc}(pp_{enc}) \to (pk_{enc}, sk_{enc})$. Given public parameters pp_{enc}, K_{enc} samples a public key and a secret key for a single user.
- $E_{enc}(pk_{enc}, m) \to Ct$. Given a public key pk_{enc} and a message m, E_{enc} encrypts m to obtain a cipher text Ct.
- $D_{enc}(sk_{enc}, Ct) \to m$. Given a secret key sk_{enc} and a cipher text Ct, D_{enc} decrypts Ct to obtain the plain message m (or $\perp$ if decryption fails).

The encryption scheme Enc is secure against chosen-ciphertext attack and provides ciphertext indistinguishability IND-CCA and key indistinguishability IK-CCA.
Elliptic curve integrated encryption scheme. We use an elliptic curve integrated encryption scheme $ECIES = (G_{ecies}, K_{ecies}, KEM, SEC.Enc, SEC.Dec)$.

- $G_{ecies}(1^\lambda) \to pp_{ecies}$. Given a security parameter λ, G_{ecies} samples public parameters pp_{ecies} for the encryption scheme.
- $K_{ecies}(pp_{ecies}) \to (pk^a, sk^a)$. Given public parameters pp_{ecies}, K_{enc} samples a public key and a secret key for a single user.
- $KEM(pk_i^a, sk_j^a) \to k^a$. Given a public key from user i and a private key from user j, KEM generates a shared secret key k^a.
- $SEC.Enc_{k^a}(m) \to msg^a$. Given a secret key k^a and a message m, $SEC.Enc$ encrypts m to obtain a ciphter text msg^a.
- $SEC.Dec_{k^a}(msg^a) \to m$. Given a secret key sk_{enc} and a cipher text msg^a, $SEC.Dec$ decrypts msg^a to obtain the plain message m (or $\perp$ if decryption fails).

The encryption scheme $ECIES$ is secure against chosen-ciphertext attack and provides ciphertext indistinguishability IND-CCA and key indistinguishability IK-CCA.
Threshold secret sharing. We use a threshold secret sharing scheme $SS = (Share, Recover)$.

- $Share(x) \to [x_1, x_2, \ldots, x_n]$. Given a secret x generates n secret shares $[x_1, x_2, \ldots, x_n]$.
- $Revocer([x_i, x_{i+1}, \ldots, x_{i+t-1}]) \to x$. Given t secret shares $[x_i, x_{i+1}, \ldots, x_{i+t-1}]$ generates the secret x.

The secret sharing is a perfect t out of n secret sharing PER-SS, i.e., the secret sharing scheme outputs n shares, and given any t shares, we can recover the secret. We learn nothing about x given less than t shares.

4.2 zk-SNARKs for withdrawing coins

We use zk-SNARK to prove a NP statement $WITHDRAW$. For the definition of zk-SNARK, we refer to [BCI+13] for a detailed explanation. We first give an informal definition of zk-SNARKs. Given a field $\mathbb{F}$, a **zk-SNARK** for $\mathbb{F}-$arithmetic circuit satisfiability is a triple of polynomial-time algorithm (KeyGen, $Prove, Verify$):

- $\mathsf{KeyGen}(1^\lambda, C) \to (pk, vk)$. On input a security parameter λ and an $\mathbb{F}-$arithmetic circuit C, the *key generator* KeyGen probabilistically samples a *proving key* **pk** and a *verification key* **vk**.
- $Prove(pk, x, a) \to \pi$. On input a proving key **pk** and any $(x, a) \in R_C$, the *prover* **Prove** outputs a non-interactive proof π for the statement $x \in L_C$.
- $Verify(vk, x, \pi) \to b$. On input a verification key **vk**, an input x, and a proof π, the *verifier* **Verify** outputs $b = 1$ if the *verifier* is convinced that $x \in L_C$.

We recall the corresponding withdraw transaction $tx_{withdraw} = (\ [(rt, sn)]\ , cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}, [msg_1^a, msg_2^a, \ldots, msg_n^a])$. To withdraw a coin c, a user u should show that

1. u owns c
2. commitment of c appears on the ledger
3. sn is the calculated correctly as the serial number of c
4. balance is preserved
5. the commitment is well encrypted

which is formalized as a statement $WITHDRAW$ and proved with zk-SNARK. We then define the statement as follows.

- Instances is $x := ([(rt, sn, h)], v^{pub}, cm_1^{new}, cm_2^{new}, h_{sig}, pk_u^a, [pk_{enc}^a], [msg^a])$, which specifies a set $[(rt, sn, h)]$ for each old coin, where rt is the root for a CRH-based Merkle tree, sn is the serial number, and h is the signature. It also specifies the public value v^{pub}, two commitments of new coins cm_1^{new}, cm_2^{new}, and fields h_{sig} used for non-malleability. pk_u^a is the user's private key for audit, pk_{enc}^a are auditors' public keys, and $[msg^a]$ are encrypted secret sharings of commitments for audit.
- Witnesses are of the form $a := ([(path, c, addr_{sk})], c_1^{new}, c_2^{new}, [cm^a], sk_u^a)$ where

$$\begin{aligned} c &= (addr_{pk}, v, \rho, r, s, cm) \\ addr_{pk} &= (a_{pk}, pk_{enc}) \\ c_i^{new} &= (addr_{pk,i}^{new}, v_i^{new}, \rho_i^{new}, r_i^{new}, s_i^{new}, cm_i^{new}) \\ addr_{pk,i}^{new} &= (a_{pk,i}^{new}, pk_{enc,i}^{new}) \end{aligned}$$

  Thus, the witness a specifies a authenticated path from root rt to the coin's commitment, the entirety information of the coin c, the address secret key, secret sharings of commitments, and the user's private key for audit.

Given a $WITHDRAW$ instance x, a witness a is valid for x if :

1. For any old coin c,
   (a) The coin's commitment cm appears on the ledger, i.e., $path$ is a valid authentication path for leaf cm in a CRH-based Merkle tree with root rt.
   (b) The address secret key a_{sk} matches the address public key, i.e., $a_{pk} = PRF_{a_{sk}}^{addr}(0)$.
   (c) The nullifier key nk matches the address secret key, i.e., $nk = PRF_{a_{sk,1}}^{addr}(1)$.
   (d) The serial number sn is computed correctly, i.e., $sn = PRF_{nk}^{sn}(\rho)$.
   (e) The coin c is well formatted, i.e., $cm = COMM_s(COMM_r(a_{pk}||\rho)||v)$.
   (f) The address secret key a_{sk} ties to h_{sig} to h, i.e., $h = PRF_{a_{sk}}^{pk}(h_{sig})$.
2. New coins c_1^{new} and c_2^{new} are well formatted, i.e., $cm = COMM_{s_i^{new}}(COMM_{r_i^{new}}(a_{pk,i}^{new}||\rho_i^{new})||v_i^{new})$.
3. Balance is preserved, i.e. $\sum v = v_1^{new} + v_2^{new} + v^{pub}$.
4. The commitments are well secret shared, i.e. $[cm^a] = Share^{(t,n)}([cm])$.
5. The public audit key match the private audit key, i.e., $pk_u^a = sk_u^a G$.
6. Each commitments share is well encrypted, i.e., $msg_i^a = SEC.Enc_{sk_u^a pk_{enc,i}^a}(cm_i^a)$.

4.3 zk-SNARKs for ZK-rollup

We use zk-SNARK to prove a NP statement $ROLLUP$. In this section, we use the same notions as in section 4.2. We recall the corresponding ZK-rollup transaction $tx_{rollup} = (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{\mathsf{ROLLUP}})$. To rollup a set of coin commitments $[cm]$, a user u should show that

1. u knows $[cm]$
2. u updates the old Merkle tree with $[cm]$

which is formalized as a statement $ROLLUP$ and proved with zk-SNARK. We then define the statements as follows.

- Instances is $x := (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup})$, which specifies an old Merkle root rt^{old}, a new Merkle root rt^{new}, a hash of a set of coin commitments $hash_{[cm]}$, the direction selector of the updated leaf's authentication path $pathIndices$.
- Witnesses are of the form $a := ([cm], path)$, and the rollup size N^{rollup}.

Thus, the witness a specifies a set of commitments $[cm]$, and the authentication path $path$.

