# DA node data storage check

## Problem

When reading the [BitNetwork proposal](https://github.com/windranger-io/public-docs/blob/main/research/R002.md), I think maybe we need some work to make sure DA node serves well in the long run. I think EigenDA should already has a solution on it, but as I can’t find EigenDA’s design detail, I just talk about my idea on it. If you could help share EigenDA documents with me, I’ll be very happy to learn it.

I think we have 2 risks:

* 1. Although EigenDA node committed to store and serve on datablob, it’s difficult to check it. Because when someone can’t get data from EigenDA node, it’s hard to prove whether it’s network issue or DA node just dropped the request.
  2. DA node should provide long time service on the datablob it committed, but how can we check and make sure it didn’t delete the data which’s very old?

## Possible solutions

* 1. DAS can solve the risks for blocks, but for other data, it maynot work very well. And it’s too heavy.
  2. EigenDA should have its solution, but I don’t know it.
  3. I’m trying to give some idea to help on risk2. it can’t resolve it, but it can help on it, and I think it costs much less resource than DAS. Please see it below.

## My solution

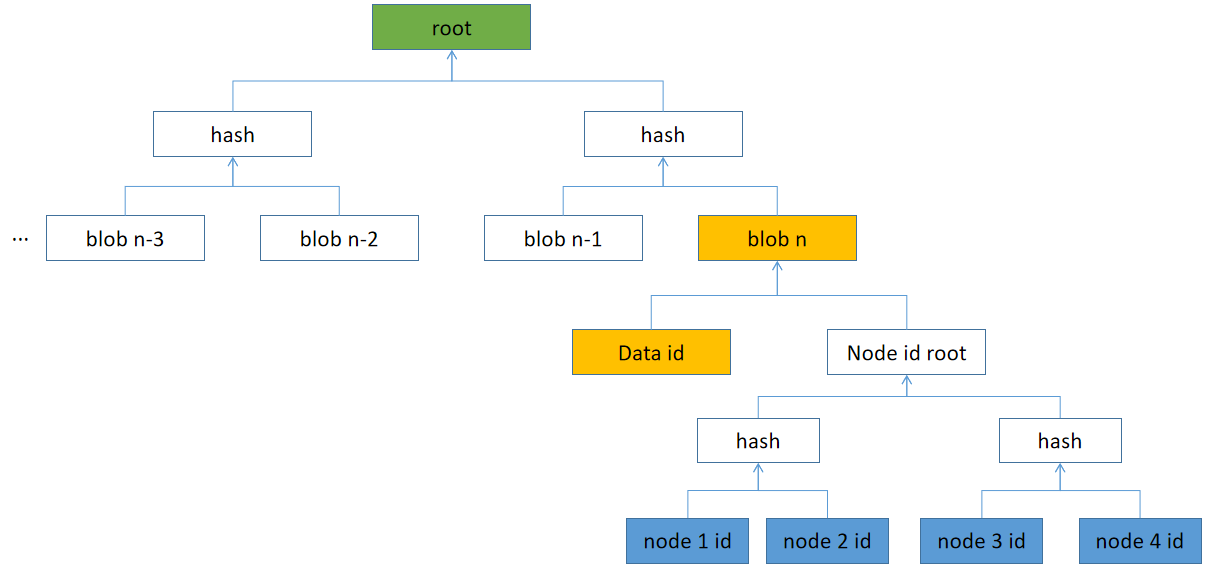
Main idea is: We can regularly , randomly select some datablob, and ask the DA nodes committed on it to give the proof that it still stores it. If the DA node can’t give the right proof, it should be slashed.

I assume datablob has index (e.x. one datablob corresponds to one block).

All the solution works via a smart contract, which’s deployed on BitNetwork, let’s call it DAProof contract.

### 3.1 preparation

In the “Commitment step” in the [proposal](https://github.com/windranger-io/public-docs/blob/main/research/R002.md) , when storing the signatures of the DA nodes to Ethereum , we could store a proof in the DAProof contract that can prove which nodes committed to store current datablob. it looks like below:



Current L2 is trying to rollup block n, the corresponding datablob is “blob n”. there’re 4 DA nodes committed to serve on “blob n”. We just need to update the green “root” in DAProof contract.

