Authors: Barry, Onur

Intro

Previously we have defined RLN which uses ZKPs to rate limit messages. Which allows for private p2p networks with strong dos resistance.

RLN ZKP has two inputs and two outputs

Inputs: Merkle Root , Message, epoch

Outputs: Shamir secret share, nullifier

The user creates a proof and broadcasts it to the network. The network sends it to every peer. Each peer validates

1. The merkle root matches the smart contract
2. The epoch is inside the last hour.
3. That the ZKP is correct.
4. Then can then check if they can decrypt the message.
5. Check if the message is a double spend with the nullifier + their list of previous nullifiers. If it is they generate the private key and slash them.

When users sign up they deposit some funds which is used to rate limit signups. We would like to take these funds a share them between the nodes that the users themselves define.

This works well but we would like to add incentives to reward the nodes that first relay the message.

Modern networks have become very sensitive to censorship attacks where the entry nodes can be blacklisted.

Here we borrow from how proof of work mining pools work in order to distribute the reward to the entry nodes.

Incentives

In order to do this we change the message such that it is the hash (message, reward_target)

the rewards target is the gateway node that is responsible for relaying the message to the rest of the network.

Note: We can also add Salt here if we want to obscure the reward_target from the rest of the network.

At the end of the current reward_epoch (1 day) each node commits to the number of messages that were signed for them by sending a message to a smart contract + a deposit (1 eth).

At the end of the claim period the smart contract generates a random number via block hash.

We define the proof of message as being the number of claimed message.nullifier % randomness.

Assuming message.nullifier is a random number produced by the random oracle (hash) know that the proof of message (message.nullifer % randomness) is a random number.

We then define the difficulty as the number of messages that they claim they have received.

message.nullifier % randomness <= 2**256 % climed_number - padding

Where padding is a parameter that we use to make it 99.999999999 % likely that they have such a number.

When the sends the RLN message to the reward contract the contract. The contract

1. Validate the snark.
2. Ensure the merkle root is correct.
3. That the epoch is in the correct range.
4. Prove that the reward_target is them. Optionally reveal the salt here or use ZKP to prove it is them without revealing the message they sent.

We then wait a period of time to allow the slashing of a double spend messages. If a double spend it detected the node is required to add another proof o message above the previously claimed difficulty.

If a node validates their claimed number of message they get paid noMessage*feePerMessage

if not they get nothing.

Note: insert optional deposit + slashing here.

Example

Users

Alice, Bob

Nodes

Node1, Node2

Epoch 1

Alice makes message for epoch 1 with

epoch = 1

nullifier = hash(alice.private_key, epoch)

secret = sss(message, alice.private_key)

message = hash(node1, hash(encrypt(“hi Bob”)))

and sends it to node 1.

Bob makes message for epoch 1 with

epoch = 1

nullifier = hash(bob.private_key, epoch)

secret = sss(message, bob.private_key)

message = hash(node1, hash(encrypt(“hi Alice”)))

and sends it to node 1.

Epoch is over claim reward

Node 1 have received 2 message and claims so in the smart contract.

Node 2 has received 0 messages but claims to have received 2.

Both deposit 1 eth collateral.

Commit is over prove it.

The smart contract generates a random number and asks the nodes to provide proofs that have

nullifier \% randomnumber < 2**253 / claimned messages

Here we use ranom number to randomize the nullfiers. If the nodes know therandom number at commit time they can cheat.

The nullifer is a number between 0 and 2^253. See table for probability a nullifier set does not contain 1 element less than 2^253 / 2 given.

Set Size

Probability

1

50%

2

25%

3

12.5%

4

6.25%

5

3.12%

As our set grows the probability that we have a really small element grows. So we can check the size of the full set by checking a single element.

Then we know that because Node1 received 2 messages the likeliest thing is that one nullifer is < 2^253 and the other is > 2^253. 25% chance that they can fulfil this requirement.

Because Node2 received no message there is 0% chance they can fulfill it.

Note: We need to tweek the parameters such that node 1 has much higher than 25% chance of being able to give a correct message.

Attacks

Analysis attack

I can see which message originate close to which coordinators

This will allow me to monitor them.

This attack only reveals a random message per epoch. Similar information could be found by monitoring the network.

Optionally we can use a ZKP to mask the ID of the node requesting the data. Nodes can also update their claim address each round.

Attack many accounts attack

I create many accounts and each epoch I claim all the rewards from each account.

The reward is proportional to the number of accounts you are basically paying your own reward. Towards the end of the valid epoch users who do not send a lot of message could have their reward claimed by these accounts.

To over come this at the end of each round we calculate how many accounts did not send any messages and burn their reward to some public good. Or else we could just leave this for the active network to share.

Conclusions

Here we defined how to reward nodes that fill important roles in the p2p network. Such incentives we hope will make such networks very resistant to censorship attacks.