Given a $ROLLUP$ instance x, let [0] be a set of N^{rollup} zeors, a witness a is valid for x if :

1. $hash_{[cm]}$ is the hash value of $[cm]$.
2. $rt^{[0]}$ is the Merkle root of [0].
3. $path$ is a valid authentication path from $rt^{[0]}$ to rt^{old}, and the corresponding director selector is $pathIndices$.
4. $rt^{[cm]}$ is the root of a CRH-based Merkle tree over $[cm]$.
5. $path$ is a valid authentication path from $rt^{[cm]}$ to rt^{new}, and the corresponding director selector is $pathIndices$.
6. The number of updated leaves is correct, e.g., let H be the height of the whole Merkle tree, $|[cm]| = |[0]|$ and $|path| + \log_2 |[cm]| - 1 = H$.

4.4 Algorithm constructions

In this section, we describe the construction of each algorithm. The intuition is given in 3.1 and 3.2. The building blocks are introduced in 4.1 and 4.2. We give the pseudocode for each algorithm in Figure 3 and Figure 4.

4.5 Concrete design

In this section, we describe how we instantiate each building block. Namely, we build CRH, PRF, $COMM$ from **Poseidon** [GKR+21], Merkle tree from **Keccak**[BDPVA13], Sig from **ECDSA**, Enc from **key-private Elliptic-Curve Integrated Encryption Scheme**.

5 Implementation and Experiment

We implemented our protocol on Ethereum and evaluated its performance on various system environments described in Table 1. This protocol employs the Groth16 proof system and implements with ZoKrates [ET18], a toolbox for zkSNARKs on Ethereum. We developed smart contracts with the Solidity language. There are three main phases in the system: withdraw, deposit, and ZK-rollup. During each phase, the protocol submits a transaction to the blockchain and executes a corresponding smart contract. During withdraw and deposit phases, the protocol also computes SNARKs for withdrawing coins and ZK-rollup.

Machine	OS	CPU	RAM
Computer_1	Ubuntu	Intel(R) Xeon(R) E-2234 CPU @ 3.6 GHz 8 core	16G
Computer_2	macOS	Intel(R) Core(R) i7 CPU @ 2.6 GHz 6 core	16G

Table 1: System specification of the different testing environments.

Setup.

- Inputs: security parameter λ
- Outputs: public parameters pp

1. Construct the arithmetic circuit C_{WITHDRAW} for the $WITHDRAW$ statement at security λ.
2. Compute $(pk_{\mathsf{WITHDRAW}}, vk_{\mathsf{WITHDRAW}}) := \mathsf{KeyGen}(1^\lambda, C_{\mathsf{WITHDRAW}})$.
3. Construct the arithmetic circuit C_{ROLLUP} for the $ROLLUP$ statement at security λ.
4. Compute $(pk_{\mathsf{ROLLUP}}, vk_{\mathsf{ROLLUP}}) := \mathsf{KeyGen}(1^\lambda, C_{\mathsf{ROLLUP}})$.
5. Compute $pp_{enc} := G_{enc}(1^\lambda)$.
6. Compute $pp_{sig} := G_{sig}(1^\lambda)$.
7. Compute $pp_{ecies} := G_{ecies}(1^\lambda)$.
8. Set $pp := (pk_{\mathsf{WITHDRAW}}, vk_{\mathsf{WITHDRAW}}, pk_{\mathsf{ROLLUP}}, vk_{\mathsf{ROLLUP}}, pp_{enc}, pp_{sig}, pp_{ecies})$.
9. Output pp.

CreateAddress

- Inputs: public parameters pp
- Outputs:
  - address key pair $(addr_{pk}, addr_{sk})$
  - nullifier key nk

1. Compute $(pk_{enc}, sk_{enc}) := K_{enc}(pp_{enc})$.
2. Randomly sample a PRF^{addr} seed a_{sk}.
3. Compute $a_{pk} = PRF^{addr}_{a_{sk}}(0)$.
4. Compute $nk = PRF^{addr}_{a_{sk}}(1)$.
5. Set $addr_{pk} := (a_{pk}, pk_{enc})$.
6. Set $addr_{sk} := (a_{sk}, sk_{enc})$.
7. Output $(addr_{pk}, addr_{sk})$ and nk.

CreateAuditKey

- Inputs: public parameters pp
- Outputs: address key pair (pk^a_u, sk^a_u)

1. Compute $(pk^a_u, sk^a_u) := K_{ecies}(pp_{ecies})$.
2. Outputs (pk^a_u, sk^a_u).

Deposit

- Inputs:
  - public parameters pp
  - coin value $v \in \mathbb{V}$
  - destination address public key $addr_{pk}$
- Outputs:
  - coin c
  - deposit transaction $tx_{deposit}$

1. Parse $addr_{pk}$ as (a_{pk}, pk_{enc}).
2. Randomly sample a PRF^{sn} seed ρ.
3. Randomly sample two $COMM$ trapdoors r, s.
4. Compute $k := COMM_r(a_{pk}||\rho)$.
5. Compute $cm := COMM_s(v||k)$.
6. Compute $Ct := E_{enc}(pk_{enc}, m)$, where $m := (v, \rho, r, s)$.
7. Set $c := (addr_{pk}, v, \rho, r, s, cm)$.
8. Set $tx_{deposit} := (cm, v, *)$, where $* := (k, s, Ct)$.
9. Output c and $tx_{deposit}$.

Audit.

- Inputs:
  - Encrypted commitments sharings $[msg^a_1, \ldots, msg^a_n]$
  - User's public key pk^a_u
  - Auditors' private keys $[sk^a_{enc,1}, sk^a_{enc,2}, \ldots, sk^a_{enc,n}]$
- Outputs: A set of commitments $[cm]$

1. For each auditors' private key $sk^a_{enc,i}$, compute $k^a_i := KEM(pk^a_u, sk^a_{enc,i})$.
2. For each encrypted commitments sharing msg^a_i, compute $cm^a_i := SEC.Dec_{k^a_i}(msg^a_i)$.
3. Compute $[cm] := Recover^{(t,n)}(cm^a_1, cm^a_2, \ldots, cm^a_n)$
4. Output $[cm]$.

Withdraw.

- Inputs:
  - public parameters pp
  - For each coin c,
    - the Merkle root rt
    - authentication path $path$ from commitment $cm(c)$ to root rt
    - the address secret key $addr_{sk}$
    - nullifier key nk
  - new address $ADDR$
  - public value v^{pub}
  - new values v^{new}_1, v^{new}_2
  - new address public keys $addr^{new}_{pk,1}, addr^{new}_{pk,2}$
  - user's audit key pair (sk^a_u, pk^a_u)
  - auditors' public keys $pk^a_{enc,1}, pk^a_{enc,2}, pk^a_{enc,3}$
- Outputs:
  - withdraw transaction $tx_{withdraw}$
  - new coins c^{new}_1, c^{new}_2

1. For each old coin c:
   (a) Parse c as $(addr_{pk}, v, \rho, r, s, cm)$.
   (b) Parse $addr_{sk}$ as (a_{sk}, sk_{enc}).
   (c) Compute $sn := PRF^{sn}_{nk}(\rho)$.
   (d) Parse $addr_{pk}$ as (a_{pk}, pk_{enc}).
2. For each $i \in 1, 2$:
   (a) Parse $addr^{new}_{pk,i}$ as $(a^{new}_{pk,i}, pk^{new}_{enc,i})$.
   (b) Randomly sample a PRF^{sn} seed ρ^{new}_i.
   (c) Randomly sample two $COMM$ trapdoors r^{new}_i, s^{new}_i.
   (d) Compute $k^{new}_i := COMM_{r^{new}_i}(a^{new}_{pk,i}||\rho^{new}_i)$.
   (e) Compute $cm^{new}_i := COMM_{s^{new}_i}(v^{new}_i||k^{new}_i)$.
   (f) Compute $Ct^{new}_i := E_{enc}(pk_{enc}, m)$, where $m := (v^{new}_i, \rho^{new}_i, r^{new}_i, s^{new}_i)$.
   (g) Set $c^{new}_i := (addr^{new}_{pk,i}, v^{new}_i, \rho^{new}_i, r^{new}_i, s^{new}_i, cm^{new}_i)$.
3. Generate $(pk_{sig}, sk_{sig}) := K_{sig}(pp_{sig})$.
4. Compute $h_{sig} := CRH(pk_{sig})$.
5. For each old coin, compute $h := PRF^{pk}_{a_{sk}}(i||h_{sig})$.
6. Compute $[cm^a_1, cm^a_2, \ldots, cm^a_n] := Share^{(t,n)}([cm])$.
7. For each auditor's public key $pk^a_{enc,i}$, compute $k^a_i = KEM(pk^a_{enc,i}, sk^a_u)$.
8. For each commitments share cm^a_i, compute $msg^a_i := SEC.Enc_{k^a_i}(cm^a_i)$.
9. Set $x := ([(rt, sn, h)], v^{pub}, cm^{new}_1, cm^{new}_2, h_{sig}, pk^a_u, pk^a_{enc,1}, pk^a_{enc,2}, pk^a_{enc,3}, [msg^a_1, msg^a_2, \ldots, msg^a_n])$.
10. Set $a = ([(path, c, addr_{sk})], c^{new}_1, c^{new}_2, cm^a_1, cm^a_2, cm^a_3, sk^a_u)$.
11. Compute $\pi_{\mathsf{WITHDRAW}} := Prove(pk_{\mathsf{WITHDRAW}}, x, a)$.
12. Set $m := (x, \pi_{\mathsf{WITHDRAW}}, ADDR, Ct^{new}_1, Ct^{new}_2)$.
13. Compute $\sigma := S_{sig}(sk_{sig}, m)$.
14. Set $tx_{withdraw} = ([(rt, sn)], cm^{new}_1, cm^{new}_2, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}, msg^a_1, msg^a_2, msg^a_3, *)$, where $* := (pk_{sig}, [h], \sigma, Ct^{new}_1, Ct^{new}_2)$.
15. Output c^{new}_1, c^{new}_2, and $tx_{withdraw}$.

