1) TL;DR

:

This post proposes a pre-consensus mechanism to secure instant finality and reduce verification gas cost, without compromising instant fund exit in zkRollup.

2) Background & Motivation

:

zkRollup can achieve instant finality by adopting a short enough commitment interval (like 10 min). The trust risk of aggregators in this method is increasing in the length of intervals, while the cost of finalization (=if groth16, verification gas cost is over 200k gas per commit) is decreasing in the length of intervals.

Let us start from reminding the over 200k gas (= $100 ~ $500) of zk's pairing verification.

This costs aggregators at each interval of zkRollup to verify and finalize the commitment.

This cost cannot be ignored since zkRollup's commitment interval is short because instant exit gets finalized with this commitment and instant finality (economic finality, 0 conf) can be allowed as far as the interval is short and a malicious aggregator has little incentives to revert.

It is hard to change this interval even if many transactions can be aggregated with recursive zk and efficient proof calculation systems. Safety will be compromised if the interval is simply lengthened.

Then we need to think how to achieve both of a safe instant finality and a long verification interval in zkRollup.

3) Approach

:

The aggregator's running cost stems from the condition that gas cost of zk-verification on contracts are high and interval is short.

Thus we make the verification interval longer without compromising safety and usability.

3.1) First Step: skipping zk-pairing-verification

The first thing we can come up with naturally is skipping pairing calculations and introducing simple fraud proof against the commitment.

Aggregators submit proof or anything they need to verify zkRollup commitment to the contract, but does not execute pairing calculation at this moment and skip paying 200k gas. This commitment gets verified after the period passes. Each state in a commitment will be the public input of the next commitment. Some Ether from aggregator needs to be funded as a penalty to incentivise verifiers (=fraud watchers).

hash(public input, zk-proof-data, last state root, next state root, tx hash, aggregator address)

is the commitment.

The preimages are emitted in events onchain, and this commitment is saved in the contract storage.

This approach has a big advantage.

Instantly, everyone can be a watchtower, who is the equivalent of "verifier" in ORU (Optimistic Rollup) context, without a fullnode or any special setup.

Data accessibility problems do not occur because everything needed to verify or to execute fraud proof is in an event emitted onchain. L2 transaction data and the storage of its result is not required to have when executing fraud proofs, because such data is already aggregated in zk's public inputs and proofs.

If a malicious aggregator submitted the malicious merkle root, and abandoned all tx data and merkle tree data, there is no need to set up a fullnode to fraud proof. Just checking the zk-proof data and executing the verify function with pairing are enough to detect such

malicious activities.

However this method has a security problem.

If 51% attack occurs on L1 to make a malicious merkle root rightful, it's hard to stop it.

As 51% execution cost is increasing in the block length, we need this period long enough to make 51% attack hard. The

ideal length of one period will be 7 days because of just the same reason that ORU's exit period is 7 days which can be calculated from the mining cost and then realistic attack incentive.

With this condition, there's no reason to avoid ORU and use this solution.

3.2) Second Step: pre-consensus commitment without zk-verification , finalize with recursive zk's pairing

We can overcome the aforementioned security issue in the following way.

We treat the commitment without zk-verification as pre-consensus which restricts the finality with zk-verification.

(consensus commit) => (pre-consensus commit) => (pre-consensus commit) => …. => (pre-consensus commit) => (consensus commit)

All of pre-consensus commitments restrict the consensus with pairing. So L2 users can have safe instant finalities of their transactions. The consensus commitment needs to be verified with all pre-consensus commitment with recursive zk. There are 2 circuits: pre-consensus circuit and recursive circuit. Pre-consensus circuit contains the logic of the dapp which is zkRolluped. Recursive circuit only has to get pre-consensus data in L1 as public inputs.

Pre-consensus can be merged horizontally over the time period by recursive zk: At the same time recursive zk can also be used vertically for mass transaction aggregation to a pre-consensus commitment.

If any fraud pre-consensus commitment disturbs the consensus verification with pairing, this is always able to be fraud proven by zk-pairing mathematically. Once this fraud is proven, an aggregator removes this with the zk-verifier-function, and restarts the aggregating and pre-consensus committing.

Fund holders can exit whenever they make the consensus from pre-consensus with 200k gas if they are in haste. (Of course they can also wait for the consensus by aggregators). As mentioned in the "first step", they don't need any special setup to make the finality of the consensus, because all inputs are already aggregated and these can be searched with onchain events. The gas cost of this recursive verification does not increase no matter how many proofs of precommitments are verified because they will be hashed to the entry hash.

51% attackers cannot finalize a malicious merkle root simply because every root should finally have the verification onchain by a logic of contract codes implemented by zk-circuit.

4) Conclusion

:

This pre-consensus protocol with fraud proof and its relevant data accessibility allows zkRollup to have a long interval commitment (like 6 hours). This approach saves a lot of gas costs of verification calculation.