Expected Consensus and Storage Power System

Protocol Labs

June 21, 2023

Contents

1 Introduction **2**
1.1 Notation . . . 2

2 Leader Election **3**
2.1 Definition . . . 3
2.2 Protocol details . . . 4
2.2.1 Randomness Chain . . . 4
2.2.2 Possible issues with the Random Chain . . . 5
2.2.3 Election PoSt . . . 5
2.2.4 Possible issues with Election PoSt . . . 6

3 Consensus [draft] **8**
3.1 Desiderata . . . 8
3.2 Protocol details . . . 8
3.2.1 Block Validity . . . 8
3.2.2 Weighting function . . . 8
3.2.3 Fork Choice Rule . . . 9
3.2.4 Chain extension . . . 9

4 Verifiable Storage Commitments [Draft] **10**

5 Power Table **13**
5.1 Definition . . . 13
5.2 Protocol Details . . . 14
5.2.1 Problems with Surprise PoSt . . . 14
5.2.2 Nuances with Periodical Rescaling . . . 16
5.3 Issues and Observations . . . 17

6 [Analysis]: How Likely is to to Fake SPoST? **17**
6.0.1 |P| is not long enough to allow (even partial) reconstruction . . . 17
6.1 PT-soundness with Surprise PoSt (Irene) . . . 18
6.2 Quality assurance . . . 18
6.3 [Analysis] Expected Number Of Challenge Rounds Needed to Prove Almost All Sectors 20

7 [Analysis]: How Likely is to fake Rescaling? 22

8 Empty Blocks Estimation 23

9 [Analysis]: Chain Quality 25

10 Game-theoretic properties 27

11 Proof details for finality 29
11.1 First case: Adversary collaborating with honest players 29
11.1.1 Normal catch up 29
11.1.2 Other strategies 30
11.2 Second case: private fork 30

1 Introduction

Our protocol is composed of three different modular sub-protocols. First a random chain and a leader election protocol that participants run at each round to determine their eligibility to mine a new block. We present this protocol in Section 2. This leader election is then used as part of a consensus protocol that participants use to extend their chain. We present this in Section 3. In order to prevent Sybil attacks, participants need to *contribute* some storage in order to participate in the aforementioned election protocol. We present our mechanism to do so in Section 4 where we explain how we ensure that participants cannot fake the storage they are contributing to the system over time. This also allows us to build a decentralized file storage network (or DSN) such that this resource is not wasted as has been argued in the case of proof-of-work.

1.1 Notation

- **Miner**: A miner M_i is a node on the network that participates in the protocol and is identified via an $\mathsf{minerID}$ derived from its public key pk_i.
- **Sector**: A miner M_i has m_i sectors $S_{i,j}, j \in [1, m_i]$ registered on the chain. A sector can be of varied sizes but all of a miner's sectors are of the same size s_i.
- The time is divided into discrete units we call *rounds*. Miners have semi-synchronised clocks that indicates the current round.
- **Power** and **Power Table** (Section 5): There is a power table PT that contains an entry for every miner in the system and is updated at each round k. Irene: *Each Round or each fallback Proving period??* The entry PT_i^k, corresponding to miner M_i, contains the integers m_i and s_i and the status (*i.e.*, a tag in $\{active, faulted\}$) of each sector $S_{i,j}$. The total power of the table at round k, $\mathsf{NetworkPower}_k$, is computed from the sum of all *active* sectors of all registered miners at round k.

  Irene: *check if considering active sectors only is okay with the spec!*

  More precisely, $\mathsf{NetworkPower}_k$ is the sum over all registered i of the terms $\mathsf{MinerPower}_i^k = n_i^k * s_i$, where n_i^k is the number of active sector of miner M_i at round k.

- **Block**: A block B contains a header H composed among other things of a set of blocks called its *parent tipeset* that we denote $B.parents$ and its body contains the list of messages. Henri: *Note that because of how tipsets work, a block's messages are only computed (i.e., used to update the state tree) at the next round, ie when that block is in a tipset of its own. Thus, the block's "state root" is the root of the state tree after the messages from block.parents have been applied to it.*

- **Tipset**: a tipset $\text{TipSet}_k = \{B_0^k, \ldots, B_j^k\}$ is a list of blocks that can be lexicographically ordered. We have $\forall i, j\ B_j^k.parents = B_i^k.parents$. A valid tipset is a set of blocks referred to as parents by another block. They must themselves point to the same parents and have been mined at the same epoch.

- **Blockchain**: A blockchain $C = \{T_0, \ldots, T_i\}$ is an ordered list of tipsets. Sarah: *it's more of a blockdag though so it may be confusing to call it a blockchain* We refer to the last element of the *chain head*.

- **Ticket**: A *random ticket* TR is included in every block header H generated and contains randomness. Ticket randomness is only derived from prior tickets in the blockchain. Taken together, the tickets of a block chain form a ticket chain. See Section 2.2.1 for more details. We distinguish such randomness tickets TR used for randomness throughout the protocols from *challenge tickets* TC used as part of the leader election (Election PoSt, see Section 2.2.3).

- **Verifiable Random Function**. There exists a verifiable random function

$$\mathsf{VRF}_{(\cdot)}(\cdot) : SK \times \{0,1\}^* \to \{0,1\}^* \times \Pi^{\mathsf{VRF}}$$

  On input of a secret key $\mathsf{sk} \in SK$ and a seed value seed, this function outputs a pseudo-random string $s \in \{0,1\}^*$ along with a publicly verifiable proof $\pi(s)$ of its correctness. We will denote this as $(s, \pi(s)) \leftarrow \mathsf{VRF}_{\mathsf{sk}}(\mathsf{seed})$.

2 Leader Election

2.1 Definition

Input: a list of N participants (miners), the $\mathsf{NetworkPower}_k$ (see Section 1.1 and 5) the PT at height k.
Output: a publicly verifiable list of leaders (could be empty) such that each participants is elected with probability equal to its power. More precisely each participants M_i outputs either $\perp$ or π_i where π_i is a publicly verifiable proof of their eligibility.
Security properties: A secure leader election protocol satisfies the following security properties (informal):

1. **Fairness**: The leader election is f-fair if for any adversary $\mathcal{A}$ controlling a fraction f of the network power it holds that

$$|\Pr[\mathcal{A} \text{ wins the election}] - f| \leq \epsilon$$

   for some $\epsilon < q$ (where q is chain quality)

2. **Liveness**. Even in the presence of f adversaries a list of leaders is output (i.e. an adversary cannot prevent the honest participants from running the protocol) Henri: *If we wanted more precision here we could introduce e the expected number of leaders per round, and define liveness as to how close to e we are on expectation. Important to note that treating e as number of leaders or proportion of reward in the round (as we do it) is equivalent, so easier to do the former.*

3. **Unpredictability**. A participant cannot predict whether they are elected or not before some pre-defined time Irene: *because of our Proof of Space, this has to be very strict: participant cannot predict at round k whether they are elected at round $k+1$*

4. **Privacy**. No one can predict the eligibility of another participant before they see published their proofs.

5. Sarah: *add any other security properties of SSLE*

2.2 Protocol details

2.2.1 Randomness Chain

In order to define our leader election protocol we first need to define a protocol that produces a random beacon. The blockchain also tracks a "randomness chain" or "ticket chain". Each block B_j^k contains a randomness ticket TR_j^k, where k is the height of the block.

Creating a new ticket: The block producer creates TR_j^k in the block header using the VRF in the following way

$$TR_j^k = y$$
$$(y, \pi(y)) \leftarrow \mathsf{VRF}_{\mathsf{sk}_i}(TR^{k-1}||j||k)$$

where TR^{k-1} is the smallest (by big-endian comparison) ticket from the previous tipset TipSet_{k-1} (if no blocks in TipSet^{k-1}, take the previous round in which there were blocks and use tickets from those), k is the height for which this ticket is derived from, and sk_i is secret key of the miner deriving the ticket za.

Usage: A miner uses the random chain to derives randomness at multiple occasion:

- Deriving a new ticket TR^k when they produced a block using TR^{k-1} (lookback is 1).
- Deriving the seed $\mathsf{EPoSt.Rand}$ for running leader election (election PoSt described in section 2.2.3, lookback is 1).
- Deriving the seed for challenging the sectors of a miner who hasn't won in some time (surprise PoSt, lookback is 1).
- Deriving the seed for challenging the newly commited power of a miner: lookback is equal to the finality parameters Nikko: *XXX link*.

2.2.2 Possible issues with the Random Chain

- **Grinding attacks** (link). NO SOLUTION YET!!

Now that we have a random beacon associated with each block, we define our leader election protocol.

2.2.3 Election PoSt

Election PoSt (EPoSt) can be considered as a lottery that works as follows. Let $\mathsf{VRF}_{(\cdot)}(\cdot)$ be a verifiable random function (see section 1.1) and recall that each round k is associated with a random ticket TR^k (see Section 2.2.1) and the power table PT^k (see Section 1.1). At height $k+1$, first the miner M_i generate $\mathsf{EPoSt.Rand}^{k+1}$ running the setup algorithm:

$\mathsf{EPoSt.Setup}(\mathsf{minerID}, k+1) \rightarrow \mathsf{EPoSt.Rand}^{k+1}$: this takes as input the tipset at height k and produces the election PoSt random seed as

$$\mathsf{EPoSt.Rand}^{k+1} = y$$

$$\text{with } (y, \pi(y)) \leftarrow \mathsf{VRF}_{\mathsf{sk}_i}(TR^k)$$

where sk_i is the secret key of the miner M_i.