Figure 3: Construction of Setup, CreateAddress, CreateAuditKey, Deposit, Audit, Withdraw algorithms.

Rollup.

- Inputs:
  - public parameters pp
  - rollup size N^{rollup}
  - a queue of deposited commitments Q^{cm}
  - an old Merkle tree root rt^{old}
  - an authentication path $path$
- Outputs:
  - a set of deposited commitments $[cm]$
  - ZK-rollup transaction tx_{rollup}

1. Set $pathIndices$ as the direction selector of $path$.
2. Set $[cm]$ as the first N^{rollup} commitments from Q^{cm}.
3. Compute $hash_{[cm]} := CRH([cm])$.
4. Compute $rt^{[cm]}$ as the root of a CRH-based Merkle tree over $[cm]$.
5. Compute rt^{new} as follows:
   (a) Let D^{path} be the length of $path$.
   (b) Let $digest = rt^{[cm]}$.
   (c) For each $i \in \{1, \ldots, D^{path}\}$, if $pathIndices[i] = 0$, compute $digest := CRH(digest, path[i])$, else $digest := CRH(path[i], digest)$.
   (d) Set $rt^{new} := digest$
6. Set $x := (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup})$.
7. Set $a := ([cm], path)$.
8. Compute $\pi_{\mathsf{ROLLUP}} := Prove(pk_{\mathsf{ROLLUP}}, x, a)$.
9. Set $tx_{rollup} := (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{\mathsf{ROLLUP}})$.
10. Ouput $[cm]$ and tx_{rollup}.

Receive.

- Inputs:
  - recipient address key pair $(addr_{pk}, addr_{sk})$
  - recipient nullifier key nk
  - the current source and destination ledgers L_{src} and L_{dst}
- Outputs: set of (unwithdrawn) received coins

1. Parse $addr_{pk}$ as (a_{pk}, pk_{enc}).
2. Parse $addr_{sk}$ as (a_{sk}, sk_{enc}).
3. For each deposit transaction $tx_{deposit}$ on the ledger:
   (a) Parse $tx_{deposit}$ as $(cm, v, *)$, where $*$ as (k, s, Ct).
   (b) Compute $m := D_{enc}(sk_{enc}, Ct)$, and parse m as (v, ρ, r, s).
   (c) If D_{enc}'s output is not $\perp$, verify that:
   - cm equals $COMM_s(v||COMM_r(a_{pk}||\rho))$;
   - $sn := PRF^{sn}_{nk}$ does not appear on L.

   (d) If both checks succeed, output $c := (addr_{pk}, v, \rho, r, s, cm)$

VerifyTransaction.

- Inputs:
  - public parameters pp
  - a (withdraw or deposit) transaction tx
  - auditors' public keys $[pk^a_{enc,1}, pk^a_{enc,2}, \ldots, pk^a_{enc,n}]$
  - the current source and destination ledgers L_{src} and L_{dst}
- Outputs: bit b, equals 1 iff the transaction is valid

1. If given a deposit transaction $tx = tx_{deposit}$:
   (a) Parse $tx_{deposit}$ as $(cm, v, *)$, and $*$ as (k, s).
   (b) If $v \notin \mathbb{V}$, output $b := 0$.
   (c) Set $cm' := COMM_s(v||k)$.
   (d) Output $b := 1$ if $cm = cm'$, else output $b := 0$.
2. If given a withdraw transaction $tx = tx_{withdraw}$:
   (a) Parse $tx_{withdraw}$ as $([(rt, sn)], cm^{new}_1, cm^{new}_2, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}, [msg^a_1, msg^a_2, \ldots, msg^a_n], *)$, where $* := (pk_{sig}, [h], \pi_{\mathsf{WITHDRAW}}, \sigma, Ct^{new}_1, Ct^{new}_2)$..
   (b) If any sn appears on L, output $b := 0$.
   (c) If any Merkle root rt does not appear on L, output $b := 0$.
   (d) Compute $h_{sig} := CRH(pk_{sig})$.
   (e) Set $x := ([(rt, sn, h)], v^{pub}, cm^{new}_1, cm^{new}_2, h_{sig}, pk^a_u, [pk^a_{enc,1}, pk^a_{enc,2}, \ldots, pk^a_{enc,n}], [msg^a_1, msg^a_2, \ldots, msg^a_n])$.
   (f) Set $m := (x, \pi_{\mathsf{WITHDRAW}}, ADDR, Ct^{new}_1, Ct^{new}_2)$.
   (g) Compute $b := V_{sig}(pk_{sig}, m, \sigma)$.
   (h) Compute $b' := Verify(vk_{\mathsf{WITHDRAW}}, x, \pi_{\mathsf{WITHDRAW}})$, and output $b \wedge b'$.
3. If given a ZK-rollup transaction $tx = tx_{rollup}$:
   (a) Parse tx_{rollup} as $(rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{\mathsf{ROLLUP}})$
   (b) If rt^{old} does not appear on L, output $b := 0$.
   (c) If rt^{new} appears on L, output $b := 0$.
   (d) If $N^{rollup} <= 0$ or $N^{rollup} > |Q^{cm}|$, output $b := 0$.
   (e) Set $x := (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup})$.
   (f) Compute $b := Verify(vk_{ROOLUP}, x, \pi_{\mathsf{ROLLUP}})$, and output b

Figure 4: Construction of Rollup, Receive, VerifyTransaction algorithms.

5.1 Deposit

During the deposit experiment, a user submitted a deposit transaction $tx_{deposit}$ to the blockchain. The blockchain then locked the token and synced the transaction to the destination chain. The experiment result shows in Table 2.

	Transaction size(byte)	Gas amount (Ethereum)
deposit	484	208,506

Table 2: Deposit experiment results.

5.2 Withdraw

During the withdraw experiment, a user computed a witness and generated the ZK-snark proof as described in Section 4.2. We evaluated the running time (in ms) of each stage in different environments as shown in Table 1. The user then submitted a withdraw transaction $tx_{withdraw}$ to the blockchain. The obtained results are reported in Table 3. We evaluated different withdraw types denoted as $n * m$ where n is the number of input coins and m counts the output coins. For example, $1 * 0$ means the user withdrew a coin to a public address, and $1 * 1$ means the user withdrew a coin from the source chain and created a new coin on the destination chain. For each type of withdraw, we evaluated the transaction forty times and averaged the measured time. We also recorded the transaction (TX) size in bytes and the required gas amount on Ethereum.

	Type	ZK-snark	$\text{Computer}_1(ms)$	$\text{Computer}_2(ms)$	TX size	Gas
withdraw	$1 * 0$	compute-witness	238	317	1,284	527,105
		generate-proof	5,140	5,652		
		verify	5	6		
	$1 * 1$	compute-witness	292	450	1,636	618,247
		generate-proof	5,195	6,175		
		verify	5	6		
	$1 * 2$	compute-witness	298	439	1,988	713,191
		generate-proof	5,284	6,289		
		verify	5	6		
	$2 * 0$	compute-witness	562	829	1,508	629,992
		generate-proof	9,779	11,685		
		verify	5	7		
	$2 * 1$	compute-witness	566	842	1,860	716,400
		generate-proof	9,892	11,789		
		verify	6	7		
	$2 * 2$	compute-witness	571	839	2,212	811,270
		generate-proof	9,975	11882		
		verify	6	7		

Table 3: Withdraw experiment results.

