Data Availability in Ethereum 2.0 Sharding Design

CAMCOS team

September 2019

We outline the Ethereum 2.0 solution to the "data availability problem" in blockchain, which uses mechanics such as the *chunk challenge* and the *bit challenge* to incentivize the participants to act honestly. We perform game-theoretical analyses of the mechanics involved and discuss their implications.

Contents

1 Introduction

As cryptocurrencies deal with increasing technological and economical demands, the *data scalability* problem (being able to process increasing amounts of data) becomes more and more relevant.

In classical centralized systems, data is kept by a centralized authority, while other participants only download data from the authority as needed. The security of these systems is dependent on the integrity of the authority. In common blockchain designs inspired by Bitcoin, each user of the blockchain must download all of the data, which is a natural but cumbersome decentralized approach. As scalability becomes more of an issue, we desire a middle ground where the system is still decentralized, but the required amount of data kept by each individual participant is lower. This gives rise to the *data availability* problem (see e.g. [ABSB18]): ensuring to each participant that the data which he/she did not download is accessible and validated by other participants in a decentralized way.

The design of the Ethereum 2.0 blockchain is *proof-of-stake*, which means participants must commit capital (in the form of protocol tokens) upfront to the blockchain. By placing capital at risk, they are incentivized to follow blockchain protocol. In the more well-known *proof-of-work* systems, participants (e.g. *miners* in blockchain) secure the system by performing difficult and time-intensive computations. By contrast, proof-of-stake protocols are designed to only require light computation, and are secured with game-theoretical incentives where validators do not want to get their stake *slashed*, or destroyed. Both types of systems require a dedication of economic resources to the protocol, though the ways the resources interact with the protocol are different.

These game-theoretical incentives that we use in proof-of-stake designs, which are usually called *cryptoeconomic incentives*, allow us to somehow make claims that things are working without directly doing the work. In this paper, we approach the data availability problem by constructing cryptoeconomic incentives to induce participants of the blockchain to have data available for all other users, who do not need to verify data availability by downloading the data themselves.

In Section 2, we define the words that appear in our work and provide some background. In Section 3 we define the main mechanisms used to address the data availability problem. In Section 6 and Section 7 we analyze these mechanisms, coming to the conclusion that [insert later]. We finish with some remarks in Section 10.

2 Preliminaries

2.1 Blockchain, Validators, Sharding

A *blockchain* is a protocol that allows a set of participants (which we call *validators* in this work; in other contexts, the participants have been called *miners* [N+08], *nodes* [CL+99], and even *bakers* [ABT19]) to agree on a consensus history of events supplied in data called *blocks*.

In the Ethereum 2.0 design [beacon][Dev19], we wish to offer a full proof-of-stake based blockchain for a set of validators. The implementation is *sharded*, which means that the design splits the data in the blockchain into subsets called *shards*, which can be thought of as individual blockchains in their own right. Specifically, the design includes a *beacon chain*, which is the central chain that stores and manages the registry of validators, and many *shard chains* to store the chain data. The idea is that each validator only needs to download data in specific shards, which addresses the scalability problem. We now focus on the resulting data availability problem, with the goal being that all the validators can trust the data integrity of every shard and potentially have access to all the data. See [al2018fraud][ABSB18].

2.2 Committees

We measure time in units of *slots*, the most basic unit of time the protocol understands (currently planned to be 12 seconds). The secondary units that we will use are:

Epoch: Roughly speaking, epochs are "checkpoint times" for the protocol. During each epoch, each validator is put on exactly one beacon committee (to be defined soon).

Custody period: This is a time length where each validator is supposed to keep the data he/she downloads available "in custody" until it is safe to delete.

[I don't think the idea of "automatic reveal" blends well in a blockchain type protocol, its not the beacon chain which reveals the keys. I assume the protocol just simply penalizes any validator who doesn't reveal, so validators will just reveal their keys. I also assume the penalty for not revealing is way higher than the anticipated slashing penalties from bit challenge. (Otherwise we get another small game theory game)][CJ] [You are completely right. Abstracting it away is just to make life a bit easier for us, and it may even make sense to write that game eventaully. I have some disclaimers on this later. See 3.1][YZ]

The validators work in subsets called *committees*. There are two types of committees motivated by the shards. Each validator is usually simultaneously on a shard and a beacon committee.

First, we have the *shard committees* (also called *persistent committees*) who actually build the shard chains. A validator on a shard committee should download and store all of the shard data to ensure that they can help build the shard chain. They are not required to store the data beyond the length of their assignment to that shard (roughly 1 day). Such a validator is incentivized to have recent shard data available for the beacon committees to download so that the data will be included in crosslinks in the beacon chain. When this happens, the shard committee members get rewards for their efforts on the shard chain.

Then, we have the *beacon committees*, which also have an assigned shard. Every slot, each shard creates a new beacon committee. Every validator is assigned to be on exactly one beacon committee per epoch. Each of these committees is supposed to download the recent work (done by the shard committee) on the

shard chain since the last *crosslink* (a "checkpoint," soon to be defined) from the last slot, and certifies the work for the inclusion into the beacon chain, creating a new checkpoint for the shard chain. This inclusion of the certification into the beacon chain is called a *crosslink*. The responsibilities of a beacon committee member to create each crosslink honestly are central to the paper; we elaborate more in Section 3.2.

sec:crosslink

["However, one of these is created every slot for every shard" Not sure how to change this, it somewhat sounds like a single committee is created for a whole slot and assigned to all shards simultaneously.][CJ] [What about now?][YZ] [Not sure if it should be slot or epoch, under optimal specs can a shard be assigned into two slots in a single epoch? (2048 committees optimal per epoch, 1024 shards)][CJ] [I checked; there are 64 shards total and they are tryng to crosslink per *slot* instead of per *epoch*. In any case, it helps to use concrete numbers, so I'm adding a remark below.][YZ]

Remark 2.1. Under the current specs, Ethereum 2.0 plans to have 32 slots per epoch, about 6 minutes. Each custody period is planned to be around 64800 slots, or 9 days. There are 64 shards in the design. During each epoch, each of the 32 slots creates 64 beacon committees. The validators (planned to be at the minimum $2^{14} \approx 16000$ and targeting 300000) are to be split between the committees, targeting 150 validators per committee. Independently running alongside these beacon committees are 64 shard committees, one per shard, which refreshes membership roughly once per day.[YZ]

While the role of the shard committee is very important, for this paper we mostly focus on the rules and incentives around the beacon committees, which are reassigned once per epoch (though the responsibility and consequences of being on the committee last longer than an epoch).

Memberships for both types of committees are randomly assigned and shuffled frequently to prevent collusion. This randomness is the foundation for the whole system and many important assumptions (for not just the data availability, but also goals such as safety, liveness, and anti-collusion). However, the quality of the randomness is out of the scope of this paper.

2.3 Merkle Trees

sec:merkle

In blockchain, it is common to store data in *Merkle trees* ever since Bitcoin nakamoto2008bitcoin [N+08], which allows for efficient query and verification of data. A *Merkle tree* is a tree where every non-leaf node is the hash (given a cryptographic hash function) of the concatenation of its children, where the leaf nodes contain the data. The *root* of a Merkle tree can then serve as a commitment to all the data contained in its leaf nodes. A slight modification in a leaf node will result in a completely different root hash of the Merkle tree. Thus, to verify the data in a leaf node, it is enough to simply supply the data and a *Merkle path*, which is a sequence of hashes in the tree that can be sequentially checked for consistency, all the way up to the root (because it would be hard, computationally, for someone without the actual data to fake the hashes that eventually hashes to the root,

which is public). This Merkle path and the relevant computations that check all the hash relationships are correct together form a *Merkle proof* that the person giving the proof has the data. See Figure 1 for a graphical visualization of a Merkle tree with 8 leaves and the associated concepts.

Remark 2.2. Ethereum 2.0 plans to use a standard binary Merkle Tree, though Ethereum 1.0 uses a modified *Merkle-Patricia tree* [W+14] combining Merkle trees and radix trees.

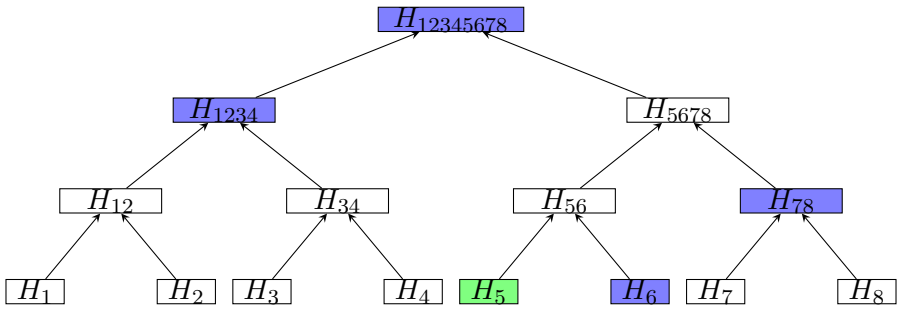


Figure 1: A Merkle tree of 8 leaves (chunks). H stands for Hash. The worst case query and verification of a piece of chunk data is $O(\log(n))$. The Merkle path of Hash5(H_5) is highlighted in green. Only the blue information is needed to verify if H_5 is part of the Merkle tree.

[This shows that the blocks start from H1, to H2, until all the blocks are filled in order to get block H12345678?][FV]

2.4 Game Theory

In this paper, we mostly deal with *finite* games, where each player has a finite set of pure *strategies*, and every combined choice of strategies for every player results in *payoffs* (measured in utility) for the players. A set of *mixed strategies* (meaning a probabilistic mixture pure strategies) is a *Nash equilibrium* if no player can improve his/her payoff by changing his/her strategy while everyone else's strategy is fixed. Nash [Nas51] proved that every finite game has a Nash equilibrium.

Game theory has already been applied to various analysis of blockchain economic and security issues. The paper [ABV+18] proposes a decentralized oracle to prevent the data manipulation attack, and demonstrates that under certain conditions a Nash equilibrium exists where all rational players are forced to behave honestly. In the paper [LLW+19], the authors argue the voter and verifier coordination game has a Nash equilibrium where both parties choose to be honest.

Unless otherwise noted, we make the standard game theory assumptions. Mainly, we convert everything into a common unit of utility (fungible with the units of stake from the protocol) and assume that we are analyzing *rational* validators (seeking to maximize utility).

3 The Protocol

sec:game

In this section, we first define some preliminaries, where the main idea is a *crosslink*, which is a type of checkpoint for a shard. Then, we outline the validators' protocol responsibilities to deal with each crosslink, which nominally requires them to each download data required for the crosslink and do some computational work. Then, we introduce the mechanisms that arise from the responsibilities, each of which can then be analyzed semi-independently of the others using elementary game theory. These analyses make up the core of our work.

