Account-Based Anonymous Rollup

(authors: Alexandre Belling, Olivier Bégassat, Nicolas Liochon)

This extends the rollup proposal (On-chain scaling to potentially ~500 tx/sec through mass tx validation) to support anonymous transactions. By anonymous we mean (1) participants in a transaction are unlinkable (2) transferred amounts are unknown (3) participants' balances are unknown. Optionally, (4) participants are hidden even to the other participants in the transaction.

Some familiarity with rollups and zCash is recommended :-).

Main ideas

The main concepts are:

- Operators and blockchain have the same responsibilities as in a standard rollup
- Accounts are not stored in clear like in a standard rollup: to update their accounts users send only states' hashes (old value, new value), plus a ZKP to prove that the transition is valid.
- To send money, users use money orders. Operators register money orders. Once registered, a money order cannot be repudiated.
- Like zCash, there is a Merkle tree to store the created money orders, and another Merkle tree to store the nullified money orders. These Merkle trees are not global but per account.
- The rollup state furthermore includes a Merkle tree to keep track of previous root hashes. This allows proving that a money order was previously registered.

Data structure

State

The state is the highest level representation of the roll-up state

Type

Description

Accounts

MerkleTree[Account]

Contains the hash of all accounts. Has depth 24.

PreviousStates

MerkleTree[State]

A Merkle tree containing the previous states of the roll-up. It is filled sequentially. When the tree is full, we loop back and overwrite the oldest state root with the newest one. This ensures the previous state roots stay accessible to the roll-up for some duration (it stays available for ~8years).

Accounts

Accounts contains user's private data

Type

Description

AccountData

AccountData

MoneyOrderSent

MerkleTree[MoneyOrder]

One-time list of created money orders. It is reset at each MoneyOrder creation.

MoneyOrderConsumed

SparseMerkleTree[Hash(MoneyOrder)]

The insert-only list of nullifiers. Essentially, this Merkle tree keeps track of the transactions received by the account. Its depth is 64.

ExtraData

string

A placeholder that can be used for more complex custom roll-ups, for instance: cross-roll-up transfers

AccountData

AccountData contains plaintext values, that we don’t want to be disclosed when the sender creates a Merkle proof of inclusion of a specific money order.

Type

Description

Balance

int

Balance of the account

PublicKey

int

The public key of the account owner

Randomness

int

Some secret randomness to hide the account data

MoneyOrder

A money order describes the intent of doing a transfer of money. Its view is restricted to the transaction participant.

Type

Description

To

int

Recipient’s public key

Amount

int

Amount of token transferred

Id

int

Unique id for each transaction. Id = hash(PubFrom Amount, To, Salt)

ExtraData

string

Extra data for real-world use cases. For instance a command identifier.

Additionally, we describe below the structure of a Money Order Receipt (MOR).

MoneyOrderReceipt

A money order receipt is a message sent by the sender to the receiver to let him that he created a money order for him and a proof that he did.

Type

Description

MoneyOrder

MoneyOrder

Money whose this receipt is about

ProofOfRegistration

MerklePath

a Merkle Path to a state root hash where the Money Order was registered. Can be a ZKP to hide the sender account from the receiver.

ZKP description

Technically, there should be a single ZKP for registering and using money orders. For clarity we detail two independent ZKP:

creationZKP

Public Inputs

Description

AccountHashBefore

AccountHash before the transaction

AccountHashAfter

AccountHash after the transaction

Constraint Summary

Name

Description

RangePBalance

Proof that the resulting balance is positive

RangeProofAmount

Proof that the transferred amount is positive

MKP_MoneyOrder

Proof that the claimed MoneyOrder root is the claimed one

OpenAccountBefore

Proof of that the claimed account content matches the public account hash

OpenAccountAfter

Proof that the new account Hash matches the new account content

Possession of private key

Proof that the sender actually owns the account

receiptZKP

Public Inputs

To

The receiver PublicKey or temporary key

Amount

The amount of token transferred

Id

Id of the transaction, see MoneyOrder.Id

StateHash

A valid former roll-up state Hash

Constraint Summary

Name

Description

OpenMonerOrder

Get the hash of the claimed money order

MKP_Account_sender

Proof of inclusion of the claimed sender account hash

MKP_MoneyOrder

Proof of inclusion of the claimed money order

OpenAccount

Proof of that the claimed account content matches the public account hash

- The receiptZKP is optional. It allows the money order creator to hide his accountID to the recipient of the money order. However, doing it implies a second level of recursion and thus, use costly MNT4-6 curves cycles or Cock-Pinchs curves.