5.3 ZK-rollup

We evaluated ZK-rollup using a similar approach to the withdraw experiment. The experiment results are reported in Table 4. The Rollup requires the number of input commitments in the power of 2; therefore, we evaluated the protocol with 2, 4, 8, and 16 commitments. For each type of ZK-rollup, we evaluated the transaction forty times and averaged the measured time.

5.4 Discussion

We showed the performance and gas consumption of our protocol. Generation of zero-knowledge proofs is the most time-consuming process, which takes up to 30 seconds when ZK-rollup 16 commitments on Computer_2. The performance is practically acceptable considering other zero-knowledge payment protocols take much more time,

	Type	ZK-snark	Computer$_1(ms)$	Computer$_2(ms)$	TX size	Gas
ZK-rollup	2	compute-witness	595	755	356	327,014
		generate-proof	6,360	7,604		
		verify	3	5		
	4	compute-witness	601	988	356	331,636
		generate-proof	6,545	8,314		
		verify	3	5		
	8	compute-witness	1,142	1,876	356	340,805
		generate-proof	11,057	14,396		
		verify	4	5		
	16	compute-witness	2,223	3,670	356	410,186
		generate-proof	20,700	27,023		
		verify	4	5		

Table 4: Zk-rollup experiment results.

e.g., 75 seconds for Zcash [6] and 120 seconds for Monero [7]. A one-to-one zero-knowledge transfer ($1 * 1$) averagely consumes 643,884 ($618247 + 410186/16$) gas when ZK-rollup 16 commitments. As a comparison, a zero-knowledge transfer in Azeroth [**?**], an auditable zero-knowledge protocol, consumes 1,555,957 gas. If we issue a $2*2$ transaction, the average one-to-one zero-knowledge transfer can further optimize to 413,146 ($811270/2 + 410186/16$) gas.

6 Completeness and Security of the Protocol

In this section, we give a formal definition of the completeness and security of the protocol and our main theorem. We then prove the theorem.

Theorem 1 *The tuple* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *is complete and secure.*

6.1 Completeness

In this part, we formally define the completeness of the protocol.

Definition 3 *A protocol* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *is complete if for every polynomial-size ledger sample* $\mathcal{S}$ *and sufficiently large* λ, $Adv_{\Pi,S}^{INCOMP}(\lambda) < negl(\lambda)$, *where* $Adv_{\Pi,S}^{INCOMP}(\lambda) := Pr[INCOMP(\Pi, S, \lambda) = 1]$ *is* $\mathcal{S}$*'s advantage in the incompleteness experiment.*

We now describe the incompleteness experiment $INCOMP$. This experiment is an interaction challenger game between a ledger sampler $\mathcal{S}$ and a challenger $\mathcal{C}$. At the beginning, $\mathcal{C}$ samples public parameters $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and sends to $\mathcal{S}$. $\mathcal{S}$ then samples a ledger L and sends back to $\mathcal{C}$. $\mathcal{C}$ also sends a set of coins $[c]$ and parameters for a withdraw transaction, i.e., secret address key $addr_{sk}$, public value v^{pub}, new coin values v_1^{new}, v_2^{new}, new address public keys $addr_{pk,1}^{new}, addr_{pk,2}^{new}$, user's audit key pair (sk_u^a, pk_u^a), auditors' public keys $[pk_{enc}^a]$, and plain text address $ADDR$. Assume, without loss of generality, $\mathcal{C}$ sends two coins c_1, c_2, and there are three auditors. After receiving message, $\mathcal{C}$ checks validations on $\mathcal{S}$'s message.

Firstly, $\mathcal{C}$ checks if c_1, c_2 are valid coins, i.e. they are well formatted as defined in section 1.2. Then, $\mathcal{C}$ checks that values are balanced, i.e. $v_1 + v_2 = v_1^{new} + v_2^{new} + v^{pub}$. $\mathcal{C}$ aborts and outputs 0 if any checks fail.

Otherwise, $\mathcal{C}$ calculate a withdraw transaction with following steps:

1. Compute the Merkle tree root rt over all coin commitments in L
2. Compute authenticated paths from c_1's commitment cm_1 and c_2's commitment cm_2 to rt
3. Compute $tx_{withdraw} \leftarrow \mathsf{Withdraw}(pp, rt, path_1, path_2, addr_{sk,1}, addr_{sk,2}, v^{pub}, v_1^{new}, v_2^{new}, addr_{pk,1}^{new}, addr_{pk,2}^{new}, (sk_u^a, pk_u^a), pk_{enc,1}^a, pk_{enc,2}^a, pk_{enc,3}^a, ADDR)$

Finally, $\mathcal{C}$ outputs 1 iff following cases hold:

[6] https://z.cash/support/faq/
[7] https://www.monero.how/how-long-do-monero-transactions-take

- $tx_{withdraw} \neq (rt, sn_1, sn_2, cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{\mathsf{WITHDRAW}}, msg_1^a, msg_2^a, msg_3^a, *)$, or
- $tx_{withdraw}$ is not valid, i.e. $\mathsf{VerifyTransaction}(pp, tx_{withdraw}, L_{src,dst})$ outputs 0.

6.2 Security

In this section, we formally define the three secure properties: ledger indistinguishability, transaction non-malleability, and balance. All properties are defined as interaction games between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. We also introduce an oracle $\mathcal{O}^{PRO}$ to simulate the behavior of honest parties. We first describe $\mathcal{O}^{PRO}$ as follows.

$\mathcal{O}^{PRO}$ initially stores a ledger L^{PRO}, a set of address $ADDR^{PRO}$, a set of user's audit key AUD^{PRO}, a set of coins $COIN^{PRO}$, and they all start out empty. $\mathcal{O}^{PRO}$ supports different queries, denoted as $\mathcal{Q}$, as described below:

- $\mathcal{Q} = (\mathsf{CreateAddress})$
  - Compute $(addr_{pk}, addr_{sk}) := \mathsf{CreateAddress}(pp)$.
  - Add the address pair $(addr_{pk}, addr_{sk})$ to $ADDR^{PRO}$.
  - Output the address public key $addr_{pk}$
- $\mathcal{Q} = (\mathsf{CreateAuditKey})$
  - Compute $(pk_u^a, sk_u^a) := \mathsf{CreateAuditKey}(pp)$.
  - Add the address pair (pk_u^a, sk_u^a) to AUD^{PRO}.
  - Output the address public key pk_u^a
- $\mathcal{Q} = (\mathsf{Deposit}, v, addr_{pk})$
  - Compute $(c, tx_{mint}) := \mathsf{Deposit}(pp, v, addr_{pk})$
  - Add the deposit transaction $tx_{deposit}$ to L
  - Output $\perp$
- $\mathcal{Q} = (\mathsf{Withdraw}, idx_1, idx_2, , addr_{sk,1}, addr_{sk,2}, v^{pub}, v_1^{new}, v_2^{new}, addr_{pk,1}^{new}, addr_{pk,2}^{new}, (sk_u^a, pk_u^a), pk_{enc,1}^a, pk_{enc,2}^a, pk_{enc,3}^a, ADDR)$
  - Compute rt, the root of a Merkle tree over all coin commitments in L^{PRO}
  - For each $i \in \{1, 2\}$: Let cm_i be the idx_i-th coin commitment in L, tx_i be the deposit/withdraw transaction in L^{PRO} that contain cm_i, c_i be the first coin in $COIN^{PRO}$ with coin commitment cm, $(addr_{pk,i}, addr_{sk,i})$ be the first key pair in $ADDR^{PRO}$ with $addr_{pk,i}$ being c_i's address. Compute $path_i$, the authentication path from cm_i to rt
  - Compute $(c_1^{new}, c_2^{new}, tx_{withdraw}) := \mathsf{Withdraw}(pp, rt, path_1, path_2, addr_{sk,1}, addr_{sk,2}, v^{pub}, v_1^{new}, v_2^{new}, addr_{pk,1}^{new}, addr_{pk,2}^{new}, (sk_u^a, pk_u^a), pk_{enc,1}^a, pk_{enc,2}^a, pk_{enc,3}^a, ADDR)$
  - Verify that $\mathsf{VerifyTransaction}(pp, tx_{withdraw}, L)$ outputs 1.
  - Add the withdraw transaction to L.
  - Output $\perp$.
- $\mathcal{Q} = (\mathsf{Rollup}, N^{rollup})$
  - Look up a queue Q^{cm} containing N^{rollup} commitments waiting for rollup in L^{PRO}. (If no such transaction is found, abort.)
  - Let $c_1, \ldots, c_n$ be the coins in Q^{cm}.
  - Compute rt^{old}, the root of a Merkle tree over all rolluped coin commitments in L^{PRO}.
  - Compute $path$, the authentication path from commitments to rt^{old}.
  - Compute $tx_{rollup} \leftarrow \mathsf{Rollup}(pp, N^{rollup}, Q^{cm}, rt^{old}, path)$.
  - Add $c_1, ..., c_n$ to $COIN^{PRO}$

– Output $\perp$.