Irene: *why don't we just use TR_j^{k+1} as EPoSt randomness used in the election of block j at height $k+1$??*

Then, the miner M_i uses the election PoSt random seed to create a list of $K = \lceil \mathsf{SampleRate} \times n_i \rceil$ sectors. Then, for each sector the miner M_i uses the seed gain and moreover the data stored in these sectors to create a list of challenge tickets $TC_{i,j}$ with $j \in [1, K]$. In short, the miner can attempt to generate a ticket for each of the sectors in the list. For each of these tickets, the miner checks if it is a "winning" ticket. When the miner has a non-negative number of winning tickets, he embeds each of them in the block. Henri: *this is how it works in practice but could be simplified to one ticket per block...*. More precisely, the miner runs the following set of algorithms:

$\mathsf{EPoSt.Sect.Sampl}(\mathsf{minerID}, R, k+1, \mathsf{EPoSt.Rand}^{k+1}) \rightarrow S_{i,1}, \ldots, S_{i,K}$:

Irene: *Check how this is currently implemented. Not clear from the spec!*

Assuming the list of sampled sectors is created by K independent and uniform sampling of i from $\{1, \ldots, n\}$,

$\mathsf{EPoSt.Ticket.Sampl}(\mathsf{minerID}, S_{i,j}, k+1, \mathsf{EPoSt.Rand}^{k+1}) \rightarrow TC_{i,j}^{k+1}$: the ticket sampling algorithm takes as input a sector $S_{i,j}$ and the ePoSt random seed $\mathsf{ePoSt.Rand}^{k+1}$. It uses the seed to generate random location from where read the data $\mathsf{answ}_i^{j,k+1}$ from the sector Si, j. Irene: *this is not very accurate, but I think it is a good abstraction to use here.* Then, it uses them to produce a challenge ticket as follows as

$$TC_{i,j}^{k+1} = \mathsf{H}(\mathsf{partialTicket})/\mathsf{maxTicket}$$

$$\text{and } \mathsf{partialTicket} = (\mathsf{EPoSt.Rand}^{k+1} || \mathsf{minerID} || j || \mathsf{answ}_i^{j,k+1})$$

Here $\mathsf{maxTicket}$ is the maximum value a challenge ticket can have, or $2^{len(\mathsf{H})}$.

$\mathsf{EPoSt.Ticket.Ver}(\mathsf{minerID}, S_{i,j}, k+1, PT^k, TC_{i,j}^{k+1}) \to 0/1$: A challenge ticket $TC_{i,j}^{k+1}$ is identified as winning ticket $TW_{i,j}^{k+1}$ if and only if $TC_{i,j}^{k+1} \leq \mathsf{Target}_{i,j}^{k+1}$ with

$$\mathsf{Target}_i^{k+1} = \frac{e \times s_i}{\mathsf{NetworkPower}_k} \times \frac{n_i}{\lceil \mathsf{SampleRate} \times n_i \rceil}$$

where s_i is the sector size for the miner and e is the expected number of winning tickets per round (a network parameter).

Note that

- on average an honest miner wins $e \times \frac{n_i s_i}{\mathsf{NetworkPower}_k}$ tickets;
- an honest miner wins the block $k+1$ with probability $1 - (1 - \mathsf{Target}_i^{k+1})^K$.

If a ticket is a winning ticket, the miner M_i runs $\mathsf{dVSC.Refresh}(\dots) \to (\pi_i^{j,k+1,R}, t)$ and includes the proofs $\pi_i^{j,k+1,R}$ and the winning ticket in the block B^{k+1}. It also includes the VRF proof. If the block is embedded in the longest-chain (see section 3.2), the miner gains a reward proportional to the number of winning tickets in their block (i.e., it gains $\mathsf{blockReward}^{k+1}/e$ reward for each ticket, where $\mathsf{blockReward}^{k+1}$ is fixed by the block reward function). In particular, a miner wins $\mathsf{blockReward}^{k+1} \times \frac{n_i s_i}{\mathsf{NetworkPower}_k}$ coins per produced block. Irene: *block reward function?*

2.2.4 Possible issues with Election PoSt

- If $\mathsf{Target}_{i,j}^{k+1} = \frac{e \times s_{i,j}}{\mathsf{NetworkPower}_k} \times \frac{n_i}{\lceil \mathsf{SampleRate} \times n_i \rceil} \geq 1$, each ticket is winning with probability 1.
- **Maximum miner size**. Consider an honest miner M_i with n active sectors and let p_i^k be the probability that one of its sectors is winning (ie, it is selected and it produces a winning ticket) in round k. Then, the probability of winning the block is equal to

  $$f_{p_i^k}(n) = 1 - (1 - p_i^k)^n$$

  (probability of at least 1 winning sector among n). Now we can see that for any p_i^k there is a maximum value of n, $n_{i,max}^k$, after which the probability $f_{p_i^k}(n)$ stays in practice the same (the smaller the p_i^k, the larger $n_{i,max}^k$). In conclusion, if at some point the total power in the network becomes stable (or it grows very slow), then it is rational for honest miners to have only a fixed number of sectors per identity.

  (For completeness: $p_i^k = \mathsf{Target}_\mathsf{i}^\mathsf{k} \times \mathsf{SampleRate}$).

- **Power Table Inflation** ("Not reporting faults attack"). What happens if a miner do not report a sector fault (or it deletes a sector)?

  Let n be the number of active sectors in the Power Table for the miner M_i. Assume that M_i is dishonest and has $n' = \mathsf{BadRate} \times n$ sectors that are not stored during the election process at round k.

  We want that this dishonest miner wins the block k with probability $<$ than a honest miner with n active sectors of the same size. Is this true?

Quick Answer: We can have this be true for any $\mathsf{BadRate}$ above a given threshold (we can choose this threshold!) with high probability if we take the minimum sector size and the sample rate accordingly to the formula below.

Analysis: Notice that the dishonest miner has not enough time to regenerate these sectors, so they can not be used to produce tickets. This implies that if one or more of the bad sectors are in the list of the sampled ones, the probability of the dishonest miner of winning a block is $<$ than the one of the honest miner. If no bad sectors are in the list of the sampled sectors used for the election, then the dishonest miner wins as an honest one. The probability of this happening is

$$P_{attackSuccess} = (1 - \mathsf{BadRate})^K$$

with $K = n \times \mathsf{SampleRate}$.

- Let's fix a value ϵ_{post}, that represents the fraction of sectors that a miner can delete (not report fault) and then we choose the **minimum miner size** n_{min} and the sample rate such that $P_{attackSuccess}$ is negligible (like equal to 2^{-10}) for any $\mathsf{BadRate} \geq \epsilon_{post}$. That is,

$$n_{min} \times \mathsf{SampleRate} \geq \frac{-\lambda}{\log_2(1 - \epsilon_{epost})}$$

  Example: say that the $n_{min} = 1000$ and $\mathsf{sampleRate} = 1/25$, thae a dishonest miner can delete the 1% ($\epsilon_{epost} = 0.01$) and win as an honest guy with probability larger than 0.66. If it deletes (and doesn't report) the 10% ($\epsilon_{epost} = 0.1$), then it wins as an honest guy woth probability 0.015.

- Notice that is $\mathsf{SampleRate} = 1$, the probability of this attack being successful is always 0.
- (**"Empty Blocks Attack**"). Even with a secure choice of parameters, a dishonest miner not reporting faults can create a disruption to the honest user because there will be more empty blocks. Indeed if $n' < n$ then the $\mathsf{NetworkPower}$ is larger that the total real persistent storage in the network and then we have $< e$ winning tickets in average per block. To avoid this we need a system to rescale down $\mathsf{NetworkPower}$. See Section 5.

Discussion (using only a subset of all the sectors for EPoSt). In the future we want to look at a leader election protocol where each miner M_i uses only a constant number, say K, of sectors (instead of all its sectors) to generate challenge tickets. This is necessary in practice since a miner has finite time to generate tickets. In this case, the setup algorithm can be modified adding the sampling of K random indices for choosing these sectors. Then, the miner M_i runs the ticket sampling and verifying algorithms only on those K randomly chosen sectors. Moreover, the target is modified in order to maintain the property that a miner wins proportionally to all the storage it has in the power table. For example, we can consider the following target:

$$\mathsf{Target}_{i,j}^{k+1} = \frac{L \times s_{i,j}}{\mathsf{NetworkPower}_k} \times \frac{n_i}{K}$$

Irene: *does this makes sense? The spec has a slightly different formula...* assuming $s_{i,j}$ is the same for all sectors

3 Consensus [draft]

Given a secure leader election as defined in the previous section, we can now define our consensus protocol.

3.1 Desiderata

Input: List of transactions and weighted list of participants
Output: Each participant outputs a chain
Security Properties Informal: (1) Chain growth (2) Chain Quality (3) Common prefix

3.2 Protocol details

3.2.1 Block Validity

We start by defining the validity rules for a block. Informally a block is valid if it contains a valid proof of eligibility as defined in Section **??** and all of the transactions are valid (i.e. signature correct and does not double-spend)Sarah: *is there anything else?*Henri: *needs a new randomness ticket, no duplicate election tickets in the proof. Also note that we allow for some invalid transactions, they just aren't counted* and it does not contain any *consensus fault* that we define now. Sarah: *add the pretty figures from the specs section 4.1.0.1 Consensus fault*

(1) double-fork mining fault: two blocks mined at the same epoch with the same proof. (2) time-offset mining fault: two blocks mined off of the same Tipset at different epochs (i.e. with different ChallengeTickets generated from the same input ticket). (3) parent-grinding fault: one block's parent is a Tipset that provably should have included a given block but does not. While it cannot be proven that a missing block was willfully omitted in general (i.e. network latency could simply mean the miner did not receive a particular block), it can when a miner has successfully mined a block two epochs in a row and omitted one. That is, this condition should be evoked when a miner omits their own prior block. When a miner's block at epoch e + 1 references a Tipset that does not include the block they mined at e both blocks can be submitted to prove this fault. Sarah: *need to add somewhere that blocks that are not created at the right time are not valid*

3.2.2 Weighting function

Forks can happen in EC and the chain selection algorithm gives a formula for miners to decide which chain is the "heaviest". Heaviest chain should reflect the most amount of storage provided. For a blockchain $C = \{\text{TipSet}_0, \cdots, \text{TipSet}_l\}$, the weight $\mathbf{w}(C)$ is given by the following: Sarah: *this is the tipset version (probably to be removed)*

$$\mathbf{w}(C) = \mathbf{w}(C[-1]) + \mathbf{wFun}(totPower(C))2^8 + \frac{(\mathbf{wFun}(totPower(C)) \times |\text{TipSet}_l|) \times wRatio_{num}2^8)}{(d \times wRatio_{den})} \tag{1}$$

where we have: $C[-1] = \{\text{TipSet}_0, \cdots, \text{TipSet}_{l-1}\}$; d, $wRatio_{num}$, $wRatio_{Den}$ are constant; **wFun** is a function Sarah: *@henri any particular properties that it should satisfy?* Henri: *I would*

treat wRatio as a single fraction. Split in spec for ease of implementation cross languages. We want wFun to be monotonically increasing. Thereafter, I chose one with a small derivative such that functions with reelatively same amount of storage behind them can have relatively same amount of weight, so miners can choose which to back. Useful in selfish mining context. e.g. **wFun=log**; and *totpower* represents the sum of the weight of the participants Sarah: *@henri is this right?*Henri: *yes*. Sarah: *@henri a couple of word to get some intuition on the above formula would be appreciated :P* Henri: *Yes. I would start with a more general subsection here on what we want out of the weight function, then dive into Filecoin's choice for one. Note that this is an area where refinements could be great. See https://observablehq.com/d/3812cd65c054082d for most of the bg behind the current one. But intutition is as follows: 1- weight function needs to promote chain with most underlying storage: goal of FIL, added security given VRC. Hence the wFun. 2- It needs to promote collab between miners to prevent selfish mining in the short term (ie before honest miners are slashed from the selfish PT). Hence counting total num of blocks in tipset to reward majority (honest) chain. 3 - Ratio tying 1 and 2 must always maintain that contribution to weight by storage is greater than by number of blocks, lest a lighter selfish chain take over (with sybils). 4- Choice of wFun depends on threat model of the VRC (ie ability for power to be bought rented, etc) but the smaller wFun's derivative, the more you allow for competition between chains (and potential reversals in case of attacks). This is hokey*. Additionally we define $\mathbf{w}(\text{TipSet}_0) = 0$. Sarah: *again is this correct?*Henri: *no. tipset_0 is the genesis block so it would have weight of bootstrapped miners included in PT at launch, + 1 for the single block in genesis tipset. But this works WLOG (save for bootstrapping issue on how to get power on if there is none to start*.

