A General Confidentiality Protocol for Blockchain Transactions

Mystiko.Network
March 14, 2023[1]

1 Introduction

In this document, we introduce a general confidentiality protocol with additional zk-rollup for cross-chain and single-chain transactions.

1.1 Motivation

Currently, the write operation is an atomic operation, i.e., after a user sends a cross-chain or single-chain transaction the source blockchain network will transfer it to the destination blockchain network. In such protocol, the sender and receiver addresses are in plain text, and one may track the transaction graph. Many research shows that this setting cannot provide privacy. To address this issue, we employ a similar solution as in Zcash: the transaction is encrypted with the public key of the receiver, and this receiver can then find the transaction and spend the coin. 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. In the new version, we support **JoinSplit** and allow internal transfers. The encrypted transaction could provide privacy; however, it could be abused for criminal purposes. We build a confidential protocol while making it auditable for auditors, and the protocol will not disclose users' transaction data unless a large enough partition of the auditors agree so.

1.2 Protocol Overview

Suppose 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 leger 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) $u1$ can scan over the public ledger and find the transaction $tx_{deposit}$. The user then decrypts Ct and gets (v, ρ, r, s).

TODO: Design of the wallet

(3) When $u1$ wants to spend 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_{SPEND} over the following statements:
For each old coins c, given the Merkle root rt, serial number sn, I know c and address secret key $a_{sk,1}$ s.t.

- c is well-formatted.
- The address secret key matches the public key, i.e., $a_{pk,1} = PRF^{addr}_{a_{sk,1}}(0)$.
- The nullifier key matches the address secret key, i.e., $nk = PRF^{addr}_{a_{sk,1}}(1)$.
- The serial number is computed correctly, i.e., $sn = PRF^{sn}_{nk}(\rho)$.
- 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 spend transaction $tx_{spend} := ([(rt, sn)], cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{SPEND})$ 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 spending a coin, the user samples a key pair (pk_{sig}, sk_{sig}) and use sk_{sig} sign every value associated with the tx_{spend} transaction. The user also computes $h_{sig} := CRH(pk_{sig})$ and $h := PRF^{pk}_{a_{sk}}(h_{sig})$, 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_{spend} transaction. The overview process is illustrated in Figure 1.

To meet regulations, when u_1 spends coins, he 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 he 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 shceme. $(pk^a_{enc,i}, sk^a_{enc,i})$ are auditors' elliptic curve key pair. To provide a zero knowledge friendly, 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 a elliptic curve key pair for audit purpose. We then set $k_i^a = sk_u^a \cdot pk^a_{enc,i} = sk^a_{enc,i} \cdot pk_u^a$ and the encrypted message $msg_i^a = SEC.Enc_{k_i^a}(cm_i^a)$. The

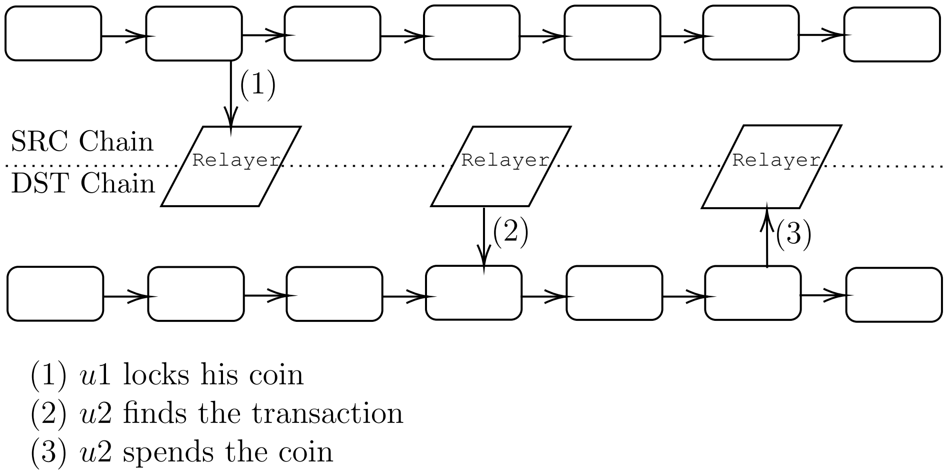

Figure 1: solution overview for write operation

user then proofs following statements along with other statements in π_{SPEND}:
Let G be the generator in the elliptic curve. Given commitments $[cm]$. encrypted messages $[msg^a_1, msg^a_2, \ldots, msg^a_n]$ and public keys pk^a_u, $[pk^a_{enc,i}]$, I know $[cm^a_1, cm^a_2, \ldots, cm^a_n]$ and sk^a_u s.t.