- $\mathcal{Q} = (\mathsf{Receive}, addr_{pk})$
  - Look up $(addr_{pk}, addr_{sk})$ in $ADDR^{PRO}$. (If no such key pair is found, abort.)
  - Compute $(c_1, ..., c_n) \leftarrow \mathsf{Receive}(pp, (addr_{sk}, addr_{pk}), L^{PRO})$.
  - Add $c_1, ..., c_n$ to $COIN^{PRO}$
  - Output $(cm_1, ..., cm_n)$ the corresponding coin commitments.
- $\mathcal{Q} = (\mathsf{Insert}, tx)$
  - Verify that $\mathsf{VerifyTransaction}(pp, tx, L)$ outputs 1. (Else, abort.)
  - Add the deposit/withdraw transaction tx to L^{PRO}
  - Run Rollup and Reveive for all address $addr_{pk}$ in $ADDR$; this updates the $COIN^{PRO}$ with any coins that might have been sent to honest parities via tx.
  - Output $\perp$.

6.2.1 Ledger indistinguishability

Definition 4 *Let* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *be a protocol. We say that* Π *is* L-IND *secure if, for every poly(λ)-size adversary* $\mathcal{A}$ *and sufficiently large* λ, $Adv^{\mathsf{L\text{-}IND}}_{\Pi,A}(\lambda) < negl(\lambda)$, *where* $Adv^{\mathsf{L\text{-}IND}}_{\Pi,A}(\lambda) := 2 \cdot Pr[\mathsf{L\text{-}IND}(\Pi, A, \lambda) = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* L-IND *experiment.*

We now describe the ledger indistinguishability experiment L-IND. This experiment is an interaction challenger game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.
Setup. At the beginning, $\mathcal{C}$ samples a random bit $b \in (0, 1)$ and public parameters $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$. $\mathcal{C}$ then initializes two oracle $\mathcal{O}_0^{PRO}$ and $\mathcal{O}_1^{PRO}$ using pp.
Main part. Let L_{left} be the current ledger in $\mathcal{O}_b^{PRO}$ and L_{right} be the current ledger in $\mathcal{O}_{1-b}^{PRO}$. $\mathcal{C}$ provides (L_{left}, L_{right}) to $\mathcal{A}$; $\mathcal{A}$ then sends two queries

$$\mathcal{Q}, \mathcal{Q}' \in \{\mathsf{CreateAddress}, \mathsf{CreateAuditKey}, \mathsf{Deposit}, \mathsf{Withdraw}, \mathsf{Rollup}, \mathsf{Receive}, \mathsf{Insert}\}$$

to $\mathcal{C}$, while $\mathcal{Q}$ and $\mathcal{Q}'$ should be public consistent. If query type is Insert, $\mathcal{C}$ forwards $\mathcal{Q}$ to $\mathcal{O}_b^{PRO}$, and $\mathcal{Q}'$ to $\mathcal{O}_{1-b}^{PRO}$. Otherwise, $\mathcal{C}$ first check if $\mathcal{Q}$ and $\mathcal{Q}'$ are public consistent and then forwards $\mathcal{Q}$ to $\mathcal{O}_0^{PRO}$ and $\mathcal{Q}'$ to $\mathcal{O}_1^{PRO}$. Let a_0 and a_1 be the two oracle answer, $\mathcal{C}$ then sends (a_b, a_{1-b}) to $\mathcal{A}$.

$\mathcal{A}$ and $\mathcal{C}$ may repeat the **Main part** several times. At the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a guess bit $b' \in (0, 1)$. $\mathcal{C}$ outputs 1 if $b = b'$, or 0 otherwise.
Public consistency Two queries $\mathcal{Q}$ and $\mathcal{Q}'$ are public consistent iff $\mathcal{Q}$ and $\mathcal{Q}'$ are the same type. Furthermore, they are well formatted, and their public information are equal.

6.2.2 Transaction non-malleability

Definition 5 *Let* Π=(Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit) *be a protocol. We say that* Π *is* TR-NM *secure if, for every poly(λ)-size adversary* $\mathcal{A}$ *and sufficiently large* λ, $Adv^{\mathsf{TR\text{-}NM}}_{\Pi,A}(\lambda) < negl(\lambda)$, *where* $Adv^{\mathsf{TR\text{-}NM}}_{\Pi,A}(\lambda) := 2 \cdot Pr[\mathsf{TR\text{-}NM}(\Pi, A, \lambda) = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* TR-NM *experiment.*

We now describe the transaction non-malleability experiment TR-NM. This experiment is an interaction challenger game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. At the beginning, $\mathcal{C}$ samples $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{C}$. $\mathcal{C}$ then initializes an oracle $\mathcal{O}^{PRO}$using pp. $\mathcal{A}$ may send several queries to $\mathcal{O}^{PRO}$. At the end of the experiment, $\mathcal{A}$ sends a withdraw transaction tx' to $\mathcal{C}$. Let $\mathbb{T}$ be the set of all withdraw transaction generated by $\mathcal{O}^{PRO}$. $\mathcal{C}$ outputs 1 iff there exists a $tx \in \mathbb{T}$ s.t. (1) $tx' \neq tx$; (2) $\mathsf{VerifyTransaction}(pp, tx', L) = 1$; and (3) a serial number revealed in tx' is also revealed in tx.

6.2.3 Balance

Definition 6 *Let* $\Pi=($Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit*) be a protocol. We say that* Π *is* BAL *secure if, for every poly(λ)-size adversary* $\mathcal{A}$ *and sufficiently large* λ, $Adv^{\mathsf{BAL}}_{\Pi,A}(\lambda) < negl(\lambda)$, *where* $Adv^{\mathsf{BAL}}_{\Pi,A}(\lambda) := 2 \cdot Pr[\mathsf{BAL}(\Pi, A, \lambda) = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* BAL *experiment.*

We now describe the balance experiment BAL. This experiment is an interaction challenger game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. At the beginning, $\mathcal{C}$ samples $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$. $\mathcal{C}$ then initializes an oracle $\mathcal{O}^{PRO}$using pp. $\mathcal{A}$ may send several queries to $\mathcal{O}^{PRO}$. At the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a set of coin $\mathbb{C}$. $\mathcal{C}$ computes the following quantities.

- $v_{unwithdrawn}$, the total withdrawable coins in $\mathbb{C}$.
- $v_{deposit}$, the total value of all coins deposited by $\mathcal{A}$.
- $v_{ADDR^{PRO}\rightarrow A}$, the total value of payment received by $\mathcal{A}$ from addresses in $ADDR^{PRO}$.
- $v_{A\rightarrow ADDR^{PRO}}$, the total value of payment sent by $\mathcal{A}$ to addresses in $ADDR^{PRO}$.
- $v_{basecoin}$, the total value of public outputs placed by $\mathcal{A}$ on the ledger.

$\mathcal{C}$ outputs 1 iff $v_{unwithdrawn} + v_{basecoin} + v_{A\rightarrow ADDR^{PRO}} > v_{deposit} + v_{ADDR^{PRO}\rightarrow A}$.

6.2.4 Auditability

Definition 7 *Let* $\Pi=($Setup, CreateAddress, CreateAuditKey, Deposit, Withdraw, Rollup, VerifyTransaction, Receive, Audit*) be a protocol. We say that* Π *is* AUD *secure if, for every poly(λ)-size adversary* $\mathcal{A}$ *and sufficiently large* λ, $Adv^{\mathsf{AUD}}_{\Pi,A}(\lambda) < negl(\lambda)$, *where* $Adv^{\mathsf{AUD}}_{\Pi,A}(\lambda) := 2 \cdot Pr[\mathsf{AUD}(\Pi, A, \lambda) = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* AUD *experiment.*

We now describe the balance experiment AUD. This experiment is an interaction challenger game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. At the beginning, $\mathcal{C}$ samples $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$. $\mathcal{C}$ then initializes an oracle $\mathcal{O}^{PRO}$using pp. $\mathcal{A}$ may send several queries to $\mathcal{O}^{PRO}$. At the end of the experiment, $\mathcal{A}$ sends a withdraw transaction tx to $\mathcal{C}$. $\mathcal{C}$ outputs 1 iff (1) $\mathsf{VerifyTransaction}(pp, tx, L) = 1$; and (2)the outputs of $\mathsf{Audit}(\mathsf{msg}^{\mathsf{a}}_1, \mathsf{msg}^{\mathsf{a}}_2, \mathsf{msg}^{\mathsf{a}}_3, \mathsf{pk}^{\mathsf{a}}_{\mathsf{u}}, \mathsf{sk}^{\mathsf{a}}_{\mathsf{enc},1}, \mathsf{sk}^{\mathsf{a}}_{\mathsf{enc},2}, \mathsf{sk}^{\mathsf{a}}_{\mathsf{enc},3})$ are not equal to the input commitments to tx.

