# Spacemint*: A Cryptocurrency Based on Proofs of Space

Sunoo Park*[§], Krzysztof Pietrzak[†][§], Albert Kwon*, Joël Alwen[†], Georg Fuchsbauer[†], and Peter Gaži[†]

* MIT
[†] IST Austria

***Abstract*—We propose a decentralized cryptocurrency called Spacemint, which is based on a block chain ledger similar to that of Bitcoin, but where the wasteful proofs of work are replaced by efficient *proofs of space*, recently introduced by Dziembowski et al. Instead of requiring that a majority of the network's computing power is controlled by honest miners (as in Bitcoin), our currency requires that honest miners dedicate more net disk space than a potential adversary.**

**In Spacemint, 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, 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 efficiently add the next block to the chain and get the mining reward. In this paper, we detail the construction of Spacemint, analyze its security and game-theoretic properties, and study its performance. Our prototype shows that it takes approximately 25 seconds to prove over a terabyte of space, and it takes a fraction of a second to verify the proof.**

## I. Introduction

Bitcoin is a decentralized digital currency which was introduced in 2009 [22] and now is by far the most successful digital currency ever deployed. The currency's decentralized bookkeeping depends on maintaining a public ledger recording all transactions that occur. This ledger is implemented by a *block chain*: that is, a sequence of blocks each of which contains transaction records and some auxiliary information, which are generated by participants in the network. To encourage participants to contribute blocks, those who add a block to the chain are rewarded with some newly minted Bitcoin.

A principal difficulty when designing a digital currency is to provide security against double-spending: that is, the owner of a coin must be able to spend it exactly once. To prevent double-spending, it must be enforced that all parties in the network agree on the same block chain (except possibly for the most recent few blocks). Before accepting a transaction, a recipient should wait until the transaction has been in the chain long enough that she can be reasonably sure it will stay there forever (that is, *consensus* has been reached).

The Bitcoin protocol achieves consensus by making it computationally hard to add a block to the chain: currently, mining a Bitcoin block requires about $2^{65}$ hash computations [1]. Accordingly, a Bitcoin block is considered to be a *proof of work* [11]: that is, a proof that a certain amount of computational resources were invested.

The Bitcoin network mines a block approximately every 10 minutes, and so it consumes computational resources – and associated natural resources, primarily in the form of electricity – at a massive scale. Current network usage is estimated to be several 100 MW of power; 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" [5] by some.

The original idea behind basing Bitcoin mining on computational power was that anyone could participate in the network by dedicating their spare CPU cycles, which incurs little marginal cost in that it uses the idle time of already-existing personal computers. However, the dynamics of modern Bitcoin mining have become very different: the majority of successful mining is done by large-scale mining farms, often in collaboration with electricity producers. Without specialized mining ASIC hardware, you don't have a chance: mining with your spare CPU cycles will *lose* money, due to overhead electricity costs. The nonlinearity of mining rewards in Bitcoin relative to resources invested has been quantified by [26] as shown in Figure 1 below. Designing a cryptocurrency where expected reward is more proportional to invested resources would be desirable for a number of reasons, including that the presence of "small players" can be important for the stability and decentralization of the currency.

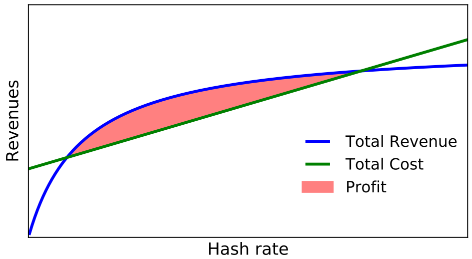

Fig. 1: Bitcoin mining profit vs. invested resources [26]

To address these issues, in this paper we propose a cryptocurrency, Spacemint, which replaces the costly proofs of work underlying Bitcoin with *proofs of space* [13]. In Spacemint, in order to mine blocks (and thereby mint coins), miners must invest *disk space* rather than computational power.

[§] Joint first author.