- The commitments are well secret shared, i.e., $[cm^a_1, cm^a_2, \ldots, cm^a_n] = Share^{(t,n)}([cm])$.
- The public key match the private key, i.e., $pk^a_u = sk^a_u G$.
- Each commitments share is well encrypted, i.e., $msg^a_i = SEC.Enc_{sk^a_u pk^a_{enc,i}}(cm^a_i)$.

1.3 Architecture Overview

In this section, we illustrated the overview of architecture. We described the overview protocols and algorithms for depositing and spending coins in section 1.2, and Mystiko implements the algorithms 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 private 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 spend the coin.

If the receiver wants to withdraw the coins, he 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 we operate it on-chain. In Mystiko, we solved this problem with **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. He then sends the proof with the root to the contract, and if the proof is validated, we update the Merkle tree root.

2 Definition of the Protocol

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

2.1 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 appended only.
Public parameters. [1] 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. [2] 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 auditable 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.
Coin. A coin is 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 spending.

[1] Taken from [BSCG$^+$14] **3.1 Data structures Public parameters**
[2] Taken from [BSCG$^+$14] **3.1 Data structures Addresses**

- address, denoted $addr_{pk}(c)$: an address public key, representing who owns c.

Transaction. We introduce three new 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 deposites a coin with commitment cm and value v, which could be spent on other chains.

- Spend transactions. A spend transaction tx_{spend} is a tuple ($[(rt, sn)]$, $cm_1^{new}, cm_2^{new}, v^{pub}$, $ADDR, \pi_{SPEND}$, $[msg_1^a, msg_2^a, \ldots, msg_n^a], *)$, 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_{spend} records that a user spends 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 track the transaction then.

- ZK-rollup transactions. A ZK-rollup transaction tx_{rollup} is a tuple $(rt^{old}, rt^{new}, hash_{[cm]}$, $pathIndices, N^{rollup}, \pi_{ROLLUP}, *)$, 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 spend 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 spend 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.

2.2 Algorithms

The protocol Π is a tuple of polynomial-time algorithms
$Setup$, $CreateAddress$, $Deposit$, $Spend$, $Rollup$, $VerifyTransaction$, $Receive$, $Audit$
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 need to generate at least one address pair. $addr_{pk}$ is public, and $addr_{sk}$ is kept secretly and used to spend the coin sent to the address.
Creating auditable keys. The $CreateAuditableKey$ 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 need to generate at least one auditable 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.
Spending coins. The $Spend$ 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 auditable key pair (sk_u^a, pk_u^a)
- auditors' public keys $[pk_{enc}^a]$

- Outputs:
  - spend transaction tx_{spend}
  - new coins c_1^{new}, c_2^{new}

For each coin c, the *Spend* algorithm takes as inputs an input coin c and its address secret key $addr_{sk}$. The *Spend* 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 *Spend* 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 *Spend* algorithm outputs a spend transaction tx_{spend}. The transaction tx_{spend} equals ($[(rt, sn)]$, $cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{SPEND}$, $[msg_1^a, msg_2^a, \ldots, msg_n^a]$). 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 $(rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{ROLLUP}, *)$, 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 (spend, deposit or ZK-rollup) transaction *tx*
  - the current source and destination ledgers L_{src} and L_{dst}
- Outputs: bit *b*, equals 1 iff the transaction is valid

Deposit, spend, and ZK-rollup transactions must be verified before executed.
Receiving coins.[3] The algorithm *Receive* scans the ledger and retrieves unspent 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 (unspent) received coins

When a user with address key pair $(addr_{pk}, addr_{sk})$ wishes to receive payments sent to $addr_{pk}$, he 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 spent by using *Spend* algorithm.
Audit. The algorithm *Audit* audits user transactions:

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

- 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.