6.3 Proof of Theorem 1

In this section, we will sketch the proof of the theorem 1. Similar to [BSCG+14], we also omit the proof of completeness. We then prove the security with three separate proofs.

6.3.1 Ledger indistinguishability.

We prove this property by hybrid experiments from the ledger indistinguishability experiment L-IND to a simulation $\mathsf{SIM}^{\mathsf{L\text{-}IND}}$. In the simulation, the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ as in the experiment, except that all answers are computed independently of the bit $\mathcal{B}$. We then proof that the simulation is indistinguishable from the real experiments.

The simulation $\mathsf{SIM}^{\mathsf{L\text{-}IND}}$ works as follows. The setup stage is similar to the L-IND experiment. However, the zk-SNARKs for withdrawing coins and ZK-rollup are initialized with two simulations $\mathsf{SIM}^{zk}_{WITHDRAW}, \mathsf{SIM}^{zk}_{ROLLUP}$. Then, the challenger $\mathcal{C}$ answers different queries as follows.

- CreateAddress. $\mathcal{C}$ behaves as in L-IND, except that $\mathcal{C}$ replaces a_{pk} in $addr_{pk}$ with a random string. Then, $\mathcal{C}$ stores $addr_{sk}$ in a table and returns $addr_{pk}$ to $\mathcal{A}$.
- CreateAuditKey. The answer is unique to the L-IND experiment.
- Deposit. $\mathcal{C}$ behaves as in L-IND, except that $\mathcal{C}$ computes k as $COMM_r(\tau||\rho)$ where τ is a random string, and $\mathcal{C}$ samples a pk_{enc} and calculates $Ct := E_{enc}(pk_{enc}, r)$ where r is a random string.

- Withdraw. $\mathcal{C}$ computes rt as the accumulation of all the coin commitments after ZK-rollup on L_i. Then, $\mathcal{C}$ samples two uniformly randoms sn_1^{old} and sn_2^{old}. For $i \in \{1,2\}$, if $addr_{pk,i}^{new}$ is generated by CreateAddress, $\mathcal{C}$ samples a random cm_i^{new} and a random pk_{enc} and calculates $Ct_i^{new} := E_{enc}(pk_{enc}, r)$ where r is a random string. For $j \in \{1,2,3\}$, $\mathcal{C}$ samples a random k_j^a and calculates $msg_j^a = SEC.Enc_{k_j^a}(r)$ where r is a random string. Otherwise, calcute cm_i^{new} and Ct_i^{new} as in the Withdraw algorithm. Let h_{sig} be a random string and compute all remain value as in Withdraw algorithm. $\mathcal{C}$ computes the proof π_{WITHDRAW} from the simulation $\mathsf{SIM}_{WITHDRAW}^{zk}$.
- Rollup. $\mathcal{C}$ behaves as in L-IND, except that $\mathcal{C}$ computes the proof π_{ROLLUP} from the simulation $\mathsf{SIM}_{ROLLUP}^{zk}$.
- Receive. The answer is unique to the L-IND experiment.
- Insert. The answer is unique to the L-IND experiment.

In each case, the answer to $\mathcal{A}$ is independent from the bit $\mathcal{B}$. When $\mathcal{A}$ guesses the bit $\mathcal{B}$, $\mathcal{A}$ can only sample a random bit b', i.e., $\mathcal{A}$'s advantage is 0. Next, we will prove that $\mathsf{SIM}^{\mathsf{L\text{-}IND}}$ is indistinguishable from L-IND.

Sketch of Proof: We now describe a sequential of hybrid experiments

$$(\mathsf{L\text{-}IND}, \mathsf{SIM}^{\mathsf{L\text{-}IND}_1}, \mathsf{SIM}^{\mathsf{L\text{-}IND}_2}, \mathsf{SIM}^{\mathsf{L\text{-}IND}_3}, \mathsf{SIM}^{\mathsf{L\text{-}IND}})$$

Let $q_{\mathbf{CA}}$ be the number of CreateAddress queries issued by $\mathcal{A}$. Let $q_{\mathbf{S}}$ be the number of Withdraw queries issued by $\mathcal{A}$. Let $q_{\mathbf{D}}$ be the number of Deposit queries issued by $\mathcal{A}$. For each intermediate experiments, we modify the experiment and show that it is distinguishable from the previous experiment.

- $\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}$: In experiment $\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}$, we simulate the zk-SNARK. For each withdraw transaction, $\mathcal{C}$ computes the proof π_{WITHDRAW} from a simulation $\mathsf{SIM}_{WITHDRAW}^{zk}$. For each ZK-rollup transaction, $\mathcal{C}$ computes the proof π_{ROLLUP} from a simulation $\mathsf{SIM}_{ROLLUP}^{zk}$. Since zk-SNARK is perfect zero knowledge, the simulation proof π_{WITHDRAW} and π_{ROLLUP} should be indistinguishable from a real proof. Hence $Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}} = 0$.
- $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$: The experiment $\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}$ modifies experiment $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ by replacing ciphertext in withdraw transactions with an encryption of a random string. More precisely, we modify $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ so that:
  - each time $\mathcal{A}$ issues a Deposit query, $\mathcal{C}$ samples a pk_{enc} and calculates $Ct := E_{enc}(pk_{enc}, r)$ where r is a random string.,
  - each time $\mathcal{A}$ issues a Withdraw query, for $i \in \{1,2\}$, if $addr_{pk,i}^{new}$ is generated by CreateAddress, $\mathcal{C}$ samples a random pk_{enc} and calculates $Ct_i^{new} := E_{enc}(pk_{enc}, r)$ where r is a random string. For $j \in \{1,2,3\}$, $\mathcal{C}$ samples a random k_j^a and calculates $msg_j^a = SEC.Enc_{k_j^a}(r)$ where r is a random string.

  By Lemma 1, we claim that $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}}| \leq 2(q_{\mathbf{D}} + 5q_{\mathbf{S}})Adv^{\mathsf{Enc}}$.
- $\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}$: The experiment $\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}$ modifies $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ by replacing all PRF results with random values. More precisely, we modify $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ so that :
  - each time $\mathcal{A}$ issues a CreateAddress query, the value a_{pk} in $addr_{pk}$ is substituted with a random string of the same length; and
  - each time $\mathcal{A}$ issues a Withdraw query, the serial number sn^{old} and the signature h are substituted with random strings fo the same length.

  By Lemma 2, we claim that $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}}| \leq q_{\mathbf{CA}} Adv^{\mathsf{PRF}}$.
- $\mathsf{SIM}^{\mathsf{L\text{-}IND}}$: We already describe the experiment $\mathsf{SIM}^{\mathsf{L\text{-}IND}}$ above. More precisely, we modify $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ so that each time $\mathcal{A}$ issues a Deposit query, the commitment cm in $tx_{deposit}$ is substituted with a commitment to a random input. If $\mathcal{A}$ issues a Withdraw query, for $i \in \{1,2\}$, if $addr_{pk,i}^{new}$ is generated by CreateAddress, $\mathcal{C}$ samples a random cm_i^{new}. By Lemma 3, we claim that $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}}| \leq (q_{\mathbf{D}} + 4q_{\mathbf{S}})Adv^{\mathsf{COMM}}$.

By summing over $\mathcal{A}$'s advantages in the hybrid experiments, we can bound $\mathcal{A}$'s advantage in L-IND by $Adv_{\Pi,A}^{\mathsf{L\text{-}IND}}(\lambda) \leq 4(q_{\mathbf{D}} + q_{\mathbf{S}})Adv^{\mathsf{Enc}} + 2(q_{\mathbf{D}} + 5q_{\mathbf{S}})Adv^{\mathsf{Enc}} + (q_{\mathbf{D}} + 4q_{\mathbf{S}})Adv^{\mathsf{COMM}}$, which is negligible in λ.