MoneyOrderRedemptionZKP

Public Input

Description

AccountHashBefore

AccountHash before the transaction

AccountHashAfter

AccountHash after the transaction

CurrentStateRootHash

State root hash used to prove correctness the used money order receipt. By current, we mean the time at which the time the redeemed user creates the proof.

Constraint Summary

Name

Description

Verify MoneyOrder

Verify the correctness of the money order

MKP_Nullifier_0

Proof that the position at which we will insert the new nullifier is empty. (IE: the transaction wasn’t already consumed)

MKP_Nullier_1

Proof that the claimed Nullifier root after insertion is the claimed one

OpenAccountBefore

Proof of that the claimed account content matches the public account hash

OpenAccountAfter

Proof that the new account Hash matches the new account content

OldStateProof

Proof that the money order was verified with a correct previous state root

Operator Proof (valid for both receiver and sender execution)

Public Inputs

Description

AccountHashBefore

AccountHash before the transaction

AccountHashAfter

AccountHash after the transaction

rollupStateBefore

The Root hash of the roll-up state before

rollupStateAfter

The root hash of the roll-up state after

oldStateInclusion*

MerkleProof of inclusion of the root proof

- Needed for Money Order Redemption execution

Constraint Summary

MKP_Account_before

Proof of inclusion of the claimed sender account hash

MKP_account_after

Proof of correctness of the resulting root hash

MKP_old_transition

Proof of inclusion of the old root

Verify transition

Verify the correctness of account hash transition

Workflow

The full workflow from user S to send money to a user R is:

- R creates its account info AI : hash(random number saltX, private key)
- R sends this information to S, off-chain

- S (1) creates a money order (2) adds its to its created orders accumulator (3) calculate the new hash value for its account (4) generate the creationZKP
- S sends the update transaction to the operator, with the old hash, the new hash and the creationZKP
- The operator generates a rollupUpdateZKP for all the transactions received, and sends a global update transaction to the blockchain.
- S watch the blockchain to see if is account was updated. Once it's done it can generate a receiptZKP to link its update to the global rollup state Sx. S creates a Money Order receipt and sends it to R.
- R checks receiptZKP, and checks on the blockchain that the root hash Sx provided correspond to an existing rollup update. If so, R knows that the money order is registered and the payment final (with caveat relating to blockhain's finality as in a standard rollup).

For R to transfer the amount of the money order to its own account, with a receipt for a money order M, going to a historical state Sx of the rollup.

- R gets a Merkle Path from a recent version of the rollup, called Sc, to Sx.
- R generates a ZKP, which includes:
- proof that M was included in the history of Sc, with the orderReceiptZKP and the Merkle Path from Sc to Sx.
- proves that M was not in R's nullifiers.
- proof that M was included in the history of Sc, with the orderReceiptZKP and the Merkle Path from Sc to Sx.
- proves that M was not in R's nullifiers.
- R send the update transaction to the operator.

Practicalities

If we want to hide the sender from the receiver, we need to use a ZKP, hence a curve that supports 3 levels of recursion (eg. MNT4 or 6).

The users should connect to the operator with a protocol such as Tor to hide their ip addresses.

The operator knows the Merkle Tree of the historical states. The users need to access this data to create the proof of past inclusion for the receipt. To prevent the operator to correlate such request to account updates the user can first get the data from the operator, then watch the blockchain for any amount of time to update the Merkle Path.

Money orders can be checked until the proof of registration leads to a state known in the state history. In other words is money order is limited in time. With 2^24 states kept and 10 seconds between two rollup update a money order is valid for more than 5 years.

Performances

With MiMC as the hash function, a cost of 20K constraints to check a proof, and 2^24 accounts, we can estimate the operator cost for a standard rollup and compare it with this anonymous rollup:

Standard:

- Transfer: 24 * 2 * 2 hashes to verify + signature (~2000 constraints) = ~70k constraints

Anonymous:

- Money order registration/utilisation/both by a single account: 24 * 2 hashes + ZKP to verify ~70k constraints. Sender's anonymity for the receiver requires specific curves (MNT4) slower and currently less optimized than BN256.

Anonymous rollups also support batching, i.e. it's possible to register/use multiple money orders in a single operator transaction. In the data structure proposed above, it's possible to create 2^8 money orders in a single transaction, while there is no limit for the operator on the number of money orders used in a single transaction.