Later when this blob is selected to be checked, sequencer could just upload the node list (the 4 blue id) and a SPV proof to the DAProof contract, because we already know the “Data id” (it’s selected to be checked), it’s very easy to prove the node list is the right list to be checked via the root.

And it should be not difficult to find an algorithm which’s “cheap” to update the root when new blob is added.

### 3.2 check

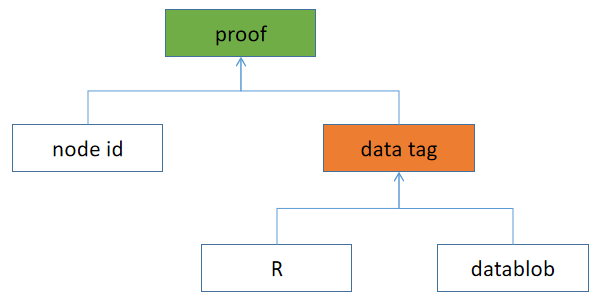
We can regularly do the check (e.x. every 600 blocks).

Assume the latest block height is “N”. We generate a random number ,named “R” (refer to chapter 4). We count “R mod N”, the remainder is the blob index which need be checked, let’s name it “blob i”. Sequencer should upload the node list who committed to serve “blob i” to DAProof contract, according to 3.1, it’s easy to prove that’s the right list.

Extend: If we assume every blob has a weight, we can increase the weight of the “newer” blobs. E.x. the latest 1% blobs have a weight of 100 for each, latest 1%-10% blobs has a weight of 10 for each, all other blobs have a weight of 1. in this way, new blobs are more likely to be checked.

### 3.3 prove

The proof looks like this:



R is the random number used in 3.2, it must be on left of datablob. So the “data tag” should be the same for every DA node. Because every node’s id is different, so their proof is different. So no one could copy and use other node’s proof.

There’re 2 steps for the DA nodes to do:

* 1. Upload proof to the DAProof contract in limited time (e.x 40 block time).
  2. After all proofs uploaded, any of the DA nodes could upload “data tag” to the DAProof contract. So the contract could verify each node’s proof. If one node don’t agree with another node’s “data tag”, he/she can upload his/her “data tag” to replace the previous one, we must make sure we use the same “data tag” to check all proofs. But if this happened, it means someone is lying.

### 3.4 slash

If a DA node can’t upload proof, or it can’t be verified, it should be slashed.

### 3.5 Values, Limitations and Risks

**Value**:

It costs much less resource than DAS, the sequencers and DA nodes just need to spend little resource to do it.   
 It could make sure the DA nodes store the blob in the long run. Because storage cost is much cheaper than be slashed , DA nodes don’t need to take risk.

**Limitations**:

This solution could only help to confirm the DA nodes stored the datablob. But if some DA node is malicious, it stores the data but don’t supply it to the requester, this solution can’t help.

**Risks**:

If DA node is not decentralized enough, this solution may fail. Because in 3.3 , if all DA nodes are controlled by the same one, even they all deleted the datablob, they can collaborate to pass the check.

If in 3.3 (2), different DA nodes supply different “data tag”, it means some one is lying. We should check out who’s lying.

* 1. An easy way is to ask all DA nodes upload the “data tag”, if most are the same, we believe that’s the right value. But if malicious collaborate, we maybe cheated.
  2. Another way is to use datablob id. The datablob id may contain the hash of the data(I guess), so we can directly verify it. But the blob maybe large, how could we do it in a decentralized way?

## Random number generate

We should find a way to generate on chain random number easily. Here’s a try.

### 4.1 solution

**Preparation:**

* Find a signature algorithm which for the same message, signature is always the same. (e.x. RSA)
* Sequencers exchanged their signature public key.

**Procedure**:

1. Assume we have a random number in the last block. When a sequencer is building the next block, it will sign on the last random number by its signature private key. The signature will be the random number in the current block.
2. Other sequencers will check the signature by the signature public key they stored, and sign on it when they’re doing BFT, the random number confirmation will be part of the block consensus.

The 1st random number could be the signature on genesis block.

### 4.2 analyze

* Each block could have a random number easily.
* This random number could be used for DApp, but can’t for system level.
* This random number could not be predicted and controlled in most cases on a decentralized chain. When a sequencer continues produces block, it could predict the random number, but it can’t control it.