Xand Network Confidentiality White Paper

Transparent Financial Systems

February 25, 2022

Contents

Abstract

Blockchain systems cause problems for businesses that would like to keep their financial information private. This paper describes a system that allows transactions on a blockchain to be confidential while maintaining the public auditability of that blockchain. This system draws from proven cryptography including zero-knowledge proofs, Pedersen commitments, and shared key encryption.

1 Introduction

A Xand Network is a decentralized network that enables Creating, Sending, and Redeeming of Claims on dollars held in trust for the purpose of payment. The Network is owned and maintained by a set of Members, who are the only participants allowed to create, send, and redeem Claims. A Trustee oracle confirms cash movement onto and off of the network during Creation and Redemption. Other entities may be paid in Claims but are only authorized to Redeem. This paper describes a system by which Members can Create, Send and Redeem Claims on a public blockchain while maintaining their privacy and enforcing the Network rules.

2 Executive Summary of the Confidentiality Design

2.1 Motivation

Xand Networks each use their own blockchain as a ledger to record financial transactions. Due to the nature of most blockchains, all participants are privy to every transaction. Most businesses would prefer not to have all their transactions be public knowledge. Some blockchains, such as Bitcoin, allow participants to use a different key for every transaction as an identity protection strategy. However, because only Members can transact on Xand and the Members are all known to each other, their identity can't be completely hidden. This paper describes our private transaction system that allows Members to transact while preserving their private information and enforcing the governance requirements of the Xand Network.

2.2 Relevant Transactions

There are three Xand Network actions that privacy applies to:

- Creation - Creation allows Members to create new Xand Claims by transferring funds into a bank account owned by the Trust
- Send - Sends allow Members to send Xand Claims that they own to another Member
- Redemption - Redemption allows a participant who holds Xand Claims to destroy those Claims and the Trustee will then transfer funds from the Trust into a specified bank account

2.3 Requirements

Every financial transaction on Xand has a signer and a counterparty. All other parties on the network are simply observers. For Creation and Redemption, the counterparty is the Trustee, while for Sends, the counterparty is the Member being paid. For a table summarizing the privacy of each transaction, please see Appendix A.

- For all transactions the signer is hidden from observers but revealed to the counterparty
- For Send the amount is hidden from observers but revealed to the counterparty
- For Creation and Redemption the amount is public knowledge such that any observer can know how many Claims exist
- For Creation and Redemption the bank account information is hidden from observers but revealed to the counterparty

- Only Members are allowed to transact, but their identity is not revealed by proving their Membership
- Members can be added and removed from the network. Removed Members are not allowed to transact
- In a Send the recipient of the Claims is hidden from observers but revealed to the counterparty
- In a Send the sum of inputs always equals the sum of outputs
- All expected transactional invariants can be verified by any observer including the validators. These include:
  - The Claims being consumed are owned by the signer
  - The Claims being consumed have not been spent previously
  - The inputs and outputs in a Send balance
  - The amounts of creation and redemption are correct

2.4 Approach

In order to fulfill the above requirements, our system adopts a number of cryptographic techniques. They will be presented here in brief and explained in depth later in the paper.

2.4.1 Claim Ownership

In order to protect the identity of the owner of Claims, every Claim is addressed to a One Time Key. This One Time Key has a mathematical relationship to the owner's private key such that the owner can always identify Claims sent to them.

2.4.2 Membership Proof

In order for Members to prove their membership without revealing their identity, they use a tokenized version of membership that we call Identity Tags. These are created on every new transaction to form an ever expanding pool. As observers cannot link them to Members it is necessary to include an additional proof that the Identity Tag does not belong to a removed Member.

2.4.3 Claims Amount

Claims amounts are hidden using a homomorphic encryption scheme that allows addition and subtraction. This combined with Bulletproofs to prevent overflow issues allows observers to ensure that the sum of a Send's inputs is non-negative, and always equals the sum of the outputs. When doing Creations and Redemptions, the signer "opens" the Claims by revealing the hidden values – allowing observers to verify that the amounts involved are correct.

2.4.4 Signer Identity

Signer identity is partially protected by the above measures, but more is needed to prevent an observer from linking everything back to the signer's original key. Inputs to every transaction are mixed in with decoy inputs. The true input is verified using a zero-knowledge proof while not revealing which input is true. This is an adaptation of the Monero Ring Signature. As the recipient is expected to know the signer, the signer includes a proof of their identity that only the recipient can decipher.

2.4.5 Bank Data

Bank account and routing numbers are hidden using a deterministic shared key encryption scheme. This allows the counterparty to decrypt the data, as well as the original signer.

3 Payment Activities

There are three activities on the Xand Network covered by the confidentiality system: Creation, Send, and Redemption. This section provides a brief overview of the purpose and mechanism of each transaction type.

3.1 Creation

Creation is the means by which Members create Claims. It involves an initiating and a completing transaction.

1. Initiating Transaction: Create Request - This transaction is submitted by the Member and contains all the details needed for creating Claims. It gives the Trustee the information it needs to correlate dollars deposited into the trust account with new Claims generated on the Network.
2. Completing Transaction:
    - Cash Confirmation - This transaction is submitted by the Trustee if the expected cash movement is observed. Once this transaction is completed the Claims are available to be used.
    - Create Cancellation - This transaction is submitted by the Trustee if the bank data is bad or malformed or by the Network if the expiry time has passed. This prevents the Claims from the Create Request from ever being spent.

3.2 Send

Sending of Claims is done via a single transaction that spends some Claims and creates new Claims for the recipient.

3.3 Redemption

Redemption is the means by which Members redeem Claims for cash. It involves an initiating and a completing transaction.

1. Initiating Transaction: Redemption Request - This transaction is submitted by the Member and contains all the details needed for redeeming Claims. It describes to the Trustee how funds are expected to be sent to the Member.
2. Alternate Initiating Transaction: Exiting Redemption - A specially formed Redemption that is used exclusively by Members who have been voted off the network.
3. Completing Transaction:
    - Redemption Fulfillment - This transaction is submitted by the Trustee after it has followed the payment instruction.
    - Redemption Cancellation - This transaction is submitted by the Trustee if the bank data is bad or malformed. This makes the Claims that were intended to be redeemed usable on the network.

4 Data Model

This section presents an overview of the data model for all transactions. It describes what data is within the payload of each transaction, the purpose of that payload, as well as how transactions interact with the state of the system.

4.1 Components

4.1.1 Permanent Public Key

Every Member of the Network has a single Private Key that they use for signing. This is a scalar value henceforth referred to as p. This private key has a corresponding curve point P where $P = pG$ and is called the Permanent Public Key. P represents the public identity of the Member with private key p and is known to all parties on the network.

4.1.2 One-Time Addresses

In order to obscure the owner of Claims, all Claims are given a One-Time Address. They are named such because each one is unique. A One-Time Address is two points (A, B) such that $pA = B$ where p is the owner's private key. Anyone can generate a new One-Time Address for anyone else using the below method:

- Select recipient P
- Choose $r \leftarrow \mathbb{F}_q$.
- Compute the temporary public key $(A, B) = (rG, rP)$.

Any Member can know whether a TxO (A, B, V) belongs to them with the help of the private key p. This is done using the check $B \stackrel{?}{=} pA$.

4.1.3 Identity Tags

Identity tags share an identical structure to One-Time Addresses. The only difference is their usage. Identity Tags are used as a tokenized version of a Member's authorization to transact on the network. Every transaction requires an Identity Tag from the pool of available tags and produces a new tag as the output to be added to the pool. Since there will be no Identity Tags available at the start of the network, (G, P) is considered a valid Identity Tag for any Member with the Permanent Public Key P.

4.1.4 Claims

Confidential Claims on the network take the form of (A, B, V), where A and B are a One-Time Addresses for the recipient as described above and V is a point representing a value commitment for the dollar value of the Claim. The value commitment can be computed as below:

TxO : Given a value v

- Choose $r \leftarrow \mathbb{F}_q^*$
- Compute the value representation $V = rG + vL$.

Given a value commitment V, the only way to know the corresponding Claims value v is to be given v and r and check $V \stackrel{?}{=} rG + vL$

4.1.5 Opened Claims

Opening a Claim allows any observer to know and verify the value of the Claim. Opening a Claim is as simple as providing v and r. Any observer can verify that the v provided is correct by checking $V \stackrel{?}{=} rG + vL$

4.1.6 Key Images

Key Images are used to prevent Claims from being double spent. Since there are multiple input sets to every transaction – and the real input is obscure – it is never publicly known which Claims have been spent. Each Key Image is linked to a one-time key. For any Claim (A_{kl}, B_{kl}, V_{kl}) t the hash H_{kl} is computed as $H_{kl} = H_g(A_{kl}, B_{kl})$. The key-image I_l is then computed as $I_l = pH_{kl}$. This formulation ensures that every Claim always produces the same Key Image and only the owner of the Claim who knows p can produce the Key Image for a given Claim.

4.2 Transactions

4.2.1 Creation Request

Data:

- *identity_inputs* : Vec⟨IdentityTags⟩, a set of identity tags. One of which is owned by the signer
- *outputs* : Vec⟨OpenedClaims⟩, a set of new Opened Claims
- *identity_output*, a single newly created Identity Tag
- *correlation_id* : [u8; 16], some randomly sampled value
- *encrypted_sender* : $VerifiableEncryptionOfSignerKey$, this is an encryption of the signer key that can be proven to be correct by any observer
- *bank_data* : Vec⟨u8⟩, this contains the encrypted bank account and routing number
- β : OutputProof, proof that the outputs belong to a verified Member identity

Preconditions:

- All the *identity_inputs* are valid known Identity Tags
- *correlation_id* has not been used before
- *outputs* are not in the set of known Claims

Effects:

- The *identity_output* is added to the set of known Identity Tags
- A pending Creation is recorded with the *correlation_id*
- *outputs* are added to the set of known Claims as Pending

4.2.2 Cash Confirmation

Data:

- *correlation_id* : [u8; 16], The correlation ID of a previous Create Request
- β : Signature, a regular Schnorr signature by the Trustee

Preconditions:

- *correlation_id* refers to a currently pending Creation

Effects:

- The pending creation with *correlation_id* is removed
- *outputs* from the pending Creation are changed to be spendable
- The amount of all the *outputs* is added to the total amount created

4.2.3 Creation Cancellation

Data:

- *correlation_id* : [u8; 16], the correlation ID of a previous create request
- β : Signature, a regular Schnorr signature by the Trustee

Preconditions:

- *correlation_id* refers to a currently pending Creation

Effects:

- The pending creation with *correlation_id* is removed
- *outputs* from the pending Creation are removed from the set of known Claims

4.2.4 Send

Data:

- *input_candidates* : Vec⟨TransactionInputSet⟩, a collection of TransactionInputSets, each of which contains a set of Claims. One of these sets is the real set being spent while the rest are decoys
- *identity_inputs* : Vec⟨Identity_Tag⟩, a set of identity tags. One of which is owned by the signer
- *outputs* : Vec⟨Claims⟩, a set of new Claims
- *identity_output* : IdentityTag, a single newly created Identity Tag
- *key_images* : Vec⟨KeyImage⟩, a set of key images ensuring spent Claims cannot be spent again
- Z : EdwardsPoint, used to store the randomness required to hide which of the input sum equals the output sum. Without this randomness, all anonymity will be lost.
- α : Proof, a proof that Z is a Pedersen commitment to 0. It also implies that pZ is also a commitment to 0 where p is any scalar.
- *range_proof* : BulletRangeProof, proof that all the transaction output(s) contain commitments to values represented by a maximum number of bits. This is to stop the sum from rolling over to create money out of nothing
- *encrypted_sender* : $VerifiableEncryptionOfSignerKey$, this is an encryption of the signer key that can be proven to be correct by any observer
- *extra* : Vec⟨u8⟩, a collection of encrypted pieces of data, one for each Claim output. This contains the information needed for the recipient to open the Claim and see the amount

Preconditions:

- All the *identity_inputs* are valid known Identity Tags
- All the Claims in all InputSets are known Claims in a spendable state
- None of the Key Images are known
- None of the *outputs* are in the set of known Claims

Effects:

- The *identity_output* is added to the set of known Identity Tags
- All the *outputs* are added to the set of known Claims as Spendable
- All the Key Images are added to the set of known Key Images

4.2.5 Redemption Request

Data:

- *input_candidates* : Vec⟨TransactionInputSet⟩, a collection of TransactionInputSets, each of which contains a set of Claims. One of these sets is the real set being spent while the rest are decoys
- *identity_inputs* : Vec⟨Identity_Tag⟩, a set of identity tags. One of which is owned by the signer
- *redeem_output* : OpenedClaim, a new Opened Claims which will be destroyed if the redeem is successful
- *change_output* : Claim, a newly created Claim
- *identity_output* : IdentityTag, a single newly created Identity Tag
- *key_images* : Vec⟨KeyImage⟩, the set of key images ensuring spent Claims cannot be spent again
- Z : EdwardsPoint, used to store the randomness required to hide which of the input sum equals the output sum. Without this randomness, all anonymity will be lost
- α : Proof, a proof that Z is a Pedersen commitment to 0. It also implies that pZ is also a commitment to 0 where p is any scalar
- *range_proof* : BulletRangeProof, proof that all the transaction output(s) contain commitments to values represented by a maximum number of bits. This is to stop the sum from rolling over to create money out of nothing
- *encrypted_sender* : $VerifiableEncryptionOfSignerKey$, this is an encryption of the signer key that can be proven to be correct by any observer
- *bank_data* : Vec⟨u8⟩, this contains the encrypted bank account and routing number

Preconditions:

- All the *identity_inputs* are valid known Identity Tags
- *correlation_id* has not been used before
- All the Claims in all InputSets are known Claims in a spendable state
- None of the Key Images are in the set of known Key Images
- None of the *outputs* are in the set of known Claims

Effects:

- The *identity_output* is added to the set of known Identity Tags
- A pending Redemption is recorded with the *correlation_id*
- *redeem_output* is added to the set of known Claims as Pending
- *change_output* is added to the set of known Claims as Spendable
- The amount of the *redeem_outputs* is added to the total amount redeemed

4.2.6 Exiting Redemption

Data:

- *input_candidates* : Vec⟨TransactionInputSet⟩, a collection of TransactionInputSets, each of which contains a set of Claims. One of these sets is the real set being spent while the rest are decoys
- *redeem_output* : OpenedClaim, a new Opened Claims which will be destroyed if the Redeem is successful
- *key_images* : Vec⟨KeyImage⟩, the set of key images ensuring spent Claims cannot be spent again
- *public_key* : $PermanentPublicKey$, the issuing Member's Permanent Public Key
- Z : EdwardsPoint, used to store the randomness required to hide which of the input sum equals the output sum. Without this randomness, all anonymity will be lost
- α : Proof, a proof that Z is a Pedersen commitment to 0. It also implies that pZ is also a commitment to 0 where p is any scalar
- *bank_data* : Vec⟨u8⟩, this contains the encrypted bank account and routing number

Preconditions:

- The public key belongs to a Member in the Exiting state
- *correlation_id* has not been used before
- All the Claims in all InputSets are known Claims in a spendable state
- None of the Key Images are in the set of known Key Images
- None of the *outputs* are in the set of known Claims

Effects:

- A pending Redemption is recorded with the *correlation_id*
- *redeem_output* is added to the set of known Claims as Pending
- The amount of the *redeem_outputs* is added to the total amount redeemed

4.2.7 Redemption Fulfillment

Data:

- *correlation_id* : [u8; 16], the correlation ID of a previous redemption
- β : Signature, a regular Schnorr signature by the Trustee

Preconditions:

- *correlation_id* refers to a currently pending Redemption

Effects:

- The pending redemption with *correlation_id* is removed
- *redeem_output* from the pending Redemption is removed from the set of known Claims

4.2.8 Redemption Cancellation

Data:

- *correlation_id* : [u8; 16], the correlation ID of a pending redemption
- β : Signature, a regular Schnorr signature by the Trustee

Preconditions:

- *correlation_id* refers to a currently pending Redemption

Effects:

- The pending Redemption with *correlation_id* is removed
- *redeem_output* from the pending Redemption is set to be spendable
- The amount of the *redeem_outputs* is subtracted from the total amount redeemed

5 Cryptographic Mechanisms

This section provides a description of the cryptographic functions used to construct and verify each transaction.

5.1 ZkPLMT

5.1.1 Overview

The primary proof mechanism used in the Xand Protocol is ZkPLMT or Zero-knowledge Proof of Linear Member Tuple. ZkPLMT is an adaptation of the CryptoNote protocol [1] used by Monero that was expanded to allow for more general proving capability. The protocol allows the constructor to prove that given a tuple of curve vectors that all the curve vectors in one of the tuples are associated with the same private key (or linear). This can be used for many purposes, such as verifying the signer is authorized to transact, as well as that Claims belong to them. The other tuples function as decoys to hide the signer's identity.

5.1.2 Notation

- We use i as the index of a component of a tuple, j to index a tuple in a set.
- We will use the operator $\forall$ to signify a sequence indexed by a variable. For example, $\forall_j(X_j, Y_j)$ represents $(X_1, Y_1), (X_2, Y_2), ...$ for all values of j. $\forall_{j=1}^{m}(c_j, d_j)$ represents $(c_1, d_1), (c_2, d_2), ..., (c_m, d_m)$. Similarly, for a set $\mathcal{S}$, $\forall_{j\in\mathcal{S}}X_j$ represents a sequence of X_j for all values of $j \in \mathcal{S}$. Notice that we use subscript and superscript with $\forall$ for this custom notation. When used with a standard text position, $\forall$ has the usual meaning.
- $\mathbb{G}$ is an elliptic curve group of prime order q.
- G is a system-wide fixed generator of $\mathbb{G}$.
- L is a system-wide fixed generator of $\mathbb{G}$, such that the discrete logarithm of G with respect to L is not known.
- $\mathbb{F}_q$ is the field integer modulo q.
- $\mathbb{F}_q^*$ is $\mathbb{F}_q \setminus \{0\}$.
- H_g is a hash function with output in the group.

- H_q is a hash function with output in $\mathbb{F}_q^*$.
- H_b is a hash function with output in $\{0,1\}^b$ for a fixed b.
- ϕ is an empty string of bits or an empty message.
- Bold **P** is the prover algorithm for ZkPLMT.
- Bold **V** is the verification algorithm for ZkPLMT.

5.1.3 Terminology

- *Curve-Point Vector* : A curve point vector is a pair of curve points.
- *Slope* : The slope of a curve-point vector is discrete logarithm of the second point with respect to the first point. If (X, Y) is a curve point vector and the slope is w, then $Y = wX$.
- *Linear Tuple* : A linear tuple is a tuple of curve-point vectors with the same slope. If $((X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n))$ is a linear tuple, then $\exists w[\forall i \in \{1, 2, .., n\}[Y_i = wX_i]]$.
- *Non-linear Tuple* : A non-linear tuple is a tuple of curve-point vector that is not a linear tuple.
- *Linear Member Tuple* : Given a list of tuples, it is computationally hard to discover that the list contains a linear tuple. ZkPLMT is a cryptographic method to prove that a given list of tuples contains at least one linear tuple, and that the prover knows the slope of the linear tuple in the list.

5.1.4 Derivation

To understand how the ZkPLMT works, we must start from the Schnorr identification protocol. This is an interactive protocol that proves that the prover has knowledge of a discrete logarithm, as shown in Figure 1. Alice claims that she knows the discrete logarithm of Y with respect to X. She wants to prove it to Bob such that she does not have to tell him the value of the discrete logarithm. To do that, Alice first makes a commitment L to Bob, and Bob provides her a random scalar challenge d, then asks her to find c such that $L = cX + dY$.

Now, since d was provided by Bob, and it could be anything, if Alice is able to do it for some $d = d_1$, she must also be to do it for another value $d = d_2$. Let her reply to the challenges be c_1 and c_2 respectively. So, $L = c_1A + d_1B = c_2A + d_2B$. Therefore:

$$\begin{aligned} & c_1X + d_1Y = c_2X + d_2Y \\ \Rightarrow & (c_1 - c_2)X = (d_2 - d_1)Y \\ \Rightarrow & log_X Y = \frac{c_1 - c_2}{d_2 - d_1} \end{aligned}$$

So, if Alice could find a c for an arbitrary d, she must be able to compute the discrete logarithm as well.

Now, the question is whether the knowledge of the discrete logarithm p will always allow Alice to convince Bob. Suppose Alice already knows some c_1 and d_1, such that $R = c_1X + d_1Y$. If she gets a new challenge d_2, she can easily compute $c_2 = c_1 + p(d_1 - d_2)$. Notice that:

$$\begin{aligned} & c_2X + d_2Y \\ =& (c_1 + p(d_1 - d_2))X + d_2Y \\ =& c_1X + p(d_1 - d_2)X + d_2Y \\ =& c_1X + (d_1 - d_2)Y + d_2Y \quad \textit{[Since } pX = Y\textit{]} \\ =& c_1X + d_1Y - d_2Y + d_2Y \\ =& c_1X + d_1Y \end{aligned}$$

Alice simply initially constructs $R = c_1X + d_1Y$, where she assigns $c_1 = r$ and $d_1 = 0$. So, after she gets the challenge $d_2 = d$, she computes $c = c_2 = c_1 + p(d_1 - d_2) = r + p(0 - d) = r - pd$.

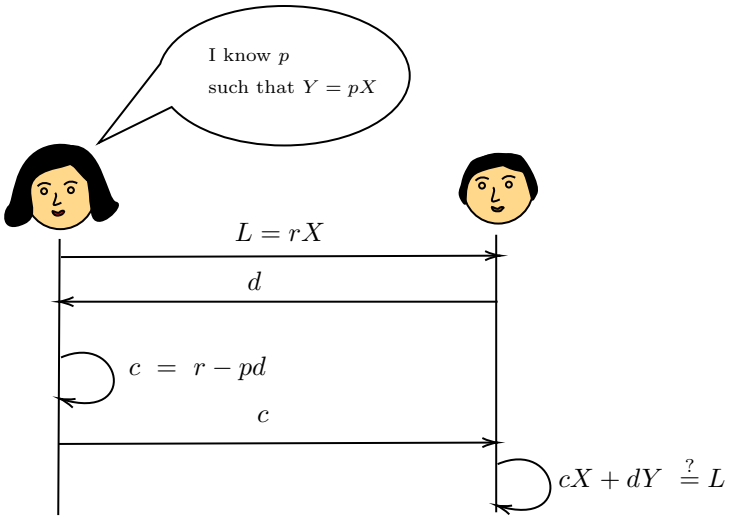

Figure 1: Schnorr identification protocol

Linear Tuple Instead of having only one vector (X, Y), Alice has a tuple of vectors $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$, which we denote as $\forall_i(X_i, Y_i)$. Alice claims that she knows some p such that for all i, $Y_i = pX_i$. In other words, she is claiming that tuple $\forall_i(X_i, Y_i)$ is linear, and she knows the discrete log. She can prove this to Bob with a very similar protocol as before, as shown in Figure 2. The only difference is that instead of one single value of L, Alice needs to commit to a point L_i for each vector (X_i, Y_i).

Just like the Schnorr identification protocol, Bob could send any value for d and Alice needs to be able to find a c for arbitrary d once $R_1, R_2, ..., R_n$ has been committed. Now, if she computes the values c_1 and c_2 for the challenges d_1 and d_2 respectively, for the same reasoning as the previous protocol, we have $log_{X_i} Y_i = \frac{c_1 - c_2}{d_2 - d_1}$ for every i. Now, we can easily see that all the discrete logarithm values have the same formula, and hence are all equal.

Linear member tuple for ZkPLMT: In the final protocol described in Figure 3, instead of a single tuple that is linear, Alice has a list of tuples of the vectors $\forall_j \forall_i(X_{ij}, Y_{ij})$, such that for a particular value of j, say k, the tuple $\forall_i(X_{ik}, Y_{ik})$ is a linear tuple. Notice that Alice is claiming only one of the tuples is linear and that she knows the discrete logarithm for that particular tuple; she is not making any claims about the other tuples in the list.

To design this proof, Alice first commits the curve-points L_{ij} for each i and each j. Bob then gives a random challenge h and asks Alice to provide (c_j, d_j) for each j, so that the sum $\sum_j d_i = s$. Alice needs control over only one of the values of d_j to be able to match the sum. As for the tuples where she does not know the discrete logarithm, she simply chooses (c_j, d_j) randomly and computes $L_{ij} = c_j X_{ij} + d_j Y_{ij}$ directly. For the index $j = k$, for which she does know the common discrete logarithm, she computes $L_{ik} = rX_{ik}$ for each i just like the previous protocol. She then finds the value of d_k that is required for the sum to match, $d_k = h - \sum_{j \neq k} d_j$. She then finds $c_k = r - pd_k$ just like the previous protocol.

For the values of $j \neq k$, the values (c_j, d_j) do not change, so the values of L_{ij} remains the same. For the values L_{ik}, the computation of the c_k is done just like the previous protocol. Hence, the proof works. Notice that Alice does need to be able to find c_k for an arbitrary d_k for some k; hence this does prove to Bob that one of the tuples is linear and that she does know the common discrete logarithm for that tuple.