* In a previous version, this cryptocurrency proposal was called "Spacecoin" rather than "Spacemint", but this has been changed due to name conflicts.

The first author's research is 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. The second, fourth, fifth, and sixth authors' research is supported by ERC Starting Grant (259668-PSPC).

Miners who dedicate more disk space have a proportionally higher expectation of successfully mining a block and reaping the reward. These days, tremendous amounts of disk space are lying around unused, and all of this storage capacity carries potential for mining. We note that in a space-based scheme, it is clear that miners will be incentivized to invest in hard drive capacity, just as Bitcoin miners are incentivized to invest in electricity. However, we highlight a couple of key differences:

1) In Spacemint, the investment is in the form of capital expenditure, and the mining process after the hard drives are bought incurs negligible overhead cost (both in terms of monetary and natural resources). In contrast, in Bitcoin, the mining process requires perpetual energy expenditure from miners.
2) In Bitcoin, resources are "used up" by mining: electricity is a *depletable* resource which once used is gone, and Bitcoin mining hardware is specialized, single-purpose resource that is not useful for anything once the need for Bitcoin mining is removed. In contrast, the resource consumed by Spacemint is *recyclable*, in that it can be used over and over, and *multi-purpose*, since hard drives have intrinsic value in their ability to store useful data.[1]

The distinction between capital expenditure and recurring overhead costs is significant also in that it changes the trade-off between expected reward and mining resources invested. More concretely, due to the low marginal cost of mining a block, the shape of the curve depicted in Figure 1 will be much flatter for Spacemint than Bitcoin, and thus the profitable area would be spread more evenly over the horizontal axis.

### A. *Background and challenges*

**A "greener" cryptocurrency.** The community has looked for alternative decentralized consensus protocols and found a potentially promising candidate in *proofs of stake* (PoStake). In such schemes, the probability that a party mines the next block is proportional to the fraction of coins (out of all coins belonging to participating miners) 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 schemes based on proof of work (PoW).

Trying to adapt Bitcoin in a straightforward way by replacing PoW by PoStake, one runs into at least three major problems which are outlined below. Intuitively, the first two problems are related to the fact that producing a PoStake proof is computationally cheap and thus opens up potential for cheating in ways which are not possible in Bitcoin. We will see that analogous challenges arise from the computational ease of mining a block by proof of space, too; and our Spacemint construction will propose ways to resolve these challenges.

*1) Multiple chains:* In Bitcoin, a rational miner will always work towards extending the longest chain of which he is aware, as working on any other chain would only lower the probability that his mined block will end up in the block chain. When using PoStake instead of PoW, checking whether one can extend a chain is very cheap, and thus miners may try extending many different chains in parallel. This impedes quick consensus finding, unlike in Bitcoin where all rational miners concentrate on the longest chain, and thus it always grows faster than others.

*2) Grinding:* In Bitcoin, the miner adding the $i^{th}$ 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).

*3) Participation:* In a PoStake-based scheme, the parties holding coins must also participate in securing the currency by providing blocks when their coins "win". Typically, only a fraction of the parties holding coins will participate, and this is an inherent challenge in designing a PoStake-based scheme (as the scheme must work no matter what the fraction is and how it changes over time). Moreover, this can become a security issue when participation is very low. In the most popular PoStake-based currency, Peercoin [19], participation is below $10\%$.

Largely due to these and related issues, many existing proof-of-stake based cryptocurrency proposals combine PoStake with some PoW, rather than basing purely on PoStake.

**Proofs of space.** A *proof of space* (PoSpace), recently introduced by Dziembowski et al. [13], is a protocol between a prover and a verifier that has two phases. After an *initialization phase*, the prover $\mathcal{P}$ is supposed to store some data $S_\gamma$ of size $N$, and the verifier stores a short commitment $\gamma$ to this data. In a later *execution phase* $\mathcal{V}$ sends a challenge $c$ to $\mathcal{P}$, who efficiently responds with a short answer $a$ after reading a small fraction of $S_\gamma$. For application to cryptocurrency, it is necessary to modify the format of the PoSpace as defined in [13], because:

- in our setting, there is no (single) entity to act as the verifier in the protocol, yet
- we require that anyone who sees the public ledger (i.e. blockchain) can efficiently verify the validity of a proof.

