Spacecoin: A Cryptocurrency Based on Proofs of Space

Sunoo Park*[1], Krzysztof Pietrzak†[2], Joël Alwen†[2],
Georg Fuchsbauer†[2], and Peter Gaži†[2]

[1]MIT
[2]IST Austria

Abstract

We propose a decentralized cryptocurrency based on a block-chain ledger similar to that of Bitcoin, but where the extremely wasteful proofs of work are replaced by proofs of space, recently introduced by Dziembowski et al. (CRYPTO 2015). Instead of requiring that a majority of the computing power is controlled by honest miners (as in Bitcoin), our currency requires that honest miners dedicate more disk space than a potential adversary.

Once a miner has dedicated and initialized some space, participating in the mining process is very cheap. A new block is added to the chain every fixed period of time (say, every minute), and in every period a miner just has to make a small number of lookups to the stored space to check if she "wins", and thus can add the next block to the chain and get the mining reward. Because this check is cheap, proof-of-space-based currencies share some (but not all) issues with currencies based on "proofs of stake", like Peercoin. Concretely, a naïve solution that simply replaces proofs of work with proofs of space raises two main issues which we address:

Grinding: A miner who can add the next block has some degree of freedom in shaping how the chain looks, e.g. by trying out different sets of transactions to include in her block. The miner can try many possible choices until she finds one which results in a chain that allows her to also mine the next block, thus hijacking the chain forever while dedicating only a small amount of the space. We solve this problem fully by "decoupling" the hash chain from the transactions, so that there is nothing to grind. To bind the transactions back to the hash chain, we add an extra signature chain, which guarantees that past transactions cannot be altered once an honest miner adds a block. Our solution also gives a simple and novel way to solve the grinding problem in currencies based on proofs of stake.

Mining multiple chains: Since checking whether one can add a block is cheap, rational miners will not only try to extend the so-far-best chain, but also try other chains, in the hope that they can extend one of them which will ultimately catch up and overtake the currently-best chain. (In the context of proof-of-stake-based currencies this is known as the "nothing-at-stake" problem.) This not only gives rational miners a larger-than-expected reward (compared to what honest miners get), but also makes consensus very slow, if not impossible. Our solution to this problem is based on penalizing miners who try to work on more than one branch of the chain.

Finally, we show formally that the proposed cryptocurrency has desirable game-theoretic properties at least as strong as those shown of Bitcoin (namely, honest mining behavior is an equilibrium if no single party holds more than half of all space invested in the currency).

*Research supported by NSF Eager CNS1347364, NSF Frontier CNS1413920, the Simons Foundation (agreement dated June 5, 2012), Air Force Laboratory FA875011-20225, and Lincoln Lab PO7000261954.

†Research supported by ERC Starting Grant (259668-PSPC).

1 Introduction

Bitcoin is a decentralized digital currency which was introduced in 2009 [Nak09] and by now is by far the most successful digital currency ever deployed. Informally, in the Bitcoin scheme a digital coin is attached to a public key pk of a digital signature scheme, and it belongs to the person holding the corresponding secret key sk. In order to transfer a coin from pk to pk', a transaction (encoding the statement that the coin belonging to pk is transferred to pk') is signed using sk. The complete record of transactions is kept in a public ledger that has the form of a *block chain*, which is a sequence $\beta_0, \beta_1, \ldots$ of blocks. Each block $\beta_i = (tx_i, \ldots)$ contains, among other information discussed below, a set tx_i of new transactions to be added.

The main difficulty when designing a digital currency is to provide security against double-spending. In the scheme just outlined, a dishonest party P holding sk can sign two different transactions, the first being a "real" transaction tx' transferring the coin to the public key pk' of a recipient R (typically in exchange for some goods), the second tx'' transferring it to itself, i.e. to some pk'' whose corresponding sk'' is known to P. The recipient R will only possess the coin if ultimately tx' (and not tx'') ends up in the ledger.

To prevent double-spending, one thus must enforce that all parties in the network agree on the same block chain, except for the last few blocks. A party should wait before accepting a transaction, until it has been in the chain long enough that she can be reasonably sure that it will stay there forever. The Bitcoin protocol achieves consensus by making it computationally hard to add a block to the chain, as detailed in the following.

1.1 Proofs of work (or, how to mine Bitcoins)

In order to extend a block chain $B^i = \beta_0, \ldots, \beta_i$ with a new block β_{i+1}, the block β_{i+1} must contain a so-called proof of work (PoW) [DN93]. In the case of Bitcoin, the PoW is simply a nonce μ_i such that the hash $\alpha_{i+1} = hash(\mu_i, \alpha_i, \tau_i, pk_{miner})$ of μ_i together with the hash α_i of the previous block, the set of transactions τ_i to be added and some public key pk_{miner} (whose function we explain below) starts with some sufficiently large number of 0's. If $hash(\cdot)$ is modeled as a random oracle (in reality it is SHA-256), finding a nonce that starts with t zeros requires an expected 2^t number of evaluations of $hash(\cdot)$.

The number t in the Bitcoin scheme is set dynamically (and adapted roughly every 2 weeks) so that the total computational power of the network is expected to find a fresh block every 10 minutes. Searching for such a block is called mining. To incentivize mining, every newly added block generates some fresh coins, which are given to the public key pk_{miner}, whose corresponding secret key is known by the miner who found the block. There is also a mechanism to incentivize mining via transaction fees.

Once a miner solves the PoW, she generates a block and sends it out to the network. As this block needs some time to propagate through the network, it can occur that two blocks extending the same chain are being sent out, and thus there is an inconsistent view on the chain. The Bitcoin protocol specifies that every miner should try to extend the *longest valid branch*[1] it is aware of; this way, even if the block chain branches, ultimately one branch will become longer, and the shorter ones will eventually be ignored. Thus, once a transaction has been added to the block chain, and sufficiently many blocks have been added after it, it will stay in the chain forever (waiting for 6 blocks, which takes roughly one hour, is considered safe).

In order to double-spend a coin, a cheating party would have to branch off sufficiently far in the past and then make this branch "catch up" with the currently longest branch. A party cannot do this with reasonable probability, unless it controls close to 50% of the entire hash

[1] Morey precisely, it is not necessarily the longest branch, but the one that required most computation to find.

power. A crucial property of the Bitcoin protocol is the fact that miners have a strong incentive to follow the "work on the longest chain" rule. A rational miner wants the blocks he finds to end up in the block chain (and not some dead branch), and to achieve this, working towards extending the longest known chain is a Nash equilibrium.

Consensus in Bitcoin. One can think of the core of the Bitcoin consensus protocol as a random selection protocol. In every round a miner is picked at random, and this miner is allowed to add a block. The probability of a particular miner being chosen is proportional to the fraction of the total hash power it controls. If honest miners control more than half of the hash power, this will result in a consensus protocol for the ledger.

Honest miners must constantly invest more computing power than a potential adversary could accumulate in order to prevent double spending. It is estimated that the mining currently uses several 100 MW of energy; moreover, most mining is currently done by dedicated hardware, which has no use beyond mining Bitcoins. For these reasons Bitcoin is considered an "environmental disaster" [And13] by some.

1.2 Proofs of stake

In order to find a "greener" decentralized cryptocurrency, the community has looked for alternative decentralized consensus protocols and found a potentially promising candidate in *proofs of stake* (PoStake). Whereas in the PoW-based Bitcoin scheme the probability that a party mines the next block is proportional to the fraction of computing power it contributes, in a PoStake-based scheme this probability should correspond to the fraction of coins (out of all coins ever minted) that it holds. This idea is very appealing as no resources (like energy, hardware, etc.) are wasted, but unfortunately, making this approach actually work turns out much more delicate than for PoW-based schemes. Trying to adapt Bitcoin in a straightforward way by replacing PoW by PoStake, one runs into at least three major problems which we discuss now.

Multiple chains: In Bitcoin, a rational miner will always work towards extending the longest chain known to him, as working on any other chain would only lower the probability that a potentially mined block will end up in the block chain. If we use PoStake instead of PoW, then checking whether one can extend a chain is very cheap, and thus there is no reason not to try extending many different chains in parallel. This impedes quick consensus finding like in Bitcoin, where rational miners concentrate on the longest chain, which will make it grow faster than others.

Grinding: In Bitcoin the miner adding the ith block can influence the hash of the chain up to block i by his choice of transactions to be included. Influencing this value does not result in any advantage in a PoW-based scheme like Bitcoin. In a PoStake-based scheme on the other hand, the miner adding block i can try out many different hashes until he finds a "good" one which will allow him to also add block $i+1$ (and thus the miner could hijack the chain forever). As before, the reason this is only a problem with PoStake but not with PoW-based schemes is that checking whether one can extend after a block is cheap.

Participation: In a PoStake-based scheme the parties holding coins must also participate in securing the currency by providing blocks when their coins "win". As typically only a fraction of the parties holding coins will participate, this makes designing a PoStake-based scheme difficult (as the scheme must work no matter what the fraction is, and it will also constantly change over time). Moreover, this is a security issue when participation is very low (as for double-spending it is sufficient to control a fraction of the coins that is slightly larger than that of coins participating). In the most popular PoStake-based currency, Peercoin, participation is already below 10%.

1.3 Proofs of space

In this paper we propose digital currency schemes which are based on so-called *proofs of space* (PoSpace), recently introduced by Dziembowski et al. [DFKP15]. A PoSpace is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ that has two phases. After an *initialization phase*, the prover $\mathcal{P}$ is supposed to store some data S_γ of size $N \in \mathbb{N}$, and the verifier stores a short commitment γ to this data. In a later *execution phase* $\mathcal{V}$ sends a challenge c to $\mathcal{P}$, who can efficiently answer with a short answer a after reading a small fraction of S_γ.