5.1.4.1 Making the proof non-interactive

Bob's only job is to generate the random challenge. To make the proof non-interactive, Bob is replaced by a hash function. Whatever Alice needs to commit goes as the input to the hash function, and the output of the hash function is the challenge. Since the hash function output looks exactly like random values, this

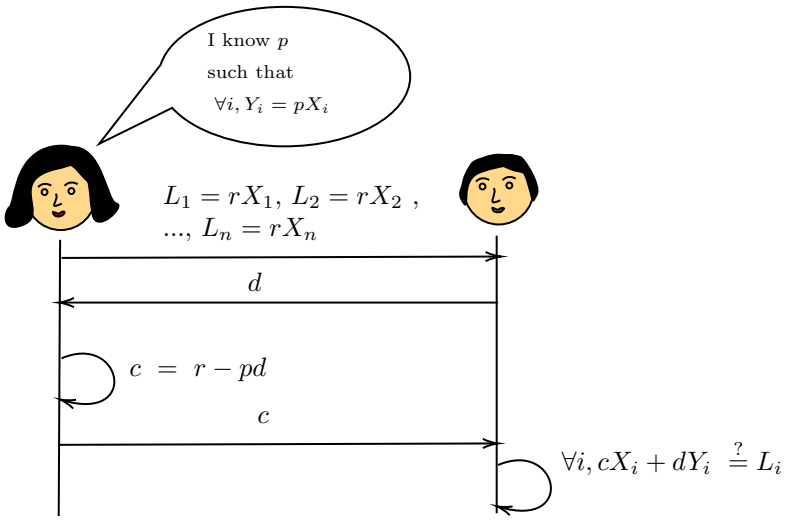

Figure 2: Proof of linearity of a tuple

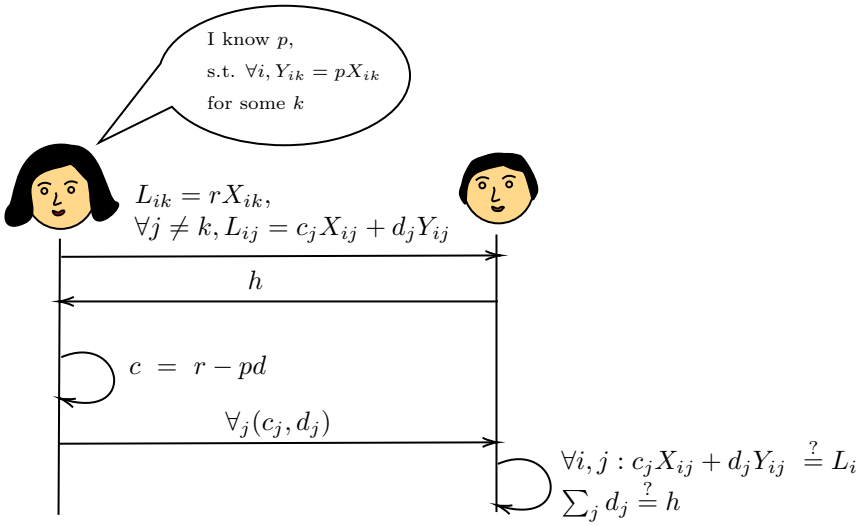

Figure 3: Proof of the presence of a linear member tuple / ZkPLMT

allows us to replace Bob with a predictable function. This, in turn, enables us to store the proof and be verified later by anyone.

5.1.5 Proof Construction and Verification

Prover $\mathbf{P}(M, \mathbf{S}, k, w)$ Where:

- M is the message.
- $\mathbf{S}$ is a set of curve-point tuples containing at least one linear tuple.
- k is the index of the linear tuple in the set of tuples.
- w is the slope the curve-point vectors in the linear tuple.

To calculate the proof:

- Choose $r \leftarrow \mathbb{F}_q^*$
- $\forall j \neq k$, choose $c_j, d_j \leftarrow \mathbb{F}_q^*$
- $\forall i, \forall j \neq k$, set $L_{ij} = c_j X_{ij} + d_j Y_{ij}$
- $\forall i$, Set $L_{ik} = r X_{ik}$
- Compute $h = H_q(M, \mathbf{S}, \forall_i \forall_j L_{ij}))$
- Set $d_k = h - \sum_{j \neq k} d_j$
- Set $c_k = r - d_k w$
- Output $\pi = (\forall_j (c_j, d_j))$

Verifier $\mathbf{V}(\pi, M, \mathbf{S})$ Given the token M and the proof π, the verifier does the following:

- $\forall i, \forall j$, Set $L_{ij} = c_j X_{ij} + d_j Y_{ij}$
- Compute $h = H_q(M, \mathbf{S}, \forall_i \forall_j L_{ij})$
- Check $h \stackrel{?}{=} \sum_j d_j$. If the check works, return $true$, else return $false$

5.2 Deterministic Shared Key Encryption

5.2.1 Overview

Deterministic Shared Key Encryption is a mechanism by which constructors of transactions can share information with the intended recipients. It is used to communicate things that are not proven by the validators, such as the secret information needed by the recipient of a transfer to see the amount of the Claim, as well as bank information needed during Creation and Redemption. The deterministic aspect of the scheme means the sender can always go back and decrypt the transaction themselves, giving them a full view of their transaction history using only their private keys.

5.2.2 Requirements

- Does not reveal the transaction signer's identity.
- Both the signer and the recipient can decode the payload without having to rely on off-chain storage (of a nonce, for example). This makes it easy for the signer to view their transaction history.
- Instances of the same data should not be identifiable.

5.2.3 Encryption

We use shared key encryption – a non-interactive variant of the x25519 Diffie-Hellman ECDH protocol – to fulfill the requirements. Before being able to perform an encrypted transfer, both the sender and the recipient must have generated x25519 keys and registered the public half of their key on the chain.

Once the keys are registered, the sender uses the following protocol to encrypt messages on-chain:

- The sender deterministically generates a temporary key derived from the primary private encryption key by hashing it together with some public, unique plaintext which is also included in the transaction. Specifically:
  - Compute the randomness $f = H_q(M', x)$. Where M' is the content of the transaction up to this point. x is the primary encryption secret key.
  - Compute the sender's temporary public encryption key $F = fG$.
- Compute the Diffie-Hellman shared secret $C = fD$. f is temporary sender private key, and D the receiver's public encryption key.
- Derive a new shared encryption key from the shared secret $K_e = H_b(C); K_m = H_b(K_e||C)$.
- Compute the encrypted message $\omega' = AEnc(K_e||K_m, \omega)$, where ω is the plaintext message.

The publicly transmissible portions of the encryption are the outputs F and ω'.

After this process, both sender and recipient have a shared secret which is specific to the transaction at hand. Both parties are also able to decrypt the transaction without storing any extra information as long as they retain their original x25519 keys. It is recommended that participants periodically rotate these keys, since if obtained, all encrypted data created with that key could be revealed. Old keys should be securely stored to enable decrypting old transactions.

The reason we derive a temporary sender key from the primary encryption key is to enable revealing one transaction without revealing others between the same two parties. If we did not perform the extra hashing/derivation step, revealing the shared secret to an entity would allow it to decrypt the data in all other transactions between the two parties.

5.2.4 Decryption

The decryption process for the receiver of an encrypted message is the following:

- Compute $C = dF$, where d is the receiver private encryption key and F is the temporary sender public key included in the message payload.
- Derive the shared encryption key from the shared secret, $K_e = H_b(C); K_m = H_b(K_e||C)$.
- Decrypt the ciphertext $\omega = ADec(K_e||K_m, \omega')$.

The decryption process for the sender of an encrypted message requires more steps, as the receiver public key is not included in the encrypted message for confidentiality:

For Creations or Redemptions, the D will always be the Trust encryption key. For Sends between members, D is found using the following process:

- Compute the receiver's permanent public key P_t given a TxO for t, $(G, P_t) = z_t^{-1}(A_t, B_t)$.
- Let D be the encryption key registered to P.

With D resolved, the sender can decrypt a message it sent, ω':

- Compute the randomness $f = H_q(M', x)$.
- Compute the Diffie-Hellman shared secret $C = fD$. f is temporary sender private key, and D the receiver's public encryption key.
- Derive the shared encryption key from the shared secret $K_e = H_b(C); K_m = H_b(K_e||C)$.
- Decrypt the ciphertext $\omega = ADec(K_e||K_m, \omega')$.

5.3 Bulletproofs

5.3.1 Overview

Bulletproofs [2] are used to construct range proofs to verify that the claim amount is within an expected range. This is necessary to prevent overflow attacks which would allow users to create claims of arbitrary value. As we simply use Bulletproofs and they are complex, we do not describe their mechanism in this paper.

6 Construction and Verification of Transactions

6.1 Overview

The following is an informal description of the pieces that go into the proof and what their purpose is. Not all transactions contain all components. The Xand network requires proof of a set of facts about a transaction before it can be added to the ledger. Primarily we must prove the following facts about the transaction:

- The submitter of the transactions owns the real inputs they are using
- Those real inputs have not previously spent in another transaction
- The total of the real inputs is equal to the outputs
- The submitter is a Xand Member

In order to do this we construct a linear tuple with the following form

$$\mathbf{Y}_k = (\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \sum_l V_{kl} - \sum_t V_t), (S_k, T_k), (S, T))$$

Each element in this tuple corresponds to one of the four facts about the transaction above and will be explained below. We then provide privacy by hiding this constructed linear tuple among other generated non-linear tuples and providing a proof that one of the tuples is linear called a ring signature. This proof of linearity proves the four facts above.

Transaction Inputs: We start with list of m sets of input commitments, each of length n: $\forall_j \forall_l (A_{jl}, B_{jl}, V_{jl})$ where $j \in \{1, \ldots, m\}$, $l \in \{1, \ldots, n\}$, and one set of output commitments $\forall_t (A_t, B_t, V_t)$ where $t \in \{1, \ldots, o\}$. o, j and l all represent indices. k is particular value of j used to identify the index of the true inputs. We also consider a list of the permanent public keys of the banned Members $\forall_e P_e$, where $e \in \{1...u\}$.

$$\mathsf{Inputs} = \begin{bmatrix} \iota_{1,1} & \cdots & \iota_{1,n} \\ \vdots & \ddots & \\ \iota_{m,1} & & \iota_{m,n} \end{bmatrix} \quad \text{where} \quad \iota_{j,l} = (A_{jl}, B_{jl}, V_{jl})$$

Choose $m-1$ instances $\forall_{j \in \{1,2,..,m\} \setminus \{k\}} \forall_l (A_{jl}, B_{jl}, V_{jl})$ from other available TxOs in the blockchain these are the input TxOs that will be used to mask which inputs the transaction constructor owns.

Transaction Outputs: These transactions have outputs of the form:

$$\mathsf{Outputs} = \{\psi_1, \ldots, \psi_o\} \quad \text{where} \quad \psi_t = (A_t, B_t, V_t)$$

, where t is the index of the output.

Proof that the submitter owns the real inputs $\forall_l (A_{kl}, B_{kl})$: The submitter's private key is a scalar p. We can have multiple public keys for the same private key. The pair of curve points (A, B) is a public key for the private key p if and only if $B = pA$. So, if we have a list of public keys $\forall_{l \in \{1...n\}} (A_{kl}, B_{kl})$, then for every $l \in \{1...n\}$, $B_{kl} = pA_{kl}$. So, the tuple $(\forall_{l \in \{1...n\}} (A_{kl}, B_{kl}))$ is a linear tuple by our definition.

Proof that the real inputs have not been spent before $\forall_l(H_{kl}, I_l)$: Since the validators cannot know which TxOs are being spent, we need some nullifier or key-image to mark them down after they are spent. For every TxO (A_{kl}, B_{kl}, V_{kl}) that is being used, the hash H_{kl} is computed as $H_{kl} = H_g(A_{kl}, B_{kl})$. This is a unique hash identifying the TxO being used. The key-image I_l is then computed as $I_l = pH_{kl}$. As the hashes are fixed for a given TxO, so is the key-image. Since the key-images cannot be used more than once, the input to the hash, in this case (A_{kl}, B_{kl}), can be used as the real input only once. We could have used (A_{kl}, B_{kl}, V_{kl}) instead as the input to the hash, which would allow use of the same public key in another TxO with a different value. This is not necessary as temporary public keys are only meant to be used once. We also need a proof that the key-image is computed correctly. Notice that the both slope of the vector (A_{kl}, B_{kl}) and (H_{kl}, I_l) are the same value p. So, the tuple $(\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l))$ is a linear tuple.

Proof that the total of the real inputs is equal to the outputs $(Z, \sum_l V_{kl} - \sum_t V_t)$: We must also ensure the sum of the input commitments being used is the same as the sum of the output commitments to verify no value was created or destroyed. However, we cannot do it directly since we don't want to disclose which tuple of the inputs we are actually using. Instead, we prove that the difference between the sum of the output commitments minus the sum of the input commitments is a commitment to zero. The commitment to a value v is $rG + vL$ for any r, therefore the commitment to zero is rG for some scalar r. For the input commitments V_{kl} and output commitments V_t, we make sure that $\sum_l V_{kl} - \sum_t V_t = rG$ for some scalar r. We also want to make it part of the linear tuple we are building, so a natural choice for the value of r is p. However, if we take that, rG would always be $pG = P$, the permanent public key of the signer. So, we must add additional randomness. We first choose a random scalar z, and then choose $r = pz$. We then compute $Z = zG$ and compute $\mathbf{Z} = pZ$. This means $\mathbf{Z} = \sum_l V_{kl} - \sum_t V_t$. Now, the slope of $(Z, \mathbf{Z}) = (Z, \sum_l V_{kl} - \sum_t V_t)$ is also p. Hence, the tuple $(\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \sum_l V_{kl} - \sum_t V_t))$ is a linear tuple.

Proof that the submitter is a Xand Member $(S_k, T_k), (S, T)$: We must also prove that the signer is an authorized Member. We use Membership keys of the form (S, T) for this. The Membership keys are structurally the same as the temporary public keys, i.e. if (S, T) is a Membership key for the private key p, then $T = pS$. The Membership keys for a particular Member forms a chain. Every transaction has a true input Membership key which is (S_k, T_k) and an output Membership key (S, T). Since they are the Membership keys for the same Member, the slope of both of them is p. Hence, the tuple $\mathbf{Y}_k = (\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \sum_l V_{kl} - \sum_t V_t), (S_k, T_k), (S, T))$ is a linear tuple.

Protecting the Submitter's privacy: To obfuscate our source transaction, we make other tuples with the same structure, which are not necessarily linear. The values $\forall_l I_l, (S, T), \forall_t(A_t, B_t, V_t)$ are the outputs of the transaction, i.e. these values are created in the transaction. We keep these the same in all of the fake inputs. Others we fill in with different values from the blockchain. For example, the fake input TxO (A_{jl}, B_{jl}, V_{jl}) are picked from existing TxOs in the blockchain. $\mathbf{Y}_j = (\forall_l(A_{jl}, B_{jl}), \forall_l(H_{jl}, I_l), (Z, \sum_l V_{jl} - \sum_t V_t), (S_j, T_j), (S, T))$. We hide the linear tuple within the non-linear tuples with the same structure that we have created.

Constructing the Proof: Finally, we simply have to prove that in the list of tuples $\forall_j \mathbf{Y}_j$, there is some k, for which $\mathbf{Y}_k$ is a linear tuple, and all of the above properties will apply to the tuple at that index k. This proof of the existence of the linear tuple would be called π.

However, we have to add one more thing. Notice that π only proves $\sum_l V_{kl} - \sum_t V_t = pZ$. pZ is a commitment to zero only when there is some z known to the prover such that $Z = zG$. The knowledge of the value of z is proved with a second ZkPLMT called α.

Banned List: We need to prevent previous Members who have been banned from participating on the Network, therefore need to check that a particular signer is not on the banned list. α also proves that for every e, $Q_e = zP_e$. Since π already proves that $\sum_l V_{kl} - \sum_t V_t = pZ$, we only need to check that for every e and every j, $\sum_l V_{jl} - \sum_t V_t \neq Q_e$ to make sure that for every e, $Q_e \neq pZ$. Now, since $Q_e = zP_e$, this check

also proves that $P_e \neq pG$. pG is the permanent public key of the signer, so this proves that the signer is not in the banned list.

Disclosure of the sender's identity: Let an output TxO be (V_t, A_t, B_t). Let the public key of the signer be P. The public key is simply encrypted using the El-Gamal encryption with the public key (A_t, B_t). To do that, first a random x is chosen from $\mathbb{F}_q$, and then the encryption is computed as $E = xB_t + P$. Both $X = xA_t$ and E are stored in the transaction. The receiver knows the private key of the receiver (say q_t); so, the receiver can compute the value of P as $E - qX$. Notice that $B_t = qA_t$, so $qX = qxA_t = xB_t$. This means $E - qX = xB_t + P - xB_t = P$.

Now, the trick is to prove that E indeed is equal to $xB_t + P$ to the validators who don't know q and must not be able to decrypt E. The trick is to 'promote' everything by some secret random y and then check the sum. We compute $Y = yG, E' = yE, P' = yP$. We now want to show that $E' = xyB_t + P'$, which would then prove that $E = xB_t + P$. To do this, we must also compute the value of xyB_t. We first compute $B'_t = yB_t$ and then $Q' = xB'_t = xyB_t$. To prove the promotion is correct, we compute a ZkPLMT (π_1) for the linear tuple $(G, Y), (E, E'), (B'_t, yB_t)$. We also compute a ZkPLMT (π_2) to prove the linearity of the tuple $(A_t, X), (B'_t, Q')$. And finally, we add the pair (Y, P') to the main ZkPLMT to prove that $P' = pY = yP$. π_1 proves that $B'_t = yB_t$ and $E' = yE$. π_2 proves $Q' = xB'_t = xyB_t$. The main signature proves that $P' = yP$. Now, the verifier can simply check whether $E' = Q' + P'$ to know whether the encryption is correctly done.

Unsigned Transaction: We assume that all relevant unsigned transaction data is $M \in \{0, 255\}^*$.

6.2 Input Selection

We propose that transaction constructors use the following methods for selecting real and decoy inputs.

6.2.1 Input TxO Selection

For minimizing dust, select TxOs to make up nearly double the amount that is actually being paid so as to make the real payment and the change amount equal. This minimizes the dust creation as most UTxOs would end up having value almost the average transaction amount.

6.2.2 Decoy Selection

We propose an N-age-bound random selection -

- We first fix a system-wide integer N that is the bound in the transaction age
- Choose $k \leftarrow \{1...N\}$
- Select $k - 1$ transactions created just before the true TxO in the order of creation in the blockchain. Let the set selected be $\mathcal{P}$
- Select $N - k$ transactions after the true TxO in the order of creation in the blockchain. If $N - k$ transactions are not available after the true TxO, repeat from the beginning. Let the set selected be $\mathcal{S}$. Let the true TxO be called $TTxO$
- Select $m - 1$ TxOs from the set $(\mathcal{P} \cup \mathcal{S}) \setminus \{TTxO\}$ with a uniform distribution

6.3 Activity Proofs

6.3.1 Confidential Transaction

The signature of a transaction is similar to the CryptoNote signature, but we add a lot more features. Our signature is closer to the MLSAG signature used in Monero in terms of features, except we also include proof of Membership of the signer. Our signature is, in general, more space efficient than MLSAG.

The signature creator must choose input TxOs that they control, which we represent as: (A_l, B_l, V_l) where A_l and B_l are the two curve points representing the RecipientOneTimeKey defined in our description of a TxO above. The subscript l is an index into the list of all input TxOs.

6.3.2 Create Transaction Proof

Inputs

- True Membership tag (S_1, T_1)
- Private Key p
- Encryption private key x
- Output Values $\forall_t v_t$
- List of m masking input Membership-tags $\forall_{j\in\{2...m\}}(S_j, T_j)$
- List of u banned Member public keys $\forall_e P_e$
- Account number a
- Routing number b
- Decryption key of the trust D
- Banking Correlation ID corrId (16 bytes)
- Receiver or trustee's public key (A, B), where $B = qA$, q is the private key of the receiver or trustee

Construction

- Generate random $z \leftarrow \mathbb{F}_q^*$
- For all e, compute $Q_e = zP_e$
- Compute $Z = zG$
- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Compute $\alpha = \mathbf{P}(\phi, \mathbf{X}', 1, z)$
- Compute $\mathbf{Z} = pZ$
- Shuffle Membership tags $\forall_j(S_j, T_j)$ randomly, so that (S_k, T_k) is a Membership tag of the signer (S, T) for some $k \in \{1...m\}$. This operation will reset the index j
- Construct output Membership tag (S, T) such that $T = pS$
- For all t, construct $r_t, z_t \leftarrow \mathbb{F}_q$
- For all t, construct output TxO $(A_t, B_t, V_t) = (z_t G, z_t pG, r_t G + v_t L)$
- Compute ω', F using 5.2, where $\omega = (a, b)$ and D is the Trust encryption key
- Sample $x, y \leftarrow \mathbb{F}_q^*$
- Compute $X = xG, Y = yG, E = xQ + P, P' = yP, E' = yE, B' = yQ, Q' = xyQ$. Note that $E = qX + P$, where q is the private key of the trustee and $Q = qG$ is the public key of the trustee
- Compute proof $\pi_1 = \mathbf{P}(\emptyset, \{(G, Y), (E, E'), (Q, B')\}, 1, y)$
- Compute proof $\pi_2 = \mathbf{P}(\emptyset, \{(G, X), (B', Q')\}, 1, x)$
- For the output TxOs $\forall_t(A_t, B_t, V_t)$, construct linear tuple $\mathbf{Y}_k = (\forall_t(A_t, B_t), (Z, \mathbf{Z}), (S_k, T_k), (S, T), (Y, P'))$
- Construct nonlinear tuples $\mathbf{Y}_j = (\forall_t(A_t, B_t), (Z, \mathbf{Z}), (S_j, T_j), (S, T), (Y, P'))$ for all $j \neq k$
- Compute $\beta = \mathbf{P}(M, \forall_j \mathbf{Y}_j, k, p)$, where M is the body of the transaction

- Compute $\bar{r} = \sum_{t=1}^{u} r_t$ and $\bar{v} = \sum_{t=1}^{u} v_t$
- Output

$$(\forall_t(A_t, B_t, V_t), \bar{r}, \bar{v}, (Z, \mathbf{Z}), \forall_j(S_j, T_j),$$
$$(S, T), (Y, P'), Y, (E, E'), (B, B'), X, (Q, Q'), \forall_e Q_e,$$
$$\alpha, \beta, \pi_1, \pi_2, \omega', F, \mathsf{corrId})$$

Verification

- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Verify $\mathbf{V}(\phi, \mathbf{X}', \alpha)$. If the verification fails, return $false$
- $\forall_e$, check that $\mathbf{Z} \neq Q_e$. If the check fails, return $false$
- Verify that the values $\forall_j(S_j, T_j)$ are present as Membership-tag outputs of some transaction
- Construct nonlinear tuples $\mathbf{Y}_j = (\forall_t(A_t, B_t), (S_j, T_j), (S, T))$ for all j
- Verify $\mathbf{V}(M, \forall_j \mathbf{Y}_j, \beta)$. If the verification succeeds, return $true$ or return $false$
- Verify $\mathbf{V}(\emptyset, \{(G, Y), (E, E'), (Q, B')\}, \pi_1)$. If the verification succeeds, return true or return false
- Verify $\mathbf{V}(\emptyset, \{(G, X), (B', Q')\}, \pi_2)$. If the verification succeeds, return true or return false
- Verify that the corrId has never been used before for any other create or redeem request
- Verify that $E' = Q' + P'$. If the verification fails, return false
- Verifier can reveal the sender's identity by computing $P = E - qX$

Explanation

- α proves that if $Z = zG$, then for each e, $Q_e = zP_e$.
- β only proves that for some k, $(\forall_t(A_t, B_t), (Z, \mathbf{Z}), (S_k, T_k), (S, T))$ is a linear tuple. This means that for all t, (A_t, B_t) has the same slope as (S_k, T_k). This means that (A_t, B_t) belongs to a Member. It also means that (S, T) has the same slope and hence can work as a Membership tag in future transactions. It also means, that $\mathbf{Z} = pZ$ which is then checked to not match with any Q_e.
- π_2 proves that $Q' = xB'$ given $X = xG$.
- π_1 proves that $E' = yE$ and $B' = yQ$ given $Y = yG$. Combining π_1 and π_2, we prove that $E' = xB' + P' = Q' + P'$.

Also, in the main proof β we prove that $T = pS$ given $yP = pY$. Thus combining π_1, π_2, β, we have proved that E is an encryption of P such that $P = pG$ and $T = pS$. Thus, the verifier can reveal the real identity of sender by computing $P = E - qX$ using his own private key q.