2.3 Completeness

Completeness of a protocol requires that unspent coins can be spent. Suppose a ledger sampler 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 spent using *Spend* transaction. Informality, if *Spend* outputs a tx_spend 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, Deposit, Spend, Rollup,* $VerifyTransaction$*, Receive, Audit) is complete if no polynomial-size ledger sample S wins* $INCOMP$ *with more than negligible probability.*

2.4 Security

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

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

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 spend transaction.
Balance. This property requires no bounded adversary could spend more coins than what he received from the deposit transaction.

3 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.

3.1 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 $CRH : \{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}$.

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 random 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, $\{COMM_x : \{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}\}_x$ where x denotes the trapdoor parameter.

One-time strongly-unforgeable digital signatures. We use a digital signature scheme $Sig = (G_{sig}, K_{sig}, S_{sig}, V_{sig})$.

- $G_{sig}(1^\lambda) \rightarrow pp_{sig}$. Given a security parameters λ, G_{sig} samples public parameters pp_{sig} for the signature scheme.

- $K_{sig}(pp_{sig}) \rightarrow (pk_{sig}, sk_{sig})$. Given public parameters pp_{sig}, K_{sig} samples a public key and a secret key for a single user.

- $S_{sig}(sk_{sig}, m) \rightarrow \sigma$. GIven a secret key sk_{sig} and a message m, S_{sig} signs m to obtain a signature σ.

- $V_{sig}(pk_{sig}, m, \sigma) \rightarrow b$. 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 $Enc = (G_{enc}, K_{enc}, E_{enc}, D_{enc})$.

- $G_{enc}(1^\lambda) \rightarrow pp_{enc}$. Given a security parameter λ, G_{enc} samples public parameters pp_{enc} for the encryption scheme.

- $K_{enc}(pp_{enc}) \rightarrow (pk_{enc}, sk_{enc})$. 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 and key indistinguishability.
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).

Threshold secret sharing. We use a threshold secret sharing scheme $SS = (Share, Recover)$.

- $Share(x) \to [x_1, x_2, \dots, x_n]$. Given a secret x generates n secret shares $[x_1, x_2, \dots, x_n]$.

- $Revocer([x_i, x_{i+1}, \dots, x_{i+t-1}]) \to x$. Given t secret shares $[x_i, x_{i+1}, \dots, x_{i+t-1}]$ generates the secret x.

The secret shareing is t out of n secret sharing, 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.

3.2 zk-SNARKs for spending coins

We use zk-SNARK to prove a NP statement $SPEND$. For the definition of zk-SNARK, we refer to [BCI+13] for a detailed explanation. We first give a 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 $(KeyGen, Prove, Verify)$:

- $KeyGen(1^\lambda, C) \to (pk, vk)$. 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 he is convinced that $x \in L_C$.

We recall the corresponding spend transaction $tx_{spend} = (\ [(rt, sn)]\ , cm_1^{new}, cm_2^{new}, v^{pub}, ADDR, \pi_{SPEND}, [msg_1^a, msg_2^a, \ldots, msg_n^a])$. To spend 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 $SPEND$ 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 $SPEND$ 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., $a_{pk} = PRF^{addr}_{a_{sk}}(0)$.
   (c) The nullifier key nk matches the address secret key, i.e., $nk = PRF^{addr}_{a_{sk,1}}(1)$.
   (d) The serial number sn is computed correctly, i.e., $sn = PRF^{sn}_{nk}(\rho)$.
   (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., $h = PRF^{pk}_{a_{sk}}(h_{sig})$.
2. New coins c^{new}_1 and c^{new}_2 are well formatted, i.e., $cm = COMM_{s^{new}_i}(COMM_{r^{new}_i}(a^{new}_{pk,i}||\rho^{new}_i)||v^{new}_i)$.
3. Balance is preserved, i.e. $\sum v = v^{new}_1 + v^{new}_2 + v^{pub}$.
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., $pk^a_u = sk^a_u G$.
6. Each commitments share is well encrypted, i.e., $msg^a_i = SEC.Enc_{sk^a_u pk^a_{enc,i}}(cm^a_i)$.