In [DFKP15] a PoSpace is constructed where any prover that can convince the verifier with constant probability must either be honest (meaning it has dedicated N bits of space), or run in time $\Theta(N)$ in the execution phase. Note that this is the best we can hope for, as a cheating prover can aways just store the short communication from the initialization phase and then simply re-initialize the entire storage during the execution phase, which takes time $\Theta(N)$.[2]

The constructions from [DFKP15] are based on *hard-to-pebble graphs*. The vertices of such a graph are labeled, where for some unique nonce μ, vertex i gets label $hash(\mu, i, l_1, \ldots, l_t)$ where $l_1, \ldots, l_t$ are the labels of its children. The prover computes and stores those labels and sends a Merkle-hash of all labels to the verifier. In the execution phase the verifier simply choses a subset of those labels to be opened. Some simpler solutions to construct a PoSpace that come to mind do not work. A tempting "solution" is to let $\mathcal{P}$ store the function table $(1, f(1)), \ldots, (N, f(N))$ of a random-looking function $f(\cdot)$ sorted by the outputs, and a challenge would then ask to invert the function on value $f(x)$ for some random $x \in \{1, \ldots, N\}$, which an honest prover can do in time $\log(N)$ using binary search in the sorted table; Unfortunately, this doesn't work due to time/memory trade-offs [Hel80], which allow a cheating prover to only store roughly $N^{2/3}$ input/output tuples, while being able to invert the function in time $N^{2/3}$ (see [DFKP15, Appendix A] for details). Another simple idea would be to let $\mathcal{V}$ send N (pseudo)random bits to $\mathcal{P}$ during initialization, and simply query $\mathcal{P}$ for some of these bits at random positions during execution. Unfortunately, this requires N bits of communication, whereas a PoSpace requires that the verifier's efficiency only depends on some security parameter, but must be basically independent of N, which is crucial for all applications of PoSpace discussed in [DFKP15] and also for this paper.

Formally, a proof of space is defined by four algorithms $\mathsf{PoS} = \{\mathsf{Init}, \mathsf{Challenge}, \mathsf{Answer}, \mathsf{Verify}\}$, and is executed between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. The protocol runs in two phases. We first describe the *initialization phase*:

1. $\mathcal{P}(N)$ and $\mathcal{V}(N)$ have a common input $N \in \mathbb{N}$ which denotes the amount of space $\mathcal{P}$ should dedicate.

2. $\mathcal{V}$ samples some unique nonce μ and sends it to $\mathcal{P}$.[3]

3. $\mathcal{P}$ computes and stores $(\gamma, S_\gamma) := \mathsf{Init}(\mu, N)$ and sends the commitment γ to $\mathcal{V}$.[4]

[2] An alternative and much simpler construction in [DFKP15] only implies that the cheating prover runs in time $\Omega(N/\log(N))$, but here one provably also requires $\Omega(N/\log(N))$ space.

[3] The nonce ensures that the same space cannot be used for two different proofs (this will be discussed more later).

[4] In the PoSpace schemes from [DFKP15] the initialization phase actually has two rounds. In the first one the verifier sends the commitment γ to the prover, and then, in a second phase, the prover asks the verifier to open the commitment on some random positions (concretely, the labels corresponding to some random nodes in the hard-to-pebble graph, together with the labels of their children). This second phase is used to ensure that the prover committed to correctly computed labels for most of the vertices. For the application to Spacecoin, we can move this check to the execution phase. In fact, it is sufficient when a cheating prover gets caught only with some constant probability, which informally is the reason why we can set the parameter λ below to be a constant.

This concludes the initialization phase, after which $\mathcal{P}$ stores the data S_γ of size N, whereas $\mathcal{V}$ only stores the short commitment γ and the nonce μ. The size N of the committed space is part of the commitment γ and we'll denote it by N_γ.

In the *execution phase*, $\mathcal{P}$ convinces $\mathcal{V}$ that she really stored S_γ.

1. $\mathcal{V}$ samples a challenge $c \leftarrow \mathsf{Challenge}(N)$ and sends it to $\mathcal{P}$.
2. $\mathcal{P}$ computes the answer $a := \mathsf{Answer}(\mu, S_\gamma, c)$ and sends it to $\mathcal{V}$.
3. $\mathcal{V}$ runs the verification procedure $b := \mathsf{Verify}(\mu, \gamma, c, a)$ and accepts if $b = 1$.

Proposition 1.1 ([DFKP15]). *There exists a PoSpace in the random oracle model with the following properties:*

- **Efficiency:** *For a statistical security parameter λ, the verifier runs in time $O(1)$ during initialization and in $O(\lambda \cdot \log(N))$ during execution. The (honest) prover runs in time $O(N \cdot \log\log(N))$ during initialization and in $O(\lambda \cdot \log\log(N))$ during execution.*
- **Security:** *Assume that a (potentially cheating) prover makes $\mathcal{V}$ accept during the execution phase with probability*[5] $2^{-\Theta(\lambda)}$. *Then the prover either stores $\Theta(N)$ bits (i.e., as much as an honest prover) or runs in time $\Theta(N)$ (on average).*

Moreover, the PoSpace satisfies the following two properties (not explicitly mentioned in [DFKP15]), which will be crucial for our application to cryptocurrencies.

- **Public-coin verifier:** *The verifier is public-coin in the execution phase.*
- **Unique accepting answer:** *It is computationally hard to find two accepting transcripts $(c, a), (c, a')$ for the execution phase where $a \neq a'$. (Concretely, finding such a pair implies breaking the collision-resistance of the underlying hash function).*

1.4 Game theory of Spacecoin

The miners in a cryptocurrency are strategic agents who seek to maximize the reward that they get for mining blocks. As such, it is a crucial property of a cryptocurrency that "following the rules" is an equilibrium strategy: in other words, it is important that the protocol rules are designed in such a way that miners never find themselves in a situation where "cheating" and deviating from the rules yields more expected profit than mining honestly.

In contrast to previous work in the cryptocurrency literature, this paper fully specifies an extensive game that corresponds to Spacecoin mining, and prove that to follow the protocol rules is a sequentially rational Nash equilibrium in this extensive game, as long as no single party holds more than half of all space invested in the currency[6].

Prior work related to equilibria in Bitcoin mining has given only an informal treatment of the problem: notably, [KDF13] presents a thorough, but still informal, analysis of equilibrium strategies in Bitcoin, and concludes that honest mining is a Nash equilibrium in Bitcoin (as long as no single party has more than half of the network's computing power). We remark that the Nash equilibrium is widely considered to be unsatisfactory as a solution concept for games that are played over multiple time steps, for reasons which will be detailed in Section 5.1. While

[5]For the application to Spacecoin, it suffices if the error probability is a sufficiently small constant, and thus (as mentioned in the previous footnote) it's sufficient when λ is a constant.

[6]We argue that this is an unlikely scenario, and we remark that the stability of Bitcoin is also known to depend on no single party controlling more than half the network's computing power. In Section 5.2, we analyze the potential risks for Spacecoin in the case that a single party does indeed hold such a large proportion of space.

previous game-theoretic discussion in this literature has been restricted to the Nash equilibrium concept, we prove in Section 5 that the Spacecoin protocol satisfies a stronger solution concept called *sequentially rational Nash equilibrium*, which is considered the standard for extensive games over many time steps.

1.5 Other related work

A concept similar to proofs of space are *proofs of storage* and *proofs of retrievability* (cf. [GJM03, BJO09, ABC$^+$07, JK07, DPML$^+$03] and many more), these are proof systems where a verifier sends a file to a prover, and later the prover can convince the verifier that it really stored or received the file. Proving that one stores a (random) file certainly shows that one dedicates space, but these proof systems are not proofs of space because the verifier has to send the entire file to the verifier, whereas from a PoSpace we require that the verifiers computation (and thus also communication) is at most polylogarithmic in the size of the storage to be dedicated.

Permacoin [MJS$^+$14] is a cryptocurrency similar to Bitcoin, but where the proofs of work are replaced with proofs of retrievability. Here the miners are actually supposed to store useful data, so the currency serves as a data archive, whereas in spacecoin the dedicated storage does not store anything useful. Like in Bitcoin, in Permacoin miners are constantly racing to find a good proof (only the type of proof is different), whereas the main goal of Spacecoin is to avoid such a race: miners only have to execute a proof once every minute, but apart from that can use their resources (except the space dedicated for mining) in a useful way.

Another type of proof systems which is related to PoSpace are *Proof of Secure Erasure* (PoSE). Informally, a PoSE allows a space restricted prover to convince a verifier that he has erased its memory of size S. PoSE were suggested by Perito and Tsudik [PT10], who also proposed a scheme where the verifier sends a random file of size S to the prover, who then answers with a hash of this file. Using hard to pebble graphs, PoSE with small communication complexity have been constructed by [DKW11, KK14, ABFG14]. A PoSpace (to be precise, a PoSpace where the execution phase requires large space, not just time) implies a PoSE (by simply running the initialisation and execution phase sequentially), but PoSE seems not to imply a PoSpace. The only application of PoSE we're aware of is the one put forward in [PT10] (i.e., to prove that one has erased its memory), in particular, PoSE cannot be used for any of the applications of PoSpace put forward in [DFKP15], and also the cryptocurrency proposed in this paper. We refer the reader to [DFKP15] for a more detailed discussion on PoSpace vs. PoSE.

2 Overview of Spacecoin

2.1 High-level protocol description

Spacecoin follows a block-mining paradigm similar to the Bitcoin system, in which miners create blocks of transactions which constitute a public ledger in the form of a *block chain*. Instead of requiring miners to provide a proof of work in order to create a valid transaction block, Spacecoin requires miners to provide a much more efficiently computable proof of space.

Transactions. Transactions are performed basically identically to Bitcoin: Each coin "belongs" to some public key pk. The block chain acts as a ledger that keeps track of which coins belong to which keys (but to prevent grinding, we suggest a new design for the block chain in Section 2.5 where the transactions are decoupled from the proofs). To transfer a coin from pk to pk', a transaction specifying this must be signed by sk (the secret key for pk), and then be added to the block chain. We also allow special transactions to initialize miners, and a special type of transaction which penalizes a miner who extended two different chains using the same proof of space.

Incentivize mining. Like in Bitcoin, there are two ways to incentivize miners to contribute resources (disk space in Spacecoin, computing power in Bitcoin): (1) a reward for adding blocks and (2) transactions fees.

Reward: For adding a block a miner receives some Spacecoins, which do not come from a benefactor, but are created out of thin air when the block is added to the hash chain. How large the reward is has to be specified as part of the protocol, and will typically depend on the index of the block.[7]

Transaction fees: When generating a transaction, one can dedicate (a typically very small) amount of the transferred coins to the miner who adds the block that includes the transaction to the block chain.[8]

Initialize miner. If a miner wants to contribute N bits of space to the mining effort, she samples a public/secret key pair (pk, sk) and runs the PoSpace initialization procedure. (Being in a non-interactive setting, there is no verifier to generate the unique nonce μ, so we simply use pk for this.)

$$(\gamma, S_\gamma) := \mathsf{Init}(pk, N) \ .$$

The miner stores (S_γ, sk) and generates a special transaction which just contains (pk, γ). Once this transaction is in the block chain the miner can start mining as described next.