### B. *Our approach and contributions*

In this paper we propose digital currency schemes based on proofs of space. We first construct a variant of PoSpace that is suitable for the cryptocurrency setting (Section IV), which addresses the issues outlined in the bullets above.

Two further problems that arise when integrating PoSpace into a blockchain-based cryptocurrency are the issues of *grinding* and *mining multiple chains*. (Note that the *participation* problem which is inherent to PoStake does not arise in PoSpace-based schemes.) We address these issues as follows.

- *Grinding:* 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. Details are in Section VI-A.

[1] One may ask: won't Spacemint spur development of specialized types of storage which are tailored for mining, and thus end up in the same position as Bitcoin in this respect? We argue that this is unlikely; see Section IX.

- *Mining multiple chains:* Our approach is based on penalizing miners who work on more than one branch. It is important to discourage miners not only from announcing blocks on multiple chains, but also from "trying out" many different chains and choosing only the best one to announce. We suggest three variant solutions. In the first two schemes, a miner's block quality is fixed for any given time-step, so that trying many chains does not yield any benefit; and then we penalize miners who *announce* blocks on multiple chains. The third scheme takes a different approach, and introduces interaction between miners during the mining process, thus enabling detection of miners who try many chains. Details in Section V-C.

Finally, we perform a game-theoretic analysis of Spacemint and find that it has at least as strong equilibrium properties as Bitcoin. We model the Spacemint protocol as an *extensive game*, and prove that miners are not incentivized to deviate from the rules as long as there is an honest majority. More formally, we prove that the protocol is a *sequentially rational Nash equilibrium*, which is the standard equilibrium concept for games which happen over many time-steps. Prior work related to equilibria in Bitcoin has given only an informal treatment of the problem: notably, [20] presents a thorough, but still informal, analysis of equilibrium strategies in Bitcoin, and concludes that honest mining is a Nash equilibrium in Bitcoin (if there is an honest majority).

**Contribution.** In summary, our contribution is as follows.

- *Cryptocurrency from proofs-of-space:* Spacemint is a cryptocurrency based purely on proofs of space, and thus avoids the major drawbacks of existing proof-of-work-based schemes as discussed in this section.
- *Addressing the "nothing-at-stake" problem:* We propose novel approaches to the known problems of grinding and mining multiple chains in non-proof-of-work based systems. Our solutions can also be extended to the proof-of-stake setting where these issues were first encountered.
- *Evaluation of Spacemint and proofs-of-space:* We implemented a proof-of-space library and a prototype of Spacemint. It takes less than 20 seconds to prove over 1 TB of space, and a fraction of a second to verify.
- *Game theory of Spacemint:* Our game-theoretic analysis models Spacemint as an extensive game proves that the adhering to the protocol is a sequential Nash equilibrium.

## II. Related work

**Proofs of storage/retrievability.** Other concepts similar to proofs of space are *proofs of storage* and *proofs of retrievability* (cf. [15], [8], [7], [17], [10] 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 sends the entire file to the prover, whereas an important property of PoSpace is that the verifier's computation (and thus also communication) is at most polylogarithmic in the size of storage dedicated.

**Proofs of secure erasure.** Another type of proof system which is related to PoSpace is *proof of secure erasure* (PoSE). Informally, a PoSE allows a space-restricted prover to convince a verifier that it has erased its memory of size $S$. PoSE were suggested by Perito and Tsudik [25], 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. PoSE with small communication complexity have been constructed by [14], [18], [6]. In [6], a weaker variant of PoSE is considered[2], where a prover "passes" whenever he has access to a sufficient amount of space, but must not necessarily have erased it during protocol execution. (Surprisingly, there actually exist problems which need space to compute, but do not erase it [9].) A PoSpace 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 of which we are aware was proposed in [25] (i.e. to prove having erasing one's memory). In particular, PoSE cannot be used for any of the applications of PoSpace put forward in [13], and also the cryptocurrency proposed in this paper. We refer the reader to [13] for a more detailed discussion on PoSpace vs. PoSE.