3.3 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 3.2. We recall the corresponding ZK-rollup transaction $tx_{rollup} = (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{ROLLUP})$. 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 a 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 $|path| + \log_2 |[cm]| - 1 = H$.

3.4 Algorithm constructions

In this section, we describe the construction of each algorithm. The intuition is given in 2.1 and 2.2. The building blocks are introduced in 3.1 and 3.2. We give the pseudocode for each algorithm.

Setup.

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

1. Construct the arithmetic circuit C_{SPEND} for the $SPEND$ statement at security λ.
2. Compute $(pk_{SPEND}, vk_{SPEND}) := KeyGen(1^{\lambda, C_{SPEND}})$.
3. Construct the arithmetic circuit C_{ROLLUP} for the $ROLLUP$ statement at security λ.
4. Compute $(pk_{ROLLUP}, vk_{ROLLUP}) := KeyGen(1^{\lambda, C_{ROLLUP}})$.
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 $pp := (pk_{SPEND}, vk_{SPEND}, pk_{ROLLUP}, vk_{ROLLUP}, pp_{enc}, pp_{sig}, pp_{ecies})$.
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.

Creating auditable keys.

- 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 $tx_{Deposite} := (cm, v, *)$, where $* := (k, s, Ct)$.
9. Output c and $tx_{Deposite}$.

Spend.

- 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_1^{new}, v_2^{new}
  - new address public keys $addr_{pk,1}^{new}, addr_{pk,2}^{new}$
  - user's auditable key pair (sk_u^a, pk_u^a)
  - auditors' public keys $[pk_{enc,1}^a, pk_{enc,2}^a, \ldots, pk_{enc,n}^a]$
- Outputs:
  - spend transaction tx_{spend}
  - new coins c_1^{new}, c_2^{new}

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 $h := PRF^{pk}_{a_{sk}}(1||h_{sig})$.

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 $x := ([(rt, sn, h)], v^{pub}, cm^{new}_1, cm^{new}_2, h_{sig}, pk^a_u, [pk^a_{enc,1}, pk^a_{enc,2}, \ldots, pk^a_{enc,n}],$ $[msg^a_1, msg^a_2, \ldots, msg^a_n])$.

10. Set $a = ([(path, c, addr_{sk})], c^{new}_1, c^{new}_2, [cm^a_1, cm^a_2, \ldots, cm^a_n], sk^a_u)$.

11. Compute $\pi_{SPEND} := Prove(pk_{SPEND}, x, a)$.

12. Set $m := (x, \pi_{SPEND}, ADDR, Ct^{new}_1, Ct^{new}_2)$.

13. Compute $\sigma := S_{sig}(sk_{sig}, m)$.

14. Set $tx_{spend} = (\ [(rt, sn)]\ , cm^{new}_1, cm^{new}_2, v^{pub}, ADDR, \pi_{SPEND}, [msg^a_1, msg^a_2, \ldots, msg^a_n], *)$, where $* := (pk_{sig}, [h], \sigma, Ct^{new}_1, Ct^{new}_2)$.

15. Output c^{new}_1, c^{new}_2, and tx_{spend}.

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 $\pi_{ROLLUP} := Prove(pk_{ROLLUP}, x, a)$.
9. Set $tx_{rollup} := (rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{ROLLUP})$.
10. Ouput $[cm]$ and tx_{rollup}.

VerifyTransaction.

- Inputs:
  - public parameters pp
  - a (spend 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 spend transaction $tx = tx_{spend}$:
   (a) Parse tx_{spend} as ($[(rt, sn)]$ $, cm^{new}_1, cm^{new}_2, v^{pub}, ADDR, \pi_{SPEND},$ $[msg^a_1, msg^a_2, \ldots,$ $msg^a_n], *)$,
   where $* := (pk_{sig}, [h], \pi_{SPEND}, \sigma, Ct^{new}_1, Ct^{new}_2)$..
   (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 $m := (x, \pi_{SPEND}, ADDR, Ct^{new}_1, Ct^{new}_2)$.
   (g) Compute $b := V_{sig}(pk_{sig}, m, \sigma)$.
   (h) Compute $b' := Verify(vk_{SPEND}, x, \pi_{SPEND})$, and output $b \wedge b'$.
3. If given a ZK-rollup transaction $tx = tx_{rollup}$:
   (a) Parse tx_{rollup} as $(rt^{old}, rt^{new}, hash_{[cm]}, pathIndices, N^{rollup}, \pi_{ROLLUP})$
   (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 $b := Verify(vk_{ROOLUP}, x, \pi_{ROLLUP})$, and output b