6.3.2.1 Construction of Input for Send UTxOTX

- Choose $z \leftarrow \mathbb{F}_q^*$ (As a randomness to hide the link between the input commitments and the output commitments)
- Generate random curve point S (As a base point for the output KYC)
- Save index $k \in \{1..m\}$ for the linear tuple. k is the index of the tuple that the transaction constructor owns / has the right to spend
- Compute $Z = zG$

- Let the list of banned Members be $(P_1, P_2, ..., P_u)$
- $\forall e \in \{1...u\}$, compute $Q_e = zP_e$
- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Compute $\alpha = \mathbf{P}(\phi, \mathbf{X}', 1, z)$
- Choose output commitments $\forall_t V_t$ such that $\sum_l V_{kl} = \mathbf{Z} + \sum_t V_t$, where $\mathbf{Z} = pZ$
- $\forall j, \forall l$, compute $H_{jl} = H_g(A_{jl}, B_{jl})$
- $\forall l$, compute key-image $I_l = pH_{lk}$ and declare I_l
- Compute $T = pS$. The KYC output is (S, T)
- Sample $x, y \leftarrow \mathbb{F}_q^*$
- For each t, compute $X_t = xA_t, Y = yG, E_t = xB_t + P, P' = yP, E'_t = yE_t, B'_t = yB_t, Q'_t = xyB_t$. Note that $E_t = q_t X + P$, where q_t is the private key of the receiver, i.e. $B_t = q_t A_t$
- Choose $(m-1)$ KYC outputs from all the past transactions $\forall_{j \in \{1...m\} \setminus \{k\}}(S_j, T_j)$
- Create the linear tuple
  $\mathbf{Y}_k = (\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \mathbf{Z}), (S_k, T_k), (S, T), (Y, P'))$
- $\forall j \neq k$, create the non-linear tuples

$$\begin{aligned}\mathbf{Y}_j = (&\forall_l(A_{jl}, B_{jl}),\\ &\forall_l(H_{jl}, I_l),\\ &(Z, \sum_l V_{jl} - \sum_t V_t),\\ &(S_j, T_j), (S, T), (Y, P'))\end{aligned}$$

- For each t, compute $z_t = H_q(M', p, t)$, where M' is the body of the transaction without the output TxOs and the proof
- For each t, Compute $(A_t, B_t) = (z_t G, z_t P_t)$ where P_t is the permanent public of the t^{th} receiver
- For each t, compute ω'_t using 5.2, where $\omega'_t = (z_t, v_t)$ and D is the encryption key owned by P_t
- Create proof $\pi = \mathbf{P}(M, \forall_j \mathbf{Y}_j, k, p)$ where M is the body of the transaction without the proof
- Generate proof $\pi_1 = \mathbf{P}(\emptyset, \{(G, Y), \forall_t(E_t, E'_t), \forall_t(B_t, B'_t)\}, 1, y)$
- Generate proof $\pi_2 = \mathbf{P}(\emptyset, \forall_t\{(A_t, X_t), \forall_t(B'_t, Q'_t)\}, 1, x)$
- Output

$$\begin{aligned}&(\forall_j \forall_l(A_{jl}, B_{jl}, V_{jl}), \forall_j(S_j, T_j), (S, T),\\ &\forall_t(A_t, B_t, V_t, \omega'_t), Z, (Y, P'), (G, Y), \forall_t(E_t, E'_t), \forall_t(B_t, B'_t),\\ &\forall_t(A_t, X_t), \forall_t(B'_t, Q'_t), \pi_1, \pi_2, \forall_l I_l, \forall_e Q_e, \alpha, \pi, F)\end{aligned}$$

Verification

- $\forall j$ and $\forall l$, check that (A_{jl}, B_{jl}, V_{jl}) exists in the blockchain as outputs of the past transactions. If not, return $false$
- $\forall j$, check that (S_j, T_j) is present in the past transactions in the blockchain. If not, return $false$
- $\forall l$, check that I_l is not present in the list of key-images for any past transaction. If present, return $false$
- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Verify $\mathbf{V}(\phi, \mathbf{X}', \alpha)$. If the verification fails, return $false$
- $\forall j, \forall l$, compute $H_{jl} = H_g(A_{jl}, B_{jl})$
- $\forall j$, create the tuples

$$\begin{aligned}\mathbf{Y}_j = (&\forall_l(A_{jl}, B_{jl}),\\ &\forall_l(H_{jl}, I_l),\\ &(Z, \sum_l V_{jl} - \sum_t V_t),\\ &(S_j, T_j), (S, T), (Y, P'))\end{aligned}$$

- Verify that $\forall j$ and $\forall e$, $Q_e \neq \sum_l V_{jl} - \sum_t V_t$. If any of the inequality is violated, return false
- Verify $\mathbf{V}(M, \forall_j \mathbf{Y}_j, \pi)$. If the verification succeeds, return $true$, else return false
- Verify $\mathbf{V}(\emptyset, \{(G, Y), \forall_t(E_t, E'_t), \forall_t(B_t, B'_t)\}, \pi_1)$. If the verification fails, return false. Otherwise, return true
- Verify $\mathbf{V}(\emptyset, \{\forall_t(A_t, X_t), \forall_t(B'_t, Q'_t)\}, \pi_2)$. If the verification fails, return false. Otherwise, return true
- $\forall t$, verify that $E'_t = Q'_t + P'_t$. If the verification fails, return false. Otherwise, return true
- Verifier can reveal the sender's identity by computing $P = E - q_t X$

6.3.2.2 Construction of Input for RedeemUTxOTX

- Set the Banking Correlation ID: corrId (16 bytes)
- Choose $z \leftarrow \mathbb{F}_q^*$ (As a randomness to hide the link between the input commitments and the output commitments)
- Generate random curve point S (As a base point for the output KYC)
- Save index $k \in \{1..m\}$ for the linear tuple. k is the index of the tuple that the transaction constructor owns / has the right to spend
- Compute $Z = zG$
- Let the list of banned Members be $(P_1, P_2, ..., P_u)$
- $\forall e \in \{1...u\}$, compute $Q_e = zP_e$
- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Compute $\alpha = \mathbf{P}(\phi, \mathbf{X}', 1, z)$
- Choose output commitments V_c for change and V_r for redemption amount, such that $\sum_l V_{kl} = \mathbf{Z} + V_c + V_r$, where $\mathbf{Z} = pZ$

- $\forall j, \forall l$, compute $H_{jl} = H_g(A_{jl}, B_{jl})$
- $\forall l$, compute key-image $I_l = pH_{lk}$ and declare I_l
- Compute $T = pS$. The KYC output is (S, T)
- Sample $x, y \leftarrow \mathbb{F}_q^*$
- Choose $(m-1)$ KYC outputs from all the past transactions $\forall_{j \in \{1 \ldots m\} \setminus \{k\}}(S_j, T_j)$,
- Create the linear tuple
  $\mathbf{Y}_k = (\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \mathbf{Z}), (S_k, T_k), (S, T), (Y, P'))$
- For each $(redemption(r), change(c))$, choose random r: $r_r, r_c \leftarrow \mathbb{F}_q^*$ (As a randomness to differentiate output public keys)
- Compute $(A_r, B_r) = (r_r G, r_r P')$ where P' is the permanent public key of the redeemer
- Compute $(A_c, B_c) = (r_c G, r_c P')$ where P' is the permanent public key of the redeemer
- $\forall j \neq k$, create the non-linear tuples

$$\begin{aligned}\mathbf{Y}_j = (&\forall_l(A_{jl}, B_{jl}),\\ &\forall_l(H_{jl}, I_l),\\ &(Z, \sum_l V_{jl} - V_r - V_c),\\ &(S_j, T_j), (S, T), (Y, P'), (A_r, B_r), (A_c, B_c))\end{aligned}$$

- Compute ω', F using 5.2, where $\omega = (a, b)$ and D is the Trust encryption key
- Create proof $\pi = \mathbf{P}(M, \forall_j \mathbf{Y}_j, k, p)$ where M is the body of the transaction without the proof
- Compute $X = xG, Y = yG, E = xQ + P, P' = yP, E' = yE, B' = yQ, Q' = xyQ$. Note that $E = qX + P$, where q is the private key of the trustee and $Q = qG$ is the public key of the trustee
- Compute proof $\pi_1 = \mathbf{P}(\emptyset, \{(G, Y), (E, E'), (Q, B')\}, 1, y)$
- Compute proof $\pi_2 = \mathbf{P}(\emptyset, \{(G, X), (B', Q')\}, 1, x)$
- Output

$$\begin{aligned}&(\forall_j \forall_l(A_{jl}, B_{jl}, V_{jl}), \forall_j(S_j, T_j), (A_c, B_c, V_c), (A_r, B_r, V_r),\\ &(S, T), (Y, P'), Z, Y, (E, E'), (Q, B'), X, (B', Q'), \pi_1, \pi_2, \forall_l I_l,\\ &\forall_e Q_e, \alpha, \pi, \pi_1, \pi_2, \omega', F, \mathsf{corrId})\end{aligned}$$

Verification

- $\forall j$ and $\forall l$, check that (A_{jl}, B_{jl}, V_{jl}) exists in the blockchain as outputs of the past transactions. If not, return $false$
- $\forall j$, check that (S_j, T_j) is present in the past transactions in the blockchain. If not, return $false$
- $\forall l$, check that I_l is not present in the list of key-images for any past transaction. If present, return $false$
- Define $\mathbf{X}' = ((G, Z), \forall_e(P_e, Q_e))$
- Verify $\mathbf{V}(\phi, \mathbf{X}', \alpha)$. If the verification fails, return $false$
- $\forall j, \forall l$, compute $H_{jl} = H_g(A_{jl}, B_{jl})$

- $\forall j$, create the tuples

$$\begin{aligned}\mathbf{Y}_j = (&\forall_l(A_{jl}, B_{jl}),\\ &\forall_l(H_{jl}, I_l),\\ &(Z, \sum_l V_{jl} - V_r - V_c),\\ &(S_j, T_j), (S, T), (Y, P'), (A_r, B_r), (A_c, B_c))\end{aligned}$$

- Verify that $\forall j$ and $\forall e$, $Q_e \neq \sum_l V_{jl} - V_c - V_r$. If any of the inequality is violated, return false
- Verify $\mathbf{V}(M, \forall_j \mathbf{Y}_j, \pi)$. If the verification succeeds, return *true*, else return false
- Verify that the corrId has never been used before for any other Create or Redeem request
- Verify $\mathbf{V}(\emptyset, \{(G, Y), (E, E'), (Q, B')\}, \pi_1)$. If the verification fails, return false
- Verify $\mathbf{V}(\emptyset, \{(G, X), (B', Q')\}, \pi_2)$. If the verification fails, return false
- Verify that $E' = Q' + P'$. If the verification fails, return false
- Verifier can reveal the sender's identity by computing $P = E - qX$

Explanation α simply works as a proof of knowledge of the discrete log z of Z with respect to G. The proof π proves that one of the tuples in $\forall_j \mathbf{Y}_j$ is linear. For such a tuple $\mathbf{Y}_k = (\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l), (Z, \mathbf{Z}), (S_k, T_k), (S, T), (Y, P'))$, the following can be concluded

- All public keys $\forall_l(A_{kl}, B_{kl})$ correspond to the same private key since the tuple is linear due to the soundness theorem.
- The signer knows the private key for the public keys $\forall_l(A_{kl}, B_{kl})$ by the proof of knowledge theorem.
- Since $\forall_l(A_{kl}, B_{kl}), \forall_l(H_{kl}, I_l)$ are linear; we know that the key-images $\forall_l I_l$ are correctly computed. Since for every l, $I_l = pH_{kl}$ and H_{kl} values are fixed and publicly known for (A_{kl}, B_{kl}), this makes sure that if any of the source UTXOs is ever used again, the signer has to compute the same value for the corresponding key-image. This prevents double-spending the same way as CryptoNote.
- $(Z, \mathbf{Z})$ is linear with the public keys, meaning $\mathbf{Z} = pZ = pzG$, which makes the sum of the transaction amounts in the input is equal to the sum of the transaction amounts in the output since the discrete log of the difference in the commitments with respect to G is clearly known to the signer.
- α also proves that $\forall e, Q_e = zP_e$. Since $\forall j$ and $\forall e$, $Q_e \neq \sum_l V_{jl} - \sum_t V_t = pZ = zpG \Rightarrow pG \neq P_e$. This proves that the signer's public key is not in the banned list.
- π_2 proves that $Q' = xB'$ given $X = xG$.
- π_1 proves that $E' = yE$ and $B' = yQ$ given $Y = yG$. Combining π_1 and π_2, we prove that $E' = xB' + P' = Q' + P'$.

Also, since in the main proof β we prove that $T = pS$ given $yP = pY$. Thus combining π_1, π_2, β, we have proved that E is an encryption of P such that $P = pG$ and $T = pS$. Thus, the verifier can reveal the real identity of sender by computing $P = E - qX$ using his own private key q.

6.4 Proving Information to Third Parties

For various reasons a Member of the network might want to be able to prove the amount of Claims that they own on the network, as well as the transactions that they were involved or not involved in. The below proofs allow them to prove facts to any third party that can also see the ledger of transactions.

6.4.1 Proof of ownership and non-ownership of a TxO

Given a TxO (A, B, V), the following procedure proves the ownership or non-ownership for a key-pair (p, P).

Proof

- Compute $B' = pA$.
- Construct the linear tuple $\mathbf{X} = ((G, P), (A, B'))$.
- Compute ZkPLMT $\pi = \mathbf{P}(\phi, (\mathbf{X}), 1, p)$.
- Return (π, B').

Verification

- Construct tuple $\mathbf{X} = ((G, P), (A, B'))$.
- Verify $\mathbf{V}(\phi, (\mathbf{X}), \pi)$.
- If verification fails, reject the proof.
- If verification succeeds, $B' = B$ implies the TxO is owned by the owner of the public key P, and $B \neq B'$ implies the TxO is not owned by the owner of the public key P.

6.4.2 Proof of unspent-ness

The owner of a TxO can prove that the TxO has not been spent if that is the case. Given a TxO (A, B, V) and a key-pair (p, P), do the following –

Proof

- Compute $H = H_g(A, B)$.
- Compute $I = pH$.
- Compute the linear tuple $X = ((A, B), (H, I))$.
- Compute $\pi = \mathbf{P}(\phi, X, 1, p)$.
- Output (I, π).

Verification

- Compute $H = H_g(A, B)$.
- Compute the linear tuple $X = ((A, B), (H, I))$.
- Compute verification $\mathbf{V}(\phi, X, \pi)$.
- If the verification does not succeed, reject the proof.
- Check if I is already in the list of spend key-images. If so, reject the proof; otherwise accept the proof.

6.4.3 Proof of creation or non-creation of transaction:

For this purpose, given a transaction, we will only consider the output identity tag (S, T). The owner of the key-pair (p, P) can prove or disprove the creation of the transactions by themselves without disclosing which TxOs were actually spent in the following manner:

Proof

- Compute $T' = pS$.
- Construct the linear tuple $\mathbf{X} = ((G, P), (S, T'))$.
- Compute ZkPLMT $\pi = \mathbf{P}(\phi, (\mathbf{X}), 1, p)$.
- Return (π, T').

Verification

- Construct tuple $\mathbf{X} = ((G, P), (S, T'))$.
- Verify $\mathbf{V}(\phi, (\mathbf{X}), \pi)$.
- If verification fails, reject the proof.
- If verification succeeds, $T' = T$ implies the transaction was created by the owner of the public key P, and $T \neq T'$ implies the transaction was not created by the owner of the public key P.

6.4.4 Proof of real source of the transaction:

In case the owner of the key-pair (p, P) has originated a transaction, they can prove this fact along with the true source TxOs in the following manner. In this case, only the source identity tag (S_k, T_k) corresponding to the true TxOs at index k is chosen. (S, T) is the output identity tag.

Proof

- Construct the linear tuple $\mathbf{X} = ((G, P), (S_k, T_k), (S, T))$.
- Compute ZkPLMT $\pi = \mathbf{P}(\phi, (\mathbf{X}), 1, p)$.
- Return (π, k).

Verification

- Construct tuple $\mathbf{X} = ((G, P), (S_k, T_k), (S, T))$.
- Verify $\mathbf{V}(\phi, (\mathbf{X}), \pi)$.
- If verification fails, reject the proof.
- If verification succeeds, it proves that the owner of the key-pair (p, P) generated the transaction and also k is the true index of the source TxOs.

Appendix A Data Confidentiality

Below a chart is provided to make clear the level of confidentiality for each piece of information about the transaction. For clarity on what this information is or what the transactions do, refer to section 4 that defines the data model.

Table 1: Confidentiality Level for Data Flow

Transaction	Data	Initiating Participant	Recipient	Observers
All Transactions	Signature/Proof	Y*	Y*	Y*
	Transaction ID	Y	Y	Y
	Transaction Type	Y	Y	Y
	Constructor Public Key	Y	Y	N
Create Request	Amount	Y	Y	Y
	Account #	Y	Y	N
	Routing #	Y	Y	N
	Correlation ID	Y	Y	Y
Cash Confirmation	Correlation ID	Y	NR	Y
Create Cancellation	Correlation ID	Y	NR	Y
Send	Amount	Y	Y	N
	Receiver Public Key	Y	Y	N
Redemption Request	Amount	Y	Y	Y
	Account #	Y	Y	N
	Routing #	Y	Y	N
	Correlation ID	Y	Y	Y
Redemption Confirm	Correlation ID	Y	NR	Y
Redemption Cancellation	Correlation ID	Y	NR	Y

[1] Y: means the data is transparent.
[2] N: means the data is obscured.
[3] NR: means there is no recipient for this transaction
[4] *: means the signature is scrutable in that anyone can verify it, but it actually contains a lot of these other listed data components, so the internals are varyingly scrutable.

Appendix B Formal Proofs

B.1 Literature Review

The Xand Network provides a business-to-business on-demand settlement solution that maintains a digital ledger using a blockchain. This distributed ledger tracks Xand Claims on funds held in trust. The Xand Network is permissioned and has important governance requirements about what entities can act as Members. Additionally, Members require some level of privacy on their financial transactions. In this section we give a literature review for the techniques used in the confidentiality protocol for the Xand Network.

Our Xand confidentiality protocol is designed based on techniques pioneered by Monero, which is a standard one-dimensional distributed cryptocurrency blockchain [3]. Monero was initially known as CryptoNote [1] published under the pseudonym of Nicolas van Saberhagen. It offered receiver anonymity through the use of one-time addresses, and sender ambiguity by means of ring signatures. Since its appearance, Monero has further strengthened its privacy aspects by implementing amount hiding, as described by Greg Maxwell in [4], integrating ring signatures based on Shen Noether's recommendations in [5], and made more efficient with Bulletproofs [2]. Afterwards, there are several version updates called *Zero to Monero* that keep improving cryptographic primitives used [6, 7].

The Xand Confidentiality Protocol has leveraged many of the advantages of Monero while adding to the protocol with our governance requirements. Transactions in our protocol are based on elliptic curve cryptography using curve Ed25519 [8] and transaction inputs are signed with Schnorr-style ring signature [5], and output amounts are communicated to recipients via Elliptic Curve Diffie-Hellman(ECDH) [9] and concealed with Pedersen commitments [10] and proven in a legitimate range with Bulletproofs [2]. We use a special curve belonging to the category of so-called *Twisted Edwards* curves [8]. In particular, it uses Curve25519 which was released in [8]. The advantage this special curve offers is that basic cryptographic primitives require fewer arithmetic operations, resulting in faster cryptographic algorithms. More details are described in Bernstein et al. [11].

A zero-knowledge proof system is a key part of our confidentiality protocol. The notion of a zero-knowledge proof system was introduced by Goldwasser, Micali and Rackoff [12] which is central in cryptography. Since its introduction, the original definition has been then reformulated under several variations, trying to capture additional security and efficiency guarantees. In recent years, the concept of a zero knowledge proof has been extremely influential and useful for many other notions and applications in blockchain-based financial systems. Some notable examples are ZCash [13], Hyperledger Fabric [14], Ethereum [15] and Cardano's Ouroboros [16].

We have designed our own zero-knowledge proof protocol called *Zero-Knowledge Proof for Linear Member Tuple (ZKPLMT)*, which is a more generalized version to support proving a member's identity in a set of members. The protocol is based on the Schnorr Identification Protocol [17, 18]. Identification, also known as entity authentication, is a process by which a verifier gains assurance that the identity of a prover is as claimed, i.e., there is no impersonation. An identification scheme enables a prover holding a secret key to identify itself to a verifier holding the corresponding public key. The origin Schnorr identification protocol is interactive and not publicly verifiable, we use Fiat-Shamir transform [19]. That is, we assume the hash function as a random oracle to generate challenges in the interactive proof protocol. Our ZKPLMT protocol follows the paradigm of Schnorr Identification Protocol to construct non-interactive zero-knowledge proofs of knowledge. These are protocols which allow a prover to demonstrate knowledge of a secret while revealing no information whatsoever other than the one bit of information regarding the possession of the secret [12, 20]. In our use-cases, we particularly focus on its application that it allows someone to prove they know the private key k of a given public key K without revealing any information about it [4].

B.2 Preliminaries

B.2.1 Notation

- The left arrow ($\leftarrow$) signifies either assignment or computation or random selection. The notation $X \leftarrow Y$ means assign the value of Y to the variable X when Y is an expression. If $\mathcal{S}$ is a set, $X \leftarrow \mathcal{S}$ represents choosing X uniformly over $\mathcal{S}$. If $\mathcal{A}$ is an algorithm, $X \leftarrow \mathcal{A}$ represents running $\mathcal{A}$ once and assigning the result to X.
- We use i as the index of a component of a tuple, j to index a tuple in a set.
- We will use the operator $\forall$ to signify a sequence indexed by a variable. For example, $\forall_j(X_j, Y_j)$ represents $(X_1, Y_1), (X_2, Y_2), ...$ for all values of j. $\forall_{j=1}^{m}(c_j, d_j)$ represents $(c_1, d_1), (c_2, d_2), ..., (c_m, d_m)$. Similarly, for a set $\mathcal{S}$, $\forall_{j \in \mathcal{S}} X_j$ represents a sequence of X_j for all values of $j \in \mathcal{S}$. Notice that we use subscript and superscript with $\forall$ for this custom notation. When used with regular text position, $\forall$ has the usual meaning.
- $\mathbb{G}$ is an elliptic curve group of prime order q. We assume that q is close to 2^λ where λ is the security parameter.
- G is a system-wide fixed generator of $\mathbb{G}$.
- L is a system-wide fixed generator of $\mathbb{G}$ such that the discrete logarithm of G with respect to L is not known.
- $\mathbb{F}_q$ is the field integer modulo q.

- $\mathbb{F}_q^*$ is $\mathbb{F}_q \setminus \{0\}$.
- H_g is a hash function with output in the group.
- H_q is a hash function with output in $\mathbb{F}_q^*$.
- H_b is a hash function with output in $\{0,1\}^b$ for a fixed b.
- ϕ is an empty string of bits or an empty message.
- Bold **P** is the prover algorithm for ZkPLMT.
- Bold **V** is the verification algorithm for ZkPLMT.

B.2.2 Definitions

Definition 1 Negligible function A function $f : \mathbb{N} \to \mathbb{R}^+$ is negligible in λ if for every polynomial $\mathbf{p}$, there exists $\mathbf{n}$ such that $f(\lambda) < \frac{1}{\mathbf{p}}$ whenever $\lambda \geq \mathbf{n}$.

Definition 2 DDH Assumption We assume the DDH problem is hard to solve in the group $\mathbb{G}$. The advantage of a DDH adversary $\mathcal{C}$ is given as -

$Adv^{DDH}(\mathcal{C}) =$

$$\left| Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \rho = \\ \qquad \mathcal{C}(G, xG, yG, xyG); \end{array} \right] \right.$$
$$\left. - Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ R \leftarrow \mathbb{G}; \\ \rho = \\ \qquad \mathcal{C}(G, xG, yG, R); \end{array} \right] \right|$$

We assume that the advantage of any PPTA $\mathcal{C}$ is negligible in the security parameter λ.

B.3 Formal Requirements

The following are the types of participants in the Xand blockchain.

- **Member:** A Member is the main transactor in the blockchain. Only a Member can transfer claims to another Member.
- **Validator:** A Validator is a blockchain Validator that creates and submits blocks and votes on them.
- **Trustee:** The Trustee is responsible for confirming that for every claim created or redeemed, an equivalent bank transaction is present.

The following types of transaction are as follows -

- **CreateTransaction:** For requesting creation of claims.
- **TransferTransaction:** For transferring claims between members.
- **RedeemTransaction:** For redeeming claims to get real currency.

The following operations are required for our system. If a function is not explicitly stated as being computationally unbounded, it is assumed to be a probabilistic polynomial time algorithm.

- **GenKeys:** $(sk, pk) \leftarrow GenKeys()$ generates a pair of keys, one secret key sk and one public key pk.

- **CheckKeys:** $KeyMatch \leftarrow CheckKeys(sk, pk)$ returns whether the (sk, pk) match as valid key-pairs generated from a single call to $GenKeys()$.
- **GenID:** $ID \leftarrow GenID(sk)$ generates a random ID given a sk.
- **CheckID:** $pk \leftarrow CheckID(ID, sk)$ checks whether ID is generated from sk.
- **GenOneTimeKey:** $opk \leftarrow GenOneTimeKey(pk)$ produces a random one-time public key given a public key pk.
- **GenTxO:** $TxO \leftarrow GenTxO(pk, value)$ generates a TxO given a public key pk and $value$.
- **GenTxOWithValueApd:** $(TxO, apd) \leftarrow GenTxO(pk, value)$ produces a random one-time public key given a public key pk and $value$. It also returns an apd.
- **CheckTxO:** $TxOValidity \leftarrow CheckTxO(TxO, pk, value, apd)$ is a function that checks whether a given TxO matches the provided $value$, pk, and some additional private data apd.
- **SumTxO:** $ResultTxO \leftarrow SumTxO(TxO_1, TxO_2, ...)$ is a function that takes a set of TxOs as input and returns a sum TxO.
- **DiffTxO:** $ResultTxO \leftarrow DiffTxO(TxO_1, TxO_2)$ is a function that takes a set of TxOs as input and returns a difference TxO.
- **CheckTxOValue:** $TxOValidity \leftarrow CheckTxO(TxO, value, apd)$ is a function that checks whether a given TxO matches the provided $value$ and some additional private data apd.
- **CheckTxOPk:** $TxOValidity \leftarrow CheckTxO(TxO, pk, apd)$ is a function that checks whether a given TxO matches the provided pk and some additional private data apd.
- **GenNullifier:** $Nullifier \leftarrow GenNullifier(TxO, sk)$ is a deterministic function that generates a $Nullifier$ and the sk corresponding to the pk used to generate the TxO. Since $GenNullifier$ is a deterministic function, the $Nullifier$ for a TxO is unique.
- **GenTransfer:** $tr \leftarrow GenTransfer(InputTxos,$
  $TrueIndex, Outputs, InputIDs,$
  $BannedList, sk)$ creates a Transfer transaction tr given a set of $InputTxos, TrueIndex, Outputs, InputIDs,$ $BannedList$ and the private key sk. Here the $InputTxOs$ is a list of tuples of TxOs,$Outputs$ is a list of tuples $(PubKey, Value)$, and the $TrueIndex$ is an integer. The $GenTransfer$ function takes a list of $u - 1$ (pk, value) pairs and generates a change TxO by itself. $BannedList$ is a list of public keys that banned from creating any transactions. $InputTxOs$ is a list of transactions also written as $\forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}} TxO_{ij}$.
- **CheckTransfer:** $TransferValidity \leftarrow CheckTransfer(tr)$ is a function that checks whether a transfer transaction is valid.
- **DecomposeTransfer:** $(InputTxOs, OutputTxOs,$
  $InputIDs, OutputID, BannedList, NullifierSet) \leftarrow DecomposeTransfer(tr)$ returns $(InputDecoyTxOs,$
  $OutputTxos, InputIDs, OutputID, BannedList,$
  $NullifierSet)$ that tr was generated with.
  - $InputTxOs$ is a list of TxOs also written as $\forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}} TxO_{ij}$.
  - $OutputTxOs$ is a list of TxOs that are output of the transaction. It is also written as $\forall_{t \in \{1..u\}} \overline{TxO}_t$.
  - $InputIDs$ is a list of input IDs. One of them belong to the signer. It is also written as $\forall_{j \in \{1..m\}} InputID_j$.
  - $BannedList$ is a list of public keys that are banned for creating transactions.
  - $NullifierSet$ is a list of Nullifiers.