Sarah: *this is the longest chain part:*

$$\mathbf{w}(C) = \mathbf{w}(C[-1]) + 1 + \mathbf{wFun}(totPower(C))$$

3.2.3 Fork Choice Rule

We can now define our fork choice rule, a miner will choose the heaviest chain:

$$\mathsf{ForkChoiceRule}(C_0, \cdots, C_p) = (\arg\max_i(\mathbf{w}(C_i)))$$

In the case where two chain have the same weight, we use the random beacon introduced in the previous section to deterministically break the tie.

3.2.4 Chain extension

When a miner is elected leader for a round, they get the opportunity to extend the blockchain. They do so as follows (*):

1. Collect all the different valid chains received $(C_0, \cdots, C_p)$
2. Run TipSet $\leftarrow (\mathsf{ForkChoiceRule}(C_0, \cdots, C_p)).leaves$
3. Create a block B such that $B.parents \leftarrow$ TipSet (i.e. the block is created "on top" of the heaviest chain) and broadcast it to the network

Sarah: *need to add somewhere the case where no miners are elected leader (probably in the leader election part)* Henri: *yes. Should be straightforward. Most of it is in the randomness derivation (using the round number).*

If a miner is not elected leader in a round, they do the following (**):

1. Wait $\mathbf{T}_{wait}$
2. if a block has been received then update chain and run leader election for next height (*)
3. else: update leader election according to 2
   - if elected leader go back to (*)
   - else go back to (**)

4 Verifiable Storage Commitments [Draft]

Given the way Filecoin is designed, space is the resource which defines miner's power. As we described above, both leader elections and consensus rely on weights, determined by storage committed to the system. It is then crucial to ensure that the weight associated to each miner reflects his actual percentage of storage over time. Inspired by the work by Azouvi and Stern Sarah: *this work is not public yet :P*, we tailor here their Verifiable Resource Commitments (VRC) primitive in our setting. First of all, we consider space as a resource, according to the following definition:

Definition 4.1 (Resource)**.** A resource r is a quantifiable entity which can exist in only one of two states, $\mathsf{available}$ and $\mathsf{committed}$, denoted with $\mathsf{r.state}$.

Our unit of space to be considered is called *sector* and will be denoted as S.

In order to be committed, a sector S be in the $\mathsf{available}$ state. We assume the system is dynamic, meaning that each miner can modify the amount of storage he is committing to the chain by augmenting it (which means, transforming some amount of space he has in $S.\mathsf{available}$ form to $S.\mathsf{committed}$) or reducing it (which means transforming some amount of space he has from $S.\mathsf{committed}$ to $S.\mathsf{available}$).

We assume the existence of an economic system which regulates and quantifies the cost of acquisition of a given resource (in our case, space). Given a sector S we assume the existence of an associated cost cost_S. This means that any participant P_i is supposed to pay a cost $c \geq \mathsf{cost}_S$ for committing a sector S to the chain.

A Verifiable Resource Commitment scheme allows for dynamically account the state of sectors of space over time, by tracking space commitment.

A Verifiable resource Commitment (VRC for short) is defined as a tuple of PPT algorithms ($\mathsf{VRC.Setup}, \mathsf{VRCCommit}, \mathsf{VRC.Verify}_{\mathsf{Com}}, \mathsf{VRC.Refresh}, \mathsf{VRC.Verify}_{\mathsf{Ref}}$), defined as follows:

$\mathsf{VRC.Setup}(1^\lambda, C, |S|) \rightarrow \mathsf{pp}$. The setup algorithm takes as input a security parameter λ, a chain C and a sector size $|S|$ and outputs a set pp of public parameters, including the proving period t_S which each participant has for proving commitment refreshment of S space and a pair (msk, mpk) of master keys; master secret key has to be destroyed. The Setup algorithm also initialize the initial storage commitment of each miner to 0.

$\mathsf{VRC.Commit}(B^k, S_{i,j}.\mathsf{available}) \rightarrow \pi^k_{i,j}$. The commitment algorithm takes as input a block B^k in the current chain C, an available sector $S_{i,j}$ identified by the j-th sector of miner $i \in \mathcal{I}$ (where i could refer to a public key pk_i). It outputs a proof of commitment $\pi^k_{i,j}$. This algorithm is used by a miner P_i to commit a sector $S_{i,j}.\mathsf{available}$ to all sub-chains of C containing B^k.

$\mathsf{VRC.Verify}_{\mathsf{Com}}(mpk, S_{i,j}.\mathsf{id}.\mathsf{committed}, \pi^k_{i,j}, B^k) \rightarrow 0/1$. The commitment verification algorithm takes as inputs the master public key mpk, a sector identifier $S_{i,j}.\mathsf{id}$ amount of a sector space $S_{i,j}$ in $\mathsf{committed}$ state, indexed by j and belonging to a miner identified by i. It also gets as input the corresponding proof of commitment $\pi^k_{i,j}$, the block B^k of the current chain C where $S_{i,j}$ was supposed to be committed and outputs either 0 (reject) or 1 (accept).

$\mathsf{VRC.Refresh}(S_{i,j}.\mathsf{committed}, B^k, t_S, \mathsf{chal}) \rightarrow (\pi^{k,R}_{i,j}, t)$ The refresh algorithm takes as input a sector $S_{i,j}$ in a $\mathsf{committed}$ state, a block B^k of the current chain C, the proving period t_S and a set of challenges chal. It uses the answers answ to the challenges chal to output a concise refresh-proof $\pi^{k,R}_{i,j}$ that $S_{i,j}$, whose commitment to chain C at block B^k produced $\pi^k_{i,j}$, is still committed to C as of block $B^{k'}$, with B^k as an ancestor in the chain. It also (implicitly) output a time t needed to generate the proof. In order to be valid, it must be $t \leq t_S$.

$\mathsf{VRC.Verify}_{\mathsf{Ref}}(mpk, \mathsf{amt}_{i,j}.\mathsf{id}, (\pi^{k',R}_{i,j}, t), B^{k'}, B^k, t_S) \rightarrow 0/1$. The refresh verification algorithm takes as inputs the master public key mpk, an identifier $\mathsf{amt}_{i,j}.\mathsf{id}$ for the sector $S_{i,j}.\mathsf{committed}$, its proof of refreshment $\pi^{k,R}_{i,j}$ with the time t needed to produce it, the block $B^{k'}$ of the current chain C where $S_{i,j}$ was supposed to be originally committed, a given block B^k that has $B^{k'}$ as an ancestor and a proving period t_S which determines the time frame over which the proof must be submitted. It outputs either 0 (reject) or 1 (accept). Note that proofs $(\pi^{k,R}_{i,j}, t)$ with $t > t_S$ are not even verified and considered invalid (and thus rejected).

Security

We want a Verifiable Storage Commitment to satisfy the following properties:

Commitment Correctness: Given a security parameter λ, we have that

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Com}}(mpk, S_{i,j}.\mathsf{id}, \pi^k_{i,j} B^k) | \pi^k_{i,j} \leftarrow \mathsf{VRC.Commit}(B^k, S_{i,j}.\mathsf{available})] = 1 - \mathsf{negl}(\lambda)$$

Refreshmentment Correctness: Given a security parameter λ, we have that

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Ref}}(mpk, S_{i,j}.\mathsf{id}, (\pi^{k',R}_{i,j}, t), B^k, B^{k'}, t_S) | (\pi^{k',R}_{i,j}, t) \leftarrow \mathsf{VRC.Refresh}(S_{i,j}.\mathsf{committed}, B^k, B^{k'}, t_S) \wedge t \leq t_S, \mathsf{chal}] = 1 - \mathsf{negl}(\lambda)$$

Commitment Soundness: Given a security parameter λ, we have that

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Com}}(mpk, S_{i,j}.\mathsf{id}, \pi^k_{i,j}, B^k) | \pi^k_{i,j} \leftarrow \mathsf{VRC.Commit}(B^k, S_{i,j}..\mathsf{committed})] = \mathsf{negl}(\lambda)$$

and

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Com}}(mpk, S_{i,j}.\mathsf{id}, \pi^k_{i,j}, B^k) | \tilde{\pi}^k_{i,j} \leftarrow \mathsf{VRC.Commit}(B^k, \tilde{S}_{i,j}.\mathsf{available} \wedge |\tilde{S}_{i,j}| < |S|] = \mathsf{negl}(\lambda)$$

Refreshment Soundness: Given a security parameter λ, we have that

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Ref}}(mpk, S_{i,j}.\mathsf{id}, (\pi_{i,j}^{k',R}, t) | t > t_S] = \mathsf{negl}(\lambda)$$

and

$$Pr[1 \leftarrow \mathsf{VRC.Verify}_{\mathsf{Ref}}(mpk, S_{i,j}.\mathsf{id}, (\tilde{\pi}_{i,j}^{k',R}, t) | (\tilde{\pi}_{i,j}^{k',R}, t) \leftarrow \mathsf{VRC.Refresh}(\tilde{S}_{i,j}, B^k, B^{k'}, t_S, \mathsf{chal}) \wedge$$

$$\wedge |\tilde{S}_{i,j}| < S_{i,j} \wedge t \leq t_S] = \mathsf{negl}(\lambda)$$

.

As a consequence of not submitting a valid proof within a time t_S for a sector S (which means, either submitting a valid proof in time $t > t_S$ or submitting an invalid proof in time $t \leq t_S$ or not submitting a proof at all), the associated amount of space S is freed.