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 (unspent) 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 $tx_{Deposite}$ 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
   $c := (addr_{pk}, v, \rho, r, s, cm)$

Audit.

- Inputs:
  - Encrypted commitments sharings $[msg^a_1, msg^a_2, \ldots, msg^a_n]$
  - 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 $[cm]$.

3.5 Concrete design

This part may be updated later.

In this section, we describe how we instantiate each building block. Namely, we build CRH, PRF, $COMM$ from **SHA256**, *Sig* from **ECDSA**, *Enc* from **key-private Elliptic-Curve Integrated Encryption Scheme**.

4 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,* $CreateAddress$*, Deposit, Spend, Rollup,* $VerifyTransaction$*,* $Receive$*,* $Audit$*) is complete and security.*

4.1 Completeness

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

Definition 3 *A protocol* Π=*(Setup,* $CreateAddress$*, Deposit, Spend, Rollup,* $VerifyTransaction$*,* $Receive$*,* $Audit$*) is complete if for every polynomial-size ledger sample* 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* 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 S and a challenger C. At the beginning, C samples public parameters $pp \leftarrow Setup(1^\lambda)$ and sends to S. S then samples a ledger L and sends back to C. S also sends a coin c and parameters for a spend transaction, i.e., secret address key $addr_{sk}$, public value v^{new}, and plain text address $ADDR$. After receiving message, C checks validations on S's message.

Firstly, C checks if c is a valid coin, i.e. c is well formatted as defined in section 1.2. Then, C checks that values are balanced, i.e. $v = v^{new}$. C aborts and outputs 0 if any checks fail.

Otherwise, C calculate a spend transaction with following steps:

1. Compute the Merkle tree root rt over all coin commitments in L
2. Compute the authenticated path from c's commitment cm to $root$
3. Compute $tx_{spend} \leftarrow Spend(pp, rt, path, addr_{sk}, ADDR, v^{new})$

Finally, C outputs 1 iff following cases hold:

- $tx_{spend} \neq (rt, sn, v^{new}, ADDR, *)$, or
- tx_{spend} is not valid, i.e. $VerifyTransaction(pp, tx_{spend}, L_{src,dst})$ outputs 0.

4.2 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 a adversary A and a challenger C. We also introduce an oracle O^{PRO} to simulate the behavior of honest parties. We first describe O^{PRO} as follows.

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

- $Q = (CreateAddress)$
  - Compute $(addr_{pk}, addr_{sk}) := CreateAddress(pp)$.
  - Add the address pair $(addr_{pk}, addr_{sk})$ to $ADDR^{PRO}$.
  - Output the address public key $addr_{pk}$
- $Q = (Deposit, v, addr_{pk})$
  - Compute $(c, tx_{mint}) := Deposit(pp, v, addr_{pk})$
  - Add the coin c to $COIN^{PRO}$
  - Add the deposit transaction $tx_{deposit}$ to L
  - Output $\perp$
- $Q = (Spend, idx, addr_{pk}, ADDR, v^{new})$
  - Compute rt, the root of a Merkle tree over all coin commitments in L^{PRO}
  - Let cm be the idx-th coin commitment in L, tx be the deposit/spend transaction in L^{PRO} that contain cm, c be the first coin in $COIN^{PRO}$ with coin commitment cm, $(addr_{pk}, addr_{sk})$ be the first key pair in $ADDR^{PRO}$ with $addr_{pk}$ being c's address. Compute $path$, the authentication path from cm to rt
  - Compute $(tx_{spend} := Spend(pp, rt, c, addr_{sk}, path, ADDR, v^{new}))$
  - Verify that $VerifyTransaction(pp, tx_{spend}, L)$ outputs 1.
  - Add the spend transaction to L
  - Output $\perp$.
- $Q = (Receive, addr_{pk})$
  - Look up $(addr_{pk}, addr_{sk})$ in $ADDR^{PRO}$. (If no such key pair is found, abort.)
  - Compute $(c_1, ..., c_n) \leftarrow Receive(pp, (addr_{sk}, addr_{pk}), L^{PRO})$.
  - Add $c_1, ..., c_n$ to $COIN^{PRO}$