- **DecryptTransferSigner** The function $pks \leftarrow (tr, skt)$ decrypts the signer's permanent public key given the secret key skt for one of the output TxOs in tr.
- **GenCreate:** $ctr \leftarrow GenCreate(TrueIndex,$ $Outputs, OutputIDs, BannedList, sk)$ generates a Create transaction given a set of $OutputTxos$ and $OutputIDs$, and the private key sk.
- **CheckCreate:** $CreateValidity \leftarrow CheckCreate(ctr)$ checks the validity of a Create transaction.
- **DecomposeCreate:** $(OutputTxOs,$ $CreateValue, InputIDs, OutputID, BannedList) \leftarrow DecomposeCreate(ctr)$ decomposes the Create transaction ctr into $(OutputTxOs, CreateValue, InputIDs, OutputID,$ $BannedList)$.
- **GetCreateProof:** $proof \leftarrow GetCreateProof(ctr)$ returns the proof part of a Create transaction ctr.
- **GenRedeem:** $rtr \leftarrow GenRedeem(InputTxos,$ $TrueIndex, Outputs, InputIDs,$ $BannedList, sk)$ creates a Transfer transaction tr given a set of $InputTxos, TrueIndex, Outputs, InputIDs,$ $BannedList$ and the private key sk. Here the $InputTxOs$ is a list of tuples of TxOs,$Outputs$ is a list of tuples $(PubKey, Value)$, and the $TrueIndex$ is an integer.
- **CheckRedeem:** $RedeemValidity \leftarrow CheckRedeem(rtr)$ checks the validity of a Redeem transaction rtr.
- **DecomposeRedeem:** $(InputTxOs, OutputTxos,$ $RedeemValue, InputIDs, OutputID, BannedList,$ $NullifierSet) \leftarrow DecomposeRedeem(rtr)$ returns $(InputDecoyTxOs,$ $OutputTxos, InputIDs, OutputID, BannedList,$ $NullifierSet)$ that rtr was generated with.
- **GetCreateSenderKey:** $pk \leftarrow GetCreateSenderKey(ctr, skt)$ Computes the sender's permanent public key pk given a Create transaction ctr and the secret key skt of the trustee.
- **GetTransferSenderKey:** $pk \leftarrow GetTransferSenderKey(tr, skr)$ Computes the sender's permanent public key pk given a Transfer transaction tr and the secret key skr of the receiver.
- **GetRedeemSenderKey:** $pk \leftarrow GetRedeemSenderKey(tr, skt)$ Computes the sender's permanent public key pk given a Redeem transaction rtr and the secret key skt of the trustee.

We assume that $\mathcal{A}_1$ and $\mathcal{A}_\in$ are probabilistic polynomial time algorithm adversaries (PPTAs) that can communicate between themselves.

B.3.1 Binding:

The TxO construction commits to a unique value.

$$Pr\left[\begin{array}{l|l} & (TxO, v_1, apd_1, pk_1, v_2, \\ & apd_2, pk_2) \leftarrow \mathcal{A}_1; \\ c_1 = 1 \wedge & \\ c_2 = 1 \wedge & c_1 \leftarrow CheckTxO(TxO, v_1, \\ (v_1 \neq v_2 \vee & pk_1, apd_1); \\ pk_1 \neq pk_2) & \\ & c_2 \leftarrow CheckTxO(TxO, v_2, \\ & pk_2, apd_2); \end{array}\right] \approx 0$$

We have a very similar requirement for the $CheckTxOValue$ and the $CheckTxOPk$ function -

$$Pr\left[\begin{array}{l|l} & (TxO, v_1, apd_1, v_2, \\ & apd_2) \leftarrow \mathcal{A}_1; \\ c_1 = 1 \wedge & \\ c_2 = 1 \wedge & c_1 \leftarrow CheckTxOValue(TxO, v_1, \\ (v_1 \neq v_2) & apd_1); \\ & \\ & c_2 \leftarrow CheckTxOValue(TxO, v_2, \\ & apd_2); \end{array}\right] \approx 0$$

$$Pr\left[\begin{array}{l|l} & (TxO, pk_1, apd_1, pk_2, \\ & apd_2) \leftarrow \mathcal{A}_1; \\ c_1 = 1 \wedge & \\ c_2 = 1 \wedge & c_1 \leftarrow CheckTxOPk(TxO, pk_1, \\ (pk_1 \neq pk_2) & apd_1); \\ & \\ & c_2 \leftarrow CheckTxOPk(TxO, pk_2, \\ & apd_2); \end{array}\right] \approx 0$$

B.3.2 Hiding:

The hiding property entails that even if the adversary chooses the values, it is not possible for a PPTA adversary to tell whether two TxOs stand for the same value or different values and whether they are for the same public key or different public keys. Note that having access to the secret would tell you whether the TxO matches the secret key, so the keys are generated separately.

$$Pr\left[\begin{array}{l|l} & (sk_1, pk_1), (sk_2, pk_2) \leftarrow GenKeys() \\ & \\ & (v_1, v_2, state) \leftarrow \mathcal{A}_1; \\ & \\ & TxO = GenTxO(pk_1, v_1); \\ b = b' & \\ & b \leftarrow \{1, 2\}; \\ & \\ & TxO' = GenTxO(pk_b, v_b) \\ & \\ & b' = \mathcal{A}_2(TxO, TxO', pk_1, pk_2 \\ & v_1, v_2, state); \end{array}\right] \approx \frac{1}{2}$$

B.3.3 Sum:

The Sum and Diff properties ensure that TxOs can be summed or subtracted resulting in TxOs with the values that are sum or difference of the input TxOs respectively.

There exists an PPTA $\mathcal{T}$ such that -

$$Pr\left[\begin{array}{l|l} & (\forall_{i\in\{1..n\}}(TxO_i, v_i, \\ & \quad pk_i, apd_i)) \leftarrow \mathcal{A}_1; \\ & \\ & \forall i[c_i \leftarrow CheckTxOValue \\ & \quad (TxO_i, v_i, apd_i)]; \\ & \\ & TxO \leftarrow SumTxO \\ \bigwedge_{i\in\{1..n\}} c_i = 1 & \quad (\forall_{i\in\{1..n\}}(TxO_i)); \\ \Rightarrow c = 1 \wedge & \\ w = 0 & (v, apd) \leftarrow \\ & \quad \mathcal{T}(\forall_{i\in\{1..n\}}(TxO_i, v_i, \\ & \quad\quad apd_i)); \\ & \\ & c \leftarrow CheckTxOValue \\ & \quad\quad (TxO, v, apd); \\ & \\ & w = v - \sum_{i=1}^{n} v_i \end{array}\right] \approx 1$$

B.3.4 Diff:

There exists an PPTA $\mathcal{T}$ such that -

$$Pr\left[\begin{array}{l|l} & (TxO_1, v_1, TxO_2, v_2 \\ & \quad pk_1, apd_1, pk_2, apd_2) \leftarrow \mathcal{A}_1; \\ & \\ & c_1 \leftarrow CheckTxOValue \\ & \quad (TxO_1, v_1, apd_1)]; \\ & \\ & c_2 \leftarrow CheckTxOValue \\ & \quad (TxO_2, v_2, apd_2)]; \\ & \\ c_1 \wedge c_2 & TxO \leftarrow DiffTxO \\ \Rightarrow c = 1 \wedge & \quad (TxO_1, TxO_2); \\ w = 0 & \\ & (v, apd) \leftarrow \\ & \quad \mathcal{T}(TxO_1, v_1, TxO_2, v_2 \\ & \quad\quad apd_1, apd_2); \\ & \\ & c \leftarrow CheckTxOValue \\ & \quad\quad (TxO, v, apd); \\ & \\ & w = v - (v_1 - v_2) \end{array}\right] \approx 1$$

We also required that $CheckTxOValue$ works for $GenTxOWithValueApd$.

$$Pr\left[\begin{array}{l|l} & (pk, v) \leftarrow \mathcal{A}_1; \\ & \\ & (TxO, apd) \leftarrow \\ c = 1 & \quad GenTxOWithValueApd \\ & \quad\quad (pk, value); \\ & \\ & c \leftarrow CheckTxOValue(TxO, v, \\ & \quad apd); \end{array}\right] = 1$$

B.4 Knowledge-soundness for Transfer Transaction

For every PPTA adversary $\mathcal{A}_1$ and PPTA algorithm $\mathcal{P}$ that generates tr such that $CheckTransfer(tr)$ passes with the probability of ϵ, there exists a PPTA extractor $\mathcal{E}$ with the expected runtime of $\frac{8}{\epsilon - \frac{1}{q}}$, that outputs $(sk, pk, k, \forall_{i\in\{1..n\}} apd_i, \overline{apd})$ such that the following requirements are true.

For any PPTA algorithm $\mathcal{P}^{\mathcal{R}}$, where $\mathcal{R}$ is the source of randomness for $\mathcal{P}$ (otherwise $\mathcal{P}$ is deterministic), We define -

$$
\begin{aligned}
&SetupTrWEE(\mathcal{A}_1) := \\
&\textit{The key-pair for which the decryption key is known} \\
&\quad (skt, pkt) \leftarrow GenKeys(); \\
&\quad \mathcal{R} \leftarrow \{0,1\}^{bits} \\
&\quad state \leftarrow \mathcal{A}_1(\mathcal{R}, pkt); \\
&\quad (tr, apdt, t') \leftarrow \mathcal{P}^{\mathcal{R}}(state, pkt); \\
&\quad (sk, pk, k, \forall_{i\in\{1..n\}} apd_i, \overline{apd}) \leftarrow \mathcal{E}(\mathcal{P}, \mathcal{R}, state); \\
&\quad (\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\
&\quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\
&\qquad OutputID, BannedList, NullifierSet) \\
&\qquad\quad \leftarrow DecomposeTransfer(tr); \\
&\quad Return(sk, pk, k, \forall_{i\in\{1..n\}} apd_i, \overline{apd}, skt, pkt, apdt, t', \\
&\qquad tr, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\
&\qquad\quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\
&\qquad\quad OutputID, BannedList, NullifierSet)
\end{aligned}
$$

We now define the following requirements for the Transfer transaction -

1. **Key Match** The point of the Key Match property is to ensure that the private key and the public key returned by the extractor match.

$$
Pr\left[\begin{array}{l|l} ck = 1 & \begin{array}{l} (sk, pk, k, \\ \quad \forall_{i\in\{1..n\}} apd_i, \overline{apd} \\ \quad skt, pkt, apdt, t', \\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\ \quad OutputID, BannedList, \\ \quad NullifierSet) \\ \qquad \leftarrow SetupTrWEE(\mathcal{A}_1); \\ \\ ck \leftarrow CheckKeys(sk, pk); \end{array} \end{array}\right] = 1
$$

2. **Ownership for transfer transaction** The point of the Ownership property is to ensure that the

signer owns the private key of all the TxOs that are being spent.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, \\ & \quad \forall_{i\in\{1..n\}} apd_i, \overline{apd} \\ & \quad skt, pkt, apdt, t', \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\ c=1 & \quad OutputID, BannedList, \\ & \quad NullifierSet) \\ & \qquad \leftarrow SetupTrWEE(\mathcal{A}_1); \\ & \\ & \forall i \in \{1..n\}[c_i \leftarrow CheckTxOPk \\ & \quad (TxO_{ik}, pk, apd_i)]; \\ & \\ & c \leftarrow \bigwedge_{i\in\{1..n\}} c_i \end{array}\right] = 1$$

3. **Nullifiers for transfer transaction** The Nullifier property ensures that the signer has to compute the unique Nullifiers for each input TxO correctly. This allows detection of double spend.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, \\ & \quad \forall_{i\in\{1..n\}} apd_i, \overline{apd} \\ & \quad skt, pkt, apdt, t', \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\ N = NS & \quad OutputID, BannedList, \\ & \quad NullifierSet) \\ & \qquad \leftarrow SetupTrWEE(\mathcal{A}_1); \\ & \\ & N \leftarrow \forall_{i\in\{1..n\}} \\ & GenNullifier(TxO_{ik}, sk); \\ & \\ & NS := NullifierSet; \end{array}\right] = 1$$

4. **Value conservation** The value conservation property ensures that the sum of the input values equals the sum of the output values. Notice that it is given in terms of sum of the TxOs which we already have seen to be equivalent to the values being summed.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, \\ & \quad \forall_{i\in\{1..n\}} apd_i, \overline{apd} \\ & \quad skt, pkt, apdt, t', \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs, \\ & \quad OutputID, BannedList, \\ & \quad NullifierSet) \\ & \qquad \leftarrow SetupTrWEE(\mathcal{A}_1); \\ & \\ c=1 & TxP \leftarrow \\ & \quad SumTxO(\forall_{i\in\{1..n\}} TxO_{ik}); \\ & \\ & TxP' \leftarrow \\ & \quad SumTxO(\forall_{t\in\{1..u\}} \overline{TxO}_t); \\ & \\ & c = CheckTxOValue(\\ & \qquad DiffTxO(TxP, TxP'), \\ & \qquad 0, \overline{apd}); \end{array}\right] = 1$$

5. **Exclusion from BannedList** This property ensures that the signer is not a member of the BannedList.

$$Pr\left[\begin{array}{l|l} c=1 & \begin{array}{l}(sk,pk,k,\\ \quad \forall_{i\in\{1..n\}}apd_i,\overline{apd}\\ \quad skt,pkt,apdt,t',\\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\ \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t),InputIDs,\\ \quad OutputID,BannedList,\\ \quad NullifierSet)\\ \qquad \leftarrow SetupTrWEE(\mathcal{A}_1);\\ \\ c\leftarrow (pk\notin BannedList)\end{array}\end{array}\right]=1$$

6. **Decryption of Signer public key** This property ensures that the receivers are able to decrypt the correct public key of the signer.

$$Pr\left[\begin{array}{l|l} cktt\Rightarrow \\ c & \begin{array}{l}(sk,pk,k,\\ \quad \forall_{i\in\{1..n\}}apd_i,\overline{apd}\\ \quad skt,pkt,apdt,t',\\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\ \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t),InputIDs,\\ \quad OutputID,BannedList,\\ \quad NullifierSet)\\ \qquad \leftarrow SetupTrWEE(\mathcal{A}_1);\\ \\ cktt\leftarrow CheckTxOPk\\ \qquad (\overline{TxO}_{t'},pkt,apdt);\\ \\ pks\leftarrow\\ \quad DecryptTransferSigner\\ \qquad (tr,skt);\\ \\ c\leftarrow ps=pks\end{array}\end{array}\right]=1$$

7. **Membership** This property ensures that the signer is a member of the known members' list. This is achieved through anonymous ID tags. This property proves that the correct InputID and the OutputID belong to the signer.

$$Pr\left[\begin{array}{l|l} c\wedge c' & \begin{array}{l}(sk,pk,k,\\ \quad \forall_{i\in\{1..n\}}apd_i,\overline{apd}\\ \quad skt,pkt,apdt,t',\\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\ \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t),InputIDs,\\ \quad OutputID,BannedList,\\ \quad NullifierSet)\\ \qquad \leftarrow SetupTrWEE(\mathcal{A}_1);\\ \\ c\leftarrow CheckID(InputID_k,sk);\\ \\ c'\leftarrow CheckID(OutputID,sk);\end{array}\end{array}\right]=1$$

B.5 Knowledge-soundness for Redeem Transaction

We define -

$$SetupRmWEE(\mathcal{A}_1):=$$

$(skt, pkt) \leftarrow GenKeys();$
$\mathcal{R} \leftarrow \{0,1\}^{bits}$
$state \leftarrow \mathcal{A}_1(\mathcal{R}, pkt)$

$(rtr, apdt, t') \leftarrow \mathcal{P}^{\mathcal{R}}(state, pkt);$

$(sk, pk, k, apd) \leftarrow \mathcal{E}(\mathcal{P}, \mathcal{R}, state, tr);$

$(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),$
$\forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs,$
$OutputID, BannedList, NullifierSet)$
$\leftarrow DecomposeRedeem(rtr);$

$Return(sk, pk, k, apd, v, skt, pkt, apdt, t',$
$tr, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),$
$\forall_{t\in\{1..u\}}(\overline{TxO}_t), InputIDs,$
$OutputID, BannedList, NullifierSet)$

We now define the following requirements for the Redeem transaction -

1. **Key Match** The point of the Key Match property is to ensure that the private key and the public key returned by the extractor match.

$$Pr\left[\begin{array}{l|l} ck = 1 & \begin{array}{l}(sk, pk, k, v, apd, skt, pkt, apdt,\\ \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t)\\ \quad rtr, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\ \quad RedeemValue, \forall_{t\in\{1..u\}}(\overline{TxO}_t),\\ \quad InputIDs, OutputID,\\ \quad BannedList, NullifierSet)\\ \quad\quad \leftarrow SetupRmWEE(\mathcal{A}_1);\\ \\ ck \leftarrow CheckKeys(sk, pk); \end{array} \end{array}\right] = 1$$

2. **Ownership for Redeem transaction** The point of the Ownership property is to ensure that the signer owns the private key of all the TxOs that are being spent.

$$Pr\left[\begin{array}{l|l} \begin{array}{l} c_{tr} \Rightarrow \\ \bigwedge_{i\in\{1..n\}} c_i \end{array} & \begin{array}{l}(sk, pk, k, v, apd, skt, pkt, apdt,\\ \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), rtr,\\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\ \quad RedeemValue,\\ \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t),\\ \quad InputIDs, OutputID,\\ \quad BannedList, NullifierSet)\\ \quad\quad \leftarrow SetupRmWEE(\mathcal{A}_1);\\ \\ \forall i \in \{1..n\}[c_i \leftarrow CheckTxOPk\\ \quad (TxO_{ik}, pk, apd_i)];\\ \\ \forall t \in \{1..u\}[c_t \leftarrow CheckTxOPk\\ \quad (TxO_t, pk, \overline{apd}_t)]; \end{array} \end{array}\right] \approx 1$$

3. **Nullifiers for Redeem transaction** The Nullifier property ensures that the signer has to compute the unique Nullifiers for each input TxO correctly. This allows detection of double spend.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, v, apd, skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), rtr, \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad RedeemValue, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad InputIDs, OutputID, \\ c_{tr} \Rightarrow & \quad BannedList, NullifierSet) \\ N = NS & \quad\quad \leftarrow SetupRmWEE(\mathcal{A}_1); \\ & \\ & c_{tr} = CheckRedeem(rtr); \\ & \\ & N \leftarrow \forall_{i\in\{1..n\}} \\ & GenNullifier(TxO_{ik}, sk); \\ & \\ & NS := NullifierSet; \end{array}\right] \approx 1$$

4. **Value conservation** The value conservation property ensures that the sum of the input values equals the sum of the output values plus the redemption value. Notice that it is given in terms of sum of the TxOs which we already have seen to be equivalent to the values being summed.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, v, apd, skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), tr, \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad RedeemValue, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad InputIDs, OutputID, \\ & \quad BannedList, NullifierSet) \\ & \quad\quad \leftarrow SetupRmWEE(\mathcal{A}_1); \\ & \\ & ck \leftarrow CheckKeys(sk, pk); \\ c_{tr} \Rightarrow & \\ w = 0 \wedge & c_{tr} = CheckRedeem(rtr); \\ c = 1 \wedge & \\ \bigwedge_{t\in\{1..u\}} c'_t & TxO \leftarrow \\ & \quad SumTxO(\forall_{i\in\{1..n\}} TxO_{ik}); \\ & \\ & c = CheckTxO \\ & \quad\quad (TxO, pk, v, apd); \\ & \\ & \forall t \in \{1..u\}[c'_t \\ & \quad \leftarrow CheckTxOValue \\ & \quad\quad (\overline{TxO}_t, \bar{v}_t, \overline{apd}_t)] \\ & \\ & w \leftarrow v - RedeemValue \\ & \quad\quad\quad - \sum_{t=1}^{u} \bar{v}_t; \end{array}\right] \approx 1$$

5. **Exclusion from BannedList** This property ensures that the signer is not a member of the BannedList.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, v, apd, skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), tr, \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad RedeemValue, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ c_{tr} \Rightarrow & \quad InputIDs, OutputID, \\ c & \quad BannedList, NullifierSet) \\ & \qquad \leftarrow SetupRmWEE(\mathcal{A}_1); \\ & \\ & c_{tr} = CheckRedeem(rtr); \\ & \\ & c \leftarrow (pk \notin BannedList) \end{array}\right] \approx 1$$

6. **Decryption of Signer public key** Unlike in case of the Transfer transaction, in Redeem transaction, the receiver is the same as the signer. The signer's identity must be disclosed to the Trustee, the private key of the trustee is skt and the public key is pkt.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, v, apd, skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), tr, \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad RedeemValue, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad InputIDs, OutputID, \\ & \quad BannedList, NullifierSet) \\ c_{tr} & \qquad \leftarrow SetupRmWEE(\mathcal{A}_1); \\ \Rightarrow & \\ c & c_{tr} = CheckRedeem(rtr, pkt); \\ & \\ & pks \leftarrow \\ & \quad DecryptRedeemSigner \\ & \qquad (rtr, skt); \\ & \\ & c \leftarrow ps = pks \end{array}\right] \approx 1$$

7. **Membership** This property ensures that the signer is a member of the known members' list. This is achieved through anonymous ID tags. This property proves that the correct InputID and the OutputID belong to the signer.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, v, apd, skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\bar{v}_t, \overline{apd}_t), tr, \\ & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), \\ & \quad RedeemValue, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad InputIDs, OutputID, \\ c \wedge c' & \quad BannedList, NullifierSet) \\ & \qquad \leftarrow SetupRmWEE(\mathcal{A}_1); \\ & \\ & \\ & c \leftarrow CheckID(InputID_k, sk); \\ & \\ & c' \leftarrow CheckID(OutputID, sk); \end{array}\right] = 1$$

B.6 Knowledge-soundness for Create Transaction

For every PPTA adversary $\mathcal{A}_1$ and PPTA algorithm $\mathcal{P}$ that generates ctr such that $CheckCreate(ctr)$ passes with the probability of ϵ, there exists a PPTA extractor $\mathcal{E}$ with the expected runtime of $\frac{8}{\epsilon - \frac{1}{q}}$, that outputs (sk, pk, k, apd) such that the following requirements are true.