3.1 The Custody Period and Subkey

For every custody period of time, each validator V generates a private 256-bit *subkey* s_V in a pseudorandom manner. This will be the key they use for their beacon committee duties, to be defined soon.

At the end of every custody period, we assume that each validator V reveals s_V to all the other validators (and that once revealed, its veracy can be verified crypographically), so the validators can check each other's work. In fact, there is actually some additional game theory here as the validators are supposed to reveal the key themselves, and risk slashing if they fail to do so. While this is certainly important to the protocol, for this paper we assume the keys are automatically revealed to constrain the scope of analysis.

3.2 The Crosslink

sec:crosslink

During each *epoch*, each validator is assigned to a unique slot. Within each slot, the validators assigned to the slot are once again subdivided into beacon committees, with each committee assigned to a shard assigned within the same slot. Each validator's job is to download all the new data they see on the network for the shard they are assigned to since the last *successful* (to be defined soon) crosslink. We call this piece of data D. Then during slot s of this epoch, they are to send a *attestation* to the network. An attestation is a message that includes several pieces of data, including a *crosslink vote*.

Remark 3.1. For a concrete sense of numbers, 64 beacon committees are made for each of the 32 slots, with a total of $64 * 32$ beacon committees created per epoch (though each validator is on exactly 1 beacon committee per epoch). Also, in the actual implementation, there is some dynamic adjustment of committee sizes and number of shards, which we ignore in our abstracted version of the protocol since they are largely irrelevant from the game-theoretical analysis. For example, there is a minimum number of validators (128) for each committee, because more validators improve the safety and liveness of the consensus layer. However, to maintain this minimum, we may not always have the same number of shards if the number of validators are low. See [But18a] for more details.

crosslink-committees

A *crosslink vote for* D is a digitally-signed (by the validator) piece of data that includes some metadata, a Merkle root (explained in Section 2.3) hash

sec:merkle

of D, and a single bit called the *custody bit*, which each validator computes from D and some validator-specific key (to be explained later in Section 3.3). Besides a crosslink vote, an attestation including metadata, a LMD GHOST vote (inspired by GHOST, [SZ15]), and a FFG vote [BG17]. For the limited scope of this paper, we can basically treat the two concepts (attestations and crosslink votes) interchangeably.

An attestation / crosslink vote serves as a "signature" of the work done by the validator, a vote of confidence in the data, and a promise that the data is available to the other users of the chain. If 2/3 of the validators assigned to the beacon committee have made an attestation for D, then we are considered to have had a *(successful) crosslink* included into the chain. The beacon chain then stores these attestations, instead of the data that they attest for; the data the beacon chain is considered to "know" is all the data up to the latest crosslink included in the chain[1].

3.3 The Custody Bit

There are several viable contenders for the actual implementation of the custody bit function. Most of the technical details around the actual custody bit function are irrelevant to us, so we will first give a specific function for our paper and then explain the assumptions we will actually use. The function we will use follows:

1. Define $\mathsf{chunkify}(D, n) = (c_1, c_2, \ldots, c_k)$, an ordered list of n-bit strings ("chunks" of equal length as s_V) of D, where $k = \frac{\mathsf{len}(D)}{n}$; we assume k is an integer by padding D if necessary.

2. For each $1 \leq i \leq k$, let $h_i = H(\mathsf{xor}(c_i, s_V))$ where, H is some cryptographic hash function.

3. Let $M = (h_1||h_2||\cdots||h_k)$ be the concatenation of all the h_i.

4. Finally, define V's *custody bit* to be $\mathsf{custody}(D, s_V) = \mathsf{bitwisexor}(M)$, where $\mathsf{bitwisexor}$ is the XOR of every bit of its argument.

Where did $\mathsf{chunkify}(D, \frac{len(D)}{32})$ come from? Does this refer to a global constant k?[JS] [I assume the function in 2 comes from `https://ethresear.ch/t/bitwise-xor-custody-scheme/5139`, so a few questions: 1. Is the hash function binary? (i.e. always outputs 1 bit regardless of input), 2. If it is not binary, should the function be $\mathsf{bitwisexor}(H(c_i||s_V))$]? 3. Is || concatenation or xor (It appears to be concatenation)?[CJ] [1. no, and it doesn't really matter, but most hash functions do something like eats a bunch of bits and spit out 32 bits. 2. I think it's correct as stated now. What I had before was bad, and your suggestion would actually also work, but it is good to follow what's there. 3. I think it is

[1] There is some nuance here, because the beacon chain can fork due to latency problems, attackers, etc.; there is a consensus protocol layer on the beacon chain that decides which of the potentially conflicting versions of the chain state is the accepted one. As it is outside of the scope of our work, we ignore the intricacies of this layer and simply work as if we know we are always talking about the correct version of the chain state.

implied that —— is concatenation, right? 4. can you make a citation to the link you gave? I think it's good that we cite that. I have no idea where we got the original text from (I think Ying got it somewhere)]YZ However, the only properties of the function that we care about in this paper[2] are:

- When a validator V is asked to attest to a piece of data D, V computes some function $\mathsf{custody}(D, s_V)$ to get his/her *custody bit* b_V.
- The output b_V is statistically approximately random in $\{0, 1\}$.
- It is difficult to compute this bit without knowing either D or s_V in its entirety.
- $\mathsf{custody}()$ is a function of $\mathsf{chunkify}(D, k)$ and s_V. This design allows discrepancies in the computation to be "locally identified," which enables both the *bit challenge* to be defined later.

For example, one can also use the Legendre PRF [Dam88] damgaard1988randomness as a pseudo-random function with good aggregation features to create a function that satisfies our requirements.

It is educational to consider alternatives to this design. The most "pure" design would be to not have a custody bit but just have e.g. a Merkle hash of the data. However, as communications are public, lazy validators can just copy the attestations of other validators (lazy or not). Thus, we need the attestation data to contain a function of both the data and the private information (the subkey s_V), of which our (specific) custody bit function is one such candidate. Also, we can have more bits than 1, which has other tradeoffs. For example, more bits (or introducing prover-challenger communication rounds) would make it harder to fabricate a valid attestation, and fewer bits (or having fewer or no prover-challenger communication rounds) would make it more efficient (and have less overall information being broadcast on-chain). For this particular design, the single bit, no prover-challenger communication approach is selected to optimize for efficiency and scalability.

sec:main-goals

3.4 Main Goals

To summarize what we have so far, each validator's beacon committee duties are as follows:

1. Subkey management:
   a) Be assigned a private subkey s_V.
   b) Use the subkey for one custody period.
   c) At the end of that time, reveal the subkey s_V and obtain a new s_V
2. Concurrently, in each epoch:
   a) Be assigned to a beacon committee for a particular shard, and

[2] There are other important assumptions for the protocol that are not important for our paper. For example, we do want the custody bit to be compatible with multi-party computation, enable signature aggregation, etc. for potential use cases such as enabling validators to pool stake. However, these considerations are orthogonal from the crypto-economical analysis.

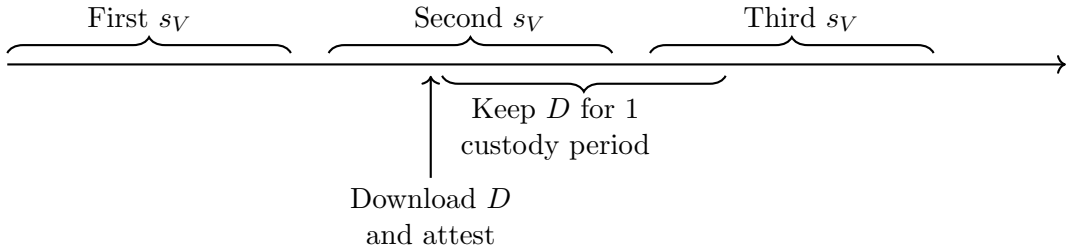
b) Download the data D, which is the difference between the state of that shard and the state of the shard at the last crosslink.

c) Create a custody bit b_V using D and s_V.

d) Attest to D, which is essentially signing the bit b_V.

e) Keep the data D for a custody period, then discard it.[YZ]

[If data D is kept only for 1 epoch, then doesn't it become unlikely that earlier false attestations will be successfully discovered and challenged, whereas later attestations have a much larger window. Since it is likely block proposers will only send their own challenges, unless the early block proposers check early attestations, dishonest people could cheat early but be honest later and get away more easily. Also, since challenge responses allow up to 72 days, would this be in conflict?][CJ] [There is a difference between early and late. Note you are supposed to keep a custody period, which is longer than an epoch. 72 days seem really long.][YZ]



[This picture needs to be put into a figure, captioned, and linked to from the text.][YZ]

Our goal is to prevent incentive misalignments in this framework. For example, while honest validators actually download the data (which can be quite large) before doing the work to compute their custody bit, a dishonest validator can cheat, saving some computation, by simply guessing a bit (and obtaining the "correct" one with 50% probability) and not actually have downloaded the data or have access to it. Thus, it makes sense to have incentive structures where such cheating is discouraged and punishable. We now list the three main things we would like to ensure with with cryptoeconomics. Each of these desiderata can then be associated to studying a different "game" whose rules are given by the relevant parts of the protocol which we (as designers) have control over.

- **We want the validators to download the data.** This leads to the *bit challenge* game, where validators making incorrect custody bits can be discovered and punished. The threat of bit challenges ensure that the validators have downloaded the data (which is required to do the computations for the bit correctly) at the time of making the attestation to the crosslink.

- **We want the data to be available for an extended period of time.** This leads to the *chunk challenge* game, where the threat of a challenge of knowing random pieces of the data (through the *chunk challenge*) ensures that the validators have (or have access to) the data for a more extended

period of the time so the blockchain as a whole "knows" the data for a while longer.

- **We want the validators to do their own work instead of outsourcing them.** Outsourcing is undesirable for us (since, among other things, we do not want these third parties to gather too much centralized power). This leads to the *early key reveal* game, which ensures that validators would take on counterparty risk by outsourcing their work.

We now give the top-down design of each validator V's duties with respect to these games. Then, we have individual subsections dedicated to the games' details and game-theoretical abstractions. Finally, we analyze the game-theoretical models in latter sections.

1. In each epoch, download the shard data D corresponding to V's beacon committee assignment in that epoch and attest to a bit b_V at time T.

2. After T, V is susceptible to being chunk challenged on D for 1 custody period worth of time. V is guaranteed to pass the challenge if V has downloaded D when challenged.

3. When s_V is revealed (which could be almost after T or up to T plus 1 custody period, depending on when T took place), V is susceptible to being bit challenged for a period of time (on the order of 1 custody period). V is guaranteed to pass the challenge if V attested to b_V correctly (by downloading D and computing).

4. Any time before s_V is revealed, V is susceptible to being whistleblown for the early key reveal if someone manages to sign a message with s_V (proving the key was revealed). V is (cryptographically) guaranteed to not be whistleblown if V did not lose his/her key, else some adversary was able to reverse digital signatures, an impossibly strong assumption.