– Output $(cm_1, ..., cm_n)$ the corresponding coin commitments.

- $Q = (Insert, tx)$
  - Verify that $VerifyTransaction(pp, tx, L)$ outputs 1. (Else, abort.)
  - Add the deposit/spend transaction tx to L^{PRO}
  - Run *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 $\perp$.

4.2.1 Ledger indistinguishability

Definition 4 *Let* Π=*(Setup, CreateAddress, Deposit, Spend, Rollup,* $VerifyTransaction$*, Receive, Audit) be a protocol. We say that* Π *is* $L - IND$ *secure if, for every poly(*λ*)-size adversary* A *and sufficiently large* λ*,* $Adv_{\Pi,A}^{L-IND}(\lambda) < negl(\lambda)$*, where* $Adv_{\Pi,A}^{L-IND}(\lambda) := 2 \cdot Pr[L - IND(\Pi, A, \lambda) = 1] - 1$ *is 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 A and a challenger C.
Setup. At the beginning, C samples a random bit $b \in (0, 1)$ and public parameters $pp \leftarrow Setup(1^\lambda)$, and sends pp to A. C then initializes two oracle O_0^{PRO} and O_1^{PRO} using pp.
Main part. Let L_{left} be the current ledger in O_b^{PRO} and L_{right} be the current ledger in O_{1-b}^{PRO}. C provides (L_{left}, L_{right}) to A; A then sends two queries

$$Q, Q' \in CreateAddress, Deposit, Spend, Receive, Insert$$

to C, while Q and Q' should be public consistent. If query type is $Insert$, C forwards Q to O_b^{PRO}, and Q' to O_{1-b}^{PRO}. Otherwise, C first check if Q and Q' are public consistent and then forwards Q to O_0^{PRO} and Q' to O_1^{PRO}. Let a_0 and a_1 be the two oracle answer, C then sends (a_b, a_{1-b}) to A.

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

4.2.2 Transaction non-malleability

Definition 5 *Let* Π=*(Setup, CreateAddress, Deposit, Spend, Rollup,* $VerifyTransaction$*, Receive, Audit) be a protocol. We say that* Π *is* $TR - NM$ *secure if, for every poly(*λ*)-size adversary* A *and sufficiently large* λ*,* $Adv_{\Pi,A}^{TR-NM}(\lambda) < negl(\lambda)$*, where* $Adv_{\Pi,A}^{TR-NM}(\lambda) := 2 \cdot Pr[TR - NM(\Pi, A, \lambda) = 1] - 1$ *is 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 A and a challenger C. At the beginning, C samples $pp \leftarrow Setup(1^\lambda)$, and sends pp to A. C then initializes an oracle O^{PRO}using pp. A may send several queries to O^{PRO}. At the end of the experiment, A sends a spend transaction tx' to A. Let $\mathbb{T}$ be the set of all spend transaction generated by O^{PRO}. C outputs 1 iff there exists a $tx \in \mathbb{T}$ s.t. (1) $tx' \neq tx$; (2) $VerifyTransaction(pp, tx', L) = 1$; and (3) a serial number revealed in tx' is also revealed in tx.

4.2.3 Balance

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

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

- $v_{unspent}$, the total spendable coins in $\mathbb{C}$.
- $v_{deposit}$, the total value of all coins deposited by A.
- $v_{ADDR^{PRO} \rightarrow A}$, the total value of payment received by A from addresses in $ADDR^{PRO}$.
- v_{spent}, the total value of public outputs placed by A on the ledger.