Mining. Blocks are added to the block chain every fixed time period (say, every minute), and we require that all parties have a clock that is roughly synchronized. To add a block in time period i, the miner retrieves the hash value of the last block in the best chain so far (this chain has $i-1$ blocks), and also a challenge c. How to derive the challenge c is the main difficulty we face. In our simplest solution we assume an unpredictable beacon that generates and broadcasts a freshly sampled random (or at least unpredictable) value from which the challenge is derived every minute (we also propose two solutions without assuming a beacon). The miner then computes the PoSpace answer from c:

$$a := \mathsf{Answer}(pk, S_\gamma, c) \ .$$

For two valid proofs (pk, γ, c, a) and (pk', γ', c', a') we denote with (recall that N_γ is the size of the space committed by γ)

$$(a', N_{\gamma'}) \prec (a, N_\gamma)$$

that the proof a is better than a'. We postpone the discussion on what properties this ordering should satisfy and how it's defined to Section 2.2. For now, we only mention that the ordering should satisfy

$$\Pr[(a', N') \prec (a, N)] = \frac{N}{N + N'}$$

that is, the probability that a wins is proportional to its fraction of the total space. The probability is taken over the choice of a random oracle used to compute the quality of a proof.

If the answer a found by a miner is so good that there is a realistic chance of it being the best answer found by any miner, the miner creates the next block – which contains the PoSpace proof and transactions – and sends it out to the network in the hope that it will end up in the chain.

[7] In Bitcoin, the reward was initially 50 Bitcoins, but it halves roughly every 4 years, and is currently at 25.

[8] Currently, in Bitcoin the transaction fees are tiny compared to the mining reward, but as the latter gets smaller, at some point transaction fees must become the main incentivizing factor.

For the remainder of this one-minute time period the miner need not do anything. As mining only requires a small amount of work (computation, communication and random access to the storage) in every time period, it can be run on any computer that has some free disk space and is connected to the internet without any noticeable slowdown.

In the introduction we discussed the three major problems of PoStake-based cryptocurrencies; below we shortly sketch how these are addressed in our PoSpace-based scheme:

Participation. Since in a scheme based on PoSpace (or PoW) stake holders are not required to participate in mining, the participation problem is not an issue in this work, which makes designing a PoSpace-based scheme significantly easier and more robust than what seems possible with pure PoStake-based approaches.

Grinding. We solve the grinding problem by de-coupling the chain containing the proofs from the transactions, so there is nothing to grind on in the proof chain. Of course, we must somehow tie the transactions to the proofs, which we do by adding a chain containing signatures; we discuss this in Section 2.5. This solution can also be used to solve the grinding problem in PoStake-based schemes.

Extending multiple chains. This problem is addressed in different ways in the schemes we propose. In a nutshell, as trying to extend multiple chains comes almost for free, we cannot prevent miners from doing so, but we can punish them afterwards.

2.2 Quality of a PoSpace Proof

Consider some valid proofs $(pk_1, \gamma_1, c_1, a_1), \ldots, (pk_m, \gamma_m, c_m, a_m)$ for space of size $N_1, \ldots, N_m$. We want to assign a quality to a proof (which will only be a function of a_i and N_i), such that the probability (over the choice of the random oracle $hash$) that the ith proof has the best "quality" corresponds to its fraction of the total space, i.e.

$$\Pr_{hash(.)}[\forall j \neq i \ : \ (a_j, N_j) \prec (a_i, N_i)] = \frac{N_i}{\sum_{j=1}^{m} N_j} \ .$$

We observe that in order to achieve this, it's sufficient to achieve this for any pair of commitments, i.e.,

$$\Pr_{hash(.)}[(a_j, N_j) \prec (a_i, N_i)] = \frac{N_i}{N_i + N_j} \ .$$

If all the N_i were of the same size N, we could simply define

$$(a_j, N) \prec (a_i, N) \iff hash(a_j) \leq hash(a_i)$$

That is, we map every a_i to a random value $hash(a_i)$, and whichever value is largest wins. We want to allow for different N_i values, so miners who want to contribute space N' only need one space commitment, and do not have to split it up in N'/N space commitments of size N, and then run a proof for each chunk separately.

For this, we define a distribution $D_N, N \in \mathbb{N}$ which is defined by sampling N values in $[0, 1]$ at random, and then outputting the largest of them.

$$D_N \sim \max \{r_1, \ldots, r_N \ : \ r_i \leftarrow [0, 1], i = 1, \ldots, N\} \tag{1}$$

With $D_N(\tau)$ we denote a sample of D_N (or rather, a distribution which is very close to it) using randomness τ to sample. We now say that (a_i, N_i) is of higher quality than (a_j, N_j) if

$$(a_j, N_j) \prec (a_i, N_i) \iff D_{N_j}(hash(a_j)) \leq D_{N_i}(hash(a_i)) \ . \tag{2}$$

It remains to show how to efficiently sample from the distribution D_N for a given N. Recall that if F_X denotes the cumulative distribution function (CDF) of some random variable X over $[0, 1]$ and the inverse F_X^{-1} exists, then $F_X^{-1}(U)$ for U uniform over $[0, 1]$ has the same distribution as X. The random variable X sampled according to the distribution D_N has CDF $F_X(z) = z^N$, since this is the probability that all N values r_i considered in (1) end up being below z (and hence also their maximum). Therefore, if we want to sample from the distribution D_N, we can simply sample $F_X^{-1}(U)$ for U uniform over $[0, 1]$, which is $U^{1/N}$. In (2) we want to sample D_{N_i} using randomness $hash(a_i)$ and $hash(\cdot)$ outputs 256-bit bitstrings instead of values in $[0, 1]$, hence we have to normalize first and we compute $D_{N_i}(hash(a_i))$ as

$$D_{N_i}(hash(a_i)) := \left(hash(a)/2^{256}\right)^{1/N} \ .$$

Note that this introduces a tiny imprecision due to the fact that $hash(a)/2^{256}$ is uniform over a discrete set instead of the continuous interval $[0, 1]$, but this can be safely disregarded.

2.3 Where the challenge comes from

The main difficulty we face when designing a PoSpace-based scheme is the generation of the PoSpace challenge c.

2.3.1 Assuming an unpredictable beacon

Our most basic solution simply assumes an unpredictable beacon which broadcasts a value every minute. The beacon being unpredictable means that no party at time t has non-negligible probability of guessing the beacon to be broadcast at time $t + 1$. Such a beacon could e.g. be instantiated as the hash of the current time and the NASDAQ chart. Given such a beacon, the challenge c for mining block i can simply be derived as a hash of the beacon value (hashing turns an unpredictable value into a random looking one).

Assuming an unpredictable beacon is a fairly strong assumption, but it doesn't trivialise the problem. It does mostly solve the grinding problem, but doesn't help with the "extending multiple chains" problem at all: As the challenge does not depend on the hash chain, a miner who finds a very good proof can try to add it to many different chains – not just the best one – to reduce the probability of his block being "orphaned", i.e., a different chain without his block ultimately becoming the best one. Such behavior is rational, but undesirable in that it prevents consensus on a single chain. To de-incentivize it, we define a special transaction which allows to "steal" the reward a miner gets for adding a block if the miner used the same proof for extending two different chains.

To add a block, a miner must provide a signature on the previous signature block in that chain. Thus, if a miner with key (pk, sk) tries to add a block to two chains, he must sign two different messages for the same time slot i. We allow for special "punishment transactions" which are basically of the form (pk', σ'_j, α) and have the following semantics: Let pk be the key of the miner that added the jth block in the current chain. If this block does not contain a signature for σ'_j and α is a signature for $\sigma'_j = (j, \ldots)$ under pk, then half of the reward that pk gets for adding this block is transferred to pk', and the other half is destroyed. Of course this reward will typically be claimed by the miner who adds a subsequent block $i > j$, as there is no point in adding such a transaction for another miner. To prevent that a miner immediately transfers his reward to another key – and thus avoid punishment – we specify that the reward for adding a block cannot be transferred until several (say 1000) blocks later (except via a punishment transaction).

This punishment strategy strongly discourages a miner from trying to extend more than one chain, as doing so will most likely lead to not getting any reward at all, even when having the best proof for a given time slot.

2.3.2 Challenge from the past

We now describe our scheme without an unpredictable beacon. This scheme is identical to the one outlined above, except that the challenge c does not come from the beacon, but is derived from the block chain itself.

The simplest solution (which does not quite work) is to let the challenge for block i be the hash of block $i-1$. Our block chain consists of a proof chain, and a separate signature chain that binds the transactions to the proof chain. If we only hash the block from the proof chain, no problem arises with miners trying to grind through several possible challenges, but another problem remains: If there are many different chains, the miner gets different challenges for different chains. A rational miner would thus compute the answers for many different chains, and if one of them is very good, try to add a block to the corresponding chain, even if this chain is not the best chain seen so far. If all miners behave rationally, this will at the very least considerably slow down consensus, as bad chains get extended with blocks of the same quality as the currently best chain; thus we expect to see a race between many different chains without the lower-quality chains falling behind rapidly. A solution to this kind of problems that is used in[9] Slasher 1.0 is to penalize miners that extend chains that do not end up in the final chain, but this seems not very robust.

The solution we propose is to compute the challenge as a hash of block $i-\mathsf{dist}$ (rather than $i-1$), for some appropriately chosen dist ($\mathsf{dist}=120$ seems like a good value for a 1-minute time slot). Now, a miner only obtains different challenges for the ith block of two different chains if those chains have forked at least d blocks ago. With $\mathsf{dist}=120$, it is extremely unlikely that two chains will survive in parallel for dist blocks. Recall that we penalize a miner who adds a block (using the same challenge) to two different chains, thus as long as we have two or more chains that all forked more than dist blocks ago, there is a strong incentive for miners to add their blocks to the best chain; which is why we expect the other chains to fall behind very fast.

The reason we cannot set dist to be arbitrary large is that a miner at time t knows its challenges for all the blocks $t+1,\ldots,t+\mathsf{dist}$. Thus, she can compute all the dist answers, and need not access her space for the next dist minutes. If dist was very large, a miner could use the same space for several space commitments, as there would be enough time to re-instantiate the space several times in the dist-blocks window. To avoid this, dist should be set so that initializing the space takes roughly dist minutes. In the next section we will discuss another reason to choose a small dist in order to prevent usage of cheap storage devices like tapes for mining.