[Tangential scenario: Lets say a shard undergoes a supermajority attack (66 percent), where the supermajority attests to some alternate data (which passes with an alternate merkle root). An honest person tries to attest for the honest merkle root, and later gets chunk challenged. Do they fail?][CJ]

4 The Mechanisms

4.1 The Bit Challenge

sec:bit-challenge

First and foremost, **we want the validators to download the data.** In our protocol, by forcing the validators to publish their custody bit, we are incentivizing them to download the data in order to compute the custody bit (otherwise they would have to resort to guessing). We define the *bit challenge* to be the mechanisms in the protocol which ensure that the custody bit is computed correctly.

The bit challenge becomes available after the custody period, when the secret keys of the validators are revealed. Now everyone who has the data can compute the custody bit of each other validator V with his/her revealed key s_V. And now any validator can take the role of a "challenger" and challenge another validator (who takes a "prover" role) about the correctness of the prover's custody bit.

Abstractly, the bit challenge itself is a Merkle proof interaction: if a challenger V_C and a prover V_P do not agree on the single bit b_P that is committed by the prover V_P, among all the k leaves of the Merkle tree of D, they must have at least one chunk where they do not agree on the sub-computation involving that chunk (this is where we use the assumption about our custody bit function that the computation is split into sub-computations on the chunks). The blockchain is then able to verify who has the correct custody bit by computing the correct local computation involving that chunk.

The bit challenge protocol is the following: If a challenger V_C disagrees with the b_P that is committed by the prover V_P, then among all the k chunks of D, there must be at least one chunk where V_C disagrees with V_P on the sub-computation involving that chunk. V_C sends a claim by providing an alternate bitfield M' such that $\mathsf{bitwisexor}(M') \neq b_P$. V_P can defend himself by submitting evidence to the blockchain that M' is incorrect.[CJ]

Example 4.1. Specifically, using our particular custody bit function $\mathsf{custody}(D, s_V)$, a challenger V_C that disagrees on the custody bit b_P from validator V_P issues the challenge by giving a k-bit bitfield M', where the i-th bit is $\mathsf{bitwisexor}(\mathsf{chunkify}(M, k)[i])$. The prover, V_P, must then disagree with V_C on some i_0-th bit with corresponding chunk $M_0 = \mathsf{chunkify}(M, k)[i_0]$. V_P can then respond with a Merkle branch of M_0, and if $H(\mathsf{xor}(M_0, s_V)) \neq M'[i_0]$, the defense is successful. See Figure 2 for a graphical visualization of this scheme. fig:game

[Paragraph formatting is broken in example][CJ]

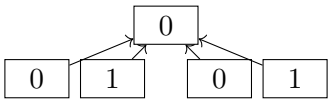


fig:game

Figure 2: A toy example of the bit challenge for a piece of data with 4 chunks. Prover A committed to a custody bit of 1 for a crosslink attestation. Challenger B disagrees, computing custody bit 0 for the same data with the A's secret key. B publishes her own intermediate bits for all 4 chunks as shown. If A can prove that any one of the 4 chunks should have a different bit, A wins, otherwise B wins.

There are many ways to implement such a challenge in practice. We can design to have fewer/more rounds of interaction (as low as a single challenger round, since the challenger can just theoretically do the computation and show that it is incorrect in some on-chain language). We can make it so that challenges can theoretically be fraudulent, or enforce that only a correct challenge can even be administered. We define the *bit challenge protocol* to be the following

protocol which abstracts away much of the specifics of the challenge[3]. [Suppose V_P responds with a chunk that is not a successful response, even though M' is incorrect. Is V_P immediately slashed, or does V_P have more opportunities to send other chunks that would prove M' incorrect? (So, 1 round or many?)]CJ

1. The challenge period begins when the secret keys have been revealed, and lasts for some specific number of slots.

2. During each crosslink, a prover V_P chooses to either:
   - **Attest**: Write an attestation by supplying an attestation bit. An attestation can be **correct** or **incorrect**. A correct attestation will pass any bit challenge, while an incorrect attestation will not pass valid challenges. The protocol gives some reward R_A for writing an attestation.
   - **No attest**: Do nothing. The protocol gives a punishment P_A for failing to write an attestation.

3. If V_P makes an attestation, the challenger V_C chooses between:
   - **Challenge**: sends a challenge message.
   - **No challenge**: Nothing happens.

   We stress that V_C has enough information (because the secret keys have been revealed) to know if the challenge is successful.

4. If V_P has no attestation, the challenger V_C can only choose **no challenge**.

5. If there is a challenge, the bit challenge mechanism will eventually resolve V_C's challenge as being **successful** (resp. V_P's response is unsuccessful) indicating that V_P's bit was incorrect or **fraudulent** (resp. V_P's response is successful) if V_P's bit was correct.

6. If the challenge is successful, V_P is slashed and penalized P_S, while V_C gains a reward R_S instead. If the challenge is fraudulent, V_P gains a reward R_F and V_C is penalized P_F.

For all of these parameters, having a number be negative would either be absurd or have trivial degenerate consequences in context. So for our entire paper, we assume these quantities are non-negative.

riable-assumptions

Remark 4.2. Note that our stated protocol hides a lot of pre-made design decisions. We could do none, some, or all of the following: reward the prover for succeeding challenges, punish the prover for failing challenges, rewarding the challenger for making a successful challenge, and/or even punishing the challenger for making an unsuccessful challenge. In particular, a reader already

[3] The current Ethereum 2.0 design does not allow fraudulent challenges to be created. However, we think it is useful to generalize the context to also look at a game where challenges can be fraudulent, since alternate designs to the Ethereum 2.0 protocol or other blockchain protocols may very well consider such choices.YZ

familiar with the Ethereum 2.0 design may know that in the chunk challenge, we do not punish the challenger for unsuccessful challenges, but in the bit challenge, we do. Rather than starting with the design as already known, our philosophy in this paper is to work with designs more general than the Ethereum implementation, even if some of the parameters end up being 0 anyway. This approach serves to clarify heuristics that actually lead to the current designs. Doing this for every possible design parameter would make for too much clutter in notation, so we aim to give a reasonable balance by including options that may be "natural" to a protocol designer a priori.

From this protocol description, it is tempting to just model the strategies of the game as the possible actions from the protocol description (e.g. have the strategies for V_P to be defined by the set $S = \{$correct attestation, incorrect attestation, no attestation$\}$. In this case, the game becomes trivial to analyze (fraudulent challenges are irrational, after which it is obviously better to give a correct attestation over an unsuccessful one, etc.). However, this result is not satisfactory, since in real-life it is intuitively rational for a prover to prefer to lazily do an incorrect attestation instead of a real one to save computational costs if they are unlikely to be caught. Furthermore, one very natural strategy (and the one we want validators to follow) for V_C is to do computation to see if a bit challenge is viable, and challenge if and only if V_P's bit is incorrect. This is is a strategy that is not really covered by something like {challenge, no challenge}. With these considerations in mind, we define the *bit challenge game* to model the following strategies for the protocol:

- V_P chooses between:
  - **Inactive prover (IP)**: do nothing.
  - **Honest prover (HP)**: process the data D, by doing C_W of work, and attest to it with the correct bit.
  - **Dishonest prover (DP)**: attest to D without processing it (and thus providing a random bit, which corresponds to a correct attestation with probability 1/2).
- V_C chooses between:
  - **Honest challenger (HC)**: actually computes the bitfield of V_P (using V_P's now public key), doing C_W work, and then challenges if and only if V_P's attestation bit is incorrect. Note that when using this strategy, challenges, when made, are always successful.
  - **Inactive challenger (IC)**: does no computation and sends no challenge.
  - **Dishonest challenger (DC)**, who does not do the computation but issues a challenge anyway. (So if V_P were honest, this challenge is always fraudulent; if V_P were dishonest, this challenge is correct 1/2 of the time.)

In Section 7, we will first take a quick look at this model, then study our main model where we extend the game to N-players. In this game, each of the N validators in the committee simultaneously chooses a prover strategy and a challenging strategy. The main differences from the 2-player game are:

- Challengers can challenge multiple players now. We model both HC and DC as families of strategies where the challenger picks β (probably different for the two strategies) random validators in their beacon committee to check (in the DC case, checking is challenging, while in the HC case, checking means doing a computation and then challenging incorrect bits). For both strategies, we assume β is the maximum number of checks the challenger is willing to make, coming from the challengers' computational/time resources (in the case of HC) and/or risk profiles (in the case of DC). Note while it may a priori make sense to consider different values of β, it will become apparent from the utility computations that by linearity, a rational challenger will move β to the maximum possible or 0 (and when $\beta = 0$ both strategies are dominated by IC). Thus, it is safe to just assume that when used as a viable strategy, β is a maximum set by the validator.

- Challenges may not be accepted. As we are in a blockchain setting, especially if the dominant strategy involves sending many challenges, the blockchain is only able to handle a certain amount of challenges. To model this, we assume that the protocol itself sets an upper bound M_B on the number of bit challenges that can be processed (included into the chain) per block.

- We also do not need or want the chain to accept more than 1 challenge for any particular prover's attestation, because after the challenge we would have definitively resolved the prover's bit as correct or incorrect. Thus, for our simplified model, we assume that for each attestation, only 1 challenge can be accepted in one iteration of the game. [To my understanding, not necessarily true - when a prover responds successfully by proving a challenge is incorrect, it does not necessarily guarantee that his committed bit is correct. I agree that we do not want more than 1 active challenge simultaneously, but if the first challenge fails, I don't see why someone else should not be able to prove the attestation bit is incorrect.][CJ]

- We model the following challenge acceptance process: the blockchain looks at all the challenged provers (up to N total) and picks M_B of them uniformly at random. Then, for each of the challenged provers V_P, we pick one of the challenges that challenged V_P uniformly at random.

- We assume that in the equilibrium the validators have the same C_*'s and β's, for simplicity. This is actually quite a strong assumption, but we feel it is not a bad estimate of what will pragmatically happen (all the validators taking similar hardware passing the leading benchmarks), and we do not see much reason why deviation from this model would significantly change the usefulness of the results.

ec:chunk-challenge

4.2 The Chunk Challenge

As stated, the bit challenge ideally incentivizes the validators to honestly download the data at the time of attestation. However, it does not necessarily prevent them from immediately deleting the data (especially if they are sure that they will not be bit challenged, or if the bit challenge response implementation does not involve them from having the data). If all validators lazily delete the data after computing their custody bits honestly, the blockchain then would have effectively "forgotten" the data, which we do not want.