Resource Accounting

We set $\mathcal{I}$ to be the set of participants, and we assume that participant $i \in \mathcal{I}$ claims to have been committing a set of sectors $\{S_{i,1}, ..., S_{i,|J|}\}$ (amt. In order to deal with completeness, we first need to define a power evaluation algorithm which depends on the VRC.

$\mathsf{VRC.PowerEval}(i, C, \{S_{i,j}.\mathsf{id}\}_{j \in J}, \{\pi_{i,j}^k\}_{j \in J, k \in K}, \{(\pi_{i,j}^{k',R}, t_{k'})\}_{j \in J, k' \in K'} \{B^k\}_{k \in K}, \{B^{k'}\}_{k' \in K'}) \rightarrow \mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})$.
The power evaluation algorithm takes as input a participant identity i, the current chain C, all the different identifier of the committed sectors $\{S_{i,j}\}_{j \in J}$, all the commitment proofs $\{\pi_{i,j}^k\}_{j \in J, k \in K}$ all the refreshment proofs $\{\pi_{i,j}^{k',R}\}_{j \in J, k' \in K'}$, the blocks $\{B^k\}_{k \in K}$ in which the commitments happened and the blocks $\{B^{k'}\}_{k' \in K'}$ where all the refreshments happened. It outputs the correspondent power $\mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})$ for participant identified with i at block $B^{k'_{\mathsf{max}}}$, where $k'_{\mathsf{max}} = \mathsf{max}\{k' | k' \in K'\}$.

Note that the total power committed to the chain C is defined as $\mathsf{Power}^C(B^{k'_{\mathsf{max}}}) = \sum_{i \in \mathcal{I}} \mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})$ where $\mathcal{I}$ is the set of all the participants in the system, at the block $B^{k'_{\mathsf{max}}}$.

It follows that, on one hand a given participant $i \in \mathcal{I}$, can prove his commitment to the system through time by providing valid the commitment and refreshment proofs. On the other hand, the network is able to verify the commitment of participant $i \in \mathcal{I}$ through time by verifying his commitment and refreshment proofs, adjusting his power accordingly. More formally, we define space power and space fraction as follows:

Definition 4.2 (Space Power). Given a chain C and a VRC over it, the space power $\mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})$ of a participant $i \in \mathcal{I}$ is a provable accounting of the total amount of space that i has committed to C as of block $B^{k'_{\mathsf{max}}}$). Moreover, $\mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})$ can be deterministically derived (and thus checked) from transactions on C.

Definition 4.3 (Space Fraction). We call space fraction at block $B^{k'_{\mathsf{max}}}$ the ratio between the space power of the participant i and the total space power of the chain, namely:

$$\mathsf{Space.Fraction}_i^C(B^{k'_{\mathsf{max}}}) = \frac{\mathsf{Power}_i^C(B^{k'_{\mathsf{max}}})}{\mathsf{Power}^C(B^{k'_{\mathsf{max}}})}$$

5 Power Table

5.1 Definition

How to Map and Account $\{S_{i,j}\}_{i\in\mathcal{I},j\in J}$ in the Power Table According to PoST Outcomes

The Power Table (PT) is the tool we use for tracking space commitment on a chain C. From the practical point of view it can be see as a bullettin board with a $\mathsf{state} = \{\mathsf{state}_i\}_{i\in\mathcal{I}}$ which collects all the different storage power of the different miners $i \in \mathcal{I}$. Power table is maintained and updated locally by each miner at every block, according to what is on-chain. The updating process can be defined via a set of three algorithms ($\mathsf{PT.Add}, \mathsf{PT.Confirm}, \mathsf{PT.FaultDeclare}$) as follows:

$\mathsf{PT.Add}(C, \mathsf{state}, B^k) \rightarrow \mathsf{state}$. The power adding algorithm takes as input a chain C, a current state state and the current block B^k. It outputs an updated state state as follows: it checks all the proofs of commitment $\{\pi^k_{i,j} \leftarrow \mathsf{VRC.Commit}(B^k, S_{i,j}.\mathsf{available})\}_{i\in\mathcal{I}}$ in B^k. For each valid proof $\pi^k_{i,j}$, it updates $\{\mathsf{state}_i = \mathsf{state}_i + |S_{i,j}|\}$. It outputs the updated state $\mathsf{state} = \bigcup_i \mathsf{state}_i$.

$\mathsf{PT.Confirm}(C, \mathsf{state}, B^{k'}, B^k, \mathsf{timeout}) \rightarrow \mathsf{state}$. The power confirmation algorithm takes as input a chain C, a past block $B^{k'}$, a state state, the current block B^k and a timeout $\mathsf{timeout}$. It checks refreshments proofs $\pi^{(\cdot),R}_{i,j} \leftarrow \mathsf{VRC.Refresh}(S_{i,j}.\mathsf{committed}, B^{(\cdot)}, t_S, \mathsf{chal})$ from block $B^{k\prime}$ and block B^k and updates the state $\mathsf{state} = \bigcup_i \mathsf{state}_i$ accordingly. Note that, if the power confirmation algorithm is run at every block, then we have $k' = k - 1$

$\mathsf{PT.FaultDeclare}(C, \mathsf{state}, B^k, i, j) \rightarrow \mathsf{state}$. The fault declaration algorithm is run by a user i who declares at height k of the chain C that his sector S_j is faulty. The output is a new state $\mathsf{state} \leftarrow \mathsf{state} - |S_j|$.

$\mathsf{PT.FallBack}(C, \mathsf{P}, B^k, \{\mathsf{state}_i\}_{i\in\mathcal{I}}, \{\pi^{\tilde{k}}_{i,j}, \pi^{\tilde{k},R}_{i,j}\}_{i\in\mathcal{I},j\in J,\tilde{k}\in\mathsf{P}}) \rightarrow \mathsf{state}$. The fallback algorithm takes as input the current chain C, a proving period P (consisting of a set of consecutive blocks in C), the current block B^k, and all the states and proofs of all the identities $i \in \mathcal{I}$. It outputs a new state state.

We want a Power Table to satisfy PT-Correctness and PT-Soundness defined as follows:

PT-Correctness: After running $\mathsf{PT.Add}$, $\mathsf{PT.Confirm}$ and $\mathsf{PT.Fallback}$, each honest miner has his power accounted in the power table with all but negligible probability. More formally:

$$Pr[\exists i \in \mathcal{I} \text{ s.t. } \mathsf{state}_i < (1-\epsilon)\mathsf{Power}^C_i(B^k)] = \mathsf{negl}(\lambda)$$

PT-Soundness: After running $\mathsf{PT.Add}$ and $\mathsf{PT.Confirm}$, and eventually $\mathsf{PT.FaultDeclare}$, the total actual storage committed to the chain C at the moment block B^k is appended to the chain is greater or equal that $(1-\epsilon)\sum_i \mathsf{state}_i$ with all but negligible probability. Moreover, the power table reflects, by a factor $(1-\epsilon)$, the amount of storage each miner $i \in \mathcal{I}$ is actually committing to the chain C with all but negligible probability. More formally, we have:

$$Pr[\exists i \in \mathcal{I} \text{ s.t. } \mathsf{Power}^C_i(B^k) < (1-\epsilon)\mathsf{state}_i] = \mathsf{negl}(\lambda)$$

5.2 Protocol Details

Adding Power in the PT

In our protocol, the instantiation of PT.Add is as described in the definition, where the proof $\pi^k_{i,j} \leftarrow \mathsf{VRC.Commit}(B^k, S_{i,j}.\mathsf{available})$ is a proof of replication.

Confirming Power in the PT

The instantiation of PT.Confirm is currently as follows:

$\mathsf{PT.Confirm}(C, \mathsf{state}, B^k, \mathsf{timeout}) \rightarrow \mathsf{state}$. The power confirmation algorithm takes as input a chain C, a current state state, the current block B^k and a timeout timeout. It checks the refreshment proofs $(\pi^{k,R}_{i,j}, t) \leftarrow \mathsf{VRC.Refresh}(S_{i,j}.\mathsf{committed}, B^{k_{i,j}}, B^k, t_S, \mathsf{chal})$ at height k produced in the EPoST step. If $\mathsf{VRC.Verify}_{\mathsf{Ref}}(mpk, S_{i,j}.\mathsf{id}, (\pi^{k,R}_{i,j}, t), B^{k_{i,j}}, B^k, t_S) \rightarrow 1$, it sets $\mathsf{state}_i \equiv \mathsf{state}_i$ (namely, it confirms the leader's space amount in the power table). Otherwise it updates $\mathsf{state}_i = \mathsf{state}_i - |S_{i,j}|$ Luca: *should we rather say fault report? If so, Over the entire storage of the leader?* The algorithm outputs $\mathsf{state} = \bigcup_i \mathsf{state}_i$

Fall Back Mechanism in the PT (Surprise PoSt)

The fallback mechanism $\mathsf{PT.Fallback}(C, \mathsf{P}, B^k, \{\mathsf{state}_i\}_{i\in\mathcal{I}}, \{\pi^{\tilde{k}}_{i,j}, \pi^{\tilde{k},R}_{i,j}\}_{i\in\mathcal{I}, j\in J, \tilde{k}\in\mathsf{P}}) \rightarrow \mathsf{state}$ in our protocol operates as follows:

- At every block, $|\mathcal{I}|/|\mathsf{P}|$ miners (randomly chosen), are challenged on a pre-determined fraction F of blocks from their committed storage (according state_i for each $i \in \mathcal{I}$) Recall that P is the proving period and $\mathcal{I}$ is the set of all miners. Irene: *I think this should be active sectors only, not all sectors.*
- Each of the challenged miners has a time t_{max} to sumbit all their proof on-chain (as a transaction). Note that this time t_{max} has to take into account both proving and propagation time. Right now we consider $t_{\mathsf{max}} =$ Luca: *XXX*

5.2.1 Problems with Surprise PoSt

- **Polling time**. An important observation to make is that, since our security is based on a timing assumption, and in particular on the fact that re-generating the 20% of a layer takes at least t_{SDR} time, in order for us to use the current fall-back mechanism we should set $t_{\mathsf{max}} > t_{SDR}$. If this is not the case, the aforementioned fallback mechanism can not guarantee persistent storage, breaking PT soundness.
  - An example of SDR parameters for which $t_{SDR} = 30$ seconds (1 block):
    Values to hash per node: 37,
    sectorsize gib: 32
    porep snark partitions: 32 (sha-pedersen),

- An example of possible SDR parameters for which $t_{SDR} = 3 \times 30$ seconds (3 blocks):
  Values to hash per node: 37,
  sectorsize gib: **96**
  porep snark partitions: 32 (sha-pedersen),
  Encoding is 3 × slower. The rest should stays the same
- An example of possible SDR parameters for which $t_{SDR} = 6 \times 30$ seconds (6 blocks):
  Values to hash per node: **54**,
  sectorsize gib: **96**
  porep snark partitions: **64** (sha-pedersen),
  Encoding is 3 × slower. SNARK times is 2× slower.