C outputs 1 iff $v_{unspent} + v_{spent} > v_{deposit} + v_{ADDR^{PRO} \rightarrow A}$.

4.3 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.

4.3.1 Ledger indistinguishability.

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

The simulation SIM^{L-IND} works as follows. The setup stage is similar to the $L - IND$ experiment. However, the zk-SNARK is initialized with a simulation SIM^{zk}. Then, the challenger C answers different queries as follows.

- **CreateAddress.** C behaves as in $L - IND$, except that C replaces a_{pk} in $addr_{pk}$ with a random string. Then, C stores $addr_{sk}$ in a table and returns $addr_{pk}$ to A.
- **Deposit.** C behaves as in $L - IND$, except that C computes k as $COMM_r(\tau||\rho)$ where τ is a random string.
- **Spend.** C computes rt as the accumulation of all the valid coin commitments on L_i. Then, C samples a uniformly random sn^{old}, which is the serial number of the coin c. Let h be a random string and compute all remain value as in *Spend* algorithm. C computes the proof π_{SPEND} from the simulation SIM^{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 A is independent from the bit b. When A guesses the bit b, A can only sample a random bit b', i.e., A's advantage is 0. Next, we will prove that SIM^{L-IND} is indistinguishable from $L - IND$.

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

$$(L - IND, SIM^{L-IND_1}, SIM^{L-IND_2}, SIM^{L-IND})$$

. For each intermediate experiments, we modify the experiment and show that it is distinguishable from the previous experiment.

- SIM^{L-IND_1} : In experiment SIM^{L-IND_1}, we simulate the zk-SNARK. For each spend transaction, C computes the proof π_{SPEND} from a simulation SIM^{zk}. Since zk-SNARK is perfect zero knowledge, the simulation proof π_{SPEND} should be indistinguishable from a real proof. Hence $Adv^{SIM^{L-IND_1}} = 0$.
- SIM^{L-IND_2} : The experiment SIM^{L-IND_2} modifies SIM^{L-IND_1} by replacing all PRF results with random values. More precisely, we modify SIM^{L-IND_1} so that :
  - each time 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 A issues a **Spend query** query, the serial number sn^{old} and the signature h are substituted with random strings fo the same length.

  We claim that $|Adv^{SIM^{L-IND_2}} - Adv^{SIM^{L-IND_1}}|$ is negligible. We omit the proof and refer to [BSCG$^+$14] Lemma D.2.

- SIM^{L-IND} : We already describe the experiment SIM^{L-IND} above. More precisely, we modify SIM^{L-IND_2} so that each time A issues a **Deposit** query, the commitment cm in $tx_{deposit}$ is substituted with a commitment to a random input. We claim that $|Adv^{SIM^{L-IND}} - Adv^{SIM^{L-IND_2}}|$ is negligible. We omit the proof and refer to [BSCG+14] Lemma D.3.

By summing over A's advantages in the hybrid experiments, we can bound A's advantage in $L-IND$ by $Adv^{L-IND}_{\Pi,A}(\lambda) \leq Adv^{SIM^{L-IND_1}} + |Adv^{SIM^{L-IND_2}} - Adv^{SIM^{L-IND_1}}| + |Adv^{SIM^{L-IND}} - Adv^{SIM^{L-IND_2}}|$, which is negligible in λ.

4.3.2 Transaction non-malleability.

Define $\epsilon := Adv^{TR-NM}_{\Pi,A}(\lambda)$. Let τ be the set of spend transactions generated by O^{PRO} in response to *Spend* 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 $Q_{CA} = \{a_{sk,1}, ..., a_{sk,q_{CA}}\}$ be the set of internal address keys created by C in response to A's *CreateAddress* queries. Let $Q_S = \{pk_{sig,1}, ..., pk_{sig,q_S}\}$ be the set of signature public keys created by C in response to A's *Spend* queries. Then, we decompose the event in which A wins into the following four disjoint events.