We define the setup function as follows -

$$
\begin{aligned}
&SetupCrWEE(\mathcal{A}_1) := \\
&\textit{skt, pkt is the key-pair for the trustee} \\
&\quad (skt, pkt) \leftarrow GenKeys(); \\
&\quad \mathcal{R} \leftarrow \{0,1\}^{bits} \\
&\quad state \leftarrow \mathcal{A}_1(pkt, \mathcal{R}); \\
&\quad (ctr, apdt) \leftarrow \mathcal{P}^{\mathcal{R}}(state, pkt); \\
&\quad (sk, pk, k, \overline{apd}_t) \\
&\qquad \leftarrow \mathcal{E}(\mathcal{A}_1, tr); \\
&(\forall_{t \in \{1..u\}} \overline{TxO}_t, CreateValue, CreateRand \\
&\quad \forall_{j \in \{1..m\}} InputID_j, \\
&\quad OutputID, BannedList) \\
&\qquad \leftarrow DecomposeCreate(ctr); \\
&\quad Return(sk, pk, k, apd, \overline{apd}_t, skt, pkt, apdt, \\
&\qquad \forall_{t \in \{1..u\}} (\overline{TxO}_t), \forall_{j \in \{1..m\}} InputID_j, \\
&\qquad OutputID, CreateValue, CreateRand, \\
&\qquad BannedList, ctr)
\end{aligned}
$$

We now define the following requirements for the Redeem transaction -

1. **Key Match** The point of the Key Match property is to ensure that the private key and the public key returned by the extractor match.

$$
Pr\left[\begin{array}{l|l} ck = 1 & \begin{array}{l} (sk, pk, k, apd, \overline{apd}_t, \\ \quad skt, pkt, apdt, \\ \quad \forall_{t \in \{1..u\}} (\overline{TxO}_t), \\ \quad \forall_{j \in \{1..m\}} InputID_j, \\ \quad OutputID, CreateValue, \\ \quad CreateRand, \\ \quad BannedList, ctr) \\ \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ \\ ck \leftarrow CheckKeys(sk, pk); \end{array} \end{array}\right] = 1
$$

2. **Ownership for Create transaction** The point of the Ownership property is to ensure that the signer

owns the private key of all the TxOs that are being spent.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, apd, \overline{apd_t}, \\ & \quad skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO_t}), \\ & \quad \forall_{j\in\{1..m\}} InputID_j, \\ & \quad OutputID, CreateValue, \\ c = 1 & \quad CreateRand, \\ & \quad BannedList, ctr) \\ & \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ & \\ & \forall t \in \{1..u\}[c_i \\ & \quad \leftarrow CheckTxOPk \\ & \quad (\overline{TxO_t}, pk, \overline{apd_t})]; \\ & \\ & c \leftarrow \bigwedge_{i\in\{1..n\}} c_i \end{array}\right] \approx 1$$

3. **Value conservation** The value conservation property ensures that the sum of the output values equals creation value.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, apd, \overline{apd_t}, \\ & \quad skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO_t}), \\ & \quad \forall_{j\in\{1..m\}} InputID_j, \\ & \quad OutputID, CreateValue, \\ & \quad CreateRand, \\ c'' = 1 & \quad BannedList, ctr) \\ & \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ & \\ & TxO' \leftarrow SumTxO \\ & \quad (\forall_{t\in\{1..u\}}\overline{TxO_t}); \\ & \\ & c = CheckTxOValue \\ & \quad (TxO', CreateValue, \\ & \quad CreateRand); \end{array}\right] \approx 1$$

4. **Exclusion from BannedList** This property ensures that the signer is not a member of the BannedList.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, apd, \overline{apd_t}, \\ & \quad skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO_t}), \\ & \quad \forall_{j\in\{1..m\}} InputID_j, \\ & \quad OutputID, CreateValue, \\ c = 1 & \quad CreateRand, \\ & \quad BannedList, ctr) \\ & \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ & \\ & c \leftarrow (pk \notin BannedList) \end{array}\right] \approx 1$$

5. **Decryption of Signer public key** Unlike in case of the Transfer transaction, in Create transaction, the receiver is the same as the signer. The signer's identity must be disclosed to the Trustee, the

private key of the trustee is skt and the public key is pkt.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, apd, \overline{apd}_t, \\ & \quad skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad \forall_{j\in\{1..m\}} InputID_j, \\ & \quad OutputID, CreateValue, \\ & \quad CreateRand, \\ c = 1 & \quad BannedList, ctr) \\ & \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ & \\ & pks \leftarrow \\ & \quad DecrypCreateSigner \\ & \quad (ctr, skt); \\ & \\ & c \leftarrow ps = pks \end{array}\right] \approx 1$$

6. **Membership** This property ensures that the signer is a member of the known members' list. This is achieved through anonymous ID tags. This property proves that the correct InputID and the OutputID belong to the signer.

$$Pr\left[\begin{array}{l|l} & (sk, pk, k, apd, \overline{apd}_t, \\ & \quad skt, pkt, apdt, \\ & \quad \forall_{t\in\{1..u\}}(\overline{TxO}_t), \\ & \quad \forall_{j\in\{1..m\}} InputID_j, \\ & \quad OutputID, CreateValue, \\ & \quad CreateRand, \\ c \wedge c' & \quad BannedList, ctr) \\ & \quad \leftarrow SetupCrWEE(\mathcal{A}_1); \\ & \\ & \\ & c \leftarrow CheckID(InputID_k, sk); \\ & \\ & c' \leftarrow CheckID(OutputID, sk); \end{array}\right] = 1$$

B.7 Completeness:

The completeness property ensures that if is possible to construct a correct transaction as long as the correct inputs are provided.

$$Pr\left[\begin{array}{l|l} & (k, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}\setminus\{k\}} \\ & (TxO_{ij}), \\ & \forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t) \\ & \forall_{j\in\{1..n\}\setminus\{k\}} InputID_j, \\ & BannedList) \leftarrow \mathcal{A}_1; \\ & \\ & \forall_{i\in\{1..n\}}(v_{ik}) \\ & \leftarrow ValueRange; \\ & (sk, pk) \leftarrow GenKeys(); \\ & \\ & \\ & \forall_{i\in\{1..n\}}(TxO_{ik}) \\ c = 1 & \leftarrow \forall_{i\in\{1..n\}} GenTxO(pk, v_{ik}); \\ & \\ & InputID_k \leftarrow GenID(sk); \\ & \\ & tr \leftarrow GenTransfer \\ & (\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), k, \\ & \forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t), \\ & \forall_{j\in\{1..n\}} InputID_j, \\ & BannedList, sk); \\ & \\ & c \leftarrow CheckTransfer(tr); \\ & \end{array}\right] \approx 1$$

$$Pr\left[\begin{array}{l|l} & (k, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}\setminus\{k\}} \\ & (TxO_{ij}), \\ & \forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t) \\ & \forall_{j\in\{1..n\}\setminus\{k\}} InputID_j, \\ & BannedList) \leftarrow \mathcal{A}_1; \\ & \\ & \forall_{i\in\{1..n\}}(v_{ik}) \\ & \leftarrow ValueRange; \\ & \\ & (sk, pk) \leftarrow GenKeys(); \\ & \\ & \forall_{i\in\{1..n\}}(TxO_{ik}) \\ c = 1 & \leftarrow \forall_{i\in\{1..n\}} GenTxO(pk, v_{ik}); \\ & \\ & InputID_k \leftarrow GenID(sk); \\ & \\ & rtr \leftarrow GenRedeem \\ & (\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), k, \\ & \forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t), InputIDs, \\ & BannedList, \\ & sk); \\ & \\ & c \leftarrow CheckRedeem(rtr); \\ & \end{array}\right] = 1$$

$$Pr\left[\begin{array}{l|l} & \forall_{t\in\{1..u\}}(\overline{pk}_t, \bar{v}_t),\\ & \forall_{j\in\{1..n\}\setminus\{k\}} OutputID_j,\\ & BannedList) \leftarrow \mathcal{A}_1;\\ & \\ & (sk, pk) \leftarrow GenKeys();\\ & \\ & OutputID_k \leftarrow GenID(sk);\\ c = 1 & \\ & ctr \leftarrow GenCreate\\ & (\forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t), OutputIDs,\\ & BannedList,\\ & sk);\\ & \\ & c \leftarrow CheckCreate(ctr);\\ & \end{array}\right] = 1$$

B.8 Confidentiality:

B.8.1 Confidentiality for the Transfer transaction

The Confidentiality property is based on computational zero-knowledge property. However, there are values like the public key of the signer and transaction values can be leaked in case of zero-knowledge since the distinguisher cannot tell whether they are correct or a random value even if they are leaked (and hence they don't help in distinguishing). The resolve this issue, we provide the correct values to the distinguisher so that if they are leaked, they can be compared to help in distinguishing the true transaction function from the simulator.

We first define a $SetUpZkTr$ that generates the inputs -

$SetupZkTr()$:

The true index

$k \leftarrow \{1..m\}$

The signer keys

$(sk, pk) \leftarrow GenKeys()$

TxO values

$\forall_{i\in\{1..n\}} v_{ik} \leftarrow ValueRange$

True TxOs

$\forall i \in \{1..n\}[(TxO_{ik}, apd_i)$
$\leftarrow GenTxOWithValueApd(pk, v_{ik});]$

Decoys TxO values

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$
$(v_{ij} \leftarrow ValueRange)$

Decoy TxO keys

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$
$[(sk_{ij}, pk_{ij}) \leftarrow GenKeys()]$

Output TxO key-pairs

$\forall_{t\in\{1..u\}}$
$[(\overline{sk}_t, \overline{pk}_t) \leftarrow GenKeys()]$

Input Decoy TxOs

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$
$[TxO_{ij} \leftarrow GenTxO(pk_{ij}, v_{ij})]$

The true input ID

$InputID_k \leftarrow GenID(sk)$

Decoy input id keys

$\forall j \in \{1..m\} \setminus \{k\}(sks_j, pks_j) \leftarrow GenKeys()$

Decoy Input IDs

$\forall j \in \{1..m\} \setminus \{k\} InputID_j \leftarrow GenID(sks_j)$

BannedList

$\forall e \in \{1..w\}[(skb_e, pkb_e) \leftarrow GenKeys();]$

$return\ (k, \forall_{i\in\{1..n\}}(v_{ik}, apd_i), \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),$

$\forall_{t\in\{1..u\}}\overline{pk}_t, \forall_{t\in\{1..u\}}\overline{sk}_t, pk$

$\forall_{j\in\{1..m\}}(InputID_j),$

$\forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e);$

$\mathcal{O}_1$ is the set of random oracles used by the prover and the verifier, $\mathcal{O}_2$ is the set of random oracles created by the simulator. The adversary $\mathcal{A}_2$ gets only selective access only one of them. There exists a simulator

Sim such that -

$$
Pr\left[\begin{array}{l|l}
 & (k, \forall_{i\in\{1..n\}}(v_{ik}, apd_i),\\
 & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
 & \quad\quad \forall_{t\in\{1..u\}}\overline{pk}_t, pk\\
 & \quad\quad \forall_{j\in\{1..m\}}(InputID_j),\\
 & \quad \forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e)\\
 & \quad\quad \leftarrow SetupZkTr\\
 & \quad\quad\quad (OutputIndeces)\\
 & \\
 & (\forall_{t\in\{1..u\}}(\bar{v}_t),\\
 & \quad\quad state)\\
 & \quad\quad \leftarrow \mathcal{A}_1(\forall_{i\in\{1..n\}}(v_{ik}));\\
 & \\
 & b \leftarrow \{1,2\};\\
 & \\
 & tr_1 \leftarrow GenTransfer^{\mathcal{O}_1}\\
 & \quad (\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
 & \quad \forall_{i\in\{1..n\}}(v_{ik}, apd_i), k,\\
 & \quad \forall_{t\in\{1..u\}}(\overline{pk}_t, \bar{v}_t),\\
 & \quad \forall_{j\in\{1..m\}}InputID_j,\\
 & \quad \forall_{e\in\{1..w\}}pkb_e, sk);\\
 & \\
 & \forall t \in \{1..u\}[(\overline{TxO}_t)\\
 & \quad \leftarrow GenTxO(\overline{pk}_t, \bar{v}_t);]\\
 & \\
 & (tr_2, \mathcal{O}_2) \leftarrow Sim(\\
 & \quad\quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
 & \quad\quad \forall_{j\in\{1..m\}}(InputID_j),\\
 & \quad\quad \forall_{t\in\{1..u\}}(\overline{TxO}_t),\\
 & \quad\quad \forall_{e\in\{1..w\}}pkb_e);\\
 & \\
 & b' \leftarrow \mathcal{A}_2^{\mathcal{O}_b}(state, k, tr_b, pk,\\
 & \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
 & \quad \forall_{t\in\{1..u-1\}}(\overline{pk}_t, \bar{v}_t), InputIDs,\\
 & \quad \forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e);\\
 & \\
 & c \leftarrow CheckTransfer(tr_1);\\
 & \\
 & b'' = c \cdot (b' - 1) + 1;
\end{array}\right] \approx \frac{1}{2}
$$

where $BannedListPvtKeys$ are the private keys of the $BannedList$.

B.8.2 Confidentiality of the Redeem Transaction

$SetupZkRTr() :$

True index

$k \leftarrow \{1..m\}$

Signer's keys

$(sk, pk) \leftarrow GenKeys()$

True input values

$\forall_{i\in\{1..n\}}v_{ik} \leftarrow ValueRange$

True input TxOs

$\forall_{i \in \{1..n\}} TxO_{ik} \leftarrow GenTxO(pk, v)$

Decoy output values

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$

$(v_{ij} \leftarrow ValueRange)$

Redeem value

$(\bar{v} \leftarrow ValueRange)$

Decoy signing keys

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$

$[(sk_{ij}, pk_{ij}) \leftarrow GenKeys()]$

Change amounts

$\forall t \in \{1..u\}[\bar{v}_t \leftarrow ValueRange;]$

Change amoutn TxOs

$\forall t \in \{1..u\}[\overline{TxO}_t \leftarrow GenTxO(pk, \bar{v}_t);]$

Decoy input TxOs

$\forall i \in \{1..n\}, \forall j \in \{1..m\} \setminus \{k\}$

$[TxO_{ij} \leftarrow GenTxO(pk_{ij}, v_{ij})]$

True input ID

$InputID_k \leftarrow GenID(sk)$

Decoy Input IDs

$\forall j \in \{1..m\} \setminus \{k\}(sks_j, pks_j) \leftarrow GenKeys()$

$\forall j \in \{1..m\} \setminus \{k\} InputID_j \leftarrow GenID(sks_j)$

BannedList

$\forall e \in \{1..w\}[(skb_e, pkb_e) \leftarrow GenKeys();]$

return $(k, \forall_{i \in \{1..n\}}(v_{ik}), \forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}}(TxO_{ij}),$

$pk, \bar{v}, \forall_{j \in \{1..m\}}(InputID_j),$

$\forall_{e \in \{1..w\}} pkb_e, \forall_{e \in \{1..w\}} skb_e)$

$$
Pr\left[\; b'' = b \;\middle|\;
\begin{array}{l}
(k, \forall_{i\in\{1..n\}}(v_{ik}),\\
\quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
\quad pk, \bar{v}\\
\quad \forall_{j\in\{1..m\}}(InputID_j),\\
\quad \forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e\\
\quad \leftarrow SetupZkRTr(OutputIndeces)\\
\\
(\forall_{t\in\{1..u\}}(\bar{v}_t),\\
\qquad state)\\
\qquad\quad \leftarrow \mathcal{A}_1(\forall_{i\in\{1..n\}}(v_{ik}), \bar{v});\\
\\
b \leftarrow \{1, 2\};\\
\\
rtr_1 \leftarrow GenRedeem^{\mathcal{O}_1}\\
\quad (\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}), k,\\
\quad \forall_{t\in\{1..u\}}\bar{v}_t, InputIDs,\\
\quad \forall_{e\in\{1..w\}}pkb_e, sk);\\
\\
\forall_{t\in\{1..u\}}(\overline{TxO}_t) \leftarrow GenTxO(pk, \bar{v}_t);\\
\\
(rtr_2, \mathcal{O}_2) \leftarrow Sim(\\
\qquad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
\qquad \forall_{j\in\{1..m\}}InputID_j, \bar{v},\\
\qquad \forall_{t\in\{1..u\}}\overline{TxO}_t, \forall_{e\in\{1..w\}}pkb_e);\\
\\
b' \leftarrow \mathcal{A}_2^{\mathcal{O}_b}(state, k, rtr_b, pk,\\
\quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(TxO_{ij}),\\
\quad \forall_{t\in\{1..u-1\}}\bar{v}_t, InputIDs,\\
\quad \forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e);\\
\\
c \leftarrow CheckTransfer(tr_1);\\
\\
b'' = c \cdot (b' - 1) + 1;
\end{array}
\right] \approx \frac{1}{2}
$$

B.8.3 Confidentiality for the Create transaction

$SetupZkCtr() :$

True Index

$k \leftarrow \{1..m\}$

Key-pair for the signer

$(sk, pk) \leftarrow GenKeys()$

Values for the output TxOs

$(\forall_{t\in\{1..u\}}(\bar{v}_t),$
$state)$
$\leftarrow \mathcal{A}_1;$

Output TxOs

$\forall t \in \{1..u\}[\overline{TxO}_t \leftarrow GenTxO(pk, \bar{v}_t)]$

Key-pair for the Trust

$(skt, pkt) \leftarrow GenKeys()$

True InputID

$InputID_k \leftarrow GenID(sk)$

$\forall j \in \{1..m\} \setminus \{k\}(sks_j, pks_j) \leftarrow GenKeys()$

Decoy InputIDs

$\forall j \in \{1..m\} \setminus \{k\} InputID_j \leftarrow GenID(sks_j)$

BannedList

$\forall e \in \{1..w\}[(skb_e, pkb_e) \leftarrow GenKeys();]$

$return\ (k, \bar{v}_t, \forall_{t\in\{1..u\}}\overline{TxO}_t$
pk, sk, pk, skt, pkt
$\forall_{j\in\{1..m\}}(InputID_j),$
$\forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e);$

$$
Pr\left[\ b'' = b\ \middle|\ \begin{array}{l}
(k, \forall_{t\in\{1..u\}}\bar{v}_t, \forall_{t\in\{1..u\}}\overline{TxO}_t \\
\quad pk, sk, pk, skt, pkt \\
\quad \forall_{j\in\{1..m\}}(InputID_j), \\
\quad \forall_{e\in\{1..w\}}pkb_e, \forall_{e\in\{1..w\}}skb_e) \\
\quad \leftarrow SetupZkCtr(); \\
\\
v \leftarrow \sum_{t=1}^{u}\bar{v}_t \\
\\
b \leftarrow \{1, 2\}; \\
\\
(ctr_2, \mathcal{O}_2) \leftarrow Sim(\\
\quad \forall_{t\in\{1..u\}}\overline{TxO}_t, v \\
\quad pkt, InputIDs, \\
\quad \forall_{e\in\{1..w\}}pkb_e) \\
\\
ctr_1 \leftarrow GenCreate^{\mathcal{O}_1} \\
\quad (k, \bar{v}_t, \\
\quad OutputIDs, \\
\quad \forall_{e\in\{1..w\}}pkb_e, \\
\quad pkt, sk); \\
\\
\\
b' \leftarrow \mathcal{A}_2^{\mathcal{O}_b} \\
\quad (state, ctr_b, pk, pkt, \\
\quad \forall_{t\in\{1..u\}}\bar{v}_t, InputIDs, \\
\quad \forall_{e\in\{1..w\}}pkb_e, \\
\quad \forall_{e\in\{1..w\}}skb_e); \\
\\
c \leftarrow CheckCreate(ctr_1); \\
\\
b'' = c \cdot (b' - 1) + 1;
\end{array}\right] \approx \frac{1}{2}
$$

B.9 Implementation of functions

The ZkPLMT function **P** is defined as follows -

$$\mathbf{P}(M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), k, p) :$$

$\forall j \in \{1..m\} \setminus \{k\}[$
$\quad c_j \leftarrow \mathbb{F}_q;$
$\quad d_j \leftarrow \mathbb{F}_q;$
$\quad \forall i \in \{1..n\}[$
$\quad\quad L_{ij} \leftarrow c_i X_{ij} + d_i Y_{ij};$
$\quad]$
$]$
$w \leftarrow \mathbb{F}_q;$
$\forall i \in \{1..n\}[$
$\quad J_{ik} \leftarrow w X_{ik};$
$]$
$h \leftarrow H_q(M, \forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}} (X_{ij}, Y_{ij}, L_{ij}))$
$d_k \leftarrow h - \sum_{j \in \{1..m\} \setminus \{k\}} d_j;$
$c_k \leftarrow w - p d_k;$
return $(\forall_{j \in \{1..m\}} (c_j, d_j));$

The verification function **V** is defined as follows -

$\mathbf{V}(M, \forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}} (X_{ij}, Y_{ij}), \forall_{j \in \{1..m\}} (c_j, d_j)) :$
$\quad \forall j \in \{1..m\}[$
$\quad\quad \forall i \in \{1..n\}[$
$\quad\quad\quad L_{ij} \leftarrow c_i X_{ij} + d_i Y_{ij};$
$\quad\quad]$
$\quad]$
$\quad h \leftarrow H_q(M, \forall_{i \in \{1..n\}} \forall_{j \in \{1..m\}} (X_{ij}, Y_{ij}, L_{ij}))$
$\quad \rho \leftarrow h = \sum_{j=1}^{m} d_j;$
$\quad$ *return* $(\rho);$

- **GenKeys:** $(sk, pk) \leftarrow GenKeys()$ generates a pair of keys, one secret key sk and one public key pk.

$GenKeys() :$
$\quad p \leftarrow \mathbb{F}_q;$
$\quad P \leftarrow pG;$
$\quad return(pP)$

- **CheckKeys:** $KeyMatch \leftarrow CheckKeys(sk, pk)$ returns whether the (sk, pk) match as valid key-pairs generated from a single call to $GenKeys()$.

$CheckKeys(p, P) :$
$\quad Return(P = pG);$

- **GenID:** $ID \leftarrow GenID(sk)$ generates a random ID given a sk.

$GenID(p) :$

$$
\begin{aligned}
&s \leftarrow \mathbb{F}_q;\\
&A \leftarrow sG;\\
&B \leftarrow spG;\\
&return(A, B);
\end{aligned}
$$

- **CheckID:** $pk \leftarrow CheckID(ID, sk)$ checks whether ID is generated from sk.

$$
\begin{aligned}
&CheckID((A, B), p) :\\
&\quad return(B = pA);
\end{aligned}
$$

- **GenOneTimeKey:** $opk \leftarrow GenOneTimeKey(pk)$ produces a random one-time public key given a public key pk.
- **GenTxO:** $TxO \leftarrow GenTxO(pk, value)$ generates a TxO given a public key pk and $value$.

$$
\begin{aligned}
&GenTxO(P, v) :\\
&\quad r \leftarrow \mathbb{F}_q;\\
&\quad s \leftarrow \mathbb{F}_q;\\
&\quad V \leftarrow rG + vL;\\
&\quad A \leftarrow sG;\\
&\quad B \leftarrow sP;\\
&\quad return(A, B, V);
\end{aligned}
$$

- **GenTxOWithValueApd:**

$$
\begin{aligned}
&GenTxOWithValueApd(P, v) :\\
&\quad r \leftarrow \mathbb{F}_q;\\
&\quad s \leftarrow \mathbb{F}_q;\\
&\quad V \leftarrow rG + vL;\\
&\quad A \leftarrow sG;\\
&\quad B \leftarrow sP;\\
&\quad return((A, B, V), r);
\end{aligned}
$$

- **CheckTxO:** $TxOValidity \leftarrow CheckTxO(TxO, pk, value, apd)$ is a function that checks whether a given TxO matches the provided $value$, pk, and some additional private data apd.

$$
\begin{aligned}
&CheckTxO(TxO : (A, B, V), pk : P,\\
&\quad value : v, apd : (r, s)) :\\
&\quad b_1 \leftarrow (A = sG);\\
&\quad b_2 \leftarrow (B = sP);\\
&\quad b_3 \leftarrow (V = rG + vL);\\
&\quad return(b_1 \wedge b_2 \wedge b_3);
\end{aligned}
$$

- **SumTxO:** $ResultTxO \leftarrow SumTxO(TxO_1, TxO_2, ...)$ is a function that takes a set of TxOs as input and returns a sum TxO.

$$
\begin{aligned}
&SumTxO(\forall_{i \in \{1..n\}}(A_i, B_i, V_i)) :\\
&\quad A \leftarrow \sum_{i=1}^{n} A_i;
\end{aligned}
$$

$$B \leftarrow \sum_{i=1}^{n} B_i;$$
$$V \leftarrow \sum_{i=1}^{n} V_i;$$
$$return(A, B, V);$$

- **CheckTxOValue:** $TxOValidity \leftarrow CheckTxO(TxO, value, apd)$ is a function that checks whether a given TxO matches the provided $value$ and some additional private data apd.

$$CheckTxOValue((A, B, V), v, r):$$
$$b \leftarrow (V = rG + vL);$$
$$return(b);$$

- **CheckTxOPk:** $TxOValidity \leftarrow CheckTxO(TxO, pk, apd)$ is a function that checks whether a given TxO matches the provided pk and some additional private data apd.

$$CheckTxOPk((A, B, V), P, p):$$
$$b_1 \leftarrow (A = pB);$$
$$b_2 \leftarrow (P = pG);$$
$$return(b_1 \wedge b_2);$$

- **GenNullifier:** $Nullifier \leftarrow GenNullifier(TxO, sk)$ is a deterministic function that generates a $Nullifier$ and the sk corresponding to the pk used to generate the TxO. Since $GenNullifier$ is a deterministic function, the $Nullifier$ for a TxO is unique.

$$GenNullifier((A, B, V), p):$$
$$I \leftarrow pH_g((A, B));$$
$$return(I);$$

- **GenTransfer:** $tr \leftarrow GenTransfer(InputTxos,$
$TrueIndex, Outputs, InputIDs,$
$BannedList, sk)$ creates a Transfer transaction tr given a set of $InputTxos, TrueIndex, Outputs, InputIDs,$ $BannedList$ and the private key sk. Here the $InputTxOs$ is a list of tuples of TxOs,$Outputs$ is a list of tuples $(PubKey, Value)$, and the $TrueIndex$ is an integer. The $GenTransfer$ function takes a list of $u - 1$ (pk, value) pairs and generates a change TxO by itself.

$$GenTransfer(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),$$
$$\forall_{i\in\{1..n\}}(r_i, v_i),$$
$$k, \forall_{t\in\{1..u\}}(\bar{P}_t, \bar{v}_t), \forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}\hat{P}_e, p):$$

$$P \leftarrow pG;$$
Output ID tags
$$r' \leftarrow \mathbb{F}_q$$
$$\bar{S} = r'G$$
$$\bar{T} = r'P$$
Banned List and Summation Check
$$z \leftarrow \mathbb{F}_1;$$
$$Z \leftarrow zG;$$

$\forall e \in \{1..w\}[\hat{Q}_e \leftarrow z\hat{P}_e]$

$\mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))$

$\alpha \leftarrow \mathbf{P}(\phi, \mathbf{X}', 1, z)$

Output TxOs

$\forall t \in \{1..u-1\}[$

$\bar{r}_t \leftarrow \mathbb{F}_q$

$\bar{s}_t \leftarrow \mathbb{F}_q$

$\bar{A}_t \leftarrow \bar{s}_t G$

$\bar{B}_t \leftarrow \bar{s}_t \bar{P}_t$

$\bar{V}_t \leftarrow \bar{r}_t G + \bar{v}_t L$

$]$

$\bar{r}_u \leftarrow (-pz) + \sum_{i=1}^{n} r_i - \sum_{t=1}^{u-1} \bar{r}_t;$

$\bar{s}_u \leftarrow \mathbb{F}_q;$

$\bar{V}_u \leftarrow \bar{r}_u G + \bar{v}_u L;$

$\bar{A}_u \leftarrow \bar{s}_u G;$

$\bar{B}_u \leftarrow \bar{s}_u \bar{P}_u;$

Nullifiers

$\forall i \in \{1..n\}, j \in \{1..m\}[H_{ij} \leftarrow H_g(A_{ij}, B_{ij})];$

$\forall i \in \{1..n\}[I_i \leftarrow pH_{ik}]$

BulletProof

$\pi_b \leftarrow \mathbf{BP}(\forall_{t\in\{1..u\}}(\bar{r}_t, \bar{v}_t));$

Verifiable encryption

$y \leftarrow \mathbb{F}_q;$

$Y \leftarrow yG;$

$P' \leftarrow pY$

$x \leftarrow \mathbb{F}_q$

$\forall t \in \{1..u\}[$

$\bar{X}_t \leftarrow x\bar{A}_t;$

$\bar{E}_t \leftarrow x\bar{B}_t + P;$

$\bar{E}'_t \leftarrow y\bar{E}_t;$

$\bar{B}'_t \leftarrow y\bar{B}_t;$

$\bar{Q}'_t \leftarrow xy\bar{B}_t;$

$]$

$\pi_1 \leftarrow \mathbf{P}(\phi, ((G, Y), \forall_{t\in\{1..u\}}(\bar{E}_t, \bar{E}'_t),$

$\forall_{t\in\{1..u\}}(\bar{B}_t, \bar{B}'_t)), 1, y);$

$\pi_2 \leftarrow \mathbf{P}(\phi,$

$(\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{X}_t), \forall_{t\in\{1..u\}}(\bar{B}'_t, \bar{Q}'_t)), 1, x);$

Main signature

$\forall j \in \{1..m\}[\mathbf{Y}_j = (\forall_{i\in\{1..n\}}((A_{ij}, B_{ij}),$

