Here and here are my previous writeups about Zeropool, built at ETHBoston.

We experimented with different architectures and selected the current way of implementation, reducing gas costs and storage.

Some properties

- TX cost is from 40k gas to 15k gas (depending from the number of transactions batched)
- Block size is 256 transactions
- Anonymity set is 2^{32}

and we have one set for all assets and all amounts

- If the relayer stop working, all assets could be safely withdrawn
- Withdrawals are not instant, there is one-challenge exit game with a timeout. And 3rd parties or users, performing deposits, could buy out the withdrawals to process them instantly.

Resuming, this is like bitcoin omni layer, but for Ethereum, stored in calldata and providing not expensive private transactions for eth and any ERC20.

State

The state was not changed from previous architecture and still Merkle tree with UTXO leaves.

UTXO structure

UTXO structure is composite of following elements:

- token (for ether equals zero)
- amount
- owner_commitment = hash(pubkey, salt)

Transactions

We use single generalized 2-to-2 joinsplit transaction snark for withdrawals, transfers, and deposits:

- in

are hidden inputs, picked inside the snark by Merkle proofs

- out

are outputs, hashes are public

- delta

is an additional amount, deposited or withdrawn from tx. Could be positive (deposit) or negative (withdrawal)

- Hash of additional data, not used directly in the circuit, but important for the contract (receiver address, hashed utxos)

Fees

We are planning to use the custodial anonymous gas station for the relayer to pay for fees. How it works:

- user deposit some ether to relayer
- user burn some his assets anonymously (with a circuit like the main Zeropool circuit)
- user attaches burning note to his transaction and send it to relayer

So, user trust to the relayer only gas. If the relayer is doing something wrong (gas stealing, going offline, etc), the user can safely withdraw his assets, excluding gas.

Zeropool layer chain

All transactions are published inside blocks up to 256 transactions. Each block item is a composite of the transaction, sent by the user and UTXO root update. All transactions are packed into block onchain (empty elements on the right are refilled with copies of the last transaction) and Merkle root of the block is published in storage.

There are following challenges for the relayer:

1. transaction challenge (checks, that snark proof is correct)
2. utxo Merkle tree challenge (checks, that Merkle tree accumulated 2 utxos correct)
3. doublespend challenge (checks, that nullifiers are unique, excepting cloned transactions on the right side of the block)

Each challenge stops the pool. All data following after the defect considered to be invalid. That's why withdrawals have a timeout - to check, that the chain is valid.

If the relayer does not process the withdrawal or pool is stopped, the user can include the force withdrawal into the chain. The timeout is double longer in this case: the first timeslot is used to prevent data races. During the second timeslot, the withdrawal could be challenged by double-spend from the chain until first timeslot end state. After that, the forced withdrawal is considered included into the chain at the current end (could be withdrawn as regular withdraw, or there could be processed a challenge (1-3)).

Receiving the assets

All UTXOs are encrypted via the receiver's public key and published onchain in calldata. Each user should scan the whole chain to find his assets. That means that the user anyway will scan and verify all chain to prove the validity of the chain part with his assets inside.

Further improvements

@BarryWhiteHat proposed to use snark+stark rollup for scaling such solutions. This approach allows us to avoid challenges and create 1-depth recursive rollup with private transactions inside the starks.

Very early alpha is alive

Attention! It is really very early alpha. Some parts are not implemented, other parts are implemented, but not fully tested. But it is working and you can touch the contract, transfer some ether privately and see the demo video.

Github

Youtube

Contract

Relayer