**Permacoin.** Permacoin [21] 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 Spacemint the dedicated storage does not store anything useful. Like in Bitcoin, Permacoin miners are constantly racing to find a good proof (but the type of proof is different). In contrast, the main goal of Spacemint 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, or not use them at all.

**Burstcoin.** The only cryptocurrency of which we are aware that uses disk space as the primary mining resource is Burstcoin [2]. The first public mention of Burstcoin we could find is from mid-August 2014, which is over one year after the first public talk on proofs of space and their potential for constructing a "green" cryptocurrency [12]. As this first proposal of a cryptocurrency based on proofs of space had several security issues, it was not published. Below, we observe that Burstcoin shares some of the security issues that the early proposals had (most notably, time/memory trade-offs), and also highlight some other issues with the Burstcoin mining process.

A major efficiency issue of Burstcoin is that a constant fraction ($0.024\%$) of dedicated disk space must be read every time a block is mined[3] Another issue is that when a miner publishes a block, one has to hash over 8 million blocks just to verify that the miner's claim is valid. In terms of security, arguably the most serious problem with Burstcoin is that it allows for time/memory trade-offs: a miner doing just a little extra computation can mine at the same rate as an honest

[2] Note that [6] refers to this type of proof as "one-stage proof of space" which has led to some confusion with the original "proof of space" notion proposed in 2013 [12] and first realised in [13]. In this work, "proof of space" always refers to the notion of [13].

[3] In contrast, Spacemint requires accessing only logarithmically many blocks in the size of dedicated space. A conservative back-of-the-envelope calculation shows that Spacemint requires less disk reading than Burstcoin already at a couple of GB of dedicated space. At 1TB, Burstcoin reads 24GB, which is 1000 times more than Spacemint requires.

miner, while using just a small fraction (say, 10%) of the space. We detail this attack in Appendix B.

## III. Bitcoin preliminaries

In Bitcoin, 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 $sk$. To transfer a coin from $pk$ to $pk'$, a transaction (encoding the statement that the coin of $pk$ 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 $\beta_0, \beta_1, \dots$ of blocks. Each block $\beta_i = (tx_i, \dots)$ contains, among other information discussed below, a set $tx_i$ of new transactions to be added.

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

The threshold $t$ in the Bitcoin scheme is set dynamically (and adapted roughly every two 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 $pk_{\text{miner}}$ associated with the block. The secret key corresponding to $pk_{\text{miner}}$ is known by the miner who found the block, who thus gains ownership of these newly minted coins. An additional mechanism to incentivize mining is the *transaction fee*, whereby the initiator of a transaction offers a small reward to the miner who includes the transaction in the blockchain. Transaction fees are optional, and typically they are much smaller than the reward for mining a block.

Once a miner finds a $\mu_{i+1}$ satisfying the requirements above, 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*[4] of which it is aware; this way, even if the block chain branches, ultimately one branch will become longer, and the shorter ones will eventually be ignored. It follows that once a transaction has been added to the block chain, and sufficiently many blocks have been added after it, the transaction will stay in the chain forever.

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 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 a "dead" branch), and thus, always working to extend the longest known chain is a Nash equilibrium.

## IV. Proofs of space

As briefly discussed in Section I, the goal of proof of space is for a prover to prove to a verifier that it is storing a certain amount of space. In this section, we first discuss two straw man approaches that do not work, and then present our variant of PoSpace for the cryptocurrency setting.

### A. Two simple approaches that don't work

**Storing a function table.** A tempting "solution" is to have $\mathcal{P}$ store a lookup table $(1, f(1)), \dots, (N, f(N))$ of a random-looking function $f$, sorted by the output. The prover's challenge would be to invert the function on value $f(x)$ for some random $x \in [N]$: an honest prover can do this in time $\log(N)$ by binary search. Unfortunately, this doesn't work due to time/memory trade-offs [16], which allow a cheating prover to only store roughly $N^{2/3}$ input/output pairs and still invert the function in time $N^{2/3}$ (in the case where $f$ is a permutation, this goes down to $N^{1/2}$ time and space, cf. [13, Appendix A] for details).