2.3.3 Request challenge from other miners

Finally, inspired by existing PoStake-based schemes, we sketch a scheme where a miner must request the challenge from other miners. A miner who tries to extend two or more chains must publish at least two requests for challenges, and any such pair of requests for challenges in the same time slot (requests contain a signature of the requesting party) can be used to create a punishing transaction. Here we must require that miners, when putting their space commitment into the chain, also put some deposit. This deposit can later be withdrawn, but the space commitment can only be used for mining as long as the deposit is there. If a miner

[9] `https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/`

posts two requests, this pair of requests can be used to get half of his deposit, whereas the other half is destroyed (this ensures that the miner gets punished even when posting the punishment transaction himself). Note that in the previous scheme we punished a miner who added two blocks using the same challenge, here we punish the mere attempt to request two different challenges for different chains.

More concretely, assume a miner pk wants to extend some chain with last block ϕ_{i-1}. He first computes an index $t = hash(pk, \phi_{i-1}) \in \{1, \dots, 10\,000\}$, which means we must ask the user who mined block $i-t$ for a challenge. The miner publishes this request (on some kind of bulletin board), hoping that the user who mined block $i-t$ is still online and will provide the challenge. To incentivize providing challenges, the user providing the challenge will get a fraction of the reward, should the requesting miner win so his block is added. We limit the domain of possible "challenge-providing miners" to those that added one of the last 10 000 blocks, so we can be reasonably sure a significant fraction of them is still active, while 10 000 is large enough so that the challenge requests for each individual miner are rather small. If a miner sees a request addressing him, he can compute the challenge c using a verifiable pseudorandom function (VRF) and publish it. We require a VRF here to avoid "grinding" through many possible challenges in the case the requesting and the providing miner collude, or the requesting miner is lucky and "asks" himself for a challenge. Even if this happens, by using a VRF the only thing gained by the miner is that she she gets one challenge extra for free, thus doubling her chances for mining the next block.

Realizing such a solution seems significantly more complicated and delicate than the previous one, and thus we do not further discuss this approach here.

2.4 Minor issues

We now discuss some minor issues that arise and how to solve them.

DoS. A party who wants to mine must have its space commitment (pk, γ) added to the hash chain. A malicious party could flood the network with countless requests of fake commitments to be added to the chain. One simple way to counter this problem is to request some small transaction fee, as is done for normal transactions. The drawback is that now miners must already possess some coins to even start mining. Another solution is to require a PoSpace proof for the commitment (pk, γ) to be added, i.e., $a := \mathsf{Answer}(pk, S_\gamma, c)$, where the challenge c can for example be computed via the Fiat-Shamir transformation as $c = hash(pk, \gamma)$. This proof is only provided to convince miners that some work went into generating the commitment, but the proof will not be added to the chain.

Reusing space. We require that a public key pk is only used once for a space commitment: a commitment (pk, γ) will not be added to the chain if some commitment (pk, γ') is already in the chain. As the PoSpace scheme from [DFKP15] uses the unique nonce (here pk) as a prefix to every random oracle query, the random oracle used in the PoSpace scheme for a given commitment (pk, γ) is independent from the random oracles used for any other commitments. This implies that space cannot be re-used for different commitments.

Why add commitments to the chain. It is not obvious why we require a miner to first add a space commitment (pk, γ) to the hash chain before it can start mining, instead of simply having miners keep this commitment locally and only send it out once they found a good PoSpace proof. The reason is that the PoSpace proofs from [DFKP15] have the property that one can take a correctly constructed commitment (pk, γ_0), and by making minor changes turn it into many other commitments $(pk, \gamma_1), (pk, \gamma_2), \dots$ that can reuse almost all space, while it is still possible to answer almost all challenges correctly.[10] Thus, if there were no requirement to have a published commitment in the block-chain with a unique public key, a cheating miner could re-use the same space for many different commitments.

Tapes. The designer(s) of Bitcoin probably were anticipating that most of the mining will be done by users on their personal computers. What happened instead is that today almost all mining is done by clusters of application-specific integrated circuits (ASICs), which can do the computation for a tiny fraction of the hardware and energy cost of a general-purpose processor. We anticipate that a PoSpace-based currency would mostly use the idle disk space on personal computers for mining. Although hard disks are rather expensive compared to other storage devices – most notably, tapes – devices like tapes are not really adequate for mining, as we also require frequent random accesses to answer the PoSpace challenges, which is more difficult on tapes which are made for long term storage. This is clearly true for the first (beacon) and third (challenge from other miners) scheme outlined in the previous section, as there the proof must be computed within a single one-minute time slot. Things are less clear for our second (challenge from the past) scheme, as here the miner only needs to compute the proof in dist minutes, recall that dist is the parameter that specifies that the challenge for block i is computed as a hash of block $i - \mathsf{dist}$.

It has to be further investigated to what extent this actually is an issue. But given that the cost advantages of tapes is being debated even for long term storage,[11] it seems unlikely that in our context, where frequent random access is necessary, tapes will bring any advantage.

2.5 The block-chain format

A block chain is a sequence of blocks $\beta_0, \beta_1, \ldots$ Each block $\beta_i = (\phi_i, \sigma_i, \tau_i)$ is created by a *miner* and consists of three main parts, which we call "sub-blocks". Each sub-block starts with the index i that specifies its position in the block chain. Below, we outline the remaining components of the three sub-blocks of a block $\beta_i, i > 0$. The genesis block β_0 necessarily has a somewhat different format as it cannot depend on previous blocks:

- The HASH sub-block ϕ_i contains:
  - A 256-bit hash $hash(\phi_{i-1})$ of the HASH sub-block from the previous block in the chain.
  - A "space proof" containing the miners identity pk (more details on this are given below).
- The TRANSACTION sub-block τ_i contains:
  - A list of *transactions* (defined in more detail below).
- The SIGNATURE sub-block σ_i contains:
  - The miner's signature $\mathsf{Sign}(sk, \tau_i)$ on the TRANSACTION sub-block τ_i associated with this block.
  - The miner's signature $\mathsf{Sign}(sk, \sigma_{i-1})$ on the SIGNATURE sub-block σ_{i-1} associated with the previous block in the chain.

[10] γ_0 is a Merkle-hash of all the labels in a hard to pebble graph. We can change γ_0 to another value by simply changing a single label, which will not be noticed in the execution phase unless this particular label with its children is requested.

[11] http://www.computerworld.com/article/2475237/data-center/tape-versus-disk--the-backup-war-exposed.html

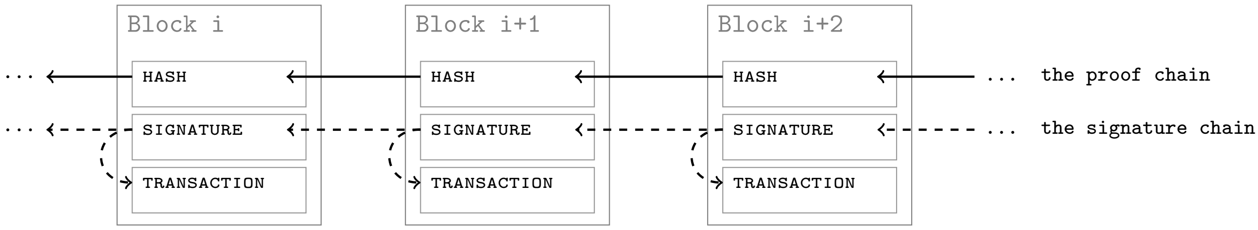

Figure 1: Our block chain consists of a proof chain that does not allow for grinding, and a signature chain that binds the transactions to the proof chain.

The links between consecutive blocks in the block chain are illustrated in Figure 1. We will also refer to the hash sub-blocks as the *proof chain*, and the signature sub-blocks with the transactions as the *signature chain*. Solid arrows represent hashes, and dotted arrows represent signatures. Notice that while the signature and transaction sub-blocks are all linked together, the hash sub-blocks are only linked to each other and not to any signature or transaction sub-blocks.

By decoupling proofs from transactions we achieve security against *grinding*: For any space commitment (pk, γ), the miner pk cannot generate two (or more) correctly formatted hash-blocks to be added to the proof chain. For this it is also crucial that the underlying PoSpace has the "unique accepting answer" property discussed in the introduction.

The signature chain binds the transactions to the proof chain. If an honest miner (honest to be defined below) adds the ith block, the transactions corresponding to this proof chain up to block i cannot be changed any more, even if an adversary controls all secret keys from miners that added the first $i-1$ blocks. Here the miner being honest means that she only signs a single block of transactions using the secret-key sk corresponding to her identity pk, and moreover keeps sk secret. To see this, note that if we want to change the transactions in block $j < i$ while keeping the current proof chain up to block i, then the signatures for blocks $j, \dots, i$ must be re-computed, which requires sk.

3 Transactions

Spacecoin is based on a secure[12] signature scheme

$$\Sigma = (\mathsf{SigParamGen}, \mathsf{SigKeyGen}, \mathsf{Sign}, \mathsf{SigVerify})$$

and a PoSpace protocol

$$\Pi = (\mathsf{Init}, \mathsf{Challenge}, \mathsf{Answer}, \mathsf{Verify}) \ .$$

In the following we specify the three types of transactions (for payments, space commitments and punishments) that we allow in Spacecoin.

Payments. Coins are held and transfered by parties identified by a verification key in the support of $\mathsf{SigKeyGen}$.[13] More specifically, a *transaction* transfers coins from m benefactors to

[12] Existentially unforgeable under chosen message attacks.

[13] In Bitcoin, the specification of payments is more general: instead of specifying beneficiaries via their verification keys, recipients are specified by writing a script *scr* in a special (non-Turing-complete) scripting language called Bitcoin Script. The output coins of a transaction can then be redeemed by any party which can produce inputs which "satisfies" the script *scr*. In practice, Spacecoin can be straightforwardly modified to accommodate such scripting; but in this work, for clarity of exposition, we assume that each payment recipient is specified by a verification key.

n beneficiaries and has the form

$$ctx = (\mathsf{payment}, txId, \vec{in}, \vec{out}) \ .$$

$txId$**:** Here and below, $txId$ always denotes a unique arbitrary *transaction identifier*. That is, no two transactions in a blockchain can share their identifier.

$\vec{out}$**:** A list of beneficiaries and the amount they receive. Specifically $\vec{out} = (out_1, \ldots, out_m)$ where each $out_i = (pk_i, v_i)$ and:

- pk_i is in the support of $\mathsf{SigKeyGen}$ and specifies a *beneficiary*, and
- v_i is the number of coins that pk_i is to be paid.