$\forall_{i\in\{1..n\}}(H_{ij}, I_i),$

$$(Z, \sum_{i=1}^{n} V_{ij} - \sum_{t=1}^{u} \bar{V}_t),$$
$$(S_j, T_j), (\bar{S}, \bar{T}), (Y, P'))];$$

M is the content of the transaction in binary

$$\beta \leftarrow \mathbf{P}(M, \forall_{j\in\{1..m\}} \mathbf{Y}_j, k, p);$$

Return transaction

$$return(\forall_{i\in\{1..n\}} \forall_{j\in\{1..m\}} (A_{ij}, B_{ij}, V_{ij})$$
$$\forall_{j\in\{1..m\}} (S_j, T_j), (\bar{S}, \bar{T}),$$
$$\forall_{t\in\{1..u\}} [(\bar{A}_t, \bar{B}_t, \bar{V}_t), Z, \pi_b$$
$$Y, P', \forall_{t\in\{1..u\}} (\bar{X}_t, \bar{E}_t, \bar{E}'_t, \bar{B}'_t, \bar{Q}'_t),$$
$$\alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}} I_i, \forall_{e\in\{1..w\}} (\hat{P}_e, \hat{Q}_e));$$

- **CheckTransfer:** $TransferValidity \leftarrow CheckTransfer(tr)$ is a function that checks whether a Transfer transaction is valid.

$$CheckTransfer((\forall_{i\in\{1..n\}} \forall_{j\in\{1..m\}} (A_{ij}, B_{ij}, V_{ij})$$
$$\forall_{j\in\{1..m\}} (S_j, T_j), (\bar{S}, \bar{T}),$$
$$\forall_{t\in\{1..u\}} [(\bar{A}_t, \bar{B}_t, \bar{V}_t), Z, \pi_b$$
$$Y, P', \forall_{t\in\{1..u\}} (\bar{X}_t, \bar{E}_t, \bar{E}'_t, \bar{B}'_t, \bar{Q}'_t),$$
$$\alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}} I_i, \forall_{e\in\{1..w\}} (\hat{P}_e, \hat{Q}_e)) :$$

Main signature verification

$$\forall j \in \{1..m\} [\mathbf{Y}_j = (\forall_{i\in\{1..n\}} ((A_{ij}, B_{ij}),$$
$$\forall_{i\in\{1..n\}} (H_{ij}, I_i),$$
$$(Z, \sum_{i=1}^{n} V_{ij} - \sum_{t=1}^{u} \bar{V}_t),$$
$$(S_j, T_j), (\bar{S}, \bar{T}), (Y, P'))];$$

M is the content of the transaction in binary

$$\rho \leftarrow \mathbf{V}(M, (\forall_{j\in\{1..m\}} \mathbf{Y}_j), \beta);$$
$$if\ (\rho \neq 1)\ return\ 0;$$

BulletProof

$$\rho_b \leftarrow \mathbf{BV}(\forall_{t\in\{1..u\}} \bar{V}_t, \pi_b);$$
$$if\ (\rho_b \neq 1)\ return\ 0;$$

Banned List and Summation Check

$$\mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}} (\hat{P}_e, \hat{Q}_e))$$
$$\rho_\alpha \leftarrow \mathbf{V}(\phi, \mathbf{X}', \alpha);$$
$$if\ (\rho_\alpha \neq 1)\ return\ 0;$$

Verifiable encryption

$$\rho_1 \leftarrow \mathbf{V}(\phi, ((G, Y), \forall_{t\in\{1..u\}} (\bar{E}_t, \bar{E}'_t),$$
$$\forall_{t\in\{1..u\}} (\bar{B}_t, \bar{B}'_t)), \pi_1);$$
$$\rho_2 \leftarrow \mathbf{V}(\phi,$$
$$(\forall_{t\in\{1..u\}} (\bar{A}_t, \bar{X}_t), \forall_{t\in\{1..u\}} (\bar{B}'_t, \bar{Q}'_t)), \pi_2);$$
$$if\ (\rho_1 \wedge \rho_2 \neq 1)\ return\ 0;$$
$$\forall_{t\in\{1..u\}} [$$

$$
\begin{aligned}
&\textit{if } (\bar{E}'_t \neq \bar{Q}'_t + P') \\
&\quad \textit{return } 0; \\
&] \\
&\textit{return } 1;
\end{aligned}
$$

- **DecomposeTransfer:** $(InputTxOs, OutputTxos, InputIDs, OutputID, BannedList, NullifierSet) \leftarrow DecomposeTransfer(tr)$ returns $(InputDecoyTxOs, OutputTxos, InputIDs, OutputID, BannedList, NullifierSet)$ that tr was generated with.

- **GenCreate:** $ctr \leftarrow GenCreate(TrueIndex, Outputs, OutputIDs, BannedList, Trustee, sk)$ generates a Create transaction given a set of $OutputTxos$ and $OutputIDs$, and the private key sk.

$$
\begin{aligned}
&GenCreate(k, \forall_{t\in\{1..u\}}\bar{v}_t, \forall_{j\in\{1..m\}}(S_j, T_j), \\
&\quad \forall_{e\in\{1..w\}}(\hat{P}_e), \hat{T}, p) : \\
&\\
&\quad P \leftarrow pG; \\
&\quad \bar{v}_0 \leftarrow \sum_{t=1}^{u} \bar{v}_t \\
&\textit{Output ID tags} \\
&\quad r' \leftarrow \mathbb{F}_q \\
&\quad \bar{S} = r'G \\
&\quad \bar{T} = r'P \\
&\textit{Outputs} \\
&\quad \forall t \in \{1..u\}[\\
&\qquad \bar{r}_t \leftarrow \mathbb{F}_q \\
&\qquad \bar{V}_t \leftarrow \bar{r}_t G + \bar{v}_t L \\
&\qquad \bar{s}_t \leftarrow \mathbb{F}_q \\
&\qquad \bar{A}_t \leftarrow \bar{s}_t G \\
&\qquad \bar{B}_t \leftarrow \bar{s}_t P \\
&\quad] \\
&\quad \bar{r} \leftarrow \sum_{t=1}^{u} \bar{r}_t \\
&\textit{BulletProof} \\
&\quad \pi_b \leftarrow \mathbf{BP}(\forall_{t\in\{1..u\}}(\bar{r}_t, \bar{v}_t)); \\
&\textit{Verifiable encryption} \\
&\quad y \leftarrow \mathbb{F}_q; \\
&\quad Y \leftarrow yG; \\
&\quad P' \leftarrow pY \\
&\quad x \leftarrow \mathbb{F}_q \\
&\quad \bar{X} \leftarrow xG; \\
&\quad \bar{E} \leftarrow x\hat{T} + P; \\
&\quad \bar{E}' \leftarrow y\bar{E};
\end{aligned}
$$

$\bar{B}' \leftarrow y\hat{T}$;

$\bar{Q}' \leftarrow xy\hat{T}$;

$\pi_1 \leftarrow \mathbf{P}(\phi, ((G, Y), (\bar{E}, \bar{E}'),$
$(\hat{T}, \bar{B}')), 1, y)$;

$\pi_2 \leftarrow \mathbf{P}(\phi,$
$((G, \bar{X}'), (\bar{B}', \bar{Q}')), 1, x)$;

Banned List

$z \leftarrow \mathbb{F}_q$;

$Z \leftarrow zG$;

$\mathbf{Z} \leftarrow pZ$;

$\forall e \in \{1..w\}[\hat{Q}_e \leftarrow z\hat{P}_e]$

$\mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))$

$\alpha \leftarrow \mathbf{P}(\phi, \mathbf{X}', 1, z)$

Main signature

$\forall j \in \{1..m\}[\mathbf{Y}_j = ($
$(S_j, T_j), (\bar{S}, \bar{T}), (Y, P'), (Z, \mathbf{Z}), \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t))]$;

M is the content of the transaction in binary

$\beta \leftarrow \mathbf{P}(M, \forall_{j\in\{1..m\}}\mathbf{Y}_j, k, p)$;

Return transaction

$return($
$\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),$
$\bar{r}, \bar{v}, \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \pi_b$
$Z, \mathbf{Z}, Y, P', (\bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}'), \hat{T}$
$\alpha, \beta, \pi_1, \pi_2, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))$;

- **CheckCreate:** $CreateValidity \leftarrow CheckCreate(ctr)$ checks the validity of a Create transaction.

$CheckCreate($
$\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),$
$\bar{r}, \bar{v}, \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \pi_b$
$Z, \mathbf{Z}, Y, P', (\bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}'), \hat{T}$
$\alpha, \beta, \pi_1, \pi_2, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)) :$

Banned List

$\mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))$

$\rho_\alpha \leftarrow \mathbf{V}(\phi, \mathbf{X}', \alpha)$;

$if\ (\rho_\alpha \neq 1)\ return\ 0$;

Output Sum Check

$\bar{V} \leftarrow \sum_{t=1}^{u} \bar{V}_t$

$if\ (\bar{V} \neq \bar{r}G + \bar{v}L)\ return\ 0$;

BulletProof

$\rho_b \leftarrow \mathbf{BV}(\forall_{t\in\{1..u\}}\bar{V}_t, \pi_b)$;

$$
\begin{aligned}
&\textit{if } (\rho_b \neq 1) \textit{ return } 0;\\
&\textit{Verifiable encryption}\\
&\rho_1 \leftarrow \mathbf{V}(\phi, ((G, Y), (\bar{E}, \bar{E}'),\\
&\qquad (\hat{T}, \bar{B}')), \pi_1);\\
&\textit{if } (\rho_1 \neq 1) \textit{ return } 0;\\
&\rho_2 \leftarrow \mathbf{V}(\phi,\\
&\qquad ((G, \bar{X}'), (\bar{B}', \bar{Q}')), \pi_2);\\
&\textit{if } (\rho_2 \neq 1) \textit{ return } 0;\\
&\textit{if } (\bar{E}' \neq \bar{Q}' + P') \textit{ return } 0;\\
&\textit{Main signature}\\
&\forall j \in \{1..m\}[\mathbf{Y}_j = (\\
&\qquad (S_j, T_j), (\bar{S}, \bar{T}), (Y, P'), (Z, \mathbf{Z}), \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t))];\\
&\textit{M is the content of the transaction in binary}\\
&\rho_\beta \leftarrow \mathbf{V}(M, \forall_{j\in\{1..m\}}\mathbf{Y}_j, \beta);\\
&\textit{if } (\rho_\beta \neq 1) \textit{ return } 0;\\
&\textit{Return transaction}\\
&\textit{return } (1);
\end{aligned}
$$

- **DecomposeCreate:** $(OutputTxOs,$ $CreateValue, InputIDs, OutputID, BannedList) \leftarrow DecomposeCreate(ctr)$ decomposes the Create transaction ctr into $(OutputTxOs, CreateValue, InputIDs, OutputID,$ $BannedList)$.

- **GenRedeem:** $rtr \leftarrow GenRedeem(InputTxos,$ $TrueIndex, Outputs, InputIDs,$ $BannedList, sk)$ creates a Transfer transaction tr given a set of $InputTxos, TrueIndex, Outputs, InputIDs,$ $BannedList$ and the private key sk. Here the $InputTxOs$ is a list of tuples of TxOs,$Outputs$ is a list of tuples $(PubKey, Value)$, and the $TrueIndex$ is an integer.

$$
\begin{aligned}
&GenRedeem(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),\\
&\qquad \forall_{i\in\{1..n\}}(r_i, v_i), \bar{v}, \hat{T},\\
&\qquad k, \forall_{t\in\{1..u\}}(\bar{v}_t), \forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}\hat{P}_e, p) :\\
&\qquad P \leftarrow pG;\\
&\textit{Output ID tags}\\
&\qquad r' \leftarrow \mathbb{F}_q\\
&\qquad \bar{S} = r'G\\
&\qquad \bar{T} = r'P\\
&\textit{Banned List and Summation Check}\\
&\qquad z \leftarrow \mathbb{F}_1;\\
&\qquad Z \leftarrow zG;\\
&\qquad \forall e \in \{1..w\}[\hat{Q}_e \leftarrow z\hat{P}_e]\\
&\qquad \mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))\\
&\qquad \alpha \leftarrow \mathbf{P}(\phi, \mathbf{X}', 1, z)\\
&\textit{Output TxOs}\\
&\qquad \bar{V} \leftarrow \bar{v}L;
\end{aligned}
$$

$\bar{A} \leftarrow G;$

$\bar{B} \leftarrow \hat{T};$

$\forall t \in \{1..u-1\}[\bar{r}_t \leftarrow \mathbb{F}_q];$

$\bar{r}_u \leftarrow (-pz) + \sum_{i=1}^{n} r_i - \sum_{t=1}^{u-1} r_t;$

$\forall t \in \{1..u\}[\bar{s}_t \leftarrow \mathbb{F}_q;]$

$\forall t \in \{1..u\}[\bar{V}_t \leftarrow \bar{r}_t G + \bar{v}_t L];$

$\forall t \in \{1..u\}[\bar{A}_t \leftarrow \bar{s}_t G;]$

$\forall t \in \{1..u\}[\bar{B}_t \leftarrow \bar{s}_t P;]$

Nullifiers

$\forall i \in \{1..n\}, j \in \{1..m\}[H_{ij} \leftarrow H_g(A_{ij}, B_{ij})];$

$\forall i \in \{1..n\}[I_i \leftarrow pH_{ik}]$

BulletProof

$\pi_b \leftarrow \mathbf{BP}((\bar{r}_c, \bar{v}_c));$

Verifiable encryption

$y \leftarrow \mathbb{F}_q;$

$Y \leftarrow yG;$

$P' \leftarrow pY$

$x \leftarrow \mathbb{F}_q$

$\bar{X} \leftarrow x\bar{G};$

$\bar{E} \leftarrow x\hat{T} + P;$

$\bar{E}' \leftarrow y\bar{E};$

$\bar{B}' \leftarrow y\hat{T};$

$\bar{Q}' \leftarrow xy\hat{T};$

$\pi_1 \leftarrow \mathbf{P}(\phi, ((G, Y), (\bar{E}, \bar{E}'),$
$(\hat{T}, \bar{B}')), 1, y);$

$\pi_2 \leftarrow \mathbf{P}(\phi,$
$((G, \bar{X}'), (\bar{B}', \bar{Q}')), 1, x);$

Main signature

$\forall j \in \{1..m\}[\mathbf{Y}_j = (\forall_{i\in\{1..n\}}((A_{ij}, B_{ij}),$
$\forall_{i\in\{1..n\}}(H_{ij}, I_i),$
$(Z, \sum_{i=1}^{n} V_{ij} - \bar{V} - \sum_{t=1}^{u} \bar{V}_t),$
$(S_j, T_j), (\bar{S}, \bar{T}), \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t), (Y, P'))];$

M is the content of the transaction in binary

$\beta \leftarrow \mathbf{P}(M, \forall_{j\in\{1..m\}}\mathbf{Y}_j, k, p);$

Return transaction

$return(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij})$
$\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),$
$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \hat{T}, Z, \pi_b$
$Y, P', \bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}',$

$$\alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}} I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e));$$

- **CheckRedeem:** $RedeemValidity \leftarrow CheckRedeem(rtr)$ checks the validity of a Redeem transaction rtr.

$$\begin{aligned}
&CheckRedeem(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij})\\
&\quad \forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\
&\quad \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \hat{T}, Z, \pi_b\\
&\quad Y, P', \bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}',\\
&\quad \alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}} I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)) :\\
&\textit{Banned List and Summation Check}\\
&\quad \forall e \in \{1..w\}[\hat{Q}_e \leftarrow z\hat{P}_e]\\
&\quad \mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))\\
&\quad \rho_\alpha \leftarrow \mathbf{V}(\phi, \mathbf{X}', \alpha);\\
&\quad \textit{if } (\rho_\alpha \neq 1) \textit{ return } 0;\\
&\textit{Output TxOs}\\
&\quad \bar{V}_r \leftarrow \bar{v}_r L;\\
&\textit{BulletProof}\\
&\quad \rho_b \leftarrow \mathbf{BV}(V_c);\\
&\quad \textit{if } (\rho_b \neq 1) \textit{ return } 0;\\
&\textit{Verifiable encryption}\\
&\quad \rho_1 \leftarrow \mathbf{V}(\phi, ((G, Y), (\bar{E}, \bar{E}'),\\
&\qquad (\hat{T}, \bar{B}')), \pi_1);\\
&\quad \textit{if } (\rho_1 \neq 1) \textit{ return } 0;\\
&\quad \rho_2 \leftarrow \mathbf{V}(\phi,\\
&\qquad ((G, \bar{X}'), (\bar{B}', \bar{Q}')), \pi_2);\\
&\quad \textit{if } (\rho_2 \neq 1) \textit{ return } 0;\\
&\quad \textit{if } (\bar{E}' \neq \bar{Q}' + P') \textit{ return } 0;\\
&\textit{Main signature}\\
&\quad \forall j \in \{1..m\}\mathbf{Y}_j = (\forall_{i\in\{1..n\}}((A_{ij}, B_{ij}),\\
&\qquad \forall_{i\in\{1..n\}}(H_{ij}, I_i),\\
&\qquad (Z, \sum_{i=1}^{n} V_{ij} - \bar{V} - \sum_{t=1}^{u} \bar{V}_t),\\
&\qquad (S_j, T_j), (\bar{S}, \bar{T}), \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t), (Y, P'))];\\
&\quad M \textit{ is the content of the transaction in binary}\\
&\quad \rho_\beta \leftarrow \mathbf{V}(M, \forall_{j\in\{1..m\}}\mathbf{Y}_j, \beta);\\
&\quad \textit{if } (\rho_\beta \neq 1) \textit{ return } 0;\\
&\textit{Return}\\
&\quad \textit{return } 1;
\end{aligned}$$

- **DecomposeRedeem:** $(InputTxOs, OutputTxos,$ $RedeemValue, InputIDs, OutputID, BannedList,$ $NullifierSet) \leftarrow DecomposeRedeem(rtr)$ returns $(InputDecoyTxOs,$ $OutputTxos, InputIDs, OutputID, BannedList,$ $NullifierSet)$ that rtr was generated with.

- **GetCreateSenderKey:** $pk \leftarrow GetCreateSenderKey(ctr, skt)$ Computes the sender's permanent public key pk given a Create transaction ctr and the secret key skt of the trustee.

$$
\begin{aligned}
&GetCreateSenderKey(\\
&\quad (\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\
&\quad \forall_{t\in\{1..u\}}[(\bar{V}_t, \bar{r}_t), (A, B), v_0, \pi_b\\
&\quad Y, P', (\bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}'), \hat{T}\\
&\quad \alpha, \beta, \pi_1, \pi_2, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e), \hat{p}) :\\
&\quad P \leftarrow \bar{E} - \hat{p}\bar{X};\\
&\quad return\ P;
\end{aligned}
$$

- **GetTransferSenderKey:** $pk \leftarrow GetTransferSenderKey(tr, skr)$ Computes the sender's permanent public key pk given a Transfer transaction tr and the secret key skr of the receiver.

$$
\begin{aligned}
&DecryptTransferSigner(\\
&\quad ((\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij})\\
&\qquad \forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\
&\qquad \forall_{t\in\{1..u\}}[(\bar{A}_t, \bar{B}_t, \bar{V}_t), Z, \pi_b\\
&\qquad Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{E}'_t, \bar{B}'_t, \bar{Q}'_t),\\
&\qquad \alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}}I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)) :\\
&\quad \forall t \in \{1..u\}[\\
&\qquad if\ (\bar{B}_t = q\bar{A}_t)\{\\
&\qquad\quad P \leftarrow \bar{E}_t - \bar{p}\bar{X}_t;\\
&\qquad\quad return\ P;\\
&\qquad \}\\
&\quad]\\
&\quad return\ \perp;
\end{aligned}
$$

- **GetRedeemSenderKey:** $pk \leftarrow GetRedeemSenderKey(tr, skt)$ Computes the sender's permanent public key pk given a Redeem transaction rtr and the secret key skt of the trustee.

$$
\begin{aligned}
&GetRedeemSenderKey(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij})\\
&\quad \forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\
&\quad \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \hat{T}, Z, \pi_b\\
&\quad Y, P', \bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}',\\
&\quad \alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}}I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)) :\\
&P \leftarrow \bar{E} - \hat{p}\bar{X};\\
&return\ P;
\end{aligned}
$$

B.10 Proofs

Let us construct a few submodules -

$$
\begin{aligned}
&createAnotherDHTuple(A, B, C, D) :\\
&\quad x \leftarrow \mathbb{F}_q;\\
&\quad y \leftarrow \mathbb{F}_q;
\end{aligned}
$$

$E \leftarrow xA + yC;$
$F \leftarrow xB + yD;$
$return\ (E, F);$

Lemma 1 *In createAnotherDHTuple, if (A, B, C, D) is a DDH tuple such that for some p, $B = pA, D = pC$, then $F = pE$ and the base point E is chosen uniformly over $\mathbb{G}$; otherwise (E, F) is a random pair of points sampled uniformly over $\mathbb{G}^2$.*

Proof If we have $B = pA, D = pC$, then $E = xA+yC$ and $F = xB+yD = xpA+ypD = p(xA+yC) = pE$. Also, since x, y are chosen uniformly, so is E.

Otherwise, if (A, B, C, D) is not a DH tuple, let use choose some generator G. Since $\mathbb{G}$ is a prime order group, there exists a, b, c, d, e, f such that $A = aG, b = bG, c = cG, D = dG, E = eG$, and $F = fG$. So, we have $e = xa + yb$ and $f = xc + yd$, or have -

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}^{-1} \begin{bmatrix} e \\ f \end{bmatrix}$$

Since, $b/a \neq d/c$ as (A, B, C, D) is not a DH tuple, $\begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is invertible. So, for any arbitrary E, F, we have a unique x, y. Since (x, y) is chosen uniformly over $\mathbb{F}_q^2$, (E, F) is chosen uniformly over $\mathbb{G}^2$.

$createNDifferentDHTuple(A, B, C, D, n)$:
$u \leftarrow \mathbb{F}_q;$
$v \leftarrow \mathbb{F}_q;$
$Y \leftarrow uA + vB;$
$Z \leftarrow uC + vD;$
$\forall_{l \in \{1..n\}}[$
$(E_l, F_l) \leftarrow createAnotherDHTuple(A, B, C, D);$
$K_l \leftarrow E_l;$
$L_l \leftarrow uE_l + vF_l$
$]$
$return\ (Y, Z, \forall_{l \in \{1..n\}}(K_l, L_l));$

Lemma 2 *Let $(Y, \forall_{l \in \{1..n\}}(K_l, L_l)) \leftarrow createNDifferentDHTuple(A, B, C, D, n)$, then if (A, B, C, D) is a DDH tuple, there exist some y such that $Y = yA$ and $\forall l\{1..n\}L_l = yK_l$; $Z = yC$; and Y is chosen uniformly over $\mathbb{G}$. Otherwise, if (A, B, C, D) is not a DDH tuple, then $(Y, Z, \forall_{l \in \{1..n\}}(K_l, L_l))$ are random points uniformly chosen over $\mathbb{G}^{2n+2}$.*

Proof If (A, B, C, D) is a DDH tuple, there exists some p, such that $B = pA$ and $D = pC$. So, $Y = uA + vB = uA + vpA = (u + vp)A$. Similarly, $Z = (u + vp)C$. Let us assume $y = u + vp$. So, $Y = yA$ and $Z = yC$. From lemma 1, we also have $F = pE$ since $B = pA, D = pC$. Hence, for any l, $L_l = uE_l + vF_l = uE_l + vpE_l = (u + vp)E_l = yE_l$. We also have $K_l = E_l$. Hence, $L_l = yK_l$. Also, since u, v are chosen uniformly, so is Y.

Now, if (A, B, C, D) is not a DH tuple, then, for every (E_l, F_l) is a random tuple uniformly chosen over $\mathbb{G}^2$. Hence, all pairs (K_l, L_l) and also random points uniformly chosen over $\mathbb{G}^2$. Also, since u, v are uniformly chosen of $\mathbb{F}_q^2$, Y is also a uniformly chosen random point.

B.10.1 Properties of TxOs

Theorem 1 *The binding properties of TxO is true.*

PROOF Given a TxO (A, B, V), the value commitment is a Pedersen commitment for which the binding property is already known. Now, if the private key is p, then $B = pA$. So, the pair (A, B) cannot match any other private key, and hence cannot match any other public key pG.

Theorem 2 *The hiding property of TxO is true.*

PROOF We first define a choice-DDH problem - Given the values G, p_1G, p_2G, aG, bG, where $b = p_1a$ or p_2a, find out which one is the case. The advantage of choice-DDH is defined as follows -

$$Adv^{Choice-DDH}(\mathcal{J}) =$$

$$\left| Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (p_1, p_1, a) \leftarrow \mathbb{F}_q^2; \\ \rho = \\ \qquad \mathcal{J}(G, p_1G, p_2G, aG, p_1aG); \end{array} \right] \right.$$

$$\left. - Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (p_1, p_1, a) \leftarrow \mathbb{F}_q^2; \\ \rho = \\ \qquad \mathcal{J}(G, p_1G, p_2G, aG, p_2aG); \end{array} \right] \right|$$

Lemma 3 *If for some algorithm $\mathcal{J}$, $Adv^{Choice-DDH}(\mathcal{J}) = \epsilon$, then we can construct a DDH-Solver $\mathcal{C}$ that has an advantage of $\epsilon/2$.*

PROOF We construct the following DDH-solver -

$$\begin{array}{l} \mathcal{C}(G, p_1G, aG, R_1) : \\ \quad (P_2, R_2) \\ \qquad \leftarrow createNDifferentDHTuple \\ \qquad\quad (G, p_1G, aG, R_1, 0); \\ \quad \rho_1 \leftarrow \{1, 2\}; \\ \quad \rho \leftarrow \mathcal{J}(G, p_1G, P_2, aG, R_{\rho_1}); \\ \quad return\ (\rho = \rho_1); \end{array}$$

When R_1 is random, the probability of $\rho = \rho_1$ is $\frac{1}{2}$ (Since R_1, R_2, P_2 are all random points). But if $R_1 = p_1aG$, we have $R_2 = aP_2$. Hence, the problem given to $\mathcal{J}$ is the correct distinguishing problem to tell the value of ρ. The probability of $\rho = \rho_1$ is $\frac{1+\epsilon}{2}$. Hence, the advantage of $\mathcal{C}$ is $\epsilon/2$.

