Recusively Labs is working on a zero knowledge rollup, where transactions are converted to zkp prior block inclusion.

This process of transactions batching into zero knowledge proofs will look approximately this way:

The transaction is modified by the zero-knowledge-proof system prior being sent to the validators. Thus, it is then assumed that the block is the block, so it is the finality. But where in this approach is the actual

bottleneck when blocks are created? A zero-knowledge-proof of every single block could be created and then there would be a compressed block, but after it has been compressed, what would happen then? Wait for new block to compress? This is not a viable option that would allow efficient increase in the total sum of transactions per second. Therefore, prior the transactions get included in the block they can be batched

together and then subsequently submitted to the block in form of a proof. So, the bottleneck of the blocksize per max gas amount will be removed. If proofs are created prior block inclusion, but after the mempool then

actually there is transactions batching after the mempool to the zero-knowledge-proof system and then “subblocks which are batched transactions transformed in zero-knowledge-proof” to the block. Thus, the

Account State would contain the zero knowledge proofs of transaction batches instead of the single transactions as it is currently done.

The journey of transactions from signing to block inclusion:

I.

mempool

II.

conversion of transactions batchwise into zero-knowledge proofs

III.

block inclusion of the created zero-knowledge-proofs into the subsequent new block

The zero-knowledge-proof creation computation is conducted locally, since the mempool is located locally and then subsequently broadcasted to others. The zero-knowledge-proofs of these transactions are created via the zero-knowledge-proofs creation process and after being added to the mempool and their inclusion into the mempool and the subsequent zero-knowledge-proofs of these are broadcasted to other nodes. Thus, the computational power requirements for running nodes will increase and blocks will include zero-knowledgeproofs and not the single transactions.

One of the nodes on the network is the block proposer for the current slot, having previously been selected pseudo-randomly using RANDAO. This very node is responsible for building and broadcasting the next block to be added to the Ethereum blockchain and updating the global state. The node is made up of three parts: an execution client, a consensus client and a validator client.

The code base of the Recursively’s modified client will also include the zero-knowledge-proofs frontend and backend, but only the one chosen by RANDAO will compute the respective zero-knowledge-proofs for the

subsequent block.

This node will be responsible for building and broadcasting the next block to be added to the Ethereum blockchain and updating the global state. The node is made up of three parts: an execution client, a consensus

client and a validator client. The execution client bundles transactions from the local mempool into an “execution payload” and executes them locally to generate a state change. This information is passed to the

consensus client where the execution payload is wrapped as part of a “beacon block” that also contains information about rewards, penalties, slashings, attestations etc. that enable the network to agree on the

sequence of blocks at the head of the chain.

The zero-knowledge-proof creation process has to come after the mempool and thus after execution payload is created as well as after state change has been performed but

before the transaction is broadcasted to other nodes. This will lead to transactions querying in a block requiring a different approach, since after this adoption zero-knowledge-proofs will be queried and thus the

transactions itself indirectly. Other nodes receive the new beacon block on the consensus layer gossip network. They pass it to their execution client where the transactions are re-executed locally to ensure the

proposed state change is valid. The validator client then attests that the block is valid and is the logical next block in their

view of the chain. The block is added to the local database in each node that attests to it.

Here it should be mentioned that from the zero-knowledge-proofs in the block the transactions can be retrieved in order to execute the transactions locally in order to ensure the proposed state change is valid.

This is here the first time that the created zero-knowledge-proofs are called to prove the transactions they are derived from.

Therefore, only the Account State contains zero-knowledge-proofs. Besides, the above listed concept has a neutral impact on the block finality.

Suggestions are welcome