This leads to our second goal for data availability, which, true to the name of the problem, is the the desire that **the data is available for an extended period of time** (this is the motivation of our *custody period* time unit). This means that we would ideally like to have the validators download the data when they are supposed to and keep it through one custody period, though our primary goal would still be (less satisfactorily) solved if the validators can ensure they can get the data somehow (such as having a third party download the data for them, or downloading the data at the last-minute when pressed). The natural way to ensure this is to create a situation where validators can be randomly challenged to provide pieces of data that they are supposed to have downloaded, which is exactly the main idea behind the *chunk challenge.* [As afore mentioned, a challenge period is 72 days, whereas a period is approximately 9, so this may be in conflict. Also, if data is only guaranteed for a single period, early attestations within a period have lower chance to be found and challenged versus later attestations][CJ]

Recall that crosslinks contain references to the shard blocks where data is stored in Merkle trees. When a crosslink for D is made, members of the beacon committee compute the Merkle tree for D. The leaf nodes in the Merkle tree are hashes of n-bit strings, or *chunks* [4] of D (for some fixed global constant n), and each parent node contains a hash derived from its child node's hashes. As in the bit challenge, a validator may take the role of "challenger" to issue a challenge to another validator ("prover") by specifying the index of a leaf of the Merkle tree for D, corresponding to one of the chunks. When challenged, a validator is responsible for providing a Merkle proof. Abstractly, we define the chunk challenge protocol as follows:

1. The challenge period begins when a crosslink has been created, and lasts for a custody period.

2. The challenger V_C can choose between:
   - **Challenge**. V_C gets a reward R_C for sending a challenge.
   - **No challenge**. Nothing happens.

3. When challenged, the chunk challenge mechanism will eventually resolve V_P's response as **successful** indicating that V_P's response is a correct

[4] We note that this is similar in spirit to the chunkify function used in the custody bit, but these two types of chunks are not necessarily of the same length and can be considered independently.

Merkle proof for his/her chunk, or **unsuccessful** if the Merkle proof is incorrect (or if V_P does not send a response).

4. If the response is successful, V_P gets a reward R_P for passing the challenge. If the response is unsuccessful, V_P is slashed and penalized P_S, while V_C gets an additional slashing reward R_S.

As before, the intuitive set of strategies from the protocol description does not satisfactorily cover what we expect to see in real life. We define the *chunk challenge game* as the game with the following strategies for the chunk challenge protocol:

- V_C chooses between (both self-explanatory):
  - **Challenge (C)**.
  - **Not Challenge (NC)**.
- V_P chooses between:
  - **Work (W)**: by downloading and processing the data, which guarantees a successful response when challenged (in exchange for doing the work, which is worth some cost C_W). [Does storing data incur cost, and is it captured in C_W?][CJ]
  - **No work (NW)**: by not downloading or processing data, and then generating a response somehow when challenged. This means V_P gets C_W more utility compared to using W, but when challenged, loses C_L utility by having to do some late work. Afterwards, V_P produces a successful response with probability q_{avail} (we use this notation because a successful response means that the data is available to V_P at the time of the challenge) and a non-successful response with probability $(1 - q_{\text{avail}})$.

In particular, the NW strategy is really a generalized strategy that includes many potentially realistic options, such as:

- Being purely lazy and never giving a response. This corresponds to spending $C_L = 0$ utility and generating a correct response with probability $q_{\text{avail}} = 0$.
- Outsourcing to a third party would correspond to spending an outsourcing cost C_L and then generating a correct response with probability q_{avail} measuring the reliability of the third party.
- Simply delaying the necessary download until asked can be modelled as $C_L = C_W$ and q_{avail} measuring the validator's reliability at being able to pass "last-minute" chunk challenges.

As for the bit challenge, we think of the 2-player game as a toy model and the real game we are interested in is an N-player model, where every validator can take both the challenger and prover roles. Thus:

- Unlike the bit challenge, where everyone chooses a challenging strategy and a proving strategy, we model the N-player model as having one challenger (considered to have been selected at random from the committee) and the other committee members being provers.
- As in the bit challenge, the challenger can challenge multiple provers. We define M_C to be the maximum number of chunk challenges allowed by the protocol to be written per block.
- However, unlike in the bit challenge, we model C as a strategy where the challenger picks $\beta = M_C$ random validators to challenge (note that this is a bit different from the "self-imposed" maximum β in the bit challenges; this is because a chunk challenge is essentially free and does not require a computation, so in practice everyone will always write the maximum).
- Unlike in the player bit challenge, we model these β challenges to be accepted automatically into the blockchain (this is because the challenger role is actually taken by the validator who is writing a block, who will immediately write his/her own challenges into the block). [Shouldn't the criterion for acceptance be actually tied to block acceptance (so not automatic acceptance but conditional, which should be easy to achieve as long as blocks are formatted correctly). Additionally there is a fee for non proposer messages in a block correct? This means that rational actors would only issue challenges when writing into their own blocks, because it would be negative utility to ask someone else to include their challenge. So, while its not forced that the challenger must be the block proposer, the incentives align both as proposer and non-proposer that only a proposer will want to make challenges.][CJ]
- Like in the bit challenge, we assume that in the equilibrium the provers have the same C_W's and q_{avail}'s, for simplicity. One justification for such an assumption is that a "leading" strategy NW (such as a common outsourcee or a particular level of network reliability to download data quickly when challenged) is likely to be commonly adapted for the validators who want to be lazy.[YZ] [Do readers understand that C_W are similar but not identical between the two challenge games?][CJ]

Some of these differences between the bit and chunk challenges may seem frivolous; we now take a quick detour to explain these design differences before introducing the early key reveal.

sec:bit-vs-chunk

4.3 Contrasting the Bit and Chunk Challenge Designs

Fundamentally, the bit challenge looks for provable errors (attesting a wrong bit) while the chunk challenge looks for hard-to-prove (and impossible-to-prove in the worst case) errors (not having data downloaded). In the bit challenge game, we are challenging a commitment to a bit, so a dishonest prover cannot respond to it successfully after acting dishonestly, [Will be true only if fraudulent challenges

are impossible][CJ] while in the chunk challenge a dishonest prover can respond to challenges successfully. Also, unlike an "honest" chunk challenge, which is essentially free computationally (because it is just sending an index), an honest bit challenge takes some computational resources. This means that from the protocol's computation-minded point of view, an erroneous bit challenge is a waste of the protocol's time, while an erroneous chunk challenge is a necessary evil. [Not sure what an erroneous chunk challenge is, nor why it would be a necessary evil][CJ] Together, these two differences between the foundations of the two challenges explain the main difference between the two challenges: **we desire bit challenges to only occur when necessary, and for chunk challenges to always occur.** This is the heart of the asymmetry between the two designs, including:

1. It makes sense to have a reward for simply making a chunk challenge (and no punishment for an incorrect one), while there is no reward for simply making a bit challenge (and a punishment for an incorrect one).

2. In the bit challenge, we desire challengers to do the work of checking others but send only honest challenges, and we would actually be happy if at least one challenger simply challenged **everyone** who made bad attestations.

One especially important consequence of these differences is that unlike bit challenges, chunk challenges are not only essentially "free" computationally, they are free utility because (slight spoiler!) C dominates NC (in both the 2- and N-player versions of the chunk challenge game). This has its own set of very important corollaries for the design:[YZ]

1. Since everyone wants to write as many chunk challenges as possible, we get the tricky consequence that it incentivizes a block proposer to include his/her own chunk challenges over others (or, say, for a savvy block proposer to steal someone else's chunk challenge, in case the proposer thinks that validators are more likely to challenge validators who are more likely to fail the chunk challenge). Thus, we might as well make it part of the protocol for the chunk challenge that only a block proposer makes chunk challenges, which explains why our N-player chunk challenge game only has one challenger and all the challenges are automatically accepted.[YZ]

2. Meanwhile, in the bit challenge game, fraudulent challenges being punished disincentivizes block proposers to just randomly make their own bit challenges (and also disincentivizes them to steal other validator's bit challenges as their own), so we use a more complicated model where all the validators can challenge other validators and only some of the challenges are accepted.[YZ]

3. Because chunk challenges are "free money," we must set an upper bound β on the number of chunk challenges accepted to the blockchain per block, and we can assume all rational validators will, when selected to propose a block, write β chunk challenges.[YZ]

4. Thus, the protocol-wide bit challenge bound β is motivated by game theory, while the protocol-wide bit challenge bound M_B is motivated by engineering purposes (such as having a predictable system throughput of challenges and having a predictable block structure) and not game theory.

5. In the chunk challenge case the best challenging strategy's number of challenges β must equal the protocol-wide chunk challenge bound M_C, while in the bit challenge case it is not predetermined whether the "self-ordained" personal bound β or the protocol-wide per-block bound M_B is the bigger bottleneck on how many challenges are accepted.

sec:key-reveal

4.4 Early Key Reveal

Recall that a validator is supposed to download the data and compute their own custody bit when they attest. We want to avoid validators being lazy and outsourcing this responsibility with on-chain incentives.

In particular, the protocol issues a reward to a validator who can prove he/she knows another validator's key s_V before the keys are supposed to be revealed (by e.g. showing they can sign a message with s_V). We model the *early key reveal protocol* as having 2 participants, a validator V and a *third-party* V_T, as follows:

- The protocol period is the custody period, and ends when the keys are revealed.
- The validator V can choose between:
  - **Outsource**: V gives key to V_T.
  - **No Outsource:** V does the work honestly him/herself.
- If V does not outsource the work, then V_T does nothing and V obtains an attestation reward R_A from the protocol.
- If V outsources the work, the third-party V_T can choose between:
  - **Whistleblow:** V_T reports[5] V to obtain a whistleblowing reward R_S from the chain, and V is penalized a slashing penality P_S. Furthermore, V does not get the attestation reward.
  - **No Whistleblow:** V_T does not report V, and proceeds to do the work for V, giving V back a valid attestation. V then submits the work as his/her own and gets the attestation reward.

Unlike the more complicated chunk and bit challenges, the actions from the protocol description map naturally onto the space of strategies of the participants, so the description of our 2-player *early key reveal game* is very similar to the protocol:

[5]Notice that this reporting is an "on-chain" protocol action, so it can only be done by a validator. In practice, this would mean that even if the third-party is an "off-chain" service, the third-party would need to have some "agent" validator, such as a throwaway validator account, to do the whistleblowing for them. This is why we also use "V" (validator) to denote the third-party.

- The validator V chooses between:
  - **Outsource (O)**: V pays V_T a small outsourcing cost C_O (e.g. US dollars transferred out-of-protocol).
  - **No Outsource (NO)**: V does the work honestly him/herself, which we model as a computational/time cost with utility C_W.
- The third-party V_T chooses between:
  - **Whistleblow (W):** we know already that V_T gets the on-chain whistleblowing reward R_S, but we should also model the exogenous costs for V_T to whistleblow (in the case that V outsources) with a variable C_E.
  - **No Whistleblow (NW):** as V_T is doing the work, we say that V_T pays a computational/time cost C'_W; note that unlike on-chain rewards, we cannot assume that $C_W = C'_W$ since the capabilities of V and V_T are probably very different.

