Blind Transactions

A blind transaction is one that is recorded on the blockchain but the recipient and amount is not seen.

This is implemented by locking value in a singleton and for the singleton to record its own coin state. This state contains additions and removals that are local to the singleton, where the amounts are not known but a proof is provided that the amounts are correct. Spends of the internal coin set can be made causing removals and more additions. An exit spend can be done to remove value (in even amounts only) from the coin. At this point the recipient and amount are revealed, but none of the intermediate transactions are known.

Singleton State

Coin name: *HASH(<PARENT COIN><OWNER BLS PUBKEY><AMOUNT>)*

Additions Merkle Root

Leaf nodes in the merkle tree contain coin names that can be spent.

Removals Merkle Root

Leaf nodes in the merkle tree contain coin names that have been spent.

Singleton Spend Paths

Spend

This function accepts a ZKP to prove the following:

1) The coins being spent exist in the additions merkle tree
2) The coins being spent do not exist in the removals merkle tree
3) The prover provided a signature for the pubkey and amount for each removal
4) The amount of the new additions and exit coin sum up to the amount of removals
5) Exit coin amount is even or zero for no exit
6) The new additions do not have the same pubkey and amount
7) The new additions merkle root after the spend
8) The new removals merkle root after the spend
9) The number of additions does not exceed maximum

ZK Circuit Design

Public Inputs

additions_root: Additions merkle root
removals_root: Removals merkle root
new_removals_names: Coin names to remove (array of up to 32, 0 if not used)
exit_puzhash: Exit coin puzzle hash (0 if not used)
exit_amount: Exit coin amount (0 if not used)

Private Inputs

additions_names: Names of all additions in the current tree (array of up to 4096, 0 if not used)
removals_names: Names of all removals in the current tree (array of up to 4096, 0 if not used)
new_removals_parents: Parent names for removed coins (array of up to 32, 0 if not used)
new_removals_pubkeys: Pubkeys for removed coins (array of up to 32, 0 if not used)
new_removals_amounts: Amounts for removed coins (array of up to 32, 0 if not used)
new_removals_sigs: Signatures for removed coins (array of up to 32, 0 if not used)
new_additions_pubkeys: Pubkeys for created coins (array of up to 32, 0 if not used)
new_additions_amounts: Amounts for created coins (array of up to 32, 0 if not used)

Public Outputs

new_additions_root: New additions merkle root
new_removals_root: New removals merkle root
new_additions_names: Names of coins being added (array of up to 128, 0 if not used)

ZK Circuit Implementation

This is the logic of the circuit

1. Construct a additions_tree with additons_names
2. Ensure additions_tree makes a root matching additions_root
3. Construct removals_tree with removals_names
4. Ensure that removals_tree makes a root matching removals_root
5. Ensure inputs are valid
    a. Ensure additions arrays have same length (null terminated, only nulls after null)

    b. Ensure removals arrays have same length (null terminated, only nulls after null)
    c. Ensure at least 1 new_removals_names
    d. Ensure exit_amount is 0 or positive even
6. Ensure that sum of new_removals_amounts = new_additions_amounts + exit_amount
7. Iterate over new_removals_names
    a. Ensure name is correct
        i. Corresponding index in new_removals_parents, new_removals_pubkeys, new_removals_amounts
    b. Ensure signature is correct
        i. Corresponding index in new_removals_pubkeys, new_removals_sigs
        ii. Signature message is the pubkey
            1. Try to memoize signature verification to reduce proving time
            2. Not worried about reuse of signature because it is not shared
    c. Ensure name is in additions_tree
    d. Ensure name is not in removals_tree
    e. Add name to removals_tree
8. Set new_removals_root to root of removals_tree
9. Iterate over new_additions_pubkeys
    a. Calculate name
        i. Use first removal as parent
        ii. Corresponding index in new_additions_pubkeys, new_additions_amounts
    b. Ensure name is not already in additions_names (duplicate child pubkey/amount)
    c. Add name to additions_tree
    d. Add name to new_additions_names
10. Set new_additions_root to root of additions_tree