- **PT-soundness**. An important observation to make here is that if we use Surprise PoST as a fallback mechanism, then proving PT-soundness is infeasible. Indeed, assuming t_{max} correspond to T blocks, at each block at height k we expect to receive the Surprise PoSTs of the $|\mathcal{I}|/|\mathsf{P}|$ miners that got randomly challenged at height $k - T$. This means that, if we call W the number of miners posting an EPoST at height k, at height k we will (eventually) know that $|\mathcal{I}|/|\mathsf{P}| + W$ miners persistently store the space that they are accounted for in the PT, but we can not say anything about the rest of the miners. Of course we could achieve PT-soundness if we assume that everyone proves all his storage (either winning an EPoST or submitting a Surprise PoST) every T blocks, but this is not practical in terms of scalability.

Other idea for $\mathsf{PT.FallBack}$: Periodical Rescaling

Instead of the current approach, what we can do is doing a local periodical rescaling (at the end of each proving period P) to update the state state_i of each miner $i \in \mathcal{I}$ according to the frequency with which i posts EPoST proofs onchain. This approach crucially depends on how EPoST and the correspondent reward is supposed to work.

We think that the updates that aim to ensure that the state of the power table is up to date can be handled differently using periodical rescaling:

Periodical Rescaling: There are periodical checkpoints where each miner locally updates his own power table according to the frequency of each miner's EPoST proofs posted onchain. A bit more formally:

- We call $|\mathsf{P}|$ the number of rounds in a proving period (after how many rounds the chain must check each miner's state).
- state_i is the active power of miner $i \in \mathcal{I}$ reported in the Power Table.
- At the end of each proving period P, each miner locally check how many times $i \in \mathcal{I}$ miner i posted an EPoST onchain. We call this number e_i
- On expectation it should be $e_i = |\mathsf{P}| \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}}$. If $e_i < |\mathsf{P}| \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} - \mathsf{threshold}$ (for some threshold value that we decide, for example via the three sigma rule), then the miner gets slashed (or rescaled).

5.2.2 Nuances with Periodical Rescaling

1. **Late Commits**. What happens if a miner commits right before the checkpoint? An honest one will be actually rescaled as a miner having less space. It seems that in the following period of time he will win more than he should according to the (incorrect) power table, so his storage will be augmented in the following checkpoint. Note that this is not affecting PT-soundness, since it is actually a (temporary) under estimation of power rather than an overestimation (as it could happen with Surprise PoST).
2. **Global Vs Per-Miner Rescaling**. Instead of a "per-miner" periodical power refresh, we can have a global power table refresh, meaning:
   - At each checkpoint, depending on the total number of EPoST proofs posted on-chain since the former checkcpoint, the total power in the power table is refreshed. Individual miners' power is re-scaled accordingly (if for instance total power goes down by a 10%, each miner's power will be cut by 10%). This approach has pros and cons: on one hand, it prevents targeted attacks to big miners, but on the other hand it does not fairly account for honest miner's power (if the total power gets cut down by X% all miners, including honest ones, will be cut down by X%. On the other hand, a dishonest miner will be cut down less than he should be).

Claim: If LE is a secure Leader Election Mechanism and $\mathsf{VRC} = (\mathsf{VRC.Setup}, \mathsf{VRCCommit}, \mathsf{VRC.Verify}_{\mathsf{Com}}, \mathsf{VRC.Refresh}, \mathsf{VRC.Verify}_{\mathsf{Ref}})$ is sound, then our instantiation of the power table is satisfies (average) PT-correctness and PT-soundness against rational adversaries.

Proof: Saying that our instantiation of the power table is secure, according to the power table security definition, means that $Pr[\mathsf{Power}^{C}_{\mathcal{A}_r}(B^k) < (1-\epsilon)\mathsf{state}_{\mathcal{A}_r}] = \mathsf{negl}(\lambda)$.

One thing to note is that in our instantiation of the power table we can not prove security against a generic adversary: indeed, it is always possible for an adversary $\mathcal{A}$to not producing any ticket in an EPoST round without being caught. Nevertheless, in the context of blockchain systems it is normal to consider rational adversaries who want to maximize their revenue, and so it is fine for us to consider security against such adversaries. If we call $\mathsf{Rev}_i(S)$ the revenue for a miner i declaring to store $S = \{S_1, ..., S_n\}$, then for a rational adversary $\mathcal{A}_r$ (whose aim is to maximize $\mathsf{Rev}_{\mathcal{A}_r}(S)$) what we want is:

$$Pr[\mathsf{Power}^{C}_{\mathcal{A}_r}(B^k) < (1-\epsilon)\mathsf{state}_{\mathcal{A}_r}] = \mathsf{negl}(\lambda)$$

.

The key observation here is that, given how EPoST works, the way $\mathcal{A}_r$ has to maximize her probability of getting a block reward is to produce as many tickets as she can (as the more tickets she has, the more likely it is that she gets winning tickets) Luca: *This is true with the current description in the specs, but not true anymore if we have to send just one ticket. It seems we have to choose which EPOST version we want to prove, or if we rather want to prove each alternative*. Since producing a ticket TR_j means answering challenges on a sector S_j, this means that an adversary is willing to answer challenges on all her sectors. The security property of the proof system ensures that the probability for an adversary to answer challenges on a sector of which an she is storing less than a $(1-\epsilon)$ fraction is negligible.

In other words, we use the rational assumption to claim that the adversary is actually answering challenges on all her sectors, and we are using the Space Hardness property of the proof of space and its timing assumption to argue that if the adversary answers a set of challenges on a sector correctly, she is actually storing the sector itself with all but negligible probability.

5.3 Issues and Observations

Here a list of issues we encountered along the way it would be useful to agree on

Sector Accountability: On aspect it was brought up is that with some frequency, each miner should prove the accountability of each sector he has (and is reflected in the power table). Current formulation of the power table, with periodical rescaling, does not provide this feature. Indeed, the periodical rescaling aims to keep under control the total storage each miner is persistently committing to the network (and thus, by consequence, the total storage committed to the network). In practice, we do not know if a particular sector $S_{i,j}$ is persistently stored or not. Nevertheless, we think this is not a property which is needed for consensus. On the other hand, it is a desirable feature for storage market, and that's why we propose to use scheduled post to address this. In this case, we should not provide persistent storage evidence, but we would give periodical (granular) proof of data possession.

Onchain Footprint Adding Rescaling: We note that introducing rescaling does not influence blockchain footprint, since it is a local procedure that only needs to look at the last proving period's blocks and no proof needs to be posted onchain.

6 [Analysis]: How Likely is to to Fake SPoST?

In this paragraph we try to evaluate how likely it is to fake a SPoST for a dishonest miner. We briefly recall how SPoST currently work:

P is the proving period and $\mathcal{I}$ is the set of all miners for the period. Irene: *what about new miners?*

- We assume to have $M = |\mathcal{I}|$ active miners and $n = |\mathsf{P}|$ to be the number of blocks in a proving period P.
- At every block, $|\mathcal{I}|/|\mathsf{P}|$ miners (randomly chosen), are challenged on a pre-determined fraction F of blocks from their committed storage (according state_i for each $i \in \mathcal{I}$).
- Each of the challenged miners has a time t_{max} to sumbit all their proof on-chain (as a transaction). This means that, every time a miner i gets challenged, he has to prove $F \cdot \mathsf{state}_i$ blocks. Note that this time t_{max} has to take into account both proving and propagation time. Right now we consider $t_{\mathsf{max}} =$ Luca: *XXX*

We assume a dishonest miner j to keep only a fraction X of the declared blocks (meaning, a total number of $X \cdot \mathsf{state}_j$).

6.0.1 |P| is not long enough to allow (even partial) reconstruction

This is the simplest case, which in out setting (with SDR) would mean $t_{SDR} > n$ blocks time.

The protocol can be analyzed as follows:

Sampling Probability: At each block, a miner i will be sampled by SPoST with probability $p = \frac{M/n}{M} = \frac{1}{n}$. This means that the probability for each miner i to be sampled k times during the proving period P is $\tilde{p} = \binom{n}{k} p^k \cdot (1-p)^{n-k} = \binom{n}{k} \left(\frac{1}{n}\right)^k \cdot \left(\frac{n-1}{n}\right)^{n-k}$.

Dishonest Success Probability: Once a miner j is chosen, he has to prove $F \cdot \mathsf{state}_j$ blocks sampled at random. Is he is keeping only $X \cdot \mathsf{state}_j$ blocks, his probability of being lucky and being challenged on sectors he is keeping only is $p_{\mathsf{pass}} = \left(\frac{X \cdot \mathsf{state}_j}{\mathsf{state}_j}\right)^{F \cdot \mathsf{state}_j} = X^{F \cdot \mathsf{state}_j}$.

If we define the following events

- $A_k^j =$ miner j gets challenged k times in a proving period
- $B_k^j =$ miner j provides k valid SPoST

What we can evaluate is the probability of a dishonest miner j to be challenged k times in a proving period passing the all the proofs as the probability of the event $C_k^j = A_k^j \cap B_k^j$.