- $EVENT_{sig}$: A wins the $TR-NM$ experiment, and there is $pk''_{sig} \in Q_S$ such that $pk'_{sig} = pk''_{sig}$.
- $EVENT_{col}$: A wins, and above event does not occur, and there is $pk''_{sig} \in Q_S$ such that $h'_{sig} = CRH(pk''_{sig})$.
- $EVENT_{mac}$: A wins, and above two events do not occur, and $h' = PRF^{pk}_{a_{sig}}(h_{sig})$ and $a_{sig} \in Q_{CA}$.
- $EVENT_{key}$: A wins, and above three events do not occur, and $h' \neq PRF^{pk}_{a_{sig}}(h_{sig})$ and $a_{sig} \in Q_{CA}$.

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

Bound the probability of $EVENT_{sig}$: Define $\epsilon_1 := Pr[EVENT_{sig}]$. We proof the statement that ϵ_1 is negligible in λ by contradiction. More precisely, if ϵ_1 is not negligible, 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 spend transaction in $tx'' \in \tau$ 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 τ contains two transactions that share the same serial number.

Next, we describe an algorithm B, which uses A as a subroutine, that wins the **SUF-1CMA** game against Sig with ϵ_1/q_P. We omit the detail and refer [BSCG$^+$14] section D.2. Because Sig is **SUF-1CMA**, ϵ_1 must be negligible in λ.

Bound the probability of $EVENT_{col}$: Define $\epsilon_2 := Pr[EVENT_{col}]$. When $EVENT_{col}$ occurs, 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 $\epsilon_3 := Pr[EVENT_{mac}]$. We state that when $EVENT_{mac}$ occurs, A could distinguish between the PRF with a truly random. We omit the detail and refer to [BSCG$^+$14] section D.2. Therefore, ϵ_3 must be negligible in λ.

Bound the probability of $EVENT_{key}$: Define $\epsilon_4 := Pr[EVENT_{key}]$. If $EVENT_{key}$ occurs, there exists an algorithm B s.t. B finds collisions for PRF^{sn}. We omit the detail and refer to [BSCG$^+$14] section D.2.

4.3.3 Balance.

To spend more coins than he owns, A may insert a transaction on the ledger. We now modify the experiment in a way that does not affect A's view. For each zk-SNARK instance $x = (rt, sn, v^{new}, h_{sig}, h)$ in a spend transaction, C computes a witness $a = (path, c, addr_{sk})$. C may do so with a knowledge extractor. Afterwards, 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_{spend}, a) where tx_{spend} is a spend transaction and a is the corresponding witness. Define $\epsilon := Adv^{BAL}_{\Pi,A}(\lambda)$. 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_{spend}, a) in $(L, \vec{a})$ contains openings of a valid coin commitment cm, and cm is a output coin commitment of a deposit transaction preceding tx_{spend} on L.
2. No two (tx_{spend}, a) and (tx'_{spend}, a') in $(L, \vec{a})$ contain openings of the same coin commitment.
3. Each (tx_{spend}, a) in $(L, \vec{a})$ contains opening of cm to value v, and $v = v^{new}$.
4. For each (tx_{spend}, a) in $(L, \vec{a})$, if cm is also the output of a deposit transaction, both transaction have the same value v.
5. For each (tx_{spend}, a) in $(L, \vec{a})$, where tx_{spend} is inserted by A, if cm is the output of a previous transaction tx', the public address is not in $ADDR$. Recall that $ADDR$ is the set of address pairs created by A's $CreateAddress$ queries.

We then prove that A cannot violate each case with more than negligible probability.

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

A **violates Condition 2**: If A violates Condition 2, L contains two spend transactions tx_{spend} and tx'_{spend} with the same cm. Since both transactions are valid, they must contain different serial numbers, namely $sn = sn'$. However, if both transactions spend 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 $COMM$.

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

A **violates Condition 4**: If A violates Condition 4, L contains a deposit transaction $tx_{deposit}$ and a spend transaction tx_{spend} s.t. both transactions have the same commitment cm but open cm to different values. This violates the binding property of the commitment scheme $COMM$.

A **violates Condition 5**: If A violates Condition 5, L contains an inserted spend transaction tx_{spend} s.t. tx_{spend} spends 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 B that, by using A as a subroutine, distinguish PRF from a random function.

References

[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.

[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.

[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.