**Storing a random file.** Another simple idea would be for $\mathcal{V}$ to send $N$ (pseudo)random bits to $\mathcal{P}$ during initialization, and simply query $\mathcal{P}$ for random subsets of these bits during execution. However, this requires $N$ bits of communication, whereas a PoSpace requires that the verifier's efficiency depends on some security parameter, but must be basically independent of $N$ – and this property is crucial for all applications of PoSpace discussed in [13] and also for this paper.

### B. Hard-to-pebble graphs

The PoSpace schemes proposed in [13] are based on *pebbling graphs*. These are directed graphs where the vertices are labeled as follows: for some unique nonce $\mu$ (which is send by $\mathcal{V}$ to $\mathcal{P}$ in the initialization phase), vertex $i$ gets label

$$l_i := \mathsf{hash}(\mu, i, l_{p_1}, \dots, l_{p_t}) \tag{1}$$

where $p_1, \dots, p_t$ are the parents of vertex $i$. The PoSpace construction requires a family of graphs which are hard to pebble: these graphs provide a provable lower bounds for the amount of computation *or* storage that must be spent in order to correctly compute the label of a random node in the graph.

[13] constructs two different families of hard-to-pebble graphs that can instantiate their PoSpace scheme. The first family has in-degree $O(\log\log(N))$ and gives a scheme where any prover that can convince the verifier with constant probability must either dedicate $\Theta(N)$ bits of space (like an honest prover), or run in time $\Theta(N)$ in the execution phase.[5] The second family of graphs has in-degree 2 and gives a scheme where any prover must either dedicate $\Omega(N/\log(N))$ bits of storage, or otherwise incur at least $\Omega(N/\log(N))$ space complexity (and thus also time) during the execution phase.

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

[5] Note that this is the best we can hope for, as a cheating prover can always just store the verifier's message from the initialization phase, and re-initialize the entire storage during the execution phase, which takes time $\Theta(N)$.

**Algorithm 1** Initialization

*Initial state:* $\mathcal{P}$ needs $N = 2 \cdot n \cdot L$ bits of storage.
*Common input:* a (hard to pebble) DAG $G$ with $n$ nodes and a function $\mathsf{hash} : \{0,1\}^* \to \{0,1\}^L$.

1) $\mathcal{V}$ samples a unique nonce $\mu$ and sends it to $\mathcal{P}$.[6]
2) $\mathcal{P}$ computes and stores $(\gamma, S_\gamma) := \mathsf{Init}(\mu, N)$ and sends the commitment $\gamma$ to $\mathcal{V}$. Here $S_\gamma$ contains the labels of all the nodes of $G$ computed as in equation (1) (of total size $n \cdot L$). $\gamma$ is a Merkle tree commitment for these $n$ labels. The intermediate values of this Merkle tree are also contained in $S_\gamma$ (and are also of total size $n \cdot L$).

**Algorithm 2** Execution

*Initial state:* $\mathcal{V}$ holds commitment $\gamma$, $\mathcal{P}$ stores $S_\gamma$.
*Common input:* $k \in \mathbb{N}$, where $k$ is determined by number of nodes needed to verify the commitment.

1) $\mathcal{V}$ samples $k$ challenges $(c_1, \ldots, c_k) \leftarrow \mathsf{Challenge}(n, k, \$)$ and sends them to $\mathcal{P}$ (the challenge sampling algorithm $\mathsf{Challenge}$ takes as input the size of the graph $n$, the number of requested challenges $k$, and the last input are the random coins to be used for the sampling).
2) $\mathcal{P}$ computes the answers $a_i := \mathsf{Answer}(\mu, S_\gamma, c_i)$ for $i = 1, \ldots, k$ and sends them back to $\mathcal{V}$. Each $a_i$ contains openings of the challenge label $\ell_{c_i}$ and also the labels of $c_i$'s parents.
3) $\mathcal{V}$ runs the verification procedure $b_i := \mathsf{Verify}(\mu, \gamma, c_i, a_i)$ and accepts if all $b_i = 1$. The procedure $\mathsf{Verify}$ outputs 1 if $a_i$ contains openings of the required labels, and the label $\ell_{c_i}$ is correctly computed, i.e., as in equation (1).