We have that

$$Pr(C_k^j) = Pr(A_k^j \cap B_k^j) = Pr(A_k^j) \cdot Pr(B_k^j) = \binom{n}{k} \left(\frac{1}{n}\right)^k \cdot \left(\frac{n-1}{n}\right)^{n-k} \cdot (X^{F \cdot \mathsf{state}_j})^k$$

It follows that the probability for a dishonest miner j to fake his storage, under the aforementioned assumptions if the probability of the event $C = C_0^j, C_1^j \cup ... \cup C_n^j$. We have

$$Pr(C) = Pr(C_0^j \cup C_1^j \cup ... \cup C_n^j) \leq \sum_{k=1}^{n} Pr(C_k^j) = \sum_{k=1}^{n} \binom{n}{k} \left(\frac{1}{n}\right)^k \cdot \left(\frac{n-1}{n}\right)^{n-k} \cdot (X^{F \cdot \mathsf{state}_j})^k$$

6.1 PT-soundness with Surprise PoSt (Irene)

Assume that the PT is frozen for each proving period, then the probability that the Network Power is wrong and we don't see it for the period is less or equal to the probability that there exist one miner M_i that is storing only $\epsilon_{spost} n_i$ active sectors (n_i is the declared one) and passes surprise PoSt. This miner passes the surprise PoSt if (1) it is never chosen or (2) it is chosen and it is lucky (that is it gets challenges only on the sectors it actually has). So, with n_{min} as minimum miner size we have that

$$\text{soundness error} \leq \left(1 - \frac{1}{N} + \frac{1}{N}\left(1 - \epsilon_{spost}\right)^{F n_{min}}\right)^N$$

6.2 Quality assurance

Scenario (Surpise PoSt): there is only 1 sector size for the entire network, there n sectors in the network, miner i has n_i sectors. At each block we sample m miners (miner i is sampled with probability $\frac{n_i}{n}$), each sampled miner must generate and post PoSt proofs for k sectors sampled uniformly at random.

Bad miner: a miner i that has more that δn_i fault sectors and he doesn't report it.

Fault sector: a sector where a fraction larger that ϵ is not stored.

1) What's the probability that when a bad miner is sampled, it passes the PoSt check (assuming he does NOT reconstruct any of the faulted sectors)?

$$p_1 = (1 - \delta + \delta(1 - \epsilon)^c)^k$$

where c is the number of challenges per sector. **Notice**: this is the same formula that give you the probability of a bad guy to pass **WindowPost** (assuming it does not reconstruct[1])

2) What's the probability that after ℓ blocks we don't see a bad guy that never reconstruct?

$$p_2 = \left(1 - \frac{n_i}{n} + \frac{n_i}{n} p_1\right)^\ell$$

[1]The non-reconstruction condition is satisfied if we can prove/argue that *Regenerating a fraction larger than ϵ of a replica is more expensive than storing the entire replica for W epochs*, when W is the frequence of the WindowPost.

6.3 [Analysis] Expected Number Of Challenge Rounds Needed to Prove Almost All Sectors

In this paragraph we want to analyze the *expected* number challenge rounds needed for a miner to be challenged on (almost) all his sectors.

Model

- A miner has N sectors in the Power Table
- Each miner, every time he is challenged, has to submit a proof on a subset of size $K < N$ of his sectors.
- We assume $K = \frac{N}{\alpha}$ for some fixed $\alpha > 1$. (meaning, we are implicitly assuming that all the miners are challenged on a fixed percentage of their sectors. Said in another way, we imagine N to change for each miner [and so does K], but not α. This is not ideal but in principle it can be adjusted assuming max miner size according to the maximum number of sectors which can be challenged per proving period).

How many times a miner has to be challenged in order to prove all his sectors?

At each round of challenges, a random subset of $K = \frac{N}{\alpha}$ sectors is sampled among the N sectors that belong to miner. We define $\mathsf{ChalSec}^j$ as the number of already challenged sectors after round j and $\mathsf{UnchalSec}^j$ as the number of still unchallenged sectors after round j. We start with $\mathsf{ChalSec}^0 = 0$ $\mathsf{UnchalSec}^j = N$. We have the following:

Round 1: After the first round, we trivially have $\mathsf{ChalSec}^1 = K$ and $\mathsf{UnchalSec}^1 = N - K$ (all the sampled sectors were unchallenged and become challenged).

Round 2: Given that the probability of sampling a unchallenged sector is $\frac{K}{N}$, the expected number of unchallenged sectors which gets sampled in round 2 is $K \cdot \frac{N-K}{N} = K \cdot (1 - \frac{1}{\alpha}) = K \cdot \frac{\alpha-1}{\alpha}$. It follows that $\mathsf{ChalSec}^2 = K \cdot (1 + \frac{\alpha-1}{\alpha})$ and $\mathsf{UnchalSec}^2 = N - K \cdot (1 + \frac{\alpha-1}{\alpha})$.

Round 3: Given that the probability of sampling a unchallenged sector is $\frac{N-K\cdot(1+\frac{\alpha-1}{\alpha})}{N}$, the expected number of unchallenged sectors which gets sampled in round 3 is

$$K(\frac{N-K\cdot(1+\frac{\alpha-1}{\alpha})}{N}) = K(\frac{N-\frac{N}{\alpha}\cdot(1+\frac{\alpha-1}{\alpha})}{N}) = K\frac{\alpha^2-2\alpha+1}{\alpha^2} = K(\frac{(\alpha-1)^2}{\alpha^2}).$$ It follows that
$$\mathsf{ChalSec}^3 = K \cdot (1 + \frac{\alpha-1}{\alpha} + (\frac{(\alpha-1)^2}{\alpha^2})) \text{ and } \mathsf{UnchalSec}^3 = N - K \cdot (1 + \frac{\alpha-1}{\alpha} - (\frac{(\alpha-1)^2}{\alpha^2}))$$

Round j: If we apply the same argument as above, the expected number of unchallenged sectors which gets sampled in round j is $(\frac{(\alpha-1)^{j-1}}{\alpha^{j-1}})$. It follows that

$$\mathsf{ChalSec}^j = K \cdot (\sum_{i=0}^{j-1} \frac{(\alpha-1)^i}{\alpha^i}) \text{ and } \mathsf{UnchalSec}^j = N - K \cdot (\sum_{i=0}^{j-1} \frac{(\alpha-1)^i}{\alpha^i}).$$

Since $K = \frac{N}{\alpha}$, we have that $\mathsf{ChalSec}^j = \frac{N}{\alpha} \cdot (\sum_{i=0}^{j-1} \frac{(\alpha-1)^i}{\alpha^i}) = N \cdot (\sum_{i=0}^{j-1} \frac{(\alpha-1)^i}{\alpha^{i+1}})$. This means that if we fix α, and we set β as the parameter that says how close do we want to get to N, we can estimate the number of rounds that are needed on average by evaluating

$$\sum_{i=0}^{j-1} \frac{(\alpha-1)^i}{\alpha^{i+1}} \geq \beta$$

Examples

1. Assume $\alpha = 2$ (every time a miner got challenged is challenged on half of his storage) and $\beta = 0.9$ (we want the 90% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(2-1)^i}{2^{i+1}} \geq 0.9$$

   which is true from $j = 4$.

2. Assume $\alpha = 4$ (every time a miner got challenged is challenged on the 25% of his storage) and $\beta = 0.9$ (we want the 90% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(4-1)^i}{4^{i+1}} \geq 0.9$$

   which is true from $j = 9$.

3. Assume $\alpha = 4$ (every time a miner got challenged is challenged on the 25% of his storage) and $\beta = 0.8$ (we want the 80% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(4-1)^i}{4^{i+1}} \geq 0.8$$

   which is true from $j = 6$.

4. Assume $\alpha = 4$ (every time a miner got challenged is challenged on the 25% of his storage) and $\beta = 0.75$ (we want the 75% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(4-1)^i}{4^{i+1}} \geq 0.75$$

   which is true from $j = 5$.

5. Assume $\alpha = 50$ (every time a miner got challenged is challenged on the 2% of his storage) and $\beta = 0.9$ (we want the 90% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(50-1)^i}{50^{i+1}} \geq 0.9$$

   which is true from $j = 114$.

6. Assume $\alpha = 10$ (every time a miner got challenged is challenged on the 10% of his storage) and $\beta = 0.75$ (we want the 75% of the sector to be challenged), than we have

$$\sum_{i=0}^{j-1} \frac{(10-1)^i}{10^{i+1}} \geq 0.75$$

   which is true from $j = 14$.

7 [Analysis]: How Likely is to fake Rescaling?

In this paragraph we try to evaluate how likely it is to fake a rescaling for a dishonest miner. We briefly recall how rescaling would work:

- We call $n = |\mathsf{P}|$ the number of blocks in a proving period P (after how many rounds the chain must check each miner's state).
- state_i is the active power of miner $i \in \mathcal{I}$ reported in the Power Table (in number of sectors).
- $\mathsf{TotalPower}$ is the total active power reported in the Power Table (in number of sectors)
- At the end of each proving period P, each miner locally check how many times $i \in \mathcal{I}$ miner i posted an EPoST onchain. We call this number e_i
- On expectation it should be $e_i = |\mathsf{P}| \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}}$. If $e_i < |\mathsf{P}| \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} - \alpha$ (for some tolerance value α that we decide, for example via the three sigma rule), then the miner gets slashed (or rescaled).

First, we assume that each miners checks all the sector he has at each round, in order to maximize revenue.

Second, For simplicity We assume that the power table gets frozen at the beginning of each proving period. This means that all the commits happened within a proving period will be added to the power table at the end of the proving period. Practically, this implies that each miner $i \in \mathcal{I}$ enters a proving period P with a fixed amount of space state_i in the power table.

What we want to evaluate now is how likely it is that a dishonest miner i, who starts a proving period with power state_i can keep his power at the end of the proving period (i.e. when the rescaling happens) while keeping only a X fraction of the declared power state_i withing the proving period. Said in another way:

- Miner i is supposed to keep state_i blocks
- Miner i only keeps $X \cdot \mathsf{state}_i$ blocks
- Miner i aims to win as if he was storing state_i blocks (so that at the next rescaling he can maintain state_i in the power table).