A natural way of modeling the actual N-player situation is to have many validators and third-parties. If we assume the third-parties are interchangeable (for example, assuming market competition have created a standard for the third party), then the analysis is essentially equivalent to the 2-player version. If we assume the third-parties were different, then the analysis simply breaks into cases depending on which third-parties are being considered by a rational validator, after which it is natural to assume that rational validators would converge to using the "best" third-party (or third-parties with equivalent capabilities and reputation) and the game would dynamically converge to the interchangeable third-party case. Because of these arguments, we cap our analysis to the 2-player game, predicting that it captures a significant amount of the intuitions of the N-player situation.

Remark 4.3. It is also reasonable to consider how the reputations of third-parties change over time, if it is feasible for the market of third-parties to diverge to multiple equilibria, such as "small" third-parties with less reputational costs (who can repeatedly change identities in the pseudonymous blockchain context to stay active) versus "large" third-parties with heavy reputational costs but more reliable clients. In our work, we choose to not actually model these complications because modeling a market and reputational costs is very different from the main types of analysis we do in this paper and does not occur in the other two games. It is, however, an interesting situation to analyze, possibly for future work, especially if we get some real-life data and precedents to calibrate on.

4.5 Notes on our Analysis

We say that the utility parameters given that are set by the protocol (such as the R's, P's, or M_B in the bit challenge) are *endogenous* variables, and the other ones are *exogenous* (such as C_*'s, or β in the bit challenge). This distinction is important for many reasons:

- When we solve for constraints from the perspective of a protocol implementer, we can only manipulate endogenous variables.
- We cannot control and can only try to bound exogenous variables. For example, it seems unreasonable for our scope to put a reasonable bound for C_E, because reputational costs are related to hard-to-model things such as the third-party's other potential clients.
- Endogenous variables are set for the game, while two different validators may have different valuations for e.g. C_W and C'_W given their different resources and capabilities.

Our approach to our upcoming analyses are as follows: First, we do a Nash equilibria analysis, which naturally includes the analysis of when certain strategies are dominant, where we already desire some strategies to be dominant over others. We also look at other desires that we have as protocol designers (e.g. collusion resistance). Our end goal is to find constraints on our parameters that allow as many of our desires to be met as possible, and to recognize when and what causes certain desires to be not enforceable by the protocol. This last point is where the endogenous / exogenous variable split is most relevant, as we can only control endogenous variables.

We also abstract away some important but distracting implementation details. Unless otherwise noted, we make these assumptions for the rest of the paper:

- **We assume challenges and responses are accepted immediately.** Challenges and responses are types of messages broadcast to the network, which, like any other messages in a blockchain context, are only confirmed if someone writes it into a block. Thus, we need the protocol to give adequate incentives and time for these messages to be written into blocks, and to disincentivize validators to ignore or even censor messages from certain validators. While this aspect can and should itself be analyzed (in fact, the protocol should have some cryptoeconomic incentives to get validators to include such messages), it is distracting to the main structure of these games to also insert roles such as the block proposer who chooses to publish the response. Thus, we analyze all of these games as if challenges and responses are written into the chain directly "in real time." [YZ]
- **We assume the data is actually available to start with.** Obviously, due to network issues, it is possible that one or more validators are simply unable to download the data to start with even if they were honest. While this is clearly relevant for our analysis, accounting for this type of issue is tangential and not very enlightening for our main issue, which is the incentives of validators during normal operation.

5 Analysis of Early Key Reveal

The early key reveal game is the simplest to analyze, so we lead with it as a warmup. In some sense, this will be the least satisfactory game to study due to

R_A	reward to V for attesting
R_S	reward to V_T for whistleblowing
P_S	slashing penalty to V for whistleblowing
C_O	cost to V and gain to V_T for outsourcing
C_E	other extrogenous costs for V_T for whistleblowing
C_W	cost to V for honestly working (download the data and compute custody bit)
C'_W	cost to V_T for honestly working (download the data and compute custody bit)

Table 1: The parameters for the early key reveal game. tab:key-reveal-params

many factors outside of our control, but we will see concepts that will reappear in the analyses of other games, where we are able to say much more.

As protocol designers, we desire V to not outsource and V_T to whistleblow when outsourced. Our main goal is to see the extent to which we can incentivize these actions.

5.1 2-Player Analysis

We start by recalling the non-negative notation for the game in Table 1. Using them, we compute the utilities of the game given these strategies and express them in Table 2.

tab:key-reveal-params

tab:key reveal

:key reveal payoff

V_T \ V	O	NO
W	$(C_O + R_S - C_E,\ -C_O - P_S)$	$(0,\ R_A - C_W)$
NW	$(C_O - C'_W, R_A - C_O)$	$(0,\ R_A - C_W)$

Table 2: Payoff matrix for the key reveal scenario. tab:key reveal

The main observation from the utility computations is that R_A is in all the utilities for V except in the (W, O) scenario, because it is assumed that V_T will not actually do any work and is whistleblowing instead.

key-reveal-collude

Remark 5.1. There is an exception to the above comment about (W,O) if the two parties are actually colluding to extract stake from the protocol, in which case V_T would both give V the work (or just have V do the work him/herself) so V can collect an attestation reward R_A, and simply slash V afterwards to collect the whistleblowing reward. This pathological case actually becomes relevant when we analyze collusion in the next subsection.

We start by looking at dominant strategies (including which strategies we prefer to be dominant):

1. **We want third-parties to prefer whistleblowing to not whistleblowing.** This requires $R_S - C_E > C'_W$, or $R_S > C_E + C'_W$. While R_S and C'_W are relatively easy to control, C_E is very difficult to control since it is determined by a lot of factors (how much the third party cares about

reputation, etc.). Thus, we are unable to guarantee this unless we have a good model of reputational utility, though obviously setting R_S high incentivizes whistleblowing.

2. **We want validators to prefer not outsourcing to outsourcing.** Because of the above, we can see that it is difficult to assume either W or NW to be a dominant strategy for V_T. If W is dominant (or close to dominant), then $R_A - C_W$ is almost always going to outclass $-C_O - P_S$, since we can assume that R_A and P_S are quite large (being in units of stake on-chain) compared with the computational cost C_W. $C_O < C_W$ is the only case worth considering, as otherwise outsourcing clearly has no benefit. If NW is close to dominant, then $R_A - C_O > R_A - C_W$ because we can assume $C_O < C_W$, so outsourcing becomes close to dominant.

We now solve for the nontrivial Nash equilibria (when no strategy is dominant). At the equilibrium, we can assume that V and V_T plays O and W with probabilities q_O and q_W respectively, both strictly between 0 and 1 (the strictness is important because otherwise it reduces to the cases with dominant strategies).

First, we need V_T to be indifferent between W and NW (because the payoffs for NO are the same for V_T). We recall that this requires $R_S = C_E - C'_W$.

Then, we need V to be indifferent between O and NO. To balance, we need

$$q_W(-C_O - P_S) + (1 - q_W)(R_A - C_O) = R_A - C_W,$$

which gives

$$q_W = \frac{C_W - C_O}{R_A + P_S}.$$

Note that q_O never shows up in our analysis, which means that this particular set of Nash equlibria, it does not matter what V does (as long as $0 < q_O < 1$.

Finally, we consider other things we may want besides dominant strategies:

1. **We want to avoid collusion where a validator and third party make money from slashing.** To avoid this, we note that the total in-protocol payout to the two parties for (W, O) is $R_S - P_S$, so we need to ensure that $P_S > R_S$ for this to be gamed. In fact, in this situation, as stated in Remark 5.1, the validator will probably collect the attestation reward (by taking V_T's work), so we should assume that the duo gets an extra R_A. Therefore, the safer constraint is actually $P_S > R_S + R_A$.

5.2 The N-player Early Key Reveal Game

As stated in Section 4.4, we do not explicitly model the N-player case, expecting most of its behavior to be captured by the 2-player case. We predict that game basically becomes a sum of many independent copies of the 2-player game, as each validator considers outsourcing or not, and additionally which third-party to use. The main difference would be that the reputational cost would probably increase with the number of players (as the third-party would have a larger

potential market cap at stake) which increases C_E significantly. Thus, we predict that the third-parties who are providing their outsourcing as a service to many customers would be very disincentivized to whistleblow, so the validators who are savvy at finding the reliable third-parties would be very likely to take the outsourcing option if the price C_O is small; conversely, we can intuit that "personal-scale" third-parties may find a niche for themselves in the market especially if sufficiently many validators are more risk-neutral, though they would be more likely to whistleblow than "service-scale" ones.

5.3 Summary

1. If $R_S > C_E - C'_W$, the rational third-party will always whistleblow, and the validator will not outsource. If $R_S < C_E - C'_W$, the third-party will never whistleblow, and the validator will always outsource.
2. If $R_S = C_E - C'_W$, we get a nontrivial family[a] of Nash equilibria where V_T plays W with probability $\frac{C_W - C_O}{R_A + P_S}$ and V plays O with any probability $0 < q_O < 1$.
3. Big P_S incentivizes validators to not outsource, while big R_S incentivizes the third party to whistleblow. However, neither of these can be guaranteed by the protocol.
4. For collusion protection, we need $P_S > R_S + R_A$. We have total control over this.
5. We expect our 2-player analysis to capture most of the insights for the N-player version. We expect C_E to be quite large for "service-level" third-parties, and quite small for personal-level third-parties.

[a] The uncertain nature of C_E is a double-edged sword: for one, it makes the equation $R_S = C_E - C'_W$ impossible to balance exactly, which seems to trivialize this result. However, by the same uncertainty, this means that C_E is basically impossible to compute **even by** V_T, so "in practice" it is possible that this lack of precision would force a mixed strategy on behalf of V_T.

6 Analysis of the Chunk Challenge Game

sec:analysis-chunk

We now analyze the chunk challenge game. Unlike the early key reveal toy example, there are many nontrivial differences between the 2-player and N-player games. So instead of completely analyzing the 2-player game, we will first gain some simple intuition from the 2-player game, then separately consider the N-player game.

As protocol designers, we desire for V_C to prefer to challenge (to keep the chain honest) and for V_P to prefer to work. We will see that the former is trivial to guarantee (with endogenous parameters) but the latter is impossible to

R_C	reward to V_C for challenging
R_S	reward to V_C for successful challenge
P_S	slashing penalty to V_P for successful challenge
R_P	reward to V_P for successful response
C_W	cost to V_P for honestly working (download the data and compute custody bit)
C_L	cost for a lazy V_P to implement his/her "when-challenged" strategy when challenged.
q_{avail}	the probability that the NW strategy produces a correct response for V_P

Table 3: The parameters for the chunk challenge game. tab:chunk-params

guarantee. All is not lost, however; **as long as q_{avail} is high in the equilibrium strategy, it means a generic prover is incentivized to somehow make the data available**, even if he/she is playing hooky by not downloading the data until asked. Thus, our more nuanced goal is to maximize q_{avail} in the equilibrium NW strategy (a useful interpretation of W is a version of NW where $C_L = C_W$ is paid "up-front," but $q_{\text{avail}} = 1$).