$\vec{in}$**:** A list of inputs coins to the transaction. Specifically $\vec{in} = (in_1, \ldots, in_n)$, a list of n *benefactors*, each comprised of a triple: $in_j = (txId_j, k_j, sig_j)$, where

- $txId_j$ is the identifier of a past transaction,
- k_j is an index that specifies a particular beneficiary pk_{k_j} of the transaction $txId_j$,[14]
- sig_j is a signature of $(txId, txId_j, k_j, \vec{out})$, which verifies under key pk_{k_j} proving ownership of the the k_jth beneficiary of transaction $txId_j$ and binding the coin to the beneficiaries.[15]

In order for a transaction to be considered valid, the following conditions must be satisfied:

1. No benefactor is referenced by more than one transaction in the block chain (to prevent double-spending).
2. The sum of the input values to the transaction (i.e. the sum of the amounts provided by each benefactor) is greater than or equal to the sum of amounts paid to beneficiaries.

Note that some of the beneficiary identities may belong to the creator of the transaction, who may thus transfer money back to himself as "change": this may be desirable if the sum of the input values exceeds the sum of the amounts of payments he wants to make to other parties. Apart from normal payments, we allow for two further types of transactions to initialize and punish miners.

Space commitment. A space commitment transaction is of the form

$$ctx = (\mathsf{commit}, txId, (pk, \gamma)) \ ,$$

where – assuming the transaction was correctly generated – pk is a public key and γ is computed as $(\gamma, S_\gamma) := \mathsf{Init}(pk, N)$, i.e., it is a space commitment to some space of size N (we assume that N is specified by γ, and thus do not explicitly add it).

Punishment. A punishment transaction is of the form

$$ctx = (\mathsf{punish}, txId, pk', (m, \alpha)) \ ,$$

where m starts with an index $j-1$ for some $j \in \mathbb{N}$. The transaction has the following semantics: Let σ_j be the jth block of the signature chain in the current chain, and pk_j the public key of

[14]That is the k_jth beneficiary of transaction $txId_j$ is the jth benefactor of transaction $txId$.

[15]$txId$ is signed in order to avoid transaction malleability https://en.bitcoin.it/wiki/Transaction_Malleability

the miner who added the jth block. If $m \neq \sigma_{j-1}$ and $\mathsf{SigVerify}(pk_j, m, \alpha) = 1$, i.e., pk_j was used to sign another message than σ_{j-1} with index $j-1$, then this transaction transfers half the reward and transactions fees that went to pk_j for adding the ith block to pk', the other half is destroyed.

We additionally require that $i - j < 1000$, thus, the punishment must happen within 1000 blocks (and as we disallow withdrawing a mining rewards for 1000 blocks, this reward cannot have already been spent).

4 Instantiation

In this section we describe the concrete steps required for setting up, mining and paying in Spacecoin. We give the instantiation for the second scheme (challenge from the past), outlined in Section 2.3. The first (random beacon) scheme is almost identical, except that the challenge c is derived from the random beacon (and not by hashing a block from the chain).

Setup. At setup we have to fix the security parameter κ to be used for the signature and PoSpace scheme. Moreover, we must specify parameters and functions:

time: A variable $\mathsf{time} \in \mathbb{N}$ which specifies the length of a timeslot in minutes. It should be sufficiently larger than the time a message needs to propagate through the network, but otherwise as small as possible. $\mathsf{time} = 1$ seems like a reasonable choice here.

dist: A variable $\mathsf{dist} \in \mathbb{N}$ which specifies that the challenge for block i is a function of block $i - \mathsf{dist}$. A reasonable value is $\mathsf{dist} = 120$.

Reward: A function $\mathsf{Reward}\colon \mathbb{N} \to \mathbb{N}$ where $\mathsf{Reward}(i)$ specifies the amount of Spacecoins a miner gets for mining the ith block (we leave this unspecified).

Quality: The function Quality takes as input a space commitment (pk, γ) for space of size N and a challenge/answer pair (c, a). If $\mathsf{Verify}(pk, \gamma, c, a) \neq 1$ (i.e., it is not a valid PoSpace proof transcript), the function outputs $-\infty$. Otherwise the output is (with $D_N(.)$ as defined in Section 2.2)

$$\mathsf{Quality}(pk, \gamma, c, a) = D_N(hash(a)) \; .$$

In order to decide which of two given proof chains is the "better" one, we also need define the quality of a proof chain $\phi_0, \ldots, \phi_i$, which we'll denote with $\mathsf{QualityPC}(\phi_0, \ldots, \phi_i)$. Each hash block ϕ_j contains a proof $(pk_j, \gamma_j, c_j, a_j)$, and we let $v_j = D_{N_j}(a_j)$ denote the quality of the jth proof in the chain. For any quality $v \in [0, 1]$, we denote with

$$N(v) = \min\{N \in \mathbb{N} \;:\; \Pr[v \prec w \mid w \leftarrow D_N] \geq 1/2\}$$

the space required to get a better proof than v on a random challenge with probability 1/2. Note that $N(v_j)$ will usually be around the total storage of all miners that were active when the jth block was mined. With this definition, a natural measure for the quality of the chain would be simply the sum[16] $\sum_{j=1}^{i} N(v_j)$. The problem with this measure is that if some miner finds an extremely good proof, say $N(v)$ is 1000 times larger than the total storage (this will happen roughly every 1000 blocks), then the miner could withhold his proof, and 1000 blocks later generate a fork using this proof followed by 999 arbitrarily bad proofs for the remaining blocks. To avoid such deep forks, we cap proofs that are too good by saying that v_j cannot contribute more to the sum than, say 10 times the median of the last 101 blocks (the median

[16] We start summing with $j = 1$ not $j = 0$ as the genesis block (still to be defined) will not contain a proof.

gives a good approximation of the total space that is dedicated towards mining). Formally, let $\hat{N}(v_j)$ be recursively defined as

$$\hat{N}(v_j) = \max\{N(v_j)\ ,\ 10 \cdot \mathsf{median}(N(v_{j-101}), \ldots, N(v_{j-1})\}$$

Another reason why defining the quality simply as $\sum_{j=1}^{i} N(v_j)$ is problematic, is that the total contributed space can increase drastically over time. In this case, in order to come up with a chain whose quality is better than the quality of the real chain it's sufficient to dedicate much less than the total space that is *currently* devoted towards mining. For this reason, we only take the last 1000 blocks into account when computing the quality:

$$\mathsf{QualityPC}(\phi_0, \ldots, \phi_i) = \sum_{j=\max\{1,i-1000\}}^{i} \hat{N}(v_j)$$

Finally, a genesis block $\beta_0 = (\phi_0, \sigma_0, \tau_0)$ is generated and published; it has a format different from other blocks. The transactions block contains only one space commitment $\tau_0 = (\mathsf{commit}, txId, (pk_0, \gamma_0))$, the hash block ϕ_0 contains only some random string,[17] and the signature block σ_0 contains the signature $\mathsf{Sign}(sk_0, \tau_0)$ of the transactions block (but not of the previous signature block, as there is none).

Initialize Mining. A party that wants to dedicate N bits of storage for mining first generates an identity and a space commitment

$$(pk, sk) \leftarrow \mathsf{SigKeyGen}\ , \qquad (\gamma, S_\gamma) := \mathsf{Init}(pk, N)\ .$$

It stores S_γ (of size N) and sk locally. The miner then generates and publishes a transaction $ctx = (\mathsf{commit}, txId, (pk, \gamma))$. Once ctx has been added as a transaction to the hash chain, the miner can start mining as described next.

Mining. As we enter time slot i, the miner retrieves[18] the so-far-best block chain $\beta_0, \ldots, \beta_{i-1}$ (that is, the chain maximizing $\mathsf{QualityPC}(\phi_0, \ldots, \phi_{i-1})$. We assume that the miner "honestly" stores space S_γ and the corresponding commitment (pk, γ) has been added to some transcription block $\tau_j, j \leq i-1$ in this chain. Next, the miner computes its challenge by hashing the hash block that is dist blocks in the past

$$c := hash(pk, \phi_{i-\mathsf{dist}})\ .$$

The miner computes the PoSpace answer

$$a := \mathsf{Answer}(pk, S_\gamma, c)\ .$$

If $q := \mathsf{Quality}(pk, \gamma, c, a)$ is very high, so it has a realistic chance to end up as the best answer of the entire network, the miner generates a hash block $\phi_i = (i, hash(\phi_{i-1}), p_i)$, where p_i is the space proof[19] (pk, γ, c, a, j, q). Then the miner retrieves transactions (typically, giving priority to the ones paying the highest fees), checks their correctness, and adds the valid ones to a transaction block τ_i. It then computes the signature block $\sigma_i = (\mathsf{Sign}(sk, \sigma_{i-1}), \mathsf{Sign}(sk, \tau_i))$

[17] Or better, some kind of timestamp like a sentence from a newspaper of the day, as is done in Bitcoin, to show that the genesis block was not generated before some date.

[18] With "publishes" and "retrieves" we mean that a party sends or downloads something from the network. Typically, there would be some servers that organize the data, i.e., keep track of the best chains and collect transactions, so a miner would only interact with one or a few such servers it trusts.

[19] The index j of the space commitment and the quality q are redundant, but they simplify verifying the proof.

and publishes block $\beta_i = (\phi_i, \sigma_i, \tau_i)$, hoping that it will end up in the block chain, earning the miner $\mathsf{Reward}(i)$ Spacecoins, plus the transactions fees of the transactions in τ_i.

Transaction. Any party can generate a transaction and publish it. If it is correctly generated, it should ultimately end up in the block chain. We have already described the format and semantics of the three types of transactions in Section 3.

5 Game theory of Spacecoin

The miners in a cryptocurrency are strategic agents who seek to maximize the reward that they get for mining blocks. As such, it is a crucial property of a cryptocurrency that "following the rules" is an equilibrium strategy: in other words, it is important that the protocol rules are designed in such a way that miners never find themselves in a situation where "cheating" and deviating from the rules yields more expected profit than mining honestly.

Intuitively, Spacecoin mining is modeled by the following n-player strategic game. Gameplay occurs over a series of discrete time steps, each of which corresponds to a block being added to the block chain. At each time step, each player (miner) must choose a strategy, specified by:

- which blocks to extend (if any), which transactions to include in the new blocks, and
- which extended blocks to publish (if any).