Now, we reduce the solver for the choice-DDH to a hiding adversary $\mathcal{A}_2$.

$$\begin{array}{l} \mathcal{J}(G, p_1G, p_2G, aG, p_\rho aG) : \\ \quad (v_1, v_2, state) \leftarrow \mathcal{A}_1 \\ \quad r \leftarrow \mathbb{F}_q; \\ \quad (V_1, V_2) \leftarrow \mathbb{G}^2; \\ \quad TxO \leftarrow (rG, r(p_1G), V_1); \\ \quad TxO' \leftarrow (aG, p_\rho aG, V_2); \\ return\ \mathcal{A}_2(TxO, TxO', p_1G, p_2G, v_1, v_2, state); \end{array}$$

When $\rho = 1$, TxO' is a TxO for the same public key p_1, otherwise, it is a TxO for the other public key p_2. Also, since any curve point can be the commitment for any value, V_2 can be seen as either the commitment of v_1 or the commitment for v_2 equally. So, this solves the choice-DDH problem with the same advantage.

Theorem 3 *The Sum and Diff properties of TxOs are true.*

PROOF The proof is trivial.

B.10.2 Knowledge-soundness for the ZkPLMT functions

We first prove the following theorem for ZkPLMT. For every PPTA $\mathcal{P}^{\mathcal{R},\mathcal{O}}$ (where $\mathcal{R}$ is the sole source of randomness for $\mathcal{P}$ and $\mathcal{O}$ is the random oracle) acting as prover and having a probability of ϵ of convincing the verifier, there exists an extractor $\mathcal{E}$ that extracts the witness (k, p) in a mean runtime of $\frac{2}{\epsilon - \frac{1}{q}}$ (where each call to $\mathcal{P}$ is considered one step) such that -

For any PPTA $\mathcal{P}$ as described above, if -

$$Pr\left[\begin{array}{l|l} b_1 = 1 & \begin{array}{l}(M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), \sigma) \\ \qquad\qquad \leftarrow \mathcal{A}_1; \\ \\ \mathcal{R} \leftarrow \{0,1\}^\theta; \\ \forall_{j\in\{1..m\}}(c_j, d_j) \\ \qquad \leftarrow \mathcal{P}^{\mathcal{R},H_q}(\sigma, M, \\ \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})) \\ \\ b_1 \leftarrow \mathbf{V}(M, \\ \qquad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), \\ \qquad \forall_{j\in\{1..m\}}(c_j, d_j)); \end{array} \end{array}\right] = \epsilon$$

then there exists an extractor $\mathcal{E}$ that runs in expected $\frac{2}{\epsilon - \frac{1}{q}}$ steps such that,

$$Pr\left[\begin{array}{l|l} b_2 = 1 & \begin{array}{l}(M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), \sigma) \\ \qquad\qquad \leftarrow \mathcal{A}_1; \\ (k, p) \leftarrow \mathcal{E}(\mathcal{P}, \mathcal{R}, \sigma, M, \\ \qquad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})); \\ \\ b_2 \leftarrow \bigwedge_{i\in\{1..n\}}(Y_{ik} = pX_{ij}); \end{array} \end{array}\right] = 1$$

Theorem 4 *The knowledge-soundness for ZkPLMT is true*

Proof First, we define a custom stateful oracle simulator $\mathcal{O}$ -

$\mathcal{O}(Q)$:
  $value\_map \leftarrow empty\_map()$
  $oracle(input)$:
    if ($value\_map[input]$ *exists*)
      return $value\_map[input]$;
    else [
      $value\_map[input] \leftarrow \mathbb{F}_q$
      return $value\_map[input]$;
    ]
  return ($oracle$);

Note that even though $\mathcal{P}$ can invoke the oracle multiple times and base its logic on the output of the oracle, it cannot possibly gain any advantage in reality by using the oracle output to take different code paths or use it as a source of new randomness over using $\mathcal{R}$ as the oracle output is also random. So, we only consider $\mathcal{P}$ that uses multiple queries only to try multiple times. Now the extractor $\mathcal{E}'$ works in the following manner -

$$\mathcal{E}'(\mathcal{P}, \sigma, M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})) :$$

$$
\begin{array}{l}
\textit{loop}[\\
\quad \mathcal{R} \leftarrow \{0,1\}^{\theta} \\
\quad (\forall_{j\in\{1..m\}}(c_j, d_j)) \leftarrow \mathcal{P}^{\mathcal{R},\mathcal{H}_q} \\
\qquad (\sigma, M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})); \\
\\
\quad b_1 \leftarrow \mathbf{V}(M, \\
\qquad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), \\
\qquad \forall_{j\in\{1..m\}}(c_j, d_j)); \\
\\
\quad \textit{if } (b_1)[\textit{break};] \\
] \\
\textit{loop}[\\
\quad \mathcal{O} \leftarrow \ \textit{new } \mathcal{O}(Q); \\
\quad (\forall_{j\in\{1..m\}}(c'_j, d'_j)) \leftarrow \mathcal{P}^{\mathcal{R},\mathcal{O}} \\
\qquad (\sigma, M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})); \\
\\
\quad b_1 \leftarrow \mathbf{V}(M, \\
\qquad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}), \\
\qquad \forall_{j\in\{1..m\}}(c'_j, d'_j)); \\
\\
\quad \textit{if } (b_1)[\textit{break};] \\
] \\
\forall j \in \{1..m\}(c'_j, d'_j))[\\
\quad \textit{if } (c'_j \neq c_j)[\\
\qquad \textit{return } \left(j, \dfrac{d_j - d'_j}{c'_j - c_j}\right); \\
\quad] \\
] \\
\textit{return } \perp;
\end{array}
$$

The extractor $\mathcal{E}$ is defined as follows -

$$
\begin{array}{l}
\mathcal{E}(\mathcal{P}, \sigma, M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})) : \\
\quad \textit{loop}[\\
\qquad ret \leftarrow \mathcal{E}'(\mathcal{P}, \sigma, M, \\
\qquad\quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij})); \\
\qquad \textit{if } (ret \neq \perp)[\\
\qquad\quad (k, p) \leftarrow ret; \\
\qquad\quad \textit{return } (k, p, pG) \\
\qquad] \\
\quad]
\end{array}
$$

For each invocation of $\mathcal{E}'$, the adversary is invoked in the second loop with the same randomness that was successful in the first loop. In the second call, the oracle simulator returns a random value at a random

call to the oracle. Since, the input to the oracle is $M, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(X_{ij}, Y_{ij}, L_{ij})$, all values of L_{ij} must have been computed beforehand. But, since the random oracle returned a different value, if the proof has to work, the values of c'_k, d'_k must be different from c_k, d_k for some k, since the sums $\sum_{j=1}^{m} d_j \neq \sum_{j=1}^{m} d'_j$. Since L_{ik} is unchanged for every i, it must be that for every i, $c_k X_{ik} + d_k Y_{ik} = c'_k X_{ik} + d'_k Y_{ik}$. Since the group has a prime order, there is some w_i such that $Y_{ik} = w_i X_{ik}$. Hence, $c_k X_{ik} + d_k Y_{ik} = c'_k X_{ik} + d'_k Y_{ik}$ implies $c_k X_{ik} + d_k w_i X_{ik} = c'_k X_{ik} + d'_k w_i X_{ik}$. Since the group order is q and the values are chosen from $\mathbb{F}_q$, we can simplify this to $c_k + d_k w_i = c'_k + d'_k w_i$, or, $w_i = \frac{d_k - d'_k}{c'_k - c_k}$. Notice that the value of w_i does not depend on i and hence we can name it p. So, for every i, $Y_{ik} = pX_{ik}$ as desired.

The probability of success in each call in the first loop in $\mathcal{E}'$ is ϵ. Hence, the expected runtime for the first loop is $1/\epsilon$. The probability of choosing a particular $\mathcal{R}$ at the end of this loop is $Pr(\mathcal{R}|success)$. Given any such $\mathcal{R}$, the probability of success in each step of the second loop is $Pr(success|\mathcal{R})$. Hence, the expected runtime for the second loop is $\sum_{\mathcal{R}\in 1^\theta} \frac{Pr(\mathcal{R}|success)}{Pr(success|\mathcal{R})}$.

From Bays' theorem, we have $Pr(success) \times Pr(\mathcal{R}|success) = Pr(\mathcal{R}) \times Pr(success|\mathcal{R})$. We also have $Pr(\mathcal{R}) = 2^{-\theta}$ and $Pr(success) = \epsilon$. Hence, we have $\epsilon \times Pr(\mathcal{R}|success) = 2^{-\theta} \times Pr(success|\mathcal{R})$. Hence, the expected runtime of the second loop is $\sum_{\mathcal{R}\in 1^\theta} \frac{Pr(\mathcal{R}|success)}{Pr(success|\mathcal{R})} = \sum_{\mathcal{R}\in 1^\theta} \frac{2^{-\theta} Pr(success|\mathcal{R})}{\epsilon Pr(success|\mathcal{R})}$.

Now, we must compute the probability of the values of c and c' being different after the second loop. Let N be the number of oracle values for which the algorithm $\mathcal{P}$ is successful for a given $\mathcal{R}$ that got chosen in the first loop. The number of oracle values different from the first loop is $N-1$. Hence, the probability of the value being different is $\frac{N-1}{N}$. But, $Pr(success|\mathcal{R}) = N/q$. So, $N = qPr(success|\mathcal{R})$. Hence, $\frac{N-1}{N} = (1 - \frac{1}{N}) = (1 - \frac{1}{qPr(success|\mathcal{R})})$. Hence, the overall probability of having a different value of c' and c at the end of the second loop is -

$$
\begin{aligned}
&\sum_{\mathcal{R}\in 1^\theta} Pr(\mathcal{R}|success) \left(1 - \frac{1}{qPr(success|\mathcal{R})}\right) \\
=&\sum_{\mathcal{R}\in 1^\theta} \frac{2^{-\theta} Pr(success|\mathcal{R})}{\epsilon} \left(1 - \frac{1}{qPr(success|\mathcal{R})}\right) \\
=&\sum_{\mathcal{R}\in 1^\theta} \frac{2^{-\theta} Pr(success|\mathcal{R})}{\epsilon} \\
&\qquad - \sum_{\mathcal{R}\in 1^\theta} \frac{2^{-\theta} Pr(success|\mathcal{R})}{\epsilon} \frac{1}{qPr(success|\mathcal{R})} \\
=&(1 - \frac{1}{\epsilon q})
\end{aligned}
$$

Hence, on an average, the entire loop in $\mathcal{E}$ must be run $\frac{1}{1-\frac{1}{\epsilon q}}$ times, each of which is $2/\epsilon$ steps long. Therefore, the expected runtime of $\mathcal{E}$ is $(2/\epsilon)\frac{1}{1-\frac{1}{\epsilon q}} = \frac{2}{\epsilon - \frac{1}{q}}$.

Theorem 5 *The knowledge soundness property of the Transfer transaction is true.*

The transaction has four ZkPLMT proofs. If the probability of successful verification of the transaction is ϵ, the probability of successfully passing each ZkPLMT verification must be at least ϵ. Hence, for each of them, there must be an extractor that extracts the values (k, p) in an expected runtime of at most $\frac{2}{\epsilon - \frac{1}{q}}$. So, with the expected runtime of $\frac{8}{\epsilon - \frac{1}{q}}$, an extractor must be able to extract (k, p, z, x, y) such that the following are true.

Given the structure of the transaction tr as follows -

$$
\begin{aligned}
(&\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}) \\
&\quad \forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}), \\
&\quad \forall_{t\in\{1..u\}}[(\bar{A}_t, \bar{B}_t, \bar{V}_t), Z, \pi_b \\
&\quad Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{E}'_t, \bar{B}'_t, \bar{Q}'_t), \\
&\quad \alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}} I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)
\end{aligned}
$$

- $\forall e \in \{1..w\}[Q_e = zP_e]$
- $Z = zG$
- $\forall i \in \{1..n\}B_{ik} = pA_{ik}$.
- $\forall i \in \{1..n\}I_i = pH_{ik}$.
- $\forall i \in \{1..n\}T_k = pS_k$.
- $\forall i \in \{1..n\}\bar{T} = p\bar{S}$.
- $P' = pY$.
- $\sum_{i=1}^{n} V_{ik} - \sum_{t=1}^{u} \bar{V}_t = pZ$
- $\forall i \in \{1..n\}B_{ik} = pA_{ik}$
- $Y = yG$.
- $\forall t \in \{1..u\}\bar{E}'_t = y\bar{E}_t$
- $\forall t \in \{1..u\}\bar{B}'_t = y\bar{B}_t$
- $\forall t \in \{1..u\}\bar{X}_t = x\bar{A}_t$
- $\forall t \in \{1..u\}\bar{Q}'_t = x\bar{B}'_t$

The extractor returns the following values -

- sk is p.
- pk is pG.
- k is k.
- For each $i \in \{1..n\}$, apd_i is p.
- $\overline{apd}$ is pz.

The extractor computes (p, k, z, x, y) from the ZkPLMT extractors and computes $P \leftarrow pG$. p is also used as apd.

Now we look at each of the properties -

- **Key Match** P was computed from $P \leftarrow pG$.
- **Ownership** We already have $\forall i \in \{1..n\}B_{ik} = pA_{ik}$.
- **Nullifier** We already have $\forall i \in \{1..n\}I_i = pH_{ik}$.
- **Value conservation** We have $\sum_{i=1}^{n} V_{ik} - \sum_{t=1}^{u} \bar{V}_t = pZ$. $pZ = pzG$ is a commitment to zero. The apd is pz
- **Exclusion from the banned list** We have $\forall e \in \{1..w\}[Q_e = zP_e]$ and $\sum_{i=1}^{n} V_{ik} - \sum_{t=1}^{u} \bar{V}_t = pZ = zP$. Since none of the differences of the commitments equal any of the Q_e values, $\forall e \in \{1..w\}[P_e \neq P]$.
- **Decryption of the signer public key** We have $\bar{E}'_t = \bar{Q}'_t + P'$. We also have $\bar{Q}'_t = x\bar{B}'_t, \bar{B}'_t = y\bar{B}_t$, which implies $\bar{Q}'_t = xy\bar{B}_t$. We also have $\bar{E}'_t = y\bar{E}_t, P' = pY = pyG = ypG = yP$. So, we have $y\bar{E}_t = xy\bar{B}_t + yP$, i.e. $\bar{E}_t = x\bar{B}_t + P$. Now, $x\bar{B}_t = x\bar{p}_t\bar{A}_t = \bar{p}_t x\bar{A}_t = \bar{p}_t\bar{X}_t$. So, $P = \bar{E}_t - \bar{p}_t\bar{X}_t$.
- **Membership** We have $T_k = pS_k$ and $\bar{T} = p\bar{S}$

B.10.3 Confidentiality property for the Transfer transaction

Definition 3 DDH-1 problem - For some generator G and some scalars $x, \forall_{t\in\{1..u\}}(b_t)$, given the values of $G, xG, \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tyG, (b_t/a_t)G, b_txyz), yG$, and values $\forall_{t\in\{1..u\}}(R_t)$ which is either a random value or b_txG with equal probability, distinguish between the case when it is random and when it is not.

The advantage Adv^{DDH-1} for any algorithm solving this problem for an algorithm $\mathcal{A}$ is defined as follows - $Adv^{DDH-1}(\mathcal{A}) =$

$$\left| Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t,b_t) \leftarrow \mathbb{F}_q^{2t}; \\ \rho = \\ \quad \mathcal{A}(G, xG, yG, \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, b_tyG, \\ \quad (b_t/a_t)G, (b_t/a_t)xG, b_txyG, b_txG)); \end{array} \right] \right.$$
$$\left. - Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t,b_t) \leftarrow \mathbb{F}_q^{2t}; \\ \forall_{t\in\{1..u\}}R_t \leftarrow \mathbb{G}^t; \\ \rho = \\ \quad \mathcal{A}(G, xG, yG, \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, b_tyG, \\ \quad (b_t/a_t)G, (b_t/a_t)xG, b_txyG, R_t)); \end{array} \right] \right|$$

Definition 4 DDH-2 problem - For some generator G and some scalars $x, \forall_{t\in\{1..u\}}(a_t, b_t)$, given the values of $G, \forall_{t\in\{1..u\}}(a_tG, a_txG, (b_t/a_t)G, b_tyG), xG, yG$, and a value $\forall_{t\in\{1..u\}}(R_t)$ which is either a random value or b_txG with equal probability, distinguish between the case when it is random and when it is not.

The advantage Adv^{DDH-2} for any algorithm solving this problem for an algorithm $\mathcal{B}$ is defined as follows - $Adv^{DDH-2}(\mathcal{B}) =$

$$\left| Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t,b_t) \leftarrow \mathbb{F}_q^{2t}; \\ \rho = \\ \quad \mathcal{B}(G, xG, yG, \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, \\ \quad (b_t/a_t)G, (b_t/a_t)xG, b_tyG, b_txG)); \end{array} \right] \right.$$
$$\left. - Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t,b_t) \leftarrow \mathbb{F}_q^{2t}; \\ \forall_{t\in\{1..u\}}R_t \leftarrow \mathbb{G}^t \\ \rho = \\ \quad \mathcal{B}(G, xG, yG, \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, \\ \quad (b_t/a_t)G, (b_t/a_t)xG, b_tyG, R_t)); \end{array} \right] \right|$$

Lemma 4 *If $\mathcal{A}$ and $\mathcal{B}$ are the most advantageous algorithms, we can construct an DDH adversary $\mathcal{C}$ such that $Adv^{DDH-1}(\mathcal{A}) \leq 2Adv^{DDH}(\mathcal{C}) + Adv^{DDH-2}(\mathcal{B})$*

Proof We start with the DDH problem of G, yG, xG, S where we have to determine weather $S = xyG$. Using the algorithms $\mathcal{A}$ and $\mathcal{A}$, we develop an algorithm $\mathcal{A}$.

$\mathcal{C}(G, yG, xG, S)$:

$\forall_{t\in\{1..u\}}(a_t, b_t) \leftarrow \mathbb{F}_q^{2t}$;
$(\rho_1) \leftarrow \{0,1\}^2$;
if (ρ_1)[
$\forall_{t\in\{1..u\}} R_t \leftarrow \mathbb{G}^t$;

]
else[
$\forall t \in \{1..u\}[R_t \leftarrow b_t(xG)]$
]
return $\neg\rho_1 \oplus \mathcal{A}(G, xG, yG$
$\forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, (b_t/a_t)G, (b_t/a_t)xG, b_tyG,$
$b_tS, R_t))$;]

Let the following table be the probabilities of getting an output of 1 in different scenarios.

$S = xyG$	$[\rho_2]R_t = b_tyG$	$\mathcal{A}$
0	0	p_{A00}
0	1	p_{A01}
1	0	p_{A10}
1	1	p_{A11}

Now, the probability of output 1 for $\mathcal{C}$ when S is a random value is $\frac{1}{2}(p_{A01} + (1 - p_{A00}))$. The probability of output 1 for $\mathcal{C}$ when $S = xyG$ is $\frac{1}{2}(p_{A11} + (1 - p_{A10}))$. Hence, the advantage of $\mathcal{C}$ is the absolute value of $\frac{1}{2}(1 + (p_{A01} - p_{A00})) - \frac{1}{2}(1 + (p_{A11} - p_{A10})) = \frac{1}{2}((p_{A01} - p_{A00}) - (p_{A11} - p_{A10}))$.

From the triangle inequality, we have -

$$\begin{aligned}
&|p_{A01} - p_{A00}| \\
=&|((p_{A01} - p_{A00}) - (p_{A11} - p_{A10})) + (p_{A11} - p_{A10})| \\
\leq&|((p_{A01} - p_{A00}) - (p_{A11} - p_{A10}))| + |(p_{A11} - p_{A10}))|
\end{aligned}$$

Note that when R_t values are random, the problem of discriminating on S reduces to the DDH-2 problem. Now, since $\mathcal{B}$ is the most advantageous algorithm, $|(p_{A11} - p_{A10}))| \leq Adv^{DDH-2}(\mathcal{B})$ That is, $Adv^{DDH-1}(\mathcal{A}) \leq 2Adv^{DDH}(\mathcal{C}) + Adv^{DDH-2}B$.

Lemma 5 *For any algorithm $\mathcal{B}$, we a construct an algorithm $\mathcal{C}$ such that $Adv^{DDH-2}(\mathcal{B}) = Adv^{DDH}(\mathcal{C})$*

PROOF We construct $\mathcal{C}$ from $\mathcal{B}$ in the following manner -

$\mathcal{C}(G, xG, cG, D)$:
$y \leftarrow \mathbb{F}_q$;
$\forall t \in \{1..u\}$[
$(b_tG, R_t) \leftarrow$
$createAnotherDHTuple(G, xG, cG, D)$;
]
$\forall_{t\in\{1..u\}} a_t \leftarrow \mathbb{F}_q$
return $\mathcal{B}(G, xG, yG,$
$\forall_{t\in\{1..u\}}(a_tG, a_t(xG), b_tG, a_t^{-1}(b_tG),$
$(b_t/a_t)xG, b_tyG, R_t))$;

By lemma 1, if (G, xG, cG, D) is a DDH tuple, then, $R_t = b_txG$, otherwise, it is a random point. Hence, $Adv^{DDH-2}(\mathcal{B}) = Adv^{DDH}(\mathcal{C})$

Definition 5 DDH-3 problem - For some generator G and some scalars $x, \forall_{t\in\{1..u\}}(a_t, b_t, z)$, given the values of $G, \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tyG, b_txyz), yG, zG, \forall_{e\in\{1..w\}}(p_eG)$, and a values $\forall_{t\in\{1..u\}}(R_t)$ and $\forall_{e\in\{1..w\}}(Q_e)$ which are either a random values or $\forall_{t\in\{1..u\}}(b_txG)$ and $\forall_{e\in\{1..w\}}(zp_eG)$ with equal probability, distinguish between the case when it is random and when it is not.

The advantage $Adv^{DDH-3}(\mathcal{D})$ is defined as follows -
$Adv^{DDH-3}(\mathcal{D}) =$

$$\left| Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t, b_t) \leftarrow \mathbb{F}_q^{2t}; \\ z \leftarrow \mathbb{F}_q; \\ \forall_{e\in\{1..w\}} p_e \leftarrow \mathbb{F}_q; \\ \rho = \\ \quad \mathcal{D}(G, yG, zG \\ \quad \forall_{e\in\{1..w\}}(p_eG, zp_eG) \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, b_tyG, \\ \quad b_txyG, b_txG)); \end{array}\right]\right.$$

$$\left. -Pr\left[\rho = 1 \;\middle|\; \begin{array}{l} G \leftarrow \mathbb{G}; \\ (x,y) \leftarrow \mathbb{F}_q^2; \\ \forall_{t\in\{1..u\}}(a_t, b_t) \leftarrow \mathbb{F}_q^{2t}; \\ \forall_{t\in\{1..u\}} R_t \leftarrow \mathbb{G}^t; \\ \forall_{e\in\{1..w\}}(P_e, Q_e) \leftarrow \mathbb{G}^{2t}; \rho = \\ \quad \mathcal{D}(G, yG, \\ \quad \forall_{e\in\{1..w\}}(P_e, Q_e) \\ \quad \forall_{t\in\{1..u\}}(a_tG, a_txG, b_tG, b_tyG, \\ \quad b_txyG, R_t)); \end{array}\right]\right|$$

Lemma 6 *For algorithms $\mathcal{D}$ and $\mathcal{A}$ with maximum advantages, we can construct a DDH solver $\mathcal{C}$ such that $Adv^{DDH-3}(\mathcal{D}) \leq 2Adv^{DDH}(\mathcal{C}) + Adv^{DDH-1}(\mathcal{A})$*

Proof It can be proved in a very similar manner to 4.