**Proofs-of-space for Spacemint.** A proof of space uses four algorithms $\mathsf{PoS} = \{\mathsf{Init}, \mathsf{Challenge}, \mathsf{Answer}, \mathsf{Verify}\}$, and is executed between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. At a high level, the PoSpace as described in [13] is carried out in two phases:

1) **Initialization:**
   a) **commitment generation:** $\mathcal{P}$ initializes its space and sends a commitment $\gamma$ to $\mathcal{V}$.
   b) **commitment verification:** $\mathcal{P}$ convinces $\mathcal{V}$ that $\gamma$ commits to labels that are correctly computed.
2) **Prove space:** $\mathcal{P}$ convinces $\mathcal{V}$ that it is holding the space.

In the cryptocurrency setting, we would like the network to act as $\mathcal{V}$, and thus it is infeasible to perform the commitment verification during the initialization phase. We therefore define a different protocol where the prove space phase and commitment verification will be in carried out in one combined step, which we call the execution phase. Our PoSpace thus consists of two phases, initialization and execution, which are described in Algorithm 1 and 2.

After initialization, $\mathcal{P}$ should have stored the data $S_\gamma$ of size $N$, whereas $\mathcal{V}$ only stores the short commitment $\gamma$ and the nonce $\mu$. We assume that the parameters $(G, \mu)$ are part of the commitment $\gamma$ and we will denote by $N_\gamma = 2 \cdot n \cdot L$, the space dedicated by an honest $\mathcal{P}$.

For verification, the construction based on the first graph family requires the verifier to check $k$ nodes and their parents, while the second construction requires opening $k \cdot \log(n)$ nodes and their parents. However, the first graph requires more parents to be opened per node, and the second graph also has a simpler structure. We therefore use the second graph for our prototype (Section VIII). We defer the exact efficiency and security parameters of the two constructions to Appendix A.

## V. Overview of Spacemint

### A. High-level protocol description

**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 propose a new design for the block chain in Section VI 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 add special transactions to initialize miners, and a special type of transaction which penalizes a miner who extends 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 Spacemint, computing power in Bitcoin): (1) a reward for adding blocks and (2) transaction fees.

*Reward:* For adding a block to the chain, a miner receives some freshly minted coins. The reward size is specified as part of the protocol, and typically depends on the block index.[7]

*Transaction fees:* When generating a transaction, one can dedicate some (usually very small) amount of the transferred coins to the miner who adds the transaction to the block chain.

**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 $\mu$, so we simply use $pk$ for this.)

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

The miner stores $(S_\gamma, sk)$ and generates a special transaction which just contains $(pk, \gamma)$. 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 $c$. This $c$ is used as randomness to sample $k_p$ challenges $c_p$ and $k_{cv}$ challenges $c_{cv}$.[8]

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

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

[8] Where, depending on which of the PoSpace discussed in the last section we use, $k_p = O(1), k_{cv} = \log(n)$ or $k_p = O(1), k_{cv} = \lambda \cdot \log(n)$. These challenges can be sampled by first using $c$ as a seed to generate a sufficient amount of randomness $(r_p, r_{cv}) := \mathsf{hash}(c)$ for the challenge sampling algorithm to get $c_p := \mathsf{Challenge}(n, k_p, r_p), c_{cv} := \mathsf{Challenge}(n, k_{cv}, r_{cv})$.

How to derive the challenge $c$ is the main difficulty we face. In our simplest solution we assume an unpredictable beacon that broadcasts a fresh 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 $c_p$:

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

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

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

that the proof $a$ is better than $a'$. We postpone the discussion on the precise definition of this ordering to Section V-B. 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 random oracle used to compute the proof quality.