Since for an honest miner, the probability of winning at each block is $p = \frac{\mathsf{state}_i}{\mathsf{TotalPower}}$, Irene: *(1) this is the probability of winning a ticket not the probability of winning a block, which is the probability of winning at least a ticket* $1-(1-p)^n$*, (2) I think here we are assuming samplerate=1, right? do you think we can generalize this analisys to any samplearete?* Luca: *wrt (1), I would say that* $1-(1-p)^n$ *is the probability of winning at least* 1 *block out of* n *total blocks. At each block, I think the probability of winning (which is, finding a winning ticket) is* p*. I think the two formula are saying the same thing.*

the expected number of winning times for miner i is $n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} = \sum_{i=1}^{n} 1 \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} + 0 \cdot (1 - \frac{\mathsf{state}_i}{\mathsf{TotalPower}})$. Likewise, a dishonest miner i storing $X \cdot \mathsf{state}_i$ blocks (instead of state_i blocks, $X < 1$) has probability $\frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}}$ of winning at each block, and thus his expected number of winning times

is $n \cdot \frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}}$.Irene: *I think we should counts number of winning tivkets, noy just blocks per miner* Luca: *I agree, that was the reason why I wrote down the note at the end of the paragraph. Is the note something you kind of buy or it seems off to you?* By definition of the protocol, a miner i gets his power confirmed if the number W of wins he has are such that $n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} - \alpha < W < n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} + \alpha$. If we call FairRescale the event for which an honest miner gets properly rescaled, then we have that

$$Pr(\mathsf{FairRescale}) = \sum_{k=n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} - \alpha}^{n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} + \alpha} \binom{n}{k} \left(\frac{\mathsf{state}_i}{\mathsf{TotalPower}}\right)^k \cdot \left(1 - \frac{\mathsf{state}_i}{\mathsf{TotalPower}}\right)^{n-k}$$

If we call UnfairRescale the probability for a dishonest miner to be maintained state_i while keeping $X \cdot \mathsf{state}_i$ only, is:

$$Pr(\mathsf{UnfairRescale}) = \sum_{k=n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} - \alpha}^{n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}} + \alpha} \binom{n}{k} \left(\frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}}\right)^k \cdot \left(1 - \frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}}\right)^{n-k}$$

Luca: *are we counting here "only" the winning rounds and not the total of winning tickets? it seems to me that this analysis is not taking into account how many ticket a miner wins in total but "only" how many round he wins (meaning, in how many rounds he wins at least 1 ticket). If so, then* FairRescale UnfairRescale *should be renamed as* FairWin *and* UnfairWin*. It seems like as we were considering a scenario where a miner was posting the first winning ticket only. The number of expected number of winning tickets per round (if a prover tries all the sector he has) should be something like:*

- $\mathsf{state}_i \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}}$ *for the honest miner*
- $X \cdot \mathsf{state}_i \cdot \frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}}$ *for the dishonest miner*

Does it mean that on expectation the total number of tickets for the honest miner in a proving period is $(n \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}}) \cdot (\mathsf{state}_i \cdot \frac{\mathsf{state}_i}{\mathsf{TotalPower}})$ *and for a dishonest miner is* $(n \cdot \frac{X \mathsf{state}_i}{\mathsf{TotalPower}}) \cdot (X \cdot \mathsf{state}_i \cdot \frac{X \cdot \mathsf{state}_i}{\mathsf{TotalPower}})$*?[which, in both cases would be the probability of a winning round times the average number of winning ticket per winning round]*

8 Empty Blocks Estimation

We want to analyse the following scenario: and adversary $\mathcal{A}$, holding the $\frac{1}{3}$ of the total power, outputs an empty block every time he wins, in order to prevent honest miners to post their proofs in time. How many blocks to we need in a proving period to have at least k honestly generated blocks with high probability (let's say 0.99)?

$k = 1$. Assume we want to know how many blocks a proving period should last in order to have at least 1 honestly generated block

Case 1: Considering binomial approximation

- $\mathcal{A}$ is winning a block with probability $\frac{1}{3}$. This means that the probability to have 1 honestly generated block in n blocks is $1 - \left(\frac{1}{3}\right)^n$.

- What we want is $1-\left(\frac{1}{3}\right)^n > 0.99$ which means $\left(\frac{1}{3}\right)^n < 0.01$. It follows $nlog(\frac{1}{3}) < log(0.01)$, that is $n > log(0.01)/log(1/3) = 4.25$.

Case 2: Taking winning probability directly form Leader Election (no approximation)

- $\mathcal{A}$ is winning a block with probability $p = 1-(1-\mathsf{Target}_{\mathcal{A}})^K$, where $\mathsf{Target}_{\mathcal{A}} = \frac{e}{\mathsf{TotalPower}} \times \frac{\frac{1}{3}\mathsf{TotalPower}}{\lceil \mathsf{SampleRate} \times \frac{1}{3}\mathsf{TotalPower} \rceil}$ and $K = \lceil \mathsf{SampleRate} \times n_i \rceil$. This means that the probability to have 1 honestly generated block in n blocks is $1-p^n$.
- What we want is $1-p^n > 0.99$ which means $p^n < 0.01$. It follows $n \cdot log(p) < log(0.01)$, that is $n > log(0.01)/log(p)$. If we assume $\mathsf{SampleRate} = 1$ and $e = 1$ we have $p = 1-(1-\frac{1}{\mathsf{TotalPower}})^{\frac{1}{3}\mathsf{TotalPower}}$. It follows $n > \frac{log(0.01)}{log(1-(1-\frac{1}{\mathsf{TotalPower}})^{\frac{1}{3}\mathsf{TotalPower}})}$

$k > 1$. Assume we want to know how many blocks a proving period should last in order to have at least k honestly generated block

Case 1: Considering binomial approximation

- $\mathcal{A}$ is winning a block with probability $\frac{1}{3}$. This means that the probability to have k honestly generated block in n blocks is $\binom{n}{k}\left(\frac{2}{3}\right)^k\left(\frac{1}{3}\right)^{n-k}$.
- We want $\binom{n}{k}\left(\frac{2}{3}\right)^k\left(\frac{1}{3}\right)^{n-k} > 0.99$ which means $log\left(\binom{n}{k}\right)+k\cdot log\left(\frac{2}{3}\right)+(n-k)\cdot log\left(\frac{1}{3}\right) > log(0.99)$.
- Since $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, we have that $log(\binom{n}{k}) = log(n!) - (log(k!) + log(n-k)!)$. If we approximate $log(n!) \approx nlog(n)$ (Stirling's approximation truncated at the first term), we obtain

$log\left(\binom{n}{k}\right) = n\cdot log(n) - k\cdot log(k) - (n-k)log(n-k) = n\cdot log(n) - k\cdot log(k) - (n-k)log(n-k))$.

If we use a better approximation we have $log(n!) \approx (n+\frac{1}{2})log(n) - n + \frac{1}{2}log(2\pi)$ we obtain

$log\left(\binom{n}{k}\right) = (n+\frac{1}{2})\cdot log(n) - (k+\frac{1}{2})\cdot log(k) - (n-k+\frac{1}{2})log(n-k) - \frac{1}{2}log(2\pi) = n\cdot log(n) - k\cdot log(k) - (n-k)log(n-k) + \frac{1}{2}(log(n) - log(k) - log(n-k) - log(2\pi))$.

- Using the simpler approximation, what we want is
$n\cdot log(n) - k\cdot log(k) - (n-k)log(n-k) + k\cdot log\left(\frac{2}{3}\right) + (n-k)\cdot log\left(\frac{1}{3}\right) > log(0.99)$
that is
$n(log(n)-log(n-k)+log\left(\frac{1}{3}\right))+k\cdot log(n-k) > k((log(k)-log(\frac{2}{3})+log(\frac{1}{3}))+log(0.99)$
- Using the second approximation we get
$(n+\frac{1}{2})\cdot log(n) - (k+\frac{1}{2})\cdot log(k) - (n-k+\frac{1}{2})log(n-k) - \frac{1}{2}log(2\pi) + k\cdot log\left(\frac{2}{3}\right) + (n-k)\cdot log\left(\frac{1}{3}\right) > log(0.99)$

Case 2: Taking winning probability directly form Leader Election (no approximation)

- $\mathcal{A}$ is winning a block with probability p as in **Case 2** above. This means that the probability to have k honestly generated block in n blocks is $\binom{n}{k}(1-p)^k(p)^{n-k}$.
- We want $\binom{n}{k}(1-p)^k(p)^{n-k} > 0.99$ which means $log\left(\binom{n}{k}\right) + k\cdot log(1-p) + (n-k)\cdot log(p) > log(0.99)$.

- Since $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, we have that $log(\binom{n}{k}) = log(n!) - (log(k!) + log(n-k)!)$. If we approximate $log(n!) \approx nlog(n)$ (Stirling's approximation truncated at the first term), we obtain

$$log\left(\binom{n}{k}\right) = n \cdot log(n) - k \cdot log(k) - (n-k)log(n-k) = n \cdot log(n) - k \cdot log(k) - (n-k)log(n-k)).$$

If we use a better approximation we have $log(n!) \approx (n + \frac{1}{2})log(n) - n + \frac{1}{2}log(2\pi)$ we obtain

$$log\left(\binom{n}{k}\right) = (n+\tfrac{1}{2}) \cdot log(n) - (k+\tfrac{1}{2}) \cdot log(k) - (n-k+\tfrac{1}{2})log(n-k) - \tfrac{1}{2}log(2\pi) = n \cdot log(n) - k \cdot log(k) - (n-k)log(n-k) + \tfrac{1}{2}(log(n) - log(k) - log(n-k) - log(2\pi)).$$

- Using the simpler approximation, what we want is
  $n \cdot log(n) - k \cdot log(k) - (n-k)log(n-k) + k \cdot log\,(1-p) + (n-k) \cdot log\,(p) > log(0.99)$
  that is
  $n(log(n) - log(n-k) + log\,(p)) + k \cdot log(n-k) > k((log(k) - log(1-p) + log(p)) + log(0.99)$
- Using the second approximation we get
  $(n + \frac{1}{2}) \cdot log(n) - (k + \frac{1}{2}) \cdot log(k) - (n-k+\frac{1}{2})log(n-k) - \frac{1}{2}log(2\pi) + k \cdot log\,(1-p) + (n-k) \cdot log\,(p) > log(0.99)$

9 [Analysis]: Chain Quality

In this paragraph we try to find a closed form expression to evaluate the chain quality EC yields. We hope to supplement this work with simulation data and an analysis of the effects grinding and other attacks have on chain quality. In all expressions α refers to adversarial power and e to the number of expected leaders per epoch.

Definition. Chain quality is defined by (k, μ) with $k \in \mathbf{N}^+$ and $\mu \in (0, 1]$ as follows:

- Consider any portion of length at least k of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most $1 - \mu$. We call μ the chain quality coefficient.

Intuitively, the chain quality determines the minimum proportion of honest blocks guaranteed to be included in the chain given a consensus protocol. It is relevant to thinking about message (transaction) censorship, as well as the quality of randomness being produced by the chain (since an adversary can grind through block production). **The larger μ the better.**

To that end, it is worth distinguishing between chain quality and randomness quality, where the latter is instead the ratio of minimal blocks (in a tipset) originating from an adversary.

It is also worth stating a stronger variant of chain quality (included in Fantomette):

- Any portion of k sequential tipsets of the chain held by an honest miner contains at least $[(1-\alpha) - \mu] * ke$ blocks generated by honest miners.

In this stronger setting, μ bounds adversarial power over block production, ensuring that the adversary cannot earn more than μ over their fair share of blocks given power α. **The smaller μ the better.**

Base Case: Chain Quality. We choose to model leader election as a binomial event, with each participant having equal power, thereby modeling power as resulting from a coalition.