6.1 2-Player Analysis

We summarize all the (nonnegative) notation introduced in Section 4.2 so far in Table 3. C_L is particularly important, because in the worst case C_L can be arbitrarily close to 0 – maybe it is very easy for this particular V_P to get a cheap way to outsource the data. Intuitively, this suggests that it is impossible for us to rule out the possibility of a rational lazy prover with just endogenous variables. We will see this formally.

We now compute the utilities of the game, with the reference utility 0 representing the case where no challenges are given and everyone honestly does the work. We express these payoffs with the matrix in Table 4.

table:payoffs1

V_C \ V_P	H	D
C	$(R_C,\ R_P)$	$(R_C + (1-q_{\text{avail}})R_S,\ C_W - C_L + q_{\text{avail}}(R_P) - (1-q_{\text{avail}})P_S)$
NC	$(0,\ 0)$	$(0,\ C_W)$

Table 4: Payoff matrix for the chunk challenge. tab:chunk

Example 6.1. As an example, we compute C versus NW, the most complex calculation. Compared to W (the default), V_P ends up saving C_W of utility from doing honest work, but now must lose C_L utility in reaction to being challenged. After the challenge, he/she gives a successful response correctly with probability p_G, in which case V_P receives R_P (and having skipped out on honest work). The other $(1 - q_{\text{avail}})$ of the time V_P receives loses P_S for being caught. In either case, V_C gets R_C for correctly sending the challenge, obtaining an additional R_S when V_P fails.

To get a handle on this toy problem, we look at when and if the strategies dominate each other:

1. **We want the challenger to prefer challenging over not challenging.** To do this, we need the reward for initiating a challenge $R_C > 0$. We stress that this is stronger than $R_C \geq 0$, in which case the challenger V_C may be indifferent between challenging and not challenging; we want V_C to actively send challenges. Thus, we can now reduce the game to the case where the strategy NC is strictly dominated by C, and V_P is deciding between W and NW, with corresponding payoffs R_P versus $C_W - C_L + q_{\text{avail}}(R_P) - (1 - q_{\text{avail}})P_S$.

2. **We want the prover to prefer working over not working.** Sadly, this is not possible. In a still fairly-realistic worst-case, we can have the prover V_P have such powerful computation/networking resources (or just having access to a third-party with powerful resources), who is happy to do nothing until challenged, at which point he/she immediately downloads the data (or outsources it). This case can be approximated by $C_W = C_L$ and $q_{\text{avail}} = 1$, in which case V_P is indifferent to being challenged. However, this is not a deal-breaker, as our goal is to have the data be available, which it still is in this case! (Just delivered late, and possibly through a third party)

There are a few other things we want besides strategies being dominant; they also impose constraints on our system:

1. **We need to prevent collusion.** Note that by default, an honest challenge/response gives total reward $R_C + R_P$ between the two validators, and when slashed, the total reward is $R_C + R_S - P_S + C_W - C_L$. Thus, validators can collude to be slashed on purpose and not actually have the data available unless $R_P > R_S - P_S + C_W - C_L$, or

$$R_P + P_S + (C_L - C_W) > R_S.$$

   As C_L and C_W are exogenous, the only constraint we can really get out of this is

$$R_P + P_S > R_S.$$

2. **We want the protocol to not leak too much money.** In practice, this means minimizing the R_*'s, which are paid out by the protocol itself. Note that as long as $(R_P + P_S)$, our incentive analyses remain fixed. This means we can raise one while lowering the other (or vice versa), even making one of them 0, though not both. Practically, it is better to make $R_P = 0$ and $P_S > 0$, because R_P is likely to be a recurrent cost paid by the protocol on every correct iteration, so the protocol saves money by minimizing R_P and just having a bigger "threat" of P_S. Furthermore, since $R_C > 0$ and $R_S \geq 0$ already incentivizes V_C to challenge, we actually do not need $R_S > 0$.

6.2 N-Player Analysis

In the actual game, each validator simultaneously plays the role of a prover and a challenger during each period. These roles are essentially independent

(besides pathological things like challenging him/herself). One key difference is that when we have multiple players, the payoffs of using a strategy depends on the distribution of the opposing strategies in the population (for example, the more people making chunk challenges, the more likely a dishonest prover may be caught). Thus, we need to introduce additional notation to describe the distributions of the players using these strategies when we write our matrix:

table:payoffs1

V_C \ V_P	H	D
C	$(\beta R_C,\ q_{\text{cha}} R_P)$	$(\beta(R_C + (1-q_{\text{avail}})q_{NW} R_S),$ $C_W + q_{\text{cha}}(-C_L + q_{\text{avail}}(R_P) - (1-q_{\text{avail}})P_S))$
NC	$(0,\ 0)$	$(0,\ C_W)$

Table 5: Payoff matrix for the N-player chunk challenge. tab:chunk-multiplayer

[@Dash: does this mean we shouldn't use a matrix to do this? Like is this kind of thing not kosher?]YZ

Here, we use the notation:

- q_W, q_{NW}: the probability that a randomly selected prover is playing the said strategy; we assume these sum to 1.
- q_C, q_{NC}: same as above, but for challengers.
- q_{cha}: the probability that a prover is challenged during the period
- β: the number of challenges a validator makes using a C strategy

First, observe that $\beta R_C > 0$ and $\beta(R_C + (1-q_{\text{avail}})q_{NW} R_S) > 0$ when $\beta > 0$, so again C dominates NC, like in the 2-player case. Thus, we know that every rational challenger will use C instead of NC ($q_C = 1$). Then, H dominates D when

$$q_{\text{cha}} R_P > C_W + q_{\text{cha}}(-C_L + q_{\text{avail}}(R_P) - (1-q_{\text{avail}})P_S),$$

or

$$q_{\text{avail}} < 1 + \frac{(q_{\text{cha}} C_L - C_W)}{q_{\text{cha}}(R_P + P_S)}. \tag{6.1}$$

eqn:chunk-availabi

Here, we can compute q_{cha} as follows: there will be on average β challenges during the custody period, and assuming that each validator can be challenged exactly once and that there are N attestations (c.f. the bit challenge analysis in Section 7, where we conclude we can incentivize everyone to attest), $q_{\text{cha}} = \frac{\beta}{N}$.

sec:analysis-bit

Remark 6.2. It is tempting to model that there will be $B\beta$ challenges during the custody period, where B is the number of blocks during the custody period; however, on average validators will only have 1 block worth of time to write chunk challenges for this crosslink, as there would be B overlapping custody periods during every instance in time for people to write chunk challenges.YZ

eqn:chunk-availability

As predicted, Equation 6.1 is impossible to enforce with just endogenous parameters. Specifically, in the worst case, we may have dishonest but resourceful validators with q_{avail} arbitrarily close to 1 and $C_L = C_W$, which breaks this inequality (because $q_{\text{cha}} \leq 1$) no matter what we set the other variables. Our best bet is to assume $C_L = C_W$ for the worst case, and conclude that for provers to

be incentivized to be dishonest, we must have dishonest strategies guaranteeing data availability probability of at least

$$q_{\text{avail}} \geq \frac{C_W(N-\beta)}{\beta(R_P+P_S)}.$$

Insert something about mixed strategies versus challenge limits here[CJ]

6.3 Summary

[TODO: can someone update this? This is a bit tagged to the 2-player game and should contain a few more things from the multiplayer analysis.][YZ]

1. We can make $R_P = R_S = 0$, but we need to keep P_S bigger than the exogenous C_W (the gain from laziness), in order to keep $(R_P + P_S) > C_W$. As C_W is likely accounted for by saving computation, P_S needs to always clearly outscale the cost of computation. As P_S is bounded above by the validator's stake, this means the smallest possible amount of stake that validators can hold must clearly outscale the computational cost.

2. We must keep R_C small but positive (to incentivize challenging). However, we need to bound the number of challenges each cycle, to not leak too much money (in practice, the current specs limit to 4 challenges per block).

3. With these assumptions, V_C is always incentivized to challenge, so his/her optimal strategy is to write as many challenges (by default random) in his/her block.

4. With these assumptions, V_P is always incentivized to pass the challenge by making the data available. The caveat is that this does not necessarily come from V_P being honest, because V_P may be lazy in only downloading / outsourcing the data downloading when challenged.

5. These conclusions are robust under collusion between the two parties.

7 Analysis of the Bit Challenge Game

sec:analysis-bit

As protocol designers, we desire both V_P and V_C to prefer being honest (and this is indeed the trivial Nash equilibrium one obtains in a simplified model with no computational costs). However, this is impossible in our more refined model, as we will soon see.

R_A	reward for V_P for attesting
P_A	punishment for V_P for failing to attest
P_S	slashing penalty to V_P when successfully challenged
R_S	reward for V_C for a successful challenge
P_F	penalty to V_C from issuing a fraudulent challenge
R_F	reward for V_P when responding successfully to a fraudulent challenge
C_W	the total cost to honestly compute the custody bit (by downloading the data and the computing the bit)

Table 6: The parameters for the bit challenge game. tab:bit-params

7.1 2-Player Analysis

t-analysis-2player
table:bit-payoffs1

V_C \ V_P	HP	IP	DP
HC	$(-C_W, -C_W + R_A)$	$(-C_W, -P_A)$	$(-C_W - \frac{R_S}{2}, -C_W - R_A - \frac{P_S}{2})$
IC	$(0, -C_W + R_A)$	$(0, -P_A)$	$(0, R_A)$
DC	$(-\frac{P_F}{2}, -C_W + R_A + \frac{R_F}{2})$	$(0, -P_A)$	$(\frac{R_S}{2} - \frac{P_F}{2}, R_A - \frac{P_S}{2} + \frac{R_F}{2})$

Table 7: Payoff matrix for the 2-player bit challenge. tab:bit

We recall the (nonnegative) parameters in the 2-player bit challenge in Table 6. tab:bit-params We express the expected payoffs between these strategies with the matrix in Table 7. tab:bit

Example 7.1. As an example, we consider DC versus DP. In this case, both validators gain C_W compared to their honest counterparts (our defaults) because they did not do any work. V_P gains R_A because he/she writes an attestation. Half of the time the challenge is fraudulent and V_P gains R_F while V_C loses P_F. The other half of the time V_P loses P_S and V_C gains R_S.