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 a block and sends it out to the network in the hope that it will end up in the chain. A block must contain transactions, the PoSpace proof $a$ and also the commitment verification output computed as $a_{cv} := \mathsf{Answer}(pk, S_\gamma, c_{cv})$. Note that the commitment verification need not be executed unless the miner has found an exceptionally good proof: thus, the computation of the vast majority of miners in the network will be very low, since they need only *check* the quality of their proof, and most of them will not proceed with verification.

For the remainder of the 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 incurring any noticeable slowdown.

### B. 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 spaces 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 $\mathsf{hash}$) that the $i$th proof has the best "quality" corresponds to its fraction of the total space, i.e.

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

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

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

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

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

That is, we map every $a_i$ to a random value $\mathsf{hash}(a_i)$, 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.

$$D_N \sim \max\{r_1, \ldots, r_N \ : \ r_i \leftarrow [0,1], i = 1, \ldots, N\} \quad (2)$$

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

$$(a_j, N_j) \prec (a_i, N_i) \Leftrightarrow D_{N_j}(\mathsf{hash}(a_j)) \leq D_{N_i}(\mathsf{hash}(a_i)). \quad (3)$$

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 (2) 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 (3) we want to sample $D_{N_i}$ using randomness $\mathsf{hash}(a_i)$, and $\mathsf{hash}$ outputs bitstrings in $\{0,1\}^{256}$ instead of values in $[0,1]$, so we have to normalize:

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

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

**Remark.** Notice that the quality function described above has the property that the quality of block that a given miner $pk$ can produce in a given time-step is fixed, regardless of which chain he chooses to extend. This property will be important to prevent the "mining multiple chains" attack which was described in Section I, as explained in the next section.

### C. Where the challenge comes from

The main difficulty we face when designing a PoSpace-based scheme is the generation of the PoSpace challenge $c$. We suggest three solutions below: note that each one protects against the "mining multiple chains" attack by penalizing miners who try to mine on more than one chain; details are given in the corresponding subsections.

*1) Assuming an unpredictable beacon:* Our most basic solution assumes an unpredictable beacon which broadcasts a value every minute. By the beacon's unpredictability, no party at time $t$ has non-negligible probability of guessing the beacon value that will be announced 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 $i$ is derived as a hash of the beacon value (hashing maps an unpredictable value to a pseudorandom one).

An unpredictable beacon is a fairly strong assumption, but it does not trivialise the problem. It does mostly solve the grinding problem, but does not help with the "mining 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 increase the probability of his block being in the ultimate block chain. Such behavior is rational, but undesirable in that it prevents consensus on a single chain. To de-incentivize this, we define a special type of transaction which allows others 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 $(pk', \sigma'_j, \alpha)$ and have the following semantics. Let $pk$ be the key of the miner that added the $j^{th}$ block in the current chain. If this block does not contain a signature for $\sigma'_j$ and $\alpha$ is a signature for $\sigma'_j = (j, \dots)$ 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) 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 that comes to mind (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 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 considerably slow down consensus, as bad chains get extended with blocks of the same quality as the currently best chain; thus, we would expect to see a race between many different chains without the lower-quality chains falling behind rapidly. A solution to this kind of problem that is used in Slasher [4] 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 $i - \delta$ (rather than $i - 1$), for some appropriately chosen $\delta$. ($\delta = 120$ seems like a good value for a 1-minute time slot.) Now, a miner only obtains different challenges for the $i^{th}$ block of two different chains if those chains have forked at least $\delta$ blocks ago. With $\delta = 120$, it is extremely unlikely that two chains will survive in parallel for $\delta$ blocks. Recall that we penalize miners who add a block (using the same challenge) to two different chains, if there are multiple chains that forked less than $\delta$ blocks ago, there is a strong incentive for miners to add their blocks to the best chain: hence, we expect the other chains to fall behind very fast.