Lemma 1 *Let Adv^{Enc} be $\mathcal{A}$'s advantages in the* IND-CCA *and* IK-CCA *experiments against the encryption scheme Enc. Since Ecies scheme is an instance of Enc, we use Adv^{Enc} to denotes $\mathcal{A}$'s advantages in the* IND-CCA *and* IK-CCA *experiments against the encryption scheme Ecies. Then after $q_{\mathbf{D}}$* Deposit *queries and $q_{\mathbf{S}}$* Withdraw *queries, $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}}| \leq 4(q_{\mathbf{D}} + q_{\mathbf{S}})Adv^{\mathsf{Enc}}$.*

Proof sketch. Let H be an intermediate simulation between $\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}$ and $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$ in which H modifies $\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}$ by replacing encrypt key with a new sampled key from the key generation algorithm. We first discuss H and $\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}$. When $\mathcal{A}$ queries CreateAddress, $\mathcal{C}$ queries the IK-CCA challenger to obtain $(pk_{enc,0}, pk_{enc,1})$ and return $pk_{enc} := pk_{enc,0}$ to $\mathcal{A}$. When $\mathcal{A}$ issues a Deposit query or a Withdraw query, $\mathcal{C}$ queries the IK-CCA challenger over the plain text m and receive $Ct^* := E_{enc}(pk_{enc,b}, m)$ where $\mathcal{B}$ is chosen by the IK-CCA challenger. $\mathcal{C}$ then replaces Ct with Ct^* and adds the result $tx_{deposit}$ to the ledger. $\mathcal{A}$ outputs a bit b', which is our answer to the IK-CCA experiment. If the maximum advantage of the IK-CCA experiment is Adv^{Enc}, by using a hybrid game, we say that $|Adv^{\mathsf{SIM}^{\mathsf{H}}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}}| \leq (q_{\mathbf{D}} + 5q_{\mathbf{S}})Adv^{\mathsf{Enc}}$.

We made a similar argument for H and $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$. Overall, the advantage is $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_1}}| \leq 2(q_{\mathbf{D}} + 5q_{\mathbf{S}})Adv^{\mathsf{Enc}}$.

Lemma 2 *Let Adv^{PRF} be $\mathcal{A}$'s advantages in distinguishing PRF from a true random function. Then after $q_{\mathbf{CA}}$* CreateAddress *queries, $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}}| \leq q_{\mathbf{CA}}Adv^{\mathsf{PRF}}$.*

Proof sketch. As mentioned in section 4.1, all pseudorandom functions $PRF^{addr}_{a_{sk}}$, $PRF^{sn}_{a_{sk}}$, and $PRF^{pk}_{a_{sk}}$ are constructed from $PRF_{a_{sk}}$. Let $\mathcal{O}$ be the oracle implementing $PRF_{a_{sk}}$ or a true random function. Let a_{sk} be the random seed generated by the oracle from the PRF experiment in answering the first CreateAddress query. If $\mathcal{O}$ implements the $PRF_{a_{sk}}$, the experiment distribution is identical to $\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}$. By using a hybrid game, we say that $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_2}}| \leq q_{\mathbf{CA}}Adv^{\mathsf{PRF}}$.

Lemma 3 *Let Adv^{COMM} be $\mathcal{A}$'s advantages against the hiding property of $COMM$. Then after $q_{\mathbf{D}}$* Deposit *queries and $q_{\mathbf{S}}$* Withdraw *queries, $|Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}}} - Adv^{\mathsf{SIM}^{\mathsf{L\text{-}IND}_3}}| \leq (q_{\mathbf{D}} + 4q_{\mathbf{S}})Adv^{\mathsf{COMM}}$.*

Proof sketch. We make a similar argument as in Lemma 2. We replace internal commitments in k in Deposit and Withdraw with a random value. By using a hybrid game, the advantage is bounded by $(q_{\mathbf{D}} + 2q_{\mathbf{S}})Adv^{\mathsf{COMM}}$. We then replace the coin commitment in Withdraw with a random value, the overall advantage is $(q_{\mathbf{D}} + 4q_{\mathbf{S}})Adv^{\mathsf{COMM}}$.

6.3.2 Transaction non-malleability.

Define $\epsilon := Adv^{\mathsf{TR\text{-}NM}}_{\Pi,A}(\lambda)$. Let $\mathbb{T}$ be the set of withdraw transactions generated by $\mathcal{O}^{PRO}$ in response to Withdraw queries. Set $h'_{sig} := CRH(pk'_{sig})$ corresponding to tx'. Let pk_{sig} be the corresponding public key in tx and set $h_{sig} := CRH(pk_{sig})$. Let $\mathcal{Q}_{\mathbf{CA}} = \{a_{sk,1}, ..., a_{sk,q_{\mathbf{CA}}}\}$ be the set of internal address keys created by $\mathcal{C}$ in response to $\mathcal{A}$'s CreateAddress queries. Let $\mathcal{Q}_{\mathbf{S}} = \{pk_{sig,1}, ..., pk_{sig,q_{\mathbf{S}}}\}$ be the set of signature public keys created by $\mathcal{C}$ in response to $\mathcal{A}$'s Withdraw queries. Then, we decompose the event in which $\mathcal{A}$ wins into the following four disjoint events.

- EVENT_{sig} : $\mathcal{A}$ wins the TR-NM experiment, and there is $pk''_{sig} \in \mathcal{Q}_{\mathbf{S}}$ such that $pk'_{sig} = pk''_{sig}$.
- EVENT_{col} : $\mathcal{A}$ wins, and above event does not occur, and there is $pk''_{sig} \in \mathcal{Q}_{\mathbf{S}}$ such that $h'_{sig} = CRH(pk''_{sig})$.
- EVENT_{mac} : $\mathcal{A}$ wins, and above two events do not occur, and $h' = PRF^{pk}_{a}(i|h_{sig})$ for some $i \in \{1, 2\}$ and $a \in \mathcal{Q}_{\mathbf{CA}}$.
- EVENT_{key} : $\mathcal{A}$ wins, and above three events do not occur, and $h' \neq PRF^{pk}_{a}(i|h_{sig})$ for all $i \in \{1, 2\}$ and $a \in \mathcal{Q}_{\mathbf{CA}}$.

Clearly, $\epsilon = Pr[\mathsf{EVENT}_{sig}] + Pr[\mathsf{EVENT}_{col}] + Pr[\mathsf{EVENT}_{mac}] + Pr[\mathsf{EVENT}_{key}]$. Then, we bound the probability of each event and show that they are all negligible to λ.

Bound the probability of EVENT_{sig}: Define $\epsilon_1 := Pr[\mathsf{EVENT}_{sig}]$. We proof the statement that ϵ_1 is negligible in λ by contradiction. More precisely, if ϵ_1 is not negligible, $\mathcal{A}$ can forge the signature with more than negligible probability, which breaks the SUF-1CMA security.

Let σ' be the signature in tx', and σ'' be the signature in the first withdraw transaction in $tx'' \in \mathbb{T}$ that contains pk''_{sig}. Let m' be everything in tx' other than σ'. Let m'' be everything in tx'' other than σ''. Observe that whenever $tx' \neq tx''$ we also have $(m', \sigma') \neq (m'', \sigma'')$. We first show that $tx' = tx''$ with negligible probability by contradiction. Since, by the definition of TR-NM, tx' and tx share the same serial number. Suppose $tx' = tx''$ then tx and tx'' also share the same serial number, which is bound by the negligible probability that $\mathbb{T}$ contains two transactions that share the same serial number.

Next, we describe an algorithm $\mathcal{B}$, which uses $\mathcal{A}$ as a subroutine, that wins the SUF-1CMA game against Sig with $\epsilon_1/q_{\mathbf{P}}$.

1. $\mathcal{B}$ chooses a random $i \in \{1, 2 \ldots, q_{\mathbf{S}}\}$
2. $\mathcal{B}$ conducts the TR-NM with $\mathcal{A}$. When $\mathcal{A}$ issues the i-th Withdraw query, $\mathcal{B}$ substitutes pk''_{sig} in the resulting transactions tx''. $\mathcal{B}$ then queries the SUF-1CMA challenger and obtain the signature σ'' for the message m'' and substitutes σ'' in tx''.
3. When $\mathcal{A}$ outputs tx', $\mathcal{B}$ looks into tx' and finds m' and σ'.
4. If $pk''_{sig} \neq pk'_{sig}$, $\mathcal{B}$ aborts; otherwise, $\mathcal{B}$ outputs m' and σ' as a forgery for Sig.

