Abstract

Ethereum needs more privacy. We are going to provide it via our ETHBoston winner ZeroPool solution. Following primitives are processed via ZeroPool's zkSNARK circuit and completely anonymous:

- multiple assets
- deposits
- withdrawals
- transfers
- atomic swaps

Here I explain more about the design of ZeroPool transactions, circuit, smart contract, and microservices.

State

ZeroPool is UTXO based solution, where hashes of UTXOs are stored in calldata, root hash of UTXO is stored in storage and UTXOs and transactions data is completely private.

We implement a Merkle tree with up to 2^{30}

elements and new elements may be added into this Merkle tree one by one. Default values are zeroes.

UTXO

Utxo is a composite of 4 parameters: assetId, amount, owner pubkey and unique utxoId. Only hashes are presented onchain without additional encryption.

To spend the utxo we need to publish onchain Nullifier

- the deterministic function from UTXO. We are using ECVRF, this approach allow us to store private keys at hardware wallets separately from the prover.

We are building chrome extension, supporting signing ECVRF and EDDSA on babyjubjub curve for ZeroPool.

Transactions

Deposits

are simple and each deposit just adds a new element into the tree.

Transfers

are based on 2to2 transfer primitive.

Some inputs and outputs may be zeroes. All kind of simple transfer (join, split, other transfers, excluding atomic swaps) may be presented as a partial case of this construction.

Withdrawals

Withdrawals structure is very close to transfer and allow the user to withdraw arbitrary values, not strictly equal of any numbers of his UTXOs

Atomic swaps

To change n_A

token A to n_B

token B maker create order at some 3rd party private platform. Taker finds the order and begins MPC procedure vs maker.

- Taker sends to maker his public key
- Maker creates a transaction

TX2: $(n_A, \text{maker}; n_B, \text{maker})\Rightarrow(n_A, \text{taker}; n_B, \text{maker})$

and send it to taker

- Taker creates a transaction

TX1: (\dots;\dots)\Rightarrow(\dots;n_B, maker)

, add TX2 hash to the transaction and send it to maker. TX1 may be completed if TX2 completed.

- Maker joins TX1 and TX2 into one meta-transaction and shows send it to ZeroPool relayer

Force withdrawals

If the relayer stops working, users can process force exit procedure. It is not anonymous in single operator case. But if we have different pools/mixers onchain, it is safe. For exit, the user must prove that he knows UTXO preimage and publish valid nullifier.

Relayer

There is an account initialization problem in Ethereum. You cannot interact with blockchain if you have no balance, including cases when you withdraw more ether from the contract than you pay for the gas.

That’s why we need the relayer:3rd party, who process all ZeroPool transactions with ethereum. To protect users from griefing from the relayer we allow to the relayer do only 10 interactions with the contract per 100 blocks and remember last 100 root hashes. Relayer can batch zkSNARK proofs via batch verifier. That means that instant transaction cost is asymptotic about 1 pairing for a batched transaction and 4 pairings for an instant transaction.

Gas

I propose to use a separate anonymous custodial pool for gas on the sidechain. User pays some ethers on the main chain to the relayer. After that, the relayer emits the ethers on the sidechain. And user can burn that ethers on the sidechain and attach such burn transactions to mainnet transactions to prove to the relayer, that he pays for the gas.

Smart contract

Smart contract process all transactions, sent by relayer or users and update the Merkle tree of UTXOs and hashmap of nullifiers. If some transaction is using the same nullifiers, it must be rejected. We use batch verifier to spend lesser values of gas per proving.

Circuit

We join all primitives into one a little bit more complex zkSNARK. This allows us to batch everything processed by ZeroPool into one batch verifier.

Here is map of all signals used with our circuit:

Deposit

Withdrawal

Transfer

AtomicSwap

Public:

0

root

root

root

1

nullifier

nullifier

nullifier

2

nullifier

nullifier

utxo_in_hash

3

utxo_out_hash

utxo_out_hash

utxo_out_hash

utxo_out_hash

4

asset

asset

utxo_out_hash

utxo_out_hash

5

0x0

(0x1<<160)+receiver

0x2<<160

0x3<<160

Private:

35

merkleproof

merkleproof

merkleproof

36

merklepath

merklepath

merklepath

66

merkleproof

merkleproof

67

merklepath

merklepath

69

ecvrf

ecvrf

ecvrf

71

ecvrf

ecvrf

ecvrf

75

utxo_input

utxo_input

utxo_input

79

utxo_input

utxo_input

utxo_input

83

utxo_output

utxo_output

utxo_output

utxo_output

87

utxo_output

utxo_output

89

eddsa

eddsa

eddsa

eddsa