The reason $\delta$ cannot be arbitrarily large is that a miner at time $t$ knows its challenges for all the blocks $t+1, \dots, t+\delta$. Thus, she can pre-compute all $\delta$ answers, and need not access her space for the next $\delta$ minutes. If $\delta$ were 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 $\delta$-block window. To avoid this, $\delta$ should be set so that initializing the space takes roughly $\delta$ minutes.[9]

*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 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 $\phi_{i-1}$. He first computes an index $t = \mathsf{hash}(pk, \phi_{i-1}) \in \{1, \dots, 10\,000\}$, which means we must ask the user who mined block $i - t$ for a challenge. The miner publishes this request to the network, 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 successfully mine a block. 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 that 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 receives a request, he computes the challenge $c$ using a verifiable pseudorandom function (VRF) and publishes it.[10]

Realizing this solution seems significantly more complicated and delicate than the previous ones, due to the added

[9] Re-instantiating mining space is expensive, since the miner must overwrite his entire disk: we refer the reader to Section VIII for timings.

[10] We require a VRF to prevent "grinding" through many challenges even if the requesting and the providing miner collude, or the requesting miner is lucky and gets to "ask" himself for a challenge. With a VRF, even in these cases, the miner gets just one extra challenge for free, at best doubling her chance of mining the next block.

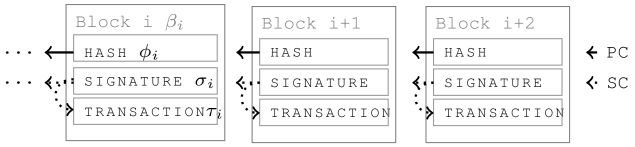

Fig. 2: Our block chain consists of a proof chain PC that does not allow for grinding, and a signature chain SC that binds transactions to the proof chain.

complexity of interaction as well as issues of availibility of online miners. We do not discuss further details of this approach in this work.

## VI. 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 $\beta_0$ necessarily has a somewhat different format as it cannot depend on previous blocks:

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

The links between consecutive blocks in the block chain are illustrated in Figure 2. 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.

### A. Solution to the grinding problem

By decoupling proofs from transactions we achieve security against *grinding*: for any space commitment $(pk, \gamma)$, the miner $pk$ cannot generate two (or more) correctly formatted hash-blocks to be added to the proof chain.[11]

The signature chain binds the transactions to the proof chain. If an honest miner (honest to be defined below) adds the $i$th 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 $j, \ldots, i$ must be re-computed, which requires $sk$.

### B. Transactions

Spacemint 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 Spacemint.

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

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

- $txId$: A unique, arbitrary *transaction identifier*. That is, no two transactions in a blockchain can have the same identifier.
- $\vec{out}$: A list of beneficiaries and the amount they receive. Specifically, $\vec{out} = (out_1, \ldots, out_m)$ with $out_i = (pk_i, v_i)$, where:
  - $pk_i$ is in the support of 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 input 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 $k_j$th 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:

[11] To prove this, we require that it is computationally hard to find two distinct accepting transcripts for the same challenge. The PoSpace of [13] satisfies this property (finding two accepting transcripts in their schemes amount to breaking collision-resistance of the underlying hash function).

[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, Spacemint 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.

[14] That is the $k_j$th beneficiary of transaction $txId_j$ is the $j$th benefactor of transaction $txId$.

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

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 at least the sum of the 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": e.g. if the sum of the input values exceeds the total payment amount he wants to transfer 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

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

consists of $pk$, a public key, and $\gamma$ which is computed as $(\gamma, S_\gamma) := \mathsf{Init}(pk, N)$. That is, $ctx$ is a space commitment to a space of size $N$ (we assume that $N$ is specified by $\gamma$, and thus do not explicitly add it).

*Why add commitments to the chain?.* It is not obvious why we require a miner to first add a space commitment $(pk, \gamma)$ 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 [13] have the property that one can take a correctly constructed commitment $(pk, \gamma_0)$, and by making minor changes turn it into many other commitments $(pk, \gamma_1), (pk, \gamma