We present the details of our game-theoretic analysis in the unpredictable-beacon model, and remark that the analysis can be extended to cover the other models too.

5.1 Game-theoretic preliminaries

The standard game-theoretic notion for a strategic game which occurs over multiple time steps (rather than in "one shot") is called an *extensive game*. In order to accurately model the probabilistic aspects of the Spacecoin protocol (e.g. the unpredictable beacon), we consider *extensive games with chance moves*: this is the standard game-theoretic notion to capture extensive games which involve exogenous uncertainty. The uncertainty is modeled by an additional player called Chance (a.k.a. Nature) which behaves according to a known probability distribution.

In the Spacecoin setting, every player (including Chance) makes an action at every time step. A player's action consists of choosing whether and how to extend the block chain, and the action of Chance determines the value of the unpredictable beacon for the next time step. We therefore omit the usual notation that specifies which players move at each time step, for clarity's sake.

An extensive game is commonly visualized as a *game tree*, with the root node representing the start of the game. Each node represents a *state* of the game, and the outward edges from any given node represent the actions that players can take at that node. Leaf nodes represent *terminal* states: once a leaf is reached, the game is over. In accordance with the literature, we refer to paths in the game tree (starting at the root) as *histories*; and histories which end at a leaf node are called *terminal histories*.

Definition 5.1. *[Extensive game] An* extensive game $\Gamma = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$ *is defined by:*

- $[N]$*, a finite set of players.*
- H*, the set of all possible histories, which must satisfy the following two properties:*
    - *the empty sequence* $()$ *is in* H*, and*

– *if* $(a_1, \ldots, a_K) \in H$ *then for all* $L \leq K$*, it holds that* $(a_1, \ldots, a_L) \in H$*.*

We write $Z \subseteq H$ *to denote the subset consisting of all* terminal histories*. For any history* h*,*

$$A(h) = \{a : (h, a) \in H\} = \times_{i\in[N]} A_i(h)$$

denotes the set of action profiles that can occur at that history, and $A_i(h)$ *denotes the set of actions that are available to player* i *at history* h*.*

- $f(\cdot, h)$ *is a probability measure on* $A_{\mathcal{C}}(h)$*, where* $h \in H$ *and* $\mathcal{C}$ *denotes the Chance player.*
- $\vec{\mathcal{I}} = (\mathcal{I}_1, \ldots, \mathcal{I}_N)$*, where each* $\mathcal{I}_i$ *is a partition of* H *into disjoint* information sets*, such that* $A_i(h) = A_i(h')$ *whenever* h *and* h' *are in the same information set* $I \in \mathcal{I}_i$*. Let* $A_i(I)$ *denote the set of actions that are available to player* i *at any history in information set* I*.*
- $\vec{u} = (u_1, \ldots, u_N)$*, where each* $u_i : Z \to \mathbb{R}$ *is the utility function of player* i*.*

5.1.1 Imperfect information and information sets

An extensive game is said to have *perfect information* if at any point during game-play, every player is perfectly informed of all actions taken so far by every other player. In the context of Spacecoin, players are only aware of each others' *announced* actions: for example, if Alice tries extending several blocks and then only announces one of them, then Bob does not know about the other blocks that Alice tried to extend. Thus, Spacecoin is a game of *imperfect information.*

The information that players do not know about other players' actions is modeled by the partitions $\vec{\mathcal{I}} = (\mathcal{I}_1, \ldots, \mathcal{I}_N)$ in Definition 5.1. Each $\mathcal{I}_i$ is a partition of H into disjoint *information sets*, and for each $i \in [N]$ and any pair of histories $h, h' \in I$ in a particular information set $I \in \mathcal{I}_i$, player i cannot tell the difference between game-play at h and at h'.

Example 5.2. ["Match my number" game]Consider a simple two-player game in two rounds: in the first round, player 1 chooses a number $a \in \{0, 1, 2\}$. In the second round, player 2 chooses a number $b \in \{0, 1, 2\}$. Player 2 wins if $b = a$, and player 1 wins otherwise. Clearly, player 2 can always win if he knows a.

However, we consider a game of *imperfect information* where player 2 must choose b without knowing a: in particular, suppose player 2 only learns whether $a = 0$. Then, the histories $(a = 1)$ and $(a = 2)$ are in the same information set in the partition $\mathcal{I}_2$. Figure 3 shows the game tree, with player 2's information sets as dotted red boxes: within each dotted box, player 2 cannot tell which history he is at.

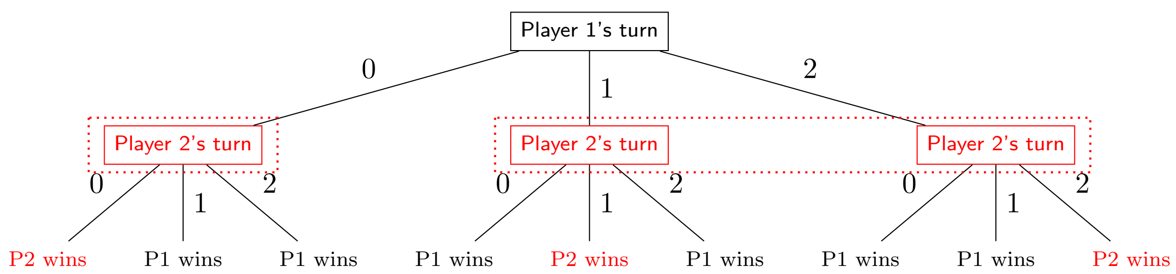

Figure 2: Game tree for the "Match my number" game

5.1.2 Strategies

A *strategy* of a player in an extensive game is defined by specifying how the player decides his next move at any given history. In games of imperfect information, the player may not know which history he is at, so we instead specify how the player decides his next move at any information set.

Definition 5.3. *[Strategy profile]A* strategy profile $\vec{\alpha} = (\alpha_1, \dots, \alpha_N)$ *of an extensive game* $\Gamma = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$ *specifies for each player* $i \in [N]$ *and each information set* $I \in \mathcal{I}_i$ *a probability distribution* $\alpha_i(I)$ *over the action set* $A_i(I)$*. We say that* α_i *is the* strategy *of player* i*.*

Let $I(h)$ denote the information set in which history h lies. The probability that a history h occurs under strategy profile α is denoted by $\Pr_{\vec{\alpha}}[h]$, and the probability that a history h' occurs given that h occurred is denoted by $\Pr_{\vec{\alpha}}[h'|h]$.

Recall that the utility functions $u_1, \dots, u_N$ were originally defined on inputs in Z, the set of terminal histories. For each $i \in [N]$, we now define $u_i(\vec{\alpha})$ to be the expected utility of player i given the strategy profile $\vec{\alpha}$. That is,

$$u_i(\vec{\alpha}) = \sum_{h \in Z} u_i(h) \cdot \Pr{}_{\vec{\alpha}}[h].$$

Moreover, we define $u_i(\vec{\alpha}|h)$ to be the expected utility of player i given $\vec{\alpha}$ and given that history h has already occurred. That is,

$$u_i(\vec{\alpha}|h) = \sum_{h' \in Z} u_i(h') \cdot \Pr{}_{\vec{\alpha}}[h'|h].$$

5.1.3 Equilibrium notions

The most widely known equilibrium concept for a strategic game is the Nash equilibrium [Nas50], given in Definition 5.4. Intuitively, in a Nash equilibrium, each player's strategy is a *best response* to the strategies of the other players.

For a strategy profile $\vec{\alpha}$, we write $\vec{\alpha}_{-i}$ to denote $(\alpha_j)_{j \in N, j \neq i}$, that is, the profile of strategies of all players other than i; and we use $(\alpha'_i, \vec{\alpha}_{-i})$ to denote the action profile where player i's strategy is α'_i and all other players' actions are as in $\vec{\alpha}$.

Definition 5.4 (Nash equilibrium of an extensive game)**.** *Let* $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ *be an extensive game. A strategy profile* $\vec{\alpha}$ *is a* Nash equilibrium of Γ *if for every player* $i \in [N]$ *and every strategy* α'_i *of player* i*,*

$$u_i(\vec{\alpha}) \geq u_i(\alpha'_i, \vec{\alpha}_{-i}).$$

The Nash equilibrium concept was originally formulated for *one-shot* games, and it is known to have some shortcomings in the setting of extensive games. Informally, the Nash equilibrium does not account for the possibility of players changing their strategy partway through the game: in particular, there exist Nash equilibria that are not "stable" in the sense that given the ability to change strategies during the game, no rational player would stick with his equilibrium strategy all the way to the end of the game.

Example 5.5. ["Unstable" game]Consider a simple two-player game in two rounds: in the first round, player 1 chooses either strategy A or B. In the second round, player 2 chooses either strategy C or D. The game tree is given below, where the notation (x, y) at the leaves denotes that player 1 gets payoff x and player 2 gets payoff y if that leaf is reached.

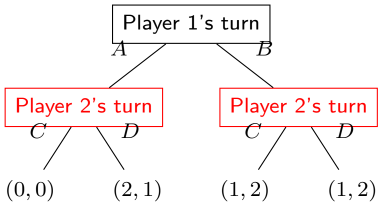

Figure 3: Game tree for the "Unstable" game.

It is a Nash equilibrium of this game for player 1 to choose B, and player 2 to always choose C[20] However, the strategy profile (B, C) seems "unstable"[21], in the following sense: player 1 does not want to switch from strategy B to A because of the "threat" that player 2 will then choose C. However, in the situation where player 1 has actually chosen strategy A, it is clearly better for player 2 to play D rather than follow through with the threatened strategy C. That is, the threat does not seem credible.

To address these shortcomings of the Nash equilibrium concept for extensive games, an alternative (stronger) notion has been proposed: the *sequentially rational* Nash equilibrium. This stronger concept ensures that players are making the best decision possible *at any point* during game-play. In a game with imperfect information, it is necessary to consider not only the strategy profile, but the players' *beliefs* at any point in time about how game-play arrived at the current information set. A strategy profile which takes into account players' beliefs is called an *assessment*.