First, HP dominates IP for V_P if and only if $R_A + P_A > C_W$. So as long as the attestation reward (or punishment for not attesting) clearly outclasses the computational work, we can eliminate IP from rational prover strategies. Second, conditioned on IP being dominated, HC strictly dominates DC for V_C if $P_F/2 > C_W$.

Thus, as rational actors eliminate dominated strategies, we can conclude that as long as we set $R_A + P_A$ and P_F to values that clearly outclass computational costs, the game is really between $\{HC, IC\}$ and $\{HP, DP\}$, which vastly simplifies the analysis.

7.2 N-Player: Utilities

Just want to confirm this is 1 player v.s. N players Game? The utility here is the expected payoff for a random player in this game?YW

Mirroring calculations from Section 7.1, sec:bit-analysis-2player we compute the utility functions for the multiplayer strategies in Table 8. tab:bit-multi-utilities We use the following notation:

- C_D: the cost of downloading the data.
- C_C: the cost of computing a custody bit.
- q_{DP}, q_{HP}, q_{IC}: the probability that a randomly selected prover is playing the said strategy; we assume these sum to 1.
- q_{HC}, q_{IC}, q_{DC}: same as above, but for challengers.
- β: the number of checks a validator makes (in context of a DC or HC strategy) The main thing to notice here is that we have divided up our work C_W into $C_D + C_C$, which is because of the possibility of multiple challenges. When performing β (honest) checks while challenging, downloading the data is a 1-time cost, but there is now a per-challenge cost to compute the other validators' custody bits.

We also have some derived quantities, which we compute later:

- q_{caught}: the probability that an attestation is caught by a correct challenge.
- q_{fcaught}: the probability that an attestation is caught by a fraudulent challenge.
- q_{cat}: the probability that a challenge (correct or not) catches the target attestation.

U_{HP}	$R_A - C_D - C_C + q_{\text{fcaught}} R_F$
U_{IP}	$-P_A$
U_{DP}	$R_A - \frac{1}{2} q_{\text{caught}} P_S + \frac{1}{2} q_{\text{fcaught}} R_F$
U_{HC}	$\beta \left[\frac{1}{2} \frac{q_{DP}}{q_{DP}+q_{HP}} q_{\text{cat}} R_S - C_C \right] - C_D$
U_{IC}	0
U_{DC}	$\beta \left[\frac{1}{2} \frac{q_{DP}}{q_{DP}+q_{HP}} q_{\text{cat}} R_S - \left(1 - \frac{3}{2} \frac{q_{DP}}{q_{DP}+q_{HP}}\right) q_{\text{cat}} P_F \right]$

Table 8: The utilities for the strategies in the N-player bit challenge game.

Example 7.2. We compute U_{DC}. For each of our challenges, we get a dishonest validator who has an incorrect bit with probability $\frac{1}{2} \frac{q_{DP}}{q_{DP}+q_{HP}}$ (the denominator is because we only select validators who have attested). Finally, our challenge is only accepted with probability q_{cat}, in which case we get R_S reward. With the same probabilility, we get $-P_F$ for a fraudulent challenge. However, if we challenged an honest validator, which happens with probability $\frac{q_{HP}}{q_{DP}+q_{HP}}$, we also get $-P_F$. Summing the two ways we make fraudulent challenges gives the expression. Finally, we were not honest, so we do not need to pay any computational cost such as C_C or C_D.

Recall that each validator takes both a proving strategy and a challenging strategy. These strategies are (almost) additive. So for example, we can combine the DP and DC strategies to get the utility function

Tried making this equation fit within the margins a little more, but let me know if it needs more adjustment or the formatting doesn't look good.[AB] How

about this? Thanks for noticing![YZ] I tend to prefer the operator starting the subsequent lines with an indent. [DF]

$$U_{DPDC} = R_A - \frac{1}{2}q_{\text{caught}}P_S + \frac{1}{2}q_{\text{fcaught}}R_F + \beta\left[\frac{1}{2}\frac{q_{DP}}{q_{DP}+q_{HP}}q_{\text{cat}}R_S - (1-\frac{3}{2}\frac{q_{DP}}{q_{DP}+q_{HP}})q_{\text{cat}}P_F\right].$$

The only exception is

$$U_{HPHC} = R_A - C_D - C_C + q_{\text{fcaught}}R_F + \beta\left[\frac{1}{2}\frac{q_{DP}}{q_{DP}+q_{HP}}q_{\text{cat}}R_S - C_C\right],$$

where we have $(-C_D)$ instead of $(-2C_D)$, which we would get from simply adding the two utilities. The reason is that an honest validator has downloaded the data already, and thus does not need to re-download the data to challenge honestly[6]. However, as these terms are likely to be dominated by the endogenous parameters, **we will now analyze the problem as if the utilities were additive**. This gives a much cleaner problem without losing any insight.

Using our intuition from the toy game, we select two desiderata to look at first:

1. **We want no inactive provers.** As long as we set $R_A+P_A > C_D+C_C$, IP becomes a dominated strategy by HP. The careful reader will recall that our utilities are not additive; however, because HP is the only prover strategy that improves utility when combined with a challenger strategy (only HC), this "local" dominance persists even when we consider the overall utility.

2. **We want no dishonest challengers. We want people to not be dishonest challengers**: Note that as long as $R_S < P_F$ (and we have control over both terms!), the bracket inside U_{DC} is negative. This means DC is dominated by IC, so we can eliminate DC when we have this assumption.

This is great news for us, as we can, by just changing **endogenous** parameters, eliminate IP and DC immediately. This has a couple of other consequences: one, as $q_{IP} = 0$ in the rational setting, the fraction $\frac{q_{DP}}{q_{DP}+q_{HP}}$ simply becomes q_{DP}. Also, no rational actor will ever make fraudulent challenges, so $q_{\text{fcaught}} = 0$. Finally, we are free to set $R_F = 0$ since the corresponding term goes to 0 in the utilities and as the protocol designer we are incentivized to lower the rewards. This gives a much cleaner set of utilities, which we revisit in Table 9. tab:bit-multi-utilities-simplified

While we would also like to dominate out DP and IC, these are intuitively impossible. To see this, notice that if every validator is an honest prover, then the expected utility for honest challenging becomes negative (which is worse than

[6] As a refresher, recall from Section 3.4 that suppose an honest validator V downloaded D and attested at time T. They are then to keep D for 1 custody period, which means all the other attestors to D will have their s_V expire and revealed during that time (since each s_V lasts for 1 custody period as well). Then they can challenged by V, who still has D at this point. sec:main-goals

U_{HP}	$R_A - C_D - C_C$
U_{DP}	$R_A - \frac{1}{2} q_{\text{caught}} P_S$
U_{HC}	$\beta \left[\frac{1}{2} q_{DP} q_{\text{cat}} R_S - C_C \right] - C_D$
U_{IC}	0

ilities-simplified

Table 9: The simplified utilities for the strategies in the N-player bit challenge game.

IC). In order to minimize loss, each honest challenger will then stop performing checks. However, then being a dishonest prover is more profitable than being an honest prover, because dishonest provers will have no risk of being caught. Consequently, honest challenging becomes a potential strategy again if the number of dishonest provers in the population is large enough to make the expected reward more than the cost of issuing challenges. This means there is a nontrivial Nash equlibrium, which we now compute.

7.3 N-Player: Nash Equilibrium

This part is not particularly clear for me. Hope i follow everything correctly. I have some questions: We want a mix-strategy (non-trivial) NE, so we set $U_{HP} = U_{DP}$? We don't want "HP dominates DP" or "DP dominates HP"? YW

We now establish the conditions for which the proving strategies and challenger strategies achieve Nash equilibrium. To do this, we again set the utility functions equal between the prover strategies and between the challenger strategies and solve for the constraints.

We make a few assumptions about the equilibrium: First, we need to compute the probabilities q_* that we need to solve for the constraints. We use N to denote the number of validators in the committee and M to denote the number of bit challenges accepted into the blockchain during the challenge period for the equlibrium (note that M depends on other factors and is not a priori a constant). We make a few assumptions about the equilibrium:

1. While everyone may have different computational resources, for aggregate phenomena we assume all the validators playing HC make the same number of checks.

2. We can assume that the total number M of accepted bit challenges in the equilibrium is going to be smaller than the number of incorrect attestations (if not, it would mean all incorrect attestations are caught, which would dominate DP and we would not be in the nontrivial equilibrium).

With these assumptions, we can now estimate q_{caught} and q_{cat}, similar to our work for the chunk challenge:

q_{caught}: From the point of view of an incorrect attestation α, during the challenging slot there will be a total of M correct bit challenges accepted by the

network. In optimal play every validator makes exactly 1 attestation during the attesting epoch and there are $(\frac{1}{2}q_{DP}N)$ incorrect attestations, so the probability of being caught is $q_{\text{caught}} = \frac{2M}{q_{DP}N}$; recall that we are assuming that this is a legitimate probability because we assumed a nontrivial equilibrium.

q_{cat}: Each of the $q_{HC}N$ honest validators do β checks, each check with probability $\frac{1}{2}q_{DP}$ of catching an incorrect attestation, so the number of (correct) bit challenges actually sent out to the network is $\frac{1}{2}\beta q_{HC}q_{DP}N$, of which M are selected by the blockchain. This means that each correct challenge has probability $q_{\text{cat}} = \frac{2M}{\beta q_{HC}q_{DP}N}$ of being selected.

At equilibrium, setting $U_{HP} = U_{DP}$ and $U_{HC} = U_{IC}$ gives us

$$\frac{1}{2}q_{\text{caught}}P_S = C_D + C_C.$$
$$\beta\left[\frac{1}{2}q_{DP}q_{\text{cat}}R_S - C_C\right] = C_D.$$

Substituting our q_* values, we obtain

$$q_{DP} = \frac{MP_S}{(C_D + C_C)N}$$
$$q_{HC} = \frac{MR_S}{(C_D + \beta C_C)N}. \tag{7.1}$$

eqn:bit-challenge-

And now, the analysis takes 2 regimes, depending on the nature of M, which has a "phase transition." Recall that M_B is a "protocol-bound" constant limit on the bit challenges accepted per block, whereas we have a "equilibrium-imposed" number of bit challenges $(\frac{\beta}{2}q_{HC}q_{DP}N)$ which comes from each honest validator simply capping him/herself at making β checks. M, the number of correct challenges accepted during the challenge period, is the minimum of these two quantities. Thus, we have two cases:

- $M = M_B$: in this case, M is already a constant, so Equations 7.1 is the solution of the q_* in terms of constants. eqn:bit-challenge-qs
- $M = \frac{\beta}{2}q_{HC}q_{DP}N$, then we now have the equations

$$q_{DP} = \frac{\frac{\beta}{2}q_{HC}q_{DP}NP_S}{(C_D + C_C)N}$$
$$q_{HC} = \frac{\frac{\beta}{2}q_{HC}q_{DP}NR_S}{(C_D + \beta C_C)N}.$$

Manipulating, we obtain

$$q_{HC} = \frac{2(C_D + C_C)}{\beta P_S}$$
$$q_{DP} = \frac{2(C_D + \beta C_C)}{\beta R_S}.$$

In this case, we need

$$M_B > \frac{\beta}{2} q_{HC} q_{DP} N = \frac{2(C_D + C_C)(C_D + \beta C_C)N}{\beta P_S R_S}.$$

7.4 Summary

Do we really find the NE for the N-player game based on the definition of the NE? I am not sure if "There is always a nontrivial nash equilibrium with a mixed HP-DP strategy playing against a mixed HC-IC strategy." But I think we want a mixed HP-DP strategy NE. The utility function depends on the value of β, and β is something we cannot control, so the NE is not fixed. Also, how do the probabilities related to the NE? Does this mean the game has a mixed NE when we can obtain these probabilities? or we have these probabilities when we have a mixed NE? Feel like chicken egg situation. YW

Can we think of this N-player game as a 1 vs many game? For an individual player, there is always a NE for the proving strategy, the NE may be HP, or DP, or the mix of the two, depending on the value of the rewards and penalties, and β. For the challenging strategy, the NE may be HC, or IC, the mix of the two. The problem here is that we cannot control the value of β. I am thinking if we can replace β with something we can control. If not, do we have an expected value or range for β? YW

1. As long as $R_A + P_A > C_D + C_C$, inactive proving is dominated. As long as $R_S < P_F$, dishonest challenging is dominated. These are both within our control.