Because Sig is SUF-1CMA, ϵ_1 must be negligible in λ.

Bound the probability of EVENT_{col}: Define $\epsilon_2 := Pr[\mathsf{EVENT}_{col}]$. When EVENT_{col} occurs, $\mathcal{A}$ find a collision $CRH(pk'_{sig}) = CRH(pk''_{sig})$. Because CRH is collision resistant, ϵ_2 must be negligible in λ.

Bound the probability of EVENT_{mac}: Define $\epsilon_3 := Pr[\mathsf{EVENT}_{mac}]$. We state that when EVENT_{mac} occurs, $\mathcal{A}$ could distinguish between the PRF with a truly random. Therefore, ϵ_3 must be negligible in λ.

Bound the probability of EVENT_{key}: Define $\epsilon_4 := Pr[\mathsf{EVENT}_{key}]$. If EVENT_{key} occurs, there exists an algorithm $\mathcal{B}$ s.t. $\mathcal{B}$ finds collisions for PRF^{sn}. Therefore, ϵ_4 must be negligible in λ.

6.3.3 Balance.

To withdraw more coins than a user owns, $\mathcal{A}$ may insert a transaction on the ledger. We now modify the experiment in a way that does not affect $\mathcal{A}$'s view. For each zk-SNARK instance $x := ([(rt, sn, h)], v^{pub}, cm_1^{new}, cm_2^{new}, h_{sig}, pk_u^a, [pk_{enc}^a], [msg^a])$ in a withdraw transaction, $\mathcal{C}$ computes a witness $a := ([(path, c, addr_{sk})], c_1^{new}, c_2^{new}, [cm^a], sk_u^a)$. $\mathcal{C}$ may do so with a knowledge extractor. Afterwards, $\mathcal{C}$ obtains an *augmented ledger* $(L, \vec{a})$ where $\vec{a}$ is a list of witness a. Note that $(L, \vec{a})$ is a list of matched pairs $(tx_{withdraw}, a)$ where $tx_{withdraw}$ is a withdraw transaction and a is the corresponding witness. Define $\epsilon := Adv_{\Pi,A}^{\mathsf{BAL}}(\lambda)$. We then define the balance property respected to the modified BAL experiment. We say an augmented ledger balanced if the following holds:

1. Each $(tx_{withdraw}, a)$ in $(L, \vec{a})$ contains openings of two valid coin commitments cm_1 and cm_2, and each cm is a output coin commitment of a deposit or withdraw transaction preceding $tx_{withdraw}$ on L.
2. No two $(tx_{withdraw}, a)$ and $(tx'_{withdraw}, a')$ in $(L, \vec{a})$ contain openings of the same coin commitment.
3. Each $(tx_{withdraw}, a)$ in $(L, \vec{a})$ contains opening of cm_1 cm_2 cm_1^{new} cm_2^{new} to value v_1 v_2 v_1^{new} v_2^{new}, and $v_1 + v_2 = v_1^{new} + v_2^{new} + v^{pub}$.
4. For each $(tx_{withdraw}, a)$ in $(L, \vec{a})$, and for each $i \in \{1, 2\}$:
   (a) If cm_i is the output of a deposit transaction $tx_{deposit}$ on L, then the value v in $tx_{deposit}$ is equal to v_i.
   (b) If cm_i is the output of a withdraw transaction $tx'_{withdraw}$ on L, then its witness a' contains an opening of cm_i to a value v' that is eqaul to v_i
5. For each $(tx_{withdraw}, a)$ in $(L, \vec{a})$, where $tx_{withdraw}$ is inserted by $\mathcal{A}$, if any cm is the output of a previous transaction tx', the public address in tx' is not in $ADDR^{PRO}$. Recall that $ADDR^{PRO}$ is the set of address pairs created by $\mathcal{A}$'s CreateAddress queries.

We then prove that $\mathcal{A}$ cannot violate each case with more than negligible probability.

$\mathcal{A}$ **violates Condition 1**: By the construction of $\mathcal{O}^{PRO}$, $\mathcal{A}$ cannot violate the condition.

$\mathcal{A}$ **violates Condition 2**: If $\mathcal{A}$ violates Condition 2, L contains two withdraw transactions $tx_{withdraw}$ and $tx'_{withdraw}$ with the same cm. Since both transactions are valid, they must contain different serial numbers, namely $sn = sn'$. However, if both transactions withdraw cm but product different serial number, then the corresponding witness a, a' contain different openings of cm. This violates the binding property of the commitment scheme $COMM$.

$\mathcal{A}$ **violates Condition 3**: By the construction of the NP statement $WITHDRAW$, this must hold. Otherwise, the zk-SNARK is violated.

$\mathcal{A}$ **violates Condition 4**: If $\mathcal{A}$ violates Condition 4, L contains

1. a deposit transaction $tx_{deposit}$ and a withdraw transaction $tx_{withdraw}$, or
2. two withdraw transactions $tx_{withdraw}$ and $tx'_{withdraw}$

s.t. both transactions have the same commitment cm but open cm to different values. This violates the binding property of the commitment scheme $COMM$.

$\mathcal{A}$ **violates Condition 5**: If $\mathcal{A}$ violates Condition 5, L contains an inserted withdraw transaction $tx_{withdraw}$ s.t. $tx_{withdraw}$ withdraws a coin deposited by a previous deposit transaction $tx_{deposit}$. Notably, $tx_{deposit}$'s public address $addr_{pk} = (a_{pk}, pk_{enc})$ lies in $ADDR$, and the witness associated to $tx_{deposit}$ contains a_{sk} s.t. $a_{pk} = PRF^{addr}_{a_{sk}}(0)$. One can construct a new adversary $\mathcal{B}$ that, by using $\mathcal{A}$ as a subroutine, distinguish PRF from a random function.

6.3.4 Auditability.

If the auditors fail to recover the commitments, it implies

1. any decryption of the messages is wrong; or
2. the secret sharing recovery result is wrong; or
3. π_{WITHDRAW} in tx is valid while $[cm^a] \neq Share^{(t,n)}([cm])$.

$\mathcal{A}$ **violates Condition 1**: If $\mathcal{A}$ violates Condition 1, this breaks the security of $ECIES$.
$\mathcal{A}$ **violates Condition 2**: If $\mathcal{A}$ violates Condition 2, this breaks the PER-SS of SS.
$\mathcal{A}$ **violates Condition 3**: If $\mathcal{A}$ violates Condition 3, this breaks the *proof of knowledge* of zk-SNARK.

7 Conclusion

In this paper, we proposed an auditable confidentiality protocol for blockchain transactions, which supports both cross-chain and single-chain transactions. We reduced the transaction cost and made the protocol affordable via a ZK-rollup scheme. To comply with regulation requirements, the protocol allows authorized auditors to track transactions. We formally proved the security and implemented the protocol on the Ethereum testnet. The evaluation result showed that the protocol is highly efficient and practical. In the future, we will realize more functionalities, e.g., cross-chain swap and general cross-chain smart contracts.

References

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography Conference*, pages 315–333. Springer, 2013.

[BDPVA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 103–112, New York, NY, USA, 1988. Association for Computing Machinery.

[Bin20] Binance. Binance smart chain: A parallel binance chain to enable smart contracts, 2020. `https://dex-bin.bnbstatic.com/static/Whitepaper_BinanceSmartChain.pdf`.

[BSCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[ET18] Jacob Eberhardt and Stefan Tai. Zokrates - scalable privacy-preserving off-chain computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091, 2018.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, August 2021.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Annual International Cryptology Conference*, pages 581–612. Springer, 2017.

[GMR85] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 321–340, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[HBHW16] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, page 1, 2016.

[KD04] Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 426–442, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[KDJL+19] Hui Kang, Ting Dai, Nerla Jean-Louis, Shu Tao, and Xiaohui Gu. Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 543–555, 2019.

[KNA21] Jaynti Kanani, Sandeep Nailwal, and Anurag Arjun. Polygon lightpaper, 2021. `https://polygon.technology/lightpaper-polygon.pdf`.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[NM16] Shen Noether and Adam Mackenzie. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[NVV18] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 65–80, Renton, WA, April 2018. USENIX Association.

[PBF+18] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

[RPX+22] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. Cryptology ePrint Archive, Paper 2022/1286, 2022. `https://eprint.iacr.org/2022/1286`.

[SBBV22] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 179–197, 2022.

[SBG+19] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1759–1776, 2019.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.