Thanks to Alex Gluchowski from zkSync and Barry Whitehat for your insights.

1) TL;DR

A zkRollup that does not require tx-history-data from an operator. This has an efficiency of gas use of txcalldata on L1 and also has the privacy of smart contract executions and assets. Only an account list of owners of changed states needs to be recorded in txcalldata for each batch.

The demerit is a need for zkp calculation of client-side for each user when she makes funds exit to L1. Another demerit is the difficulty of EVM compatibility.

2) Background & Motivation

Rollups still cost operators and transactors for their txcalldata use. This restriction is simply because of a need to restore states which are the result of transactions not to make users unavailable to generate the Merkle proofs of their funds. The most of specifications of Rollups require operators to dump all transaction history data to txcalldata on L1.

This transparency of transaction history data is not only increasing gas costs of txcalldata but also disabling the privacy of transactions.

It’s conjectured that an accumulator of transaction history data solves both the efficiency problem and the privacy problem.

3) Approach

In short,

In the first step, we construct a zkRollup that operator writes final states diff directly to txcalldata. Transaction history data will be in private inputs of a circuit of zkp.

In the second step, we remove the final states diff from txcalldata by separating commonly used storage and user state storage. This enables a user’s exit with non-inclusion proof like a state version of Plasma Prime. A user keeps her user storage and exposes only its Merkle Root. The user can prove the root transition with zkp and can update a commonly used storage of a smart contract.

The details are below.

3.1) First Step, options of txcalldata usage in zkRollup

There are 2 two options of using txcalldata to restore the full states.

Option 1 is recording all of transaction history data to txcalldata.

Option 2 is recording the diff of the final state as a result of transactions in the block (batch).

In option 2, millions of transactions with the same result of no transactions use 0 gas for the txcalldata use, since there is nothing to record in the txcalldata. The soundness of the Merkle root transition is guaranteed by zkp.

Adopting “option 2” is the first step.

3.2) The second step, optimizing “option 2”

Option 2 described above costs less gas when the transactions in a batch/block change the same storage value in a contract. Such commonly shared and changed values are like a total supply of ERC20, a total pooled amount of a swap protocol, etc.

And also this kind of storage value affects all asset holders, and the loss of this kind of data leads to the loss of liveness of zkRollup. On the other hand, the other data which is not commonly shared and changed are mostly individual asset data. The loss of this kind of data directly means the loss of funds of the owner. This risk is separated and does not affect the other’s funds.

Then separating users’ states and giving a user data of her states and its proof as a receipt of her transaction by an operator makes mass gas cost cut.

(1) A transactor send a transaction to the operator

(2) The operator make the merle proof of her user state as a receipt of the transaction

(3) The transactor sign the receipt

(4) Only transaction data with the signed receipt is accepted in the circuit

If one user makes a transaction and several users have their balances changed and they know their states including those balances and Merkle proofs, any of them can exit her funds at any time by zkp which proves that it's the last state.

This proves that it's the last state of her balance can be made by non-inclusion proof of each account list of owners of changed states for each batch. Sparse Merkle Tree of account list of owners of changed states can be used to have efficient proof. This is like Plasma Prime with states, SMT, and zkp. There is no exit game.

There are 2 ways to let the owners of changed states know their last changes.

If they are online, the operator sends the last diff, receives the signed that diff, and puts it into the input of the zkp circuit. This has the cheapest gas cost.

If they are not online, the operator posts it to txcalldata or off-chain decentralized storage.

With this separation of states, an operator no longer needs to put even any diff of the final states to txcalldata, because users have their account states safe enough for the exit, and losing commonly shared data is just meaning that the operators can not update the Merkle Root of the zkRollup and they will simply stop the service. Then both commonly shared storage and user storage can be distributed off-chain. Only an account list of owners of changed states needs to be recorded in txcalldata for each batch.

3.3) The third step, private smart contract execution

Users' transactions are not on-chain, but operators still can see and need to see the user states including balances to make zkp proofs.

If a user makes zkp by her side to prove the transition of her Merkle root of her user state and the commonly shared storage, the operator can just change that root and storage. The secret of the balance remains.

(1) The user sends a transaction to the operator.

(2) The operator returns the diff of balance and the updated common shared storage.

(3) The user makes zkp proof of the updated Merkle root of her user state and common shared storage.

An operator who makes each batch can know the balance diff by the changes of commonly shared storage in the batch, but she can not know the balance diffs in other batches since only the final diff is shared among operators. This has mixing-level privacy.

This mechanism requires recursive-zk.

4) More detailed discussion

4.1) Communication to off-line state changers at off-chain.

This is just an option. This protocol can be constructed without this part.

Even in the worst case, state changers are off-line, data availability risks for this case are very limited.

An offline user can get data to be safe to exit when she is online. She can set agents instead of receiving the data by herself.

And we can construct the exit method so that the last state update does not make the previous state dangerous due to a data availability problem.

Typical decentralized storage can be constructed as bellow,

(1) commit hash(storage)

(2) prove preimage(hash(storage)) = preimage(hash(storage, last-Ethereum-block-header)) - last-Ethereum-block-header

(3) keep watching how many nodes can do (2)

4.2) Account listing gas cost on chain

Each account can get an ID that is much shorter than an address itself.

Only an account list is needed for each batch, then this can omit duplicates, and this is much more efficient than transaction history in txcalldata uses.

4.3) Further optimization of common shared storage

On Ethereum L1, you can not erase txcalldata.

We can modify this since common shared storage needs not to be on-chain.

Unlike transaction history data, we need only last state data, not any state before.

Then operators can abandon the previous “final states data” shared in the network.

Operators can know the data which can be abandoned with the zkp logic.

5) Conclusion

Separating user states makes both efficiency and privacy in zkRollup smart contract execution. Almost all of txcalldata cost is removed from zkRollup.

ps: Please read this also.

A zkRollup with no transaction history data to enable secret smart contract execution with calldata efficiency[

General Layer 2

](/c/layer-2/32)

It is easy to misunderstand the point of data availability in this protocol. First, this protocol does not rely on off-chain decentralized storage. This is just an option.

I edited the document and added this. Second, this is important that operators don’t need to write the commonly shared storage to calldata. Let’s talk about the Uniswap V2 on this protocol spec, and let’s start with all LPs are online for the simplest case. It’s essentially a transaction between a transactor and several…