Let

- m be the number of participants in the protocol
- $\mathsf{h} = (1-\alpha)m$ be the number of honest participants in the protocol
- $\mathsf{a} = \alpha * m$ be the number of adversarial participants in the protocol
- $\mathsf{p} = \frac{e}{m}$ the likelihood that any given participant is elected leader in an epoch.
- $\mathsf{S}^{\mathsf{y}}_{\mathsf{x}}$ be the event in which an honest block is the smallest in a given Tipset with x honest blocks found and y total blocks found ($y - x$ adversarial).

In each epoch, each participant attempts leader election. Therefore, we have the variable X, the total number of honest wins in an epoch:

- (**) $Pr(X = n) = \binom{h}{n} p^n (1-p)^{h-n}$, the probability that there be n honest wins in an epoch.

  Irene: *I think considering the probability of at least n honest players winning (instead of exactly n) would be better. There should be closed formula for that as well.* Henri: *@irene: this is a good point, but actually my plan is to just use this to try and bound max adversarial wins (i.e. use same formula for adversarial power)*

  Irene: *@henri, see also the comment below.*

We try applying this to approximating quality.

Using binomial sums

Thankfully, we know about sums of binomials. So we can actually represent the variable Y, the total number of honest wins in a series of k epochs.

- $Pr(Y = n) = \binom{h*k}{n} p^n (1-p)^{h*k-n}$

We can thus find an expression for which any run of k epochs producing more than n blocks for the adversary is probabilistically negligible.

Irene: *so, if I understand correctly, say you want μ chain quality (using the first def for simplicity here) then you set $n = 1 - \mu k$ (α is fixed) and use this formula to find how large k needs to be in order to have $Pr(Y = n = \mu k)$ be negl. Correct? This gives the baseline values of different choices of μ. Is this understanding correct? If yes, I think the former formula (**) should account for at.*

Henri: *TODO: how to model adversarial grinding?*

Applying this to randomness quality. As stated above it is worth distinguishing between chain quality and randomness quality, where the latter is instead the ratio of minimal blocks (in a tipset) originating from an adversary.

Accordingly, we can adapt our above expression to get a probabilistic expression for the number of smallest blocks in tipsets found amongst the n honest blocks generated in k epochs. Note that this is a much harder problem, since it must account for both honest and adversarial blocks

produced.

We can thus approximate the likelihood that an honest participant find the smallest block in an epoch as follows:

- $\Sigma_{y=x}^{\inf}\Sigma_{x=0}^{\inf} Pr(\mathsf{S}_{\mathsf{x}}^{\mathsf{y}} \wedge X = x \wedge Y = y) = \frac{Pr(\mathsf{S}_{\mathsf{x}}^{\mathsf{y}}|X=x \wedge Y=y)}{Pr(X=x \wedge Y=y)}$, with
- $Pr(\mathsf{S}_{\mathsf{x}}^{\mathsf{y}}|X = x \wedge Y = y) = \binom{y}{x}$
- $Pr(X = x \wedge Y = y) = Pr(X = x)Pr(Y = y)$, since they are independent events

This is the sum of the probabilities that given tipsets of various sizes, the smallest block in them be honest. We can then use Chernoff bounds to bound the range of values we consider for X and Y, making this computable.

Henri: *TODO: figure out how to apply this across multiple epochs.*

10 Game-theoretic properties

In this section, we prove properties related to the game theoretic guarantees of the system.

We start by detailing the action space, i.e. the set of action available to each player.

1. When they are elected leader whether to create a block or not (or more than one)
2. (If applicable) which chain to extend
3. When they create a block: what to include in their blocks (especially whether to include other players surprise post)
4. When to broadcast a block
5. Who to broadcast it to (depending on what capabilities we assume here cf epoch boundary attacks)
6. If they receive a surprise challenge whether to respond to it
7. When they receive a block whether to include it in their chain or no
8. With respect to storage:
9. When to free storage and regenerate it
10. What to store?

We now prove that following the protocol is a coalition-proof $\epsilon-$Nash equilibrium for coalitions of up to k players.

Theorem 10.1. *A coalition of up to k players cannot increase its expected utility by more than ϵ by deviating from the protocol, assuming the rest of the players also follow the protocol.*

Proof. We start by proving that a coalition of players if better off keeping their storage over time.

Lemma 10.2. *A coalition of players if better off keeping their storage over time.*

Proof. Sarah: *don't know if this is actually true* □

Let's now focus on the content of the blocks, we show that, regardless of the rest of their strategies, the coalition is better off including the surprise post of the other parties. Sarah: *do surprise post still exist?*

Lemma 10.3. *Assuming all the other players follow the protocol, a coalition of up to k players is better off including the surprise posts of the other players in their blocks (regardless of their strategy for creating blocks).*

Proof. (Sketch) chain quality tells us that one of the other players will create at least one block out of ?. By assumption this player will include all the surprise posts it is aware of so all surprise posts will be included in the chain. The coalition is better off including them in its block in order to receive the transaction fees associated with them. Sarah: *Do we consider the case of a congested network?* □

There is no incentive to not collaborate with other players.

Lemma 10.4. *do we want a lemma for this?*

Proof. this follows from the fact that (1) the leader election is unbiasable and (2) even if multiple blocks are created at the same epoch, each block will earn exactly the same reward. □

We now show the following lemma:

Lemma 10.5. *the coalition mine on top of the chain chosen by the fork choice rule and on top of this one only.*

Proof. This follows from slashing. To start with let's assume that block rewards within each block are fixed (i.e. we do not account for transaction fees).

Let's consider the strategy over onetime slot to start with Case where a single miner of the coalition is elected leader: A miner gets slashed if they extend more than a chain for the same epoch.

In the case of two chains, for example, if the miner mines on multiple chain, then its expected utility will be **block reward- slashing**. Sarah: *can you get slashed multiple times if you mine on multiple chains?* This is because other players follow the protocol so the miner is sure to get slashed, and also sure to get the rewards from that block because one of the two chain will win with its block in it. In the case of mining on top of the fork-choice rule chain the expected utility becomes $\mathbb{P}$[this chain wins] × **block reward**.

So the miner will choose to mine on the fcr chain as long as

$\mathbb{P}$[this chain wins]× **block reward** > **block reward- slashing** and so as long as $\mathbb{P}$[this chain wins] > 1 − **slashing/ block reward** the intuition is that **slashing¿¿block reward** to ensure that the left hand side term is very small and that, since most of the players are extending only one chain and assuming good network the probability the probability that the honest chain wins is on the other hand close to one, ensuring that the above inequality holds and that a single miner mines on only one chain.

Case of multiple players: In this case the coalition could choose to spread its players across chains in order to avoid being slashed. In this case this means that they will earn less rewards than

what they could have on the winning chain because they could have earned more blocks. In the case of mining on multiple chains for one block with different players, the expected utility of the coalition is $\sum_{chain} \mathbb{P}[\text{chain wins}] \times$ **number of blocks on chain** $\times$ **block reward**.

In the case of a coalition mining on the fcr chains the expected utility of the coalition is $\mathbb{P}[\text{chain wins}] \times$ **number of winning leader** $\times$ **block reward** Because the rest of the players follow the protocol, the main chain is the more likely to win Sarah: *need to prove this* and so for each chain c we have : $\mathbb{P}[\text{chain c wins}] \times n \times$ **block reward** $< \mathbb{P}[\text{fcr chain wins}] \times n \times$ **block reward**. Hence we have:

$$\sum_{chain} \mathbb{P}[\text{chain wins}] \times \textbf{number of blocks on chain} \times \textbf{block reward} < \mathbb{P}[\text{fcr chain wins}] \times \sum_{chain} \textbf{number of blocks on ch}$$

and thus the players are better off extending the fcr chain.

The above analysis also applies to the case where the coalitions mines on multiple slots. The expected utility is always higher when putting all of its block that has the higher chance of success rather than spreading it around or getting slashed.

Now let's consider the case where the block rewards changes with the content of blocks, i.e. depends on transaction fees. A miner will then be incentivized to add in its block the transaction with the highest transaction fees. Since only the block with the smallest ticket will win transaction fees. However by omitting other players' blocks, the coalitions risks having their blocks not including in the chain and hence losing their block reward. Since block rewards are ¿¿ than transaction fees it is not worth the risk. The expected rewards in the case the coalition is following the protocol is as follows:

$$e \times f \times \textbf{block reward} +$$

Sarah: *could also solve this problem by creating more rewards in each block such that every block earns transaction fees but the sender of that transaction only pays it once.*

Sarah: *I think headstart may be the rational thing for coalitions of 1/3rd+* □

□

11 Proof details for finality

We note c_A the stochastic process that describes the number of adversarial leaders elected at each slot (indexed by t).

We note c_H the stochastic process that describes the number of adversarial leaders elected at each slot (indexed by t).

11.1 First case: Adversary collaborating with honest players

11.1.1 Normal catch up

The probability that the adversary catch up with the honest chain after one round is equal to:

$$P_1 := \mathbb{P}[c_A^1 \geq c_H^1] = \sum_{i=0}^{n_A} \sum_{j=i}^{n_A} p_i^{n_h} p_j^{n_A}$$

Where $p_i^n = \binom{n}{i} p^i (1-p)^{n-i}$ represents the binomial distribution, where $p = \frac{e}{n_A + n_H}$

After two round this probability is: Sarah: *update all thee following*

$$P_2 := \mathbb{P}[c_A^1 + c_A^2 \geq c_H^1 + c_H^2 | c_A^1 < c_H^1] = \sum_{c_A^1=0}^{n_A} \sum_{c_A^2=0}^{n_A} \sum_{c_H^1=c_A^1+1}^{n_H} \sum_{c_H^2=0}^{c_A^1+c_A^2-c_H^1} p_{c_A^1}^{n_A} p_{c_A^2}^{n_A} p_{c_H^1}^{n_H} p_{c_H^2}^{n_H}$$

Similarly, we can derive that the probability of catching up at slot l is:

The probability of catching up for the adversary is then: $\sum P_i$.

11.1.2 Other strategies

Headstart from two rounds. Probabilities. Sarah: *to do*

11.2 Second case: private fork

The probability that the adversary creates a chain of length k, as long as the honest chain is as follows:

$$\mathbb{P}[\sum_{i=1}^{k} c_A^i \geq \sum_{i=2}^{k} c_H^i] = \sum_{i=0}^{kn_A} \sum_{j=i}^{kn_A} p_i^{(k-1)n_h} p_j^{kn_A}$$