Definition 5.6. *[Assessment] An* assessment *in an extensive game is a pair* $(\vec{\alpha}, \vec{\mu})$ *where* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_N)$ *is a strategy profile and* $\vec{\mu} = (\mu_1, \ldots, \mu_N)$ *is a* belief system*, in which each* μ_i *is a function that assigns to every information set in* $\mathcal{I}_i$ *a probability measure on the histories in the information set.*

In Definition 5.6, $\mu_i(I)(h)$ represents the probability that player i assigns to the history $h \in I$ having occurred, conditioned on the information set $I \in \mathcal{I}_i$ having been reached. For each $i \in [N]$, we now define $u_i((\vec{\alpha}, \vec{\mu})|I)$ to be the expected utility of player i at the information set $I \in \mathcal{I}_i$, given the strategy profile $\vec{\alpha}$ and belief system $\vec{\mu}$. That is,

$$u_i((\vec{\alpha}, \vec{\mu})|I) = \sum_{h \in I} u_i(\vec{\alpha}|h) \cdot \mu(I)(h).$$

We write $u_i((\vec{\alpha}, \vec{\mu}))$ to denote $u_i((\vec{\alpha}, \vec{\mu})|\{()\})$, that is, the expected utility for player i at the beginning of the game.

An assessment (α, μ) is said to be *sequentially rational* if for every $i \in [N]$ and every information set $I \in \mathcal{I}_i$, the strategy of player i is a best response to the other players' strategies, given i's beliefs at I. A formal definition follows.

Definition 5.7 (Sequentially rational assessment)**.** *Let* $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ *be an extensive game. An assessment* $(\vec{\alpha}, \vec{\mu})$ *is* sequentially rational *if for every* $i \in [N]$ *and every strategy* α_i' *of player* i*, for every information set* $I \in \mathcal{I}_i$*, it holds that*

$$u_i((\vec{\alpha}, \vec{\mu})|I) \geq u_i(((\alpha_i', \vec{\alpha}_{-i}), \vec{\mu})|I).$$

[20] It is straightforward to verify that this is an equilibrium, by considering the payoff matrix of the game.
[21] In this example, we assume that the game is with perfect information.

Definition 5.7 almost fully captures the idea players should be making the best decision possible given their beliefs at any point during game-play. To fully characterize a *sequentially rational Nash equilibrium*, we require additionally that the beliefs of the players be *consistent* with $\vec{\alpha}$. For example, if an event occurs with zero probability in $\vec{\alpha}$, then we require that the players also *believe* that it will occur with zero probability.

Definition 5.8. *[Consistent assessment] Let* $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ *be an extensive game. A strategy profile* $\vec{\alpha}$ *is said to be* completely mixed *if it assigns positive probability to every action at every information set. An assessment* $(\vec{\alpha}, \vec{\mu})$ *is* consistent *if there is a sequence* $((\vec{\alpha}^n, \vec{\mu}^n))_{n \in \mathbb{N}}$ *of assignments that converges to* $(\vec{\alpha}, \vec{\mu})$ *in Euclidean space, where each* $\vec{\alpha}^n$ *is completely mixed and each belief system* $\vec{\mu}^n$ *is derived from* $\vec{\alpha}^n$ *using Bayes' rule.*

Finally, we arrive at the definition of a sequentially rational Nash equilibrium.

Definition 5.9. *[Sequentially rational Nash equilibrium]An assessment is a* sequentially rational Nash equilibrium *if it is sequentially rational and consistent.*

5.2 Game-theoretic analysis of Spacecoin

In order to analyze the game-theoretic properties of Spacecoin mining, we define an extensive game, SpacecoinGame, which models the actions that miners can take, and the associated payoffs. To facilitate analysis, we simplify the action space of the game as much as possible while still accurately modeling the incentives of Spacecoin miners. Concretely:

- We do not include the action of *creating a space commitment* because (as discussed in Section 2.1 under "Mining") we can assume that rational miners will commit to all the space they have, and nothing else.[22]
- We do not include the action of *creating transactions* because such actions do not affect the rewards that players receive from mining blocks, except in the case of punishment transactions. To deal with the case of punishment transactions, we define the payoff of a player who mines multiple blocks in the same time step to be zero. This payoff function exactly captures that of a miner in the actual Spacecoin protocol, because it is a *dominant strategy* for each other miner to create a punishment transaction (including a positive transaction fee) if she sees that a cheating player has mined multiple blocks in a time step, and hence we can assume that the cheating player will surely be punished at a later point in the protocol. Since the punishment penalizes the cheating player by the amount of the mining reward, it follows that the cheater's overall utility for the time step in which he cheated is zero.
- We do not explicitly model the amount of space that each player has[23]. Instead, we study the two critical cases: in our initial analysis, we assume that no miner controls more than 50% of the space committed by active miners. Then, we discuss potential issues that arise if a miner does control a majority of the space.

Definition 5.10. *[The Spacecoin Game] Let* $\mathcal{B}$ *denote set of all* blocks *as defined in Section 2.5. For any number of players* $N \in \mathbb{N}$*, any number of time steps* $K \in \mathbb{N}$*, and any reward function* $\rho \colon \mathbb{N} \to \mathbb{N}$*, we define the extensive game* $\mathsf{SpacecoinGame}_{\Pi,K,\rho} = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$ *as follows:*

[22] Later in this section, we address what happens if a miner gains additional space (or loses some space) during the game.

[23] We remark that the standard way to model this would be to assign a *type* to each player, representing how much space he has.

- *The set H of histories is defined inductively as follows.*
  - *The action set of the Chance player $A_{\mathcal{C}}(h) = \{0,1\}^m$ is the same for every history h.*
  - *The empty sequence $()$ is in H, and $A_i(()) = \{(\varnothing, \varnothing)\}$ for each $i \in [N]$.*
  - *Let $h = (h', a)$ be any non-terminal history where the latest action profile $a = (a_1, \ldots, a_N, a_{\mathcal{C}})$ consists of the actions of each player in $[N] \cup \{\mathcal{C}\}$ at history h', and for each player $i \in [N]$, the action $a_i = (S_i, T_i)$ is a pair of sets. Then for any $i \in [N]$, the action set $A_i(h)$ of player i at h is*

    $$A_i(h) = \mathcal{P}(T) \times \mathcal{B} \qquad \textit{where} \qquad T = \bigcup_{i \in [N]} T_i.$$

    An action $a_i = (S_i, T_i)$ can be interpreted as follows: S_i is the set of blocks from the previous time step which player i attempts to extend in this time step, and T_i is the set of extended blocks which player i announces in this time step.
- *The probability measure $f(\cdot, h)$ is uniform over $\{0,1\}^m$.*
- *For each $i \in [N]$, we define the partition $\mathcal{I}_i$ by an equivalence relation $\sim_i$. The equivalence relation $\sim_i$ is defined inductively as follows (we write $[h]_i$ to denote the equivalence class of h under $\sim_i$):*
  - *$[()]_i = \{()\}$, that is, the empty sequence is equivalent only to itself.*
  - $[(h, ((S_1, T_1), \ldots, (S_N, T_N), a_{\mathcal{C}}))]_i =$

    $$\left\{(h', ((S'_1, T'_1), \ldots, (S'_N, T'_N), a'_{\mathcal{C}})) \in H : h \sim_i h' \wedge S_i = S'_i \wedge T_i = T'_i \wedge a_{\mathcal{C}} = a'_{\mathcal{C}} \wedge \forall j \neq i,\ T_j = T'_j\right\},$$

    where h and h' are histories and the pairs (S_j, T_j) and (S'_j, T'_j) are actions of player j. That is, two histories are equivalent under $\sim_i$ if they are identical except in the "first components" S_j of the actions (S_j, T_j) taken by the players other than i.
- *$\vec{u} = (u_1, \ldots, u_N)$, where each $u_i : Z \to \mathbb{R}$ is defined as described below. For a history h, let $\mathsf{beac}(h)$ denote the sequence of actions taken by the Chance player in h, and let $\mathsf{beac}_j(h)$ denote the jth action taken by the Chance player in h. For a block B, let $B.c$ denote the challenge c within the proof of space of B. Recall that $\mathsf{Quality}(B)$ was defined in Section 4. We define*

  $$\mathsf{Quality}(B, c) = \begin{cases} \mathsf{Quality}(B) & \textit{if } B.c = c \\ 0 & \textit{otherwise.} \end{cases}$$

  Similarly, let $\mathsf{QualityPC}((B_1, \ldots, B_L), (c_1, \ldots, c_L))$

  $$= \begin{cases} \mathsf{QualityPC}((B_1, \ldots, B_L)) & \textit{if } \forall i \in [L],\ B_i.c = c_i \\ 0 & \textit{otherwise.} \end{cases}$$

  Let $\mathsf{blocks}(h)$ denote the sequence of "winning blocks" at each time step in the game, defined inductively:
  - $\mathsf{blocks}(()) = ()$
  - $\mathsf{blocks}((h', ((S_1, T_1), \ldots, (S_N, T_N), a_{\mathcal{C}}))) = \arg\max_{B \in T}(\mathsf{Quality}(B, \mathsf{beac}_{|h|}(h)))$, *where $T = \cup_{i \in [N]} T_i$.*

Let $\mathsf{blocks}_j(h)$ *denote the* j*th block in the blockchain. We assume that the winning block is unique at each time step*[24].

Let $\mathsf{winners}(h)$ *denote the sequence of players who announce the winning block at each time step in the game, defined inductively as follows:*

– $\mathsf{winners}(()) = ()$
– $\mathsf{winners}(h = (h', ((S_1, T_1), \dots, (S_N, T_N), a_{\mathcal{C}}))) = \arg\max_{i \in [N]} \max_{B \in T_i}(\mathsf{Quality}(B, \mathsf{beac}_{|h|}(h)))$.

Let $\mathsf{winners}_j(h)$ *denote the* j*th winner in the sequence* $\mathsf{winners}(h)$*. Let* $\mathsf{onlyone}_j(i, h)$ *be an indicator variable for the event that player* i*'s* j*th action* (S_i, T_i) *in the history* h *does not mine multiple blocks, i.e.* $|T_i| \leq 1$.

Finally, the players' utility functions are defined as follows: for a terminal history h *of length* K,

$$u_i(h) = \sum_{j \in [K]} \delta_{i, \mathsf{winners}_j(h)} \cdot \mathsf{onlyone}_j(i, h) \cdot \rho(\mathsf{blocks}_j(h)),$$

where $\delta_{i,j}$ *is the Kronecker delta function*[25]*. That is, a player's utility is the sum of the rewards he has received for announcing a winning block (in the time steps where he has announced at most one block).*

By Definition 5.10, for any $i \in [N]$, for any histories h, h' in the same information set $I \in \mathcal{I}_i$, it holds that $\mathsf{blocks}(h) = \mathsf{blocks}(h')$. Thus, we can associate a unique blockchain with each information set: we define $\mathsf{blocks}(I)$ to be equal to $\mathsf{blocks}(h)$ for any $h \in I$. Similarly, $\mathsf{beac}(h) = \mathsf{beac}(h')$ for any $h, h' \in I$ in the same information set I, so we define $\mathsf{beac}(I)$ to be equal to $\mathsf{beac}(h)$ for any $h \in I$.