2. With dishonest challenging dominated, we are free to set $R_F = 0$.

3. With the minor exception of honest proving and challenging saving an additional download cost (which is minor compared to endogenous protocol values), we can assume proving and challenging are independent strategies at little cost (and we know that the actual game incentivizes honesty a bit more than the one with an independence assumption).

4. There is always a nontrivial nash equilibrium with a mixed HP-DP strategy playing against a mixed HC-IC strategy.

5. If the protocol has a limit of M_B bit challenges for the period **and**

$$M_B > \frac{2(C_D + C_C)(C_D + \beta C_C)N}{\beta P_S R_S},$$

we get mixing probabilities $q_{HC} = \frac{M_B R_S}{(C_D+\beta C_C)N}$ and $q_{DP} = \frac{M_B P_S}{(C_D+C_C)N}$.

6. Otherwise (if the inequality fails or we have no protocol limit on accepting bit challenges), the probabilities are $q_{HC} = \frac{2(C_D+C_C)}{\beta P_S}$ and $q_{DP} = \frac{2(C_D+\beta C_C)}{\beta R_S}$.

8 High Level of Confidence in Data Availability

8.1 Random Sampling

If we sample the pool at random, what percentage of validator needs to be sampled to get a high level of confidence that the crosslink is actually linked to valid data? If all validators in the pool are sampled. How many samples does each validator need to do?

Suppose that we have N validators in a beacon committee. We assume that for any beacon committee, we have a minimum of 50% honest validators, so $\frac{N}{2}$ honest validators. We want to determine how many other validators each honest validator needs to check so that every validator in the beacon committee Define the set of honest validators $V = \left\{v_1, v_2, ..., v_{\frac{N}{2}}\right\}$. Each validator has a set of other validators that he/she can check; we will define these sets to be $S_1, S_2, ..., S_{\frac{N}{2}}$. Since we cannot rely on dishonest validators to check other validators, we will only consider the sets of the honest validators in our analysis. Let k be the number of other validators that each honest validator checks. We

want to find the probability that an arbitrary validator, v_i, is checked.

Case 1: v_i is honest
Then v_i appears in every honest validator's set except for v_i's own set. The total number of ways that any validator can choose k other validators to check is $\binom{N-1}{k}$, and the total number of ways that any validator can choose k other validators not including v_i is $\binom{N-2}{k}$. The probability that v_i is checked is

$$P(v_i \, checked) = 1 - P(v_i \, not \, checked)$$

$$= 1 - \left(\frac{\binom{N-2}{k}}{\binom{N-1}{k}} \right)^{\frac{N}{2}-1} = 1 - \left(\frac{N-1-k}{N-1} \right)^{\frac{N}{2}-1}$$

Case 2: v_i is dishonest
Then v_i appears in every honest validator's set. This case is similar to the previous case. Then, the probability that v_i is checked is

$$= 1 - \left(\frac{N-1-k}{N-1} \right)^{\frac{N}{2}}$$

8.2 Probabilistic analysis for a single crosslink

For the safety of crosslinks, the recommended minimum committee size of 128. 128 is chosen because it's the minimum size which is reasonably safe against attackers [But18b]. serenitydg The probability depends on the outcome of the chunk challenge. YW

8.3 Probabilistic analysis for a single validator per game period

According to the spec, each game period is 2048 epochs, a little more that 9 days, each validator will attest 2048 times in one game period. YW A validator is supposed to attest 2048 times in a game period (a little more than 9 days). If they cheat all the time, their probability of being slashed is close to 1. Even we only check 2% of all the attestations, the dishonest validator still has a big chance of being slashed at least once in one game period.

Probabilistic analysis for a single validator per game period					
2025 attestations per game period, calculation based on 9-day game period, 1 attestation/6.4 minutes (ref: beacon chain spec)					
P(being checked for each attestation)	P(wrong bit if cheat)	P(slashed for each dishonest attestation)	being slashed at least once if cheat 20 times in one game period	being slashed at least once if cheat 200 times in one game period	being slashed at least once if cheat 2000 times in one game period
P1	P2	P3=P1*P2	P= 1-(1-P3)^20	P= 1-(1-P3)^200	P= 1-(1-P3)^2000
0.02	0.5	0.01	0.182093062	0.866020325	0.999999999
0.04	0.5	0.02	0.332392028	0.982412053	1
0.06	0.5	0.03	0.456205657	0.997738759	1
0.08	0.5	0.04	0.557997566	0.999715392	1
0.1	0.5	0.05	0.641514078	0.999964947	1
0.12	0.5	0.06	0.709893759	0.999995777	1
0.14	0.5	0.07	0.765761126	0.999999503	1
0.16	0.5	0.08	0.811306671	0.999999943	1
0.18	0.5	0.09	0.848355087	0.999999994	1
0.2	0.5	0.1	0.878423345	0.999999999	1
0.22	0.5	0.11	0.902770034	1	1
0.24	0.5	0.12	0.922437206	1	1
0.26	0.5	0.13	0.938285807	1	1
0.5	0.5	0.25	0.996828788	1	1

Todo: Generalize the analysis with the number of attestations , the sampling rate using some dynamic system model. (assume the ratio of honest validator is 2/3.)

For a single validator, we define the following variables:

- Expected sampling rate for a single attestation is r. $(0 \leq r \leq 1)$
- The number of dishonest attestations made by this validator in a given period is x.

For this single validator, the probability of being slashed for this given period has the formula :

$$P(Slashed) = 1 - \left(1 - \frac{r}{2}\right)^x$$

P converges to 1 for any $r > 0$ as x goes to infinity. For large value of r, P converges to 1 quickly as x goes big.

I think x is also affected by r, can we find the relationship between x and r. Will a dynamic system help? If r =0, x will go to infinity.[YW]

8.4 High Probability of Data Being Available

Can we prove this? How confident we are?[YW]

8.5 Safety Analysis

Safety problem: can the attackers take advantage of the proof of custody game to include some invalid data in the chain? What is attacker's cost for doing this?

I believe that validating that data is downloadable is a different game compared to ensuring that data itself is valid. Ethereum has a system for fraud proofs for invalid data: https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding So I suspect if the game theory on that side is correct, it is difficult or very expensive to attack with incorrect data.[CJ]

9 Design vs Implementation

Address our concerns for the design, foresee any issues for the implementation?[YW]

10 Conclusion

sec:conclusion

[From Danny: "A discussion about the differences between the custody game and erasure encoding and how they solve similar problem and are synergistic would be a good section to have near the conclusion"][YZ]

`https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-c`

`https://arxiv.org/abs/1809.09044`

Can someone look up these sources, cite them, and write 1-2 paragraphs about them?[YZ]

In order to understand more about data availability problem, we will take a deeper look at the concept of fraud proof. Suppose there is an invalid block, a validator then provides a fraud proof, which contains the transaction and some merkle proof, to prove that the block is invalid. A third party verifies the proof and if there is error during the processing step then we concluded that the block is not valid. This system does not work at all time because a validator may create a block (can be valid or not) but does not publish the whole block. Another validator may either falsely raise an alarm that's unjustified to gain a reward. Since not all validators pay attention to the data at that specific time, it's hard to see who the malicious player is. A solution to this problem is having light clients (validators that have bandwidth that is too low to verify the whole blockchain) download random chunks of the block from the network. If selected data is correct, the block is then accepted as valid. However, if there are too many chunks and not enough light clients, a dangerous attack can still occur. Erasure code helps solve this problem by expanding a piece of data M chunks into a piece of data N chunks long. The light client checks the extended data to see if the validity of the block. This process depends on the "honest minority assumptions" which means there are more honest light clients and validators than dishonest.

I mainly used the second article to write this[KH]

References

al2018fraud

[ABSB18] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 2018.

rt2019introduction

[ABT19] Victor Allombert, Mathias Bourgoin, and Julien Tesson. Introduction to the tezos blockchain. *arXiv preprint arXiv:1909.08458*, 2019.

adler2018astraea

[ABV+18] John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A decentralized blockchain

oracle. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1145–1152. IEEE, 2018.

buterin2017casper [BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv e-prints*, page arXiv:1710.09437, Oct 2017.

osslink-committees [But18a] Vitalik Buterin. Serenity design rationale. 2018. `https://notes.ethereum.org/@vbuterin/rkhCgQteN?type=view#Crosslink-committees`.

serenitydg [But18b] Vitalik Buterin. Serenity design rationale. 2018. `https://notes.ethereum.org/@vbuterin/rkhCgQteN?type=view#Serenity-Design-Rationale`.

astro1999practical [CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *Operating Systems Design and Implementation*, volume 99, pages 173–186, 1999.

aard1988randomness [Dam88] Ivan Bjerre Damgård. On the randomness of legendre and jacobi sequences. In *Conference on the Theory and Application of Cryptography*, pages 163–172. Springer, 1988.

beacon [Dev19] Ethereum Developers. Ethereum 2.0 phase 0: The beacon chain. 2019. `https://github.com/ethereum/eth2.0-specs/blob/dev/specs/core/0_beacon-chain.md`.

liu2019survey [LLW+19] Ziyao Liu, Nguyen Cong Luong, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. A survey on applications of game theory in blockchain. *arXiv preprint arXiv:1902.10865*, 2019.

akamoto2008bitcoin [N+08] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

nash1951non [Nas51] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

polinsky2015secure [SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

wood2014ethereum [W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.