Theorem 6 *The Transfer Transaction is confidential under the DDH assumption.*

Proof We construct the submodule Sim' in the following manner -

$$\begin{array}{l} Sim'(k, P, Z, \mathbf{Z}, Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{B}'_t, \bar{Q}'_t), \\ \quad \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}), \\ \quad \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{C}_t), \forall_{j\in\{1..m\}}(S_j, T_j), \\ \quad \forall_{e\in\{1..w\}}(P_e, Q_e)) : \end{array}$$

Compute $\bar{E}'$

$\forall_{t\in\{1..u\}}[\bar{E}'_t \leftarrow \bar{Q}'_t + P';]$

Output ID tags

$(\bar{S}, \bar{T}) = createAnotherDHTuple(Y, P', Z, \mathbf{Z})$

Banned List and Summation Check

$\mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))$

$\alpha \leftarrow CreateSimulatedZkPLMT(\phi, \mathbf{X}');$

Nullifiers

$\forall_{i\in\{1..n\}}[H_{ik} \leftarrow r''_iG]$

$\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}\setminus\{k\}}[H_{ij} \leftarrow H_g(A_{ij}, B_{ij});]$
$\forall i \in \{1..n\}[r''_i \leftarrow \mathbb{F}_q;]$
$\forall i \in \{1..n\}[I_i \leftarrow r''_i P;]$

BulletProof

$\pi_b \leftarrow CreateSimulatedBulletproof;$

Verifiable encryption

$\pi_1 \leftarrow CreateSimulatedZkPLMT(\phi,$
$((G, Y), \forall_{t\in\{1..u\}}(\bar{E}_t, \bar{E}'_t),$
$\forall_{t\in\{1..u\}}(\bar{B}_t, \bar{B}'_t)));$
$\pi_2 \leftarrow CreateSimulatedZkPLMT(\phi,$
$(\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{X}'_t), \forall_{t\in\{1..u\}}(\bar{B}'_t, \bar{Q}'_t)));$

Main signature

$\forall j \in \{1..m\}[\mathbf{Y}_j = (\forall_{i\in\{1..n\}}((A_{ij}, B_{ij}),$
$\forall_{i\in\{1..n\}}(H_{ij}, I_i),$
$(Z, \sum_{i=1}^{n} V_{ij} - \sum_{t=1}^{u} \bar{V}_t),$
$(S_j, T_j), (\bar{S}, \bar{T}), (Y, P'))];$

M is the content of the transaction in binary

$\beta \leftarrow CreateSimulatedZkPLMT$
$(M, \forall_{j\in\{1..m\}}\mathbf{Y}_j, k, p);$
$\mathcal{O}_{g2}, \mathcal{O}_{q2} \leftarrow CreateOracleSimulators();$

Return transaction

$return(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij})$
$\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),$
$\forall_{t\in\{1..u\}}[(\bar{A}_t, \bar{B}_t, \bar{V}_t), Z, \pi_b$
$Y, P', Z, \mathbf{Z}, Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{B}'_t, \bar{Q}'_t),$
$\forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{E}'_t, \bar{B}'_t, \bar{Q}'_t),$
$\alpha, \beta, \pi_1, \pi_2, \forall_{i\in\{1..n\}}I_i, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e));$

Now, we define the simulator Sim as follows -

$Sim(\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),$
$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}(P_e)) :$

$k \leftarrow \{1..m\}$
$z \leftarrow \mathbb{F}_q;$
$Z \leftarrow zG$
$P \leftarrow \mathbb{G};$
$\mathbb{Z} \leftarrow zP$
$y \leftarrow \mathbb{F}_q$
$x \leftarrow \mathbb{F}_q$
$Y \leftarrow yG;$
$P' \leftarrow \mathbb{G};$
$\forall e \in \{1..w\}[Q_e \leftarrow zP_e]$
$\forall_{t\in\{1..u\}}[\bar{X}_t \leftarrow x\bar{A}_t]$

$\forall_{t\in\{1..u\}}[\bar{E}_t \leftarrow \mathbb{G}]$
$\forall_{e\in\{1..w\}}[\bar{Q}_e \leftarrow \mathbb{G}]$
$\forall_{t\in\{1..u\}}[\bar{B}'_t \leftarrow y\bar{B}_t;]$
$\forall_{t\in\{1..u\}}[\bar{Q}'_t \leftarrow y\bar{\mathbf{X}}_t);]$
$\bar{E}_t \leftarrow \bar{\mathbf{X}}_t) + P$
$Z, \mathbf{Z}, Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{B}'_t, \bar{Q}'_t)$
$return\ Sim'(k, P, \forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),$
$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t),$
$\forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}(P_e, Q_e)) :$

The implementations of $CreateSimulatedBulletproof$ and $CreateOracleSimulators$ are obvious. It is also quite simple to simulate the oracles, so we skip over that.

We construct the following algorithm to solve the DDH-1 problem using the algorithm $\mathcal{A}_1$ that distinguishes between real and a simulated transaction -

$DDH - 1 - Solver(G, xG, yG$
$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{X}_t, \bar{B}_t, \bar{B}'_t, \bar{P}_t,$
$x\bar{P}_t, \bar{Q}'_t, R_t)) :$
$k \leftarrow \{1..m\}$
$((P, P'), (S_k, T_k), (Z, \mathbf{Z})\forall_{i\in\{1..n\}}(A_{ik}, B_{ik}))$
$\leftarrow createNDifferentDHTuple$
$(G, Y, R_1, \bar{Q}'_1, 1 + n);$
$\bar{E}_t \leftarrow \bar{\mathbf{X}}_t + P$
$\forall_{i\in\{1..n\}}(v_{ik}) \leftarrow ValueRange^n;$
$(\forall_{t\in\{1..u\}}(\bar{v}_t),$
$state)$
$\leftarrow \mathcal{A}_1(\forall_{i\in\{1..n\}}(v_{ik}));$
$\forall e \in \{1..w\}[p_e \leftarrow \mathbb{F}_q]$
$\forall e \in \{1..w\}[P_e \leftarrow p_e G]$
$\forall e \in \{1..w\}[Q_e \leftarrow p_e Z]$
$\forall_{i\in\{1..n\}} V_{ik} \leftarrow \mathbb{G}$
$\forall_{j\in\{1..m\}\setminus\{k\}}\forall_{i\in\{1..n\}}(A_{ij}, B_{ij}, V_{ij}, S_j, T_j) \leftarrow \mathbb{G}^{5mn};$
$\forall_{t\in\{1..u-1\}}(\bar{V}_t) \leftarrow \mathbb{G}^{u-1};$
$V_u \leftarrow \sum_{i=1}^{n} V_{ik} - \sum_{t=1}^{u-1} \bar{V}_t - \mathbf{Z};$
$tr \leftarrow Sim'(k, P, Z, \mathbf{Z}, Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{B}'_t, \bar{Q}'_t),$
$\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),$
$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}(P_e, Q_e)));$
$return\ \mathcal{A}_2^{O_g, O_q}(state, k, tr, P,$
$\forall_{i\in\{1..n\}}\forall_{j\in\{1..m\}}(A_{ij}, B_{ij}, V_{ij}),$
$\forall_{t\in\{1..u\}}(\bar{P}_t, \bar{v}_t), \forall_{j\in\{1..m\}}(S_j, T_j),$
$\forall_{e\in\{1..w\}} p_e G, \forall_{e\in\{1..w\}} p_e);$

It is straightforward to verify that when the input to this function is a genuine DDH-1 tuple, the transaction tr matches the probability distribution of a genuine transaction and when the input has random inputs for R_t, the probability distribution matches exactly as the simulated transaction. Hence, this function can solve the DDH-1 problem with the same advantage as the distinguisher.

Theorem 7 *Knowledge-soundness of the Create transaction is true*

PROOF The transaction has four ZkPLMT proofs. If the probability of successful verification of the transaction is ϵ, the probability of successfully passing each ZkPLMT verification must be at least ϵ. Hence, for each of them, there must be an extractor that extracts the values (k, p) in an expected runtime of at most $\frac{2}{1-\frac{1}{q}}$. So, with the expected runtime of $\frac{8}{1-\frac{1}{q}}$, an extractor must be able to extract (k, p, z, x, y) such that the following are true.

Given the structure of the transaction ctr as follows -

$$\begin{aligned}&\forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\ &\forall_{t\in\{1..u\}}[(\bar{V}_t, \bar{r}_t), (A, B), v_0, \pi_b\\ &Y, P', (\bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}'), \hat{T}\\ &\alpha, \beta, \pi_1, \pi_2, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e)) :\end{aligned}$$

- $\forall e \in \{1..w\}[Q_e = zP_e]$
- $Z = zG$
- $B = pA$.
- $\forall i \in \{1..n\} T_k = pS_k$.
- $\forall i \in \{1..n\} \bar{T} = p\bar{S}$.
- $P' = pY$.
- $\sum_{i=1}^{n} V_{ik} = \bar{r}G + \bar{v}L$
- $Y = yG$.
- $\forall t \in \{1..u\} \bar{E}' = y\bar{E}$
- $\forall t \in \{1..u\} \bar{B}' = y \cap T$
- $\forall t \in \{1..u\} \bar{X} = xG$
- $\forall t \in \{1..u\} \bar{Q}' = x\bar{B}'$

The extractor computes (p, k, z, x, y) from the ZkPLMT extractors and computes $P \leftarrow pG$. p is also used as apd.

Now we look at each of the properties -

- **Key Match** P was computed from $P \leftarrow pG$.
- **Ownership** We already have $B = pA$.
- **Value conservation** The value of $CreateRand$ is returned as $\bar{r}$ and $CreateValue$ is $\bar{v}$. It can then be clearly seen that the $CheckTxOValue$ returns true.
- **Exclusion from the banned list** We have $\forall e \in \{1..w\}[Q_e = zP_e]$ and $\sum_{i=1}^{n} V_{ik} - \sum_{t=1}^{u} \bar{V}_t = pZ = zP$. Since none of the differences of the commitments equal any of the Q_e values, $\forall e \in \{1..w\}[P_e \neq P]$.

- **Decryption of the signer public key** We assume that the private key of the trust is $\hat{p}$, so that $\hat{T} = \hat{p}G$. We have $\bar{E}' = \bar{Q}' + P'$. We also have $\bar{Q}' = x\bar{B}', \bar{B}' = y\bar{B}$, which implies $\bar{Q}' = xy\bar{B}$, and $\bar{E}' = y\bar{E}_t, P' = pY = pyG = ypG = yP$. So, we have $y\bar{E} = xy\hat{T} + yP$, i.e. $\bar{E} = x\hat{T} + P$. Now, $x\hat{T} = x\hat{p}G = \hat{p}xG = \hat{p}\bar{X}$. So, $P = \bar{E} - \hat{p}\bar{X}$.
- **Membership** We have $T_k = pS_k$ and $\bar{T} = p\bar{S}$

Theorem 8 *The Confidentiality property of the creation transaction is true.*

Proof We construct the submodule Sim' in the following manner -

$$
\begin{aligned}
&Sim'(k, P, \hat{T}, Z, \mathbf{Z}, Y, P', \bar{X}, \bar{E}, \bar{B}', \bar{Q}'),\\
&\quad \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \bar{r}, \forall_{j\in\{1..m\}}(S_j, T_j),\\
&\quad \forall_{e\in\{1..w\}}(P_e, Q_e)) :\\
&\\
&\quad \textit{Compute } \bar{E}'\\
&\quad \bar{E}' \leftarrow \bar{Q}' + P';\\
&\\
&\quad \textit{Output ID tags}\\
&\qquad (\bar{S}, \bar{T}) = createAnotherDHTuple(Y, P', Z, \mathbf{Z})\\
&\quad \textit{Banned List and Summation Check}\\
&\qquad \mathbf{X}' \leftarrow ((G, Z), \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e))\\
&\qquad \alpha \leftarrow CreateSimulatedZkPLMT(\phi, \mathbf{X}');\\
&\quad \textit{BulletProof}\\
&\qquad \pi_b \leftarrow CreateSimulatedBulletproof;\\
&\quad \textit{Verifiable encryption}\\
&\qquad \pi_1 \leftarrow CreateSimulatedZkPLMT(\phi,\\
&\qquad\quad ((G, Y), (\bar{E}, \bar{E}'),\\
&\qquad\quad (\bar{B}, \bar{B}')));\\
&\qquad \pi_2 \leftarrow CreateSimulatedZkPLMT(\phi,\\
&\qquad\quad (\forall_{t\in\{1..u\}}(G, \bar{X}), \forall_{t\in\{1..u\}}(\bar{B}', \bar{Q}')));\\
&\quad \textit{Main signature}\\
&\qquad \forall j \in \{1..m\}[\mathbf{Y}_j = (\forall_{t\in\{1..u\}}((\bar{A}_t, \bar{B}_t),\\
&\qquad\quad (Z, \mathbf{Z}),\\
&\qquad\quad (S_j, T_j), (\bar{S}, \bar{T}), (Y, P'))];\\
&\qquad \textit{M is the content of the transaction in binary}\\
&\qquad \beta \leftarrow CreateSimulatedZkPLMT\\
&\qquad (M, \forall_{j\in\{1..m\}}\mathbf{Y}_j);\\
&\qquad \mathcal{O}_{g2}, \mathcal{O}_{q2} \leftarrow CreateOracleSimulators();\\
&\quad \textit{Return transaction}\\
&\qquad return(\\
&\qquad\quad \forall_{j\in\{1..m\}}(S_j, T_j), (\bar{S}, \bar{T}),\\
&\qquad\quad \bar{r}, \bar{v}, \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \pi_b\\
&\qquad\quad Z, \mathbf{Z}, Y, P', (\bar{X}, \bar{E}, \bar{E}', \bar{B}', \bar{Q}'), \hat{T}\\
&\qquad\quad \alpha, \beta, \pi_1, \pi_2, \forall_{e\in\{1..w\}}(\hat{P}_e, \hat{Q}_e));
\end{aligned}
$$

Now, we define the simulator Sim as follows -

$$Sim(\hat{T}, \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \bar{r}, \\ \forall_{j\in\{1..m\}}(S_j, T_j), \forall_{e\in\{1..w\}}(P_e)) :$$

$$k \leftarrow \{1..m\}$$
$$z \leftarrow \mathbb{F}_q;$$
$$Z \leftarrow zG$$
$$P \leftarrow \mathbb{G};$$
$$\mathbb{Z} \leftarrow zP$$
$$y \leftarrow \mathbb{F}_q$$
$$x \leftarrow \mathbb{F}_q$$
$$Y \leftarrow yG;$$
$$P' \leftarrow \mathbb{G};$$
$$\forall e \in \{1..w\}[Q_e \leftarrow zP_e]$$
$$\bar{X} \leftarrow xG;$$
$$\bar{E} \leftarrow \mathbb{G};$$
$$\bar{B}' \leftarrow y\bar{B};$$
$$\bar{Q}' \leftarrow y\bar{\mathbf{X}};$$
$$Z, \mathbf{Z}, Y, P', \forall_{t\in\{1..u\}}(\bar{X}_t, \bar{E}_t, \bar{B}'_t, \bar{Q}'_t)$$
$$return\ Sim'(k, P, \hat{T}, Z, \mathbf{Z}, Y, P', \bar{X}, \bar{E}, \bar{B}', \bar{Q}'),$$
$$\forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \bar{r}, \forall_{j\in\{1..m\}}(S_j, T_j),$$
$$\forall_{e\in\{1..w\}}(P_e, Q_e)) :$$

The implementations of $CreateSimulatedBulletproof$ and $CreateOracleSimulators$ are obvious. It is also quite simple to simulate the oracles, so we skip over that.

We construct the following algorithm to solve the DDH-1 problem using the algorithm $\mathcal{A}_1$ that distinguishes between real and a simulated transaction -

$$DDH-1-Solver(G, xG, yG$$
$$\forall_{t\in\{1..u\}}(\hat{A}_t, \hat{X}_t, \hat{B}_t, \hat{B}'_t, \hat{T}_t,$$
$$x\hat{T}_t, \hat{Q}'_t, R_t)) :$$
$$\bar{X} \leftarrow xG;$$
$$Y \leftarrow yG;$$
$$k \leftarrow \{1..m\}$$
$$(P, P', (S_k, T_k), (Z, \mathbf{Z}), \forall_{t\in\{1..u\}}(\bar{A}_t, \bar{B}_t)) \\ \leftarrow createNDifferentDHTuple \\ (G, Y, R_1, \bar{Q}'_1, 1+u);$$
$$\bar{E} \leftarrow x\hat{T}_1 + P;$$
$$\forall_{t\in\{1..u\}}(\bar{v}_t, state) \leftarrow \mathcal{A}_1;$$
$$\bar{v} \leftarrow \sum_{t=1}^{u} \bar{v}_t;$$
$$\forall e \in \{1..w\}[p_e \leftarrow \mathbb{F}_q]$$
$$\forall e \in \{1..w\}[P_e \leftarrow p_eG]$$

$$
\begin{aligned}
&\forall e \in \{1..w\}[Q_e \leftarrow p_e Z] \\
&\forall t \in \{1..u\}[r_t \leftarrow \mathbb{F}_q]; \\
&\bar{r} \leftarrow \sum_{t=1}^{u} r_t; \\
&\forall t \in \{1..u\}[\bar{V}_t \leftarrow r_t G + v_t L]; \\
&ctr \leftarrow Sim'(k, P, \hat{T}, Z, \mathbf{Z}, Y, P', \bar{X}, \bar{E}, \bar{B}', \bar{Q}'), \\
&\forall_{t \in \{1..u\}}(\bar{A}_t, \bar{B}_t, \bar{V}_t), \bar{v}, \bar{r}, \forall_{j \in \{1..m\}}(S_j, T_j), \\
&\forall_{e \in \{1..w\}}(P_e, Q_e)); \\
&return\ \mathcal{A}_2^{O_g, O_q}(state, k, ctr, P, \hat{T}_1, \\
&\qquad \forall_{t \in \{1..u\}} \bar{v}_t, \\
&\qquad \forall_{j \in \{1..m\}}(S_j, T_j), \\
&\qquad \forall_{e \in \{1..w\}} p_e G, \forall_{e \in \{1..w\}} p_e);
\end{aligned}
$$

It is straightforward to verify that when the input to this function is a genuine DDH-1 tuple, the transaction *ctr* matches the probability distribution of a genuine transaction and when the input has random inputs for R_t, the probability distribution matches exactly as the simulated transaction. Hence, this function can solve the DDH-1 problem with the same advantage as the distinguisher.

B.10.4 Confidentiality of Redeem Transaction

This can be proved very similarly to the confidentiality of the Transfer transaction.

Appendix C Supplementary Material

The following texts may be helpful to understand certain concepts

C.1 First order logic

https://formal.iti.kit.edu/~beckert/teaching/Einfuehrung-KI-WS0304/08FirstOrderLogic.pdf

C.2 Group Theory

First two chapters should work: https://pages.mtu.edu/~kreher/ABOUTME/syllabus/GTN.pdf

C.3 Elliptic Curve Cryptography

https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc
https://www.youtube.com/watch?v=muIv8I6v1aE

C.4 Zero Knowledge Proofs

https://crypto.ethz.ch/publications/files/Maurer09.pdf

Appendix D Glossary of Terms

Note: All generic cryptographic terms apply to Elliptic Curve Cryptography (ECC).

D.1 Conventions

1. All uppercase italicized letters: Ristretto points (with some noted exceptions).
2. All lowercase italicized letters: scalars (except e, h,etc. which are indices, indicated below) - also called slopes, are elements in finite fields (notated as $\mathbb{Z}_p^*$ or $\mathbb{F}_p^*$), or as values.
3. For all Ristretto points, scalars, and indices, see usage for more context.
4. For all Ristretto points, a subscript letter denotes indexing. E.g. Q_e is indexed by e.

- $\leftarrow$ - to sample a random value, to get a result from a computation
- $\forall$ - "for all", "for any" standard mathematical notation
- $\forall_j$ - sequence indexed by a variable
- $\in$ - set membership
- Σ - summation of elements
- $\cup$ - Union
- A - Ristretto point
- $ADec$ - authenticated decryption algorithm
- $AEnc$ - authenticated encryption algorithm
- α, β : OutputProof - zero-knowledge proof output
- a - bank account number in string representation
- B - Ristretto point
- b - bank routing number in string representation
- bp : BulletProof - proof that all output transaction outputs have a value less than a maximum allowed value
- C - Diffie-Hellman secret that is a Ristretto point
- c - scalar
- D - Ristretto point
- D - permanent decryption key of the trust
- D - decryption algorithm of an authenticated encryption scheme, see usage for context
- d - scalar
- e - index
- $\mathbb{F}_q^*$ - a (multiplicative) finite field of prime order q, in which the Discrete Logarithm Problem is believed to be hard
- $\mathbb{F}_q$ - an (additive) finite field of prime order q
- F - Ristretto point
- $\mathbb{G}$ - elliptic curve group of a given prime order (e.g. prime order q)

- G - system-wide fixed generator (base) point of $\mathbb{G}$
- H_b - hash function with output in the set $\{0,1\}^b$ for a fixed b
- H_g - hash function with output in the group
- H_q - hash function with output in $\mathbb{F}_q^*$
- H^j - result of a hash function over the public key P_j concatenating I
- H_{kl} - set of Ristretto points, indexed by k and l
- h - result of a hash function
- h - index
- I - key-image
- i - index
- j - index
- K - key space, see: Construction of Input for Send and Redeem
- k - index
- L - system-wide fixed generator (base) point of $\mathbb{G}$ that is obscured by the discrete logarithm of $\mathbb{G}$(see: Discrete Logarithm Problem). E.g. $\mathbb{G} = x * L$ for some scalar x
- l - index
- M - body of the transaction without proof, a private ZkPLMT message
- $\mathbf{M'}$ - body of the transaction without the transaction output and the proof, a private ZkPLMT message
- m - length of a vector, see usage for context (e.g. $m - 1$ is a subtraction operation)
- o - index
- ϕ - empty string of bits or an empty message
- π : InputProof - works as both a signature and a proof that the signer is a valid identified member
- $\mathbf{P}$ - prover algorithm for ZkPLMT
- P - Ristretto point that represents a public key
- ψ - transaction output
- $\mathcal{P}$ - Set (e.g. set of transactions)
- pG - permanent private key
- p - scalar value that represents a private key
- Q - Ristretto point
- r - scalar value that represents a random number sampled from the field
- $\mathbf{S}$ - ZkPLMT set of curve-point tuples containing at least one linear tuple
- $\mathcal{S}$ - Set (e.g. of transactions)
- S - Ristretto point that represents a random curve point

- T - Ristretto point
- t - index
- u - length of a vector, see usage for context, e.g.: length of a list of banned members' public keys
- $\mathbf{V}$ - verification algorithm for ZkPLMT
- V - Pedersen commitment over the transaction amount
- v - scalar
- $\mathbf{X}'$ - Linear tuple of Ristretto points
- X - Ristretto point
- x - scalar value that represents an encryption secret key (as in a symmetric encryption scheme)
- $\mathbf{Y}$ - linear tuple
- Z - Ristretto point
- $\mathbf{Z}$ - Ristretto point
- z - scalar
- ω - An unencrypted plaintext message
- ω' - An encrypted version of ω

D.2 Terms

- **Banned List** - list of members barred from making Create, Send or Redeem transactions on the Xand network
- **ChaCha20-Poly1305** - authenticated encryption scheme, ChaCha20 is the block cipher and Poly1305 is the authenticator (i.e., message authentication code algorithm), see: AEAD
- **commitment** - commitment to a value that is not being revealed - see: https://en.wikipedia.org/wiki/Commitment_scheme
- **CryptoNote** - reference implementation for an application layer protocol for Bitcoin that aims to solve problems around traceability and privacy, among others
- **curve vector** - curve point vector is a pair of curve points
- **Discrete Logarithm Problem (DLP)** - an algorithmic concept with applications in cryptography because it currently has no efficient solution. For our usage, see: Appendix: Cryptographic Models: DDH
- **field, Finite** - mathematical concepts for fields - see: https://en.wikipedia.org/wiki/Finite_field
- **key-image** - output of a hash function over a member's public key
- **key, Encryption Private** - secret key belonging to a member, used for authenticating encryption, see: Components, Zero-knowledge Proof of Linear Member Tuple (ZkPLMT)
- **key, Permanent** - any key that does not change - see: Confidential UTxO Model
- **key, Private** - secret key belonging to a member, used for creating a transaction proof, see: Activity Proofs, Create Transaction Proof

- **key, RecipientOneTimeKey** - address (represented as a temporary public key) that owns the transaction output, see: Components, One Time Address Generation and Discovery
- **key, RecipientPermanentKey** - address (represented as a public key) that is the permanent key for the redeem-only wallet, see: Transaction Data Model, Confidential UTxO Model
- **member** - a participant/party in a cryptographic scheme
- **Member** - a participating Member of the Xand network
- **membership tag** - a curve-point vector that is used to prove that the signer is a member in Xand
- **membership tag, true** - The membership tag belonging to the signer hidden within decoy membership tags
- **MLSAG - multilayered linkable spontaneous anonymous group signatures (Schnorr-style)** - signature scheme, see: https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf
- **N-age-bound random selection** - decoy selection method, used to randomize the transaction inputs
- **Pedersen commitment** - cryptographic primitive allowing a commitment to a private value that can later be revealed
- **proof** - in blockchain, a consensus mechanism to finalize blocks. In contrast to a "proof" in mathematics or logic, where a conventional proof of a statement is a sequence of elementary, easily verifiable steps, which, starting from the axioms (or previously proven facts), in the last step yields the statement to be proven
- **proof, Bulletproof** - a particular type of range proof, see: References for more details [2]
- **proof, non-interactive** - proof that does not require interaction between the prover and the verifier during the verification process
- **proof, range** - a proof that values exist in a range
- **salt** - one-time-use random value used as an input to a hashing function
- **signature, ring**- a signature that proves that requires that the signer knows at least the private key of one of the ring or set of public keys provided
- **slope** - slope of a curve-point vector is the discrete logarithm of the second point with respect to the first point
- **tuple, linear** - tuple of curve-point vectors with the same slope
- **tuple, linear member** - a linear tuple that is a member of a set of decoy tuples of the same structure that are not necessarily linear
- **tuple, non-linear** - tuple of curve-point vector that is not a linear tuple (with different slopes)
- **TxO** - "transaction output" - these are outputs from a transaction and represent ownership of claims on funds held in trust. They may be spent or unspent, but this status is known only by the owner of the transaction output.
- **UTxO** - "unspent transaction output", i.e., known output of some value from a previous transaction that has not yet been used as an input to another transaction. We rarely use this term because the spent status of transaction outputs are not generally known.
- **value** - the value of a TxO is the claim amount it represents
- **ZkPLMT** - non-interactive zero-knowledge proof, used to produce binding anonymous multiparty commitments, see: References, Zero Knowledge Proof

References

[1] N. Van Saberhagen, "Cryptonote v 2.0," 2013.

[2] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315–334.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[4] G. Maxwell and A. Poelstra, "Borromean ring signatures," *Accessed: Jun*, vol. 8, p. 2019, 2015.

[5] S. Noether, A. Mackenzie *et al.*, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.

[6] K. M. Alonso, "Monero - privacy in the blockchain," 2018.

[7] ——, "Zero to monero," 2020.

[8] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted edwards curves," in *International Conference on Cryptology in Africa*. Springer, 2008, pp. 389–405.

[9] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[10] G. Maxwell, "Confidential transactions," *URL: https://people. xiph. org/greg/confidential values. txt (Accessed 09/05/2016)*, 2015.

[11] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2007, pp. 29–50.

[12] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[13] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, 2016.

[14] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[15] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[16] T. Kerber, A. Kiayias, M. Kohlweiss, and V. Zikas, "Ouroboros crypsinous: Privacy-preserving proof-of-stake," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 157–174.

[17] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 239–252.

[18] U. Maurer, "Unifying zero-knowledge proofs of knowledge," in *International Conference on Cryptology in Africa*. Springer, 2009, pp. 272–286.

[19] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[20] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *Journal of cryptology*, vol. 1, no. 2, pp. 77–94, 1988.