For a block $B \in \mathcal{B}$ and a challenge $c \leftarrow \mathsf{Challenge}$, we define $\mathsf{Extend}_i(B, c)$ to be the block generated by player i when mining the next block after B using the PoSpace challenge c (the exact format of such a block is specified in Section 4).

Theorem 5.11. *For any number of players* N*, any number of time steps* $K \in \mathbb{N}$*, and any reward function* $\rho : \mathbb{N} \to \mathbb{N}$*, let* $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ *be a pure strategy profile of* $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$*, defined as follows: for each* $i \in [N]$*, for any information set* $I \in \mathcal{I}_i$ *such that* $I \neq \{()\}$,

$$\alpha_i(I)\left(\left(\{\mathsf{blocks}_j(I)\}, \{\mathsf{Extend}_i(\mathsf{blocks}_j(I), \mathsf{beac}_j(I))\}\right)\right) = 1,$$

where $j \geq 1$ *is the length of the histories in information set* I[26]*. That is, player* i*'s next action at information set* I *is*

$$\hat{\alpha}_i = \left(\{\mathsf{blocks}_j(I)\}, \{\mathsf{Extend}_i(\mathsf{blocks}_j(I), \mathsf{beac}_j(I))\}\right).$$

Then $\vec{\alpha}$ *is a Nash equilibrium of* $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$.

Proof. Take any player $i \in [N]$. By the definition of Extend, for any information set $I \in \mathcal{I}_i$ with $I \neq \{()\}$, the quality v of the extended blockchain

$$v = \mathsf{QualityPC}((\mathsf{blocks}(I), \mathsf{Extend}_i(B, \mathsf{beac}_j(I))), \mathsf{beac}(I))$$

[24] This can be achieved by breaking ties between blocks in an arbitrary way. Note that it is not possible for two different players to announce exactly the same (valid) block, because each block contains the miner's identity.

[25] Kronecker delta function: $\delta_{i,j} = 1$ if $i = j$, and 0 otherwise.

[26] All histories in an information set must be of the same length.

is the same for any block B which was announced at time step j. Therefore, no utility can be gained by choosing any block B over any other block B' to extend: that is, $u_i(\vec{\alpha}) \geq u_i(\alpha'_i, \vec{\alpha}_{-i})$ for any strategy α'_i which distributes probability over actions of the form (S, T) where $|S| = 1$.

Moreover, not extending any block *or* extending multiple blocks precludes a player from being the "winner" and receiving the reward in this time step, so extending a block is preferable to not extending any block. That is, $u_i(\vec{\alpha}) \geq u_i(\alpha'_i, \vec{\alpha}_{-i})$ for any strategy α'_i which assigns non-zero probability to any action of the form (S, T) where $|S| \neq 1$.

We have shown that $u_i(\vec{\alpha}) \geq u_i(\alpha'_i, \vec{\alpha}_{-i})$ for all strategies α'_i of player i. The theorem follows. □

Theorem 5.12. *Let* $\Pi = \{\mathsf{Init}, \mathsf{Challenge}, \mathsf{Answer}, \mathsf{Verify}\}$ *be a proof of space. For any number of players* N*, any number of time steps* $K \in \mathbb{N}$*, and any reward function* $\rho : \mathbb{N} \to \mathbb{N}$*, let* $(\vec{\alpha}, \vec{\mu})$ *be an assessment of* $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$ *where:*

- $\vec{\alpha}$ *and* $\hat{\alpha}_i$ *are defined as in Theorem 5.11, and for each* $n \in \mathbb{N}$*, we define* $\vec{\alpha}^n$ *to be the completely mixed strategy profile which (at history* h*) assigns probability* $1/|A_i(h)|^n$ *to every action except* $\hat{\alpha}_i$*, and assigns all remaining probability to* $\hat{\alpha}_i$*.*
- $\vec{\mu}$ *is derived from* $\vec{\alpha}$ *using Bayes' rule in the following way:* $\vec{\mu} = \lim_{n\to\infty} \vec{\mu}^n$*, where for each* $n \in \mathbb{N}$*,* $\vec{\mu}^n$ *is derived from* $\vec{\alpha}^n$ *using Bayes' rule.*

Then $(\vec{\alpha}, \vec{\mu})$ *is a sequentially rational Nash equilibrium of* $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$*.*

Proof. Let $I \in \mathcal{I}_i$ be any information set of player i in $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$, and let L be the length of histories in I. It follows from Definition 5.10 that the expected utility of player i at I is $u_i((\vec{\alpha}, \vec{\mu})|I) =$

$$\sum_{j\in[L]} \delta_{i,\mathsf{winners}_j(h)} \cdot \mathsf{onlyone}_j(i, h) \cdot \rho(\mathsf{blocks}_j(h)) + u'_i((\vec{\alpha}, \vec{\mu})),$$

where u'_i is the utility function of player i in the game $\mathsf{SpacecoinGame}_{\Pi,K-L,\rho}$. Since $\mathsf{winners}$, $\mathsf{onlyone}$, and blocks are invariant over histories within any given information set, the summation term can be computed explicitly by player i at I. Hence, in order to maximize his expected utility at I, the player needs simply to maximize $u'_i((\vec{\alpha}, \vec{\mu}))$. Let $(\vec{\alpha}|_{K-L}, \vec{\mu}|_{K-L})$ denote the assessment $(\vec{\alpha}, \vec{\mu})$ for the first $K - L$ time steps of the game. By Theorem 5.11, $\vec{\alpha}|_{K-L}$ is a Nash equilibrium of $\mathsf{SpacecoinGame}_{\Pi,K-L,\rho}$. Since $\vec{\mu}$ is derived from $\vec{\alpha}$ by Bayes' rule, it follows that $u_i((\vec{\alpha}, \vec{\mu})|I) \geq u_i(((\alpha'_i, \vec{\alpha}_{-i}), \vec{\mu})|I)$ for any strategy α'_i of player i. Applying this argument for every I, we conclude that $(\vec{\alpha}, \vec{\mu})$ is sequentially rational in $\mathsf{SpacecoinGame}_{\Pi,K,\rho}$.

By construction, $\lim_{n\to\infty} \vec{\alpha}^n = \vec{\alpha}$ and $\vec{\mu} = \lim_{n\to\infty} \vec{\mu}^n$, so $(\vec{\alpha}, \vec{\mu})$ is consistent. The theorem follows. □

Parameters. The Spacecoin Game is parametrized by N and K. It is natural to ask: do we require that the number of miners N is fixed in advance, or that the block-chain will end after a certain number K of time-steps? The answer is *no*. Theorem 5.12 gives a sequentially rational Nash equilibrium in which each player's strategy is independent of N, and so it makes sense for each miner to play this strategy even if N is unknown or changes over time. In light of this, from each rational player's point of view, K can be considered to be the number of time-steps that he intends to participate in the game: perhaps his goal is to use his earnings to buy a house after K time-steps, or perhaps he does not expect to live for more than K time-steps[27]. The

[27] In the latter case, K is an upper bound on the number of time-steps that the player intends to stay in the game. It is reasonable to treat K as an upper bound because maximizing expected utility after K time-steps also maximizes expected utility after any $0 < L < K$ time-steps, as shown in the proof of Theorem 5.12.

crucial observation is that even if different players have different values of K "in their heads", their equilibrium strategies are still the same.

Buying space. Players' strategies in equilibrium do not depend on the amount of space that (they believe) other players possess. Also, we showed above that the equilibrium strategies are robust to changes in N. Hence, if a player's amount of space changes (e.g. he buys/sells a hard disk), then he can simply create a new space commitment, and then behave as a "new player" with the new amount of space.

The "51% Attack". If a player P controls more than half of the total space that belongs to active miners, then following the protocol rules is no longer a Nash equilibrium, because whichever branch of the block-chain P chooses to mine on will eventually become the highest-quality chain. Thus, P can decide arbitrary rules about which blocks to extend, and the other players will be incentivized to adapt their strategies accordingly. Moreover, P can prevent certain transactions from ever getting into the block-chain, by refusing to extend blocks which contain these transactions – as a consequence, P can mine multiple blocks per time-step without ever being punished. This attack was first analyzed by [KDF13] in the context of Bitcoin, which suffers from the same problem (with respect to computing power rather than space).

It may seem unrealistic that a single party would control more than half of the total space that belongs to active miners in a widely adopted currency. A more realistic concern could be that a large group of miners (in a *mining pool*) may acquire more half of the total space. However, under the assumption that each miner is an individual strategic agent, we consider it unlikely that such a mining pool could do much damage: for this, a large group of self-interested and relatively anonymous agents would have to coordinate *and trust each other* throughout the duration of an attack. In particular, each rational miner in the pool must be convinced that he will get his share of the attack profits, and it seems highly unlikely that a large group of anonymous people would all trust each other so. The improbableness of a 51% attack by a mining pool is supported by recent events: when a large mining pool (`ghash.io`) was nearing 50% of Bitcoin computing power in 2014, self-interested miners started leaving the mining pool in order to avoid destabilizing the currency.

References

[ABC$^+$07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 598–609. ACM Press, October 2007.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 538–557. Springer, September 2014.

[And13] Nate Anderson. Mining Bitcoins takes power, but is it an "environmental disaster"?, April 2013. `http://tinyurl.com/cdh95at`.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *CCSW*, pages 43–54, 2009.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *CRYPTO 2015*, 2015.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 125–143. Springer, March 2011.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, August 1993.

[DPML+03] R. Di Pietro, L.V. Mancini, Yee Wei Law, S. Etalle, and P. Havinga. Lkhw: a directed diffusion-based secure multicast scheme for wireless sensor networks. In *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, pages 397–406, 2003.

[GJM03] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135. Springer, March 2003.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[JK07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 584–597. ACM Press, October 2007.

[KDF13] Joshua A. Kroll, Ian C. Davey, and Edward W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Workshop on the Economics of Information Security*, June 2013.

[KK14] Nikolaos P. Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 520–537, 2014.

[MJS+14] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE Computer Society Press, May 2014.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. `http://bitcoin.org/bitcoin.pdf`.

[Nas50] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[PT10] Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 643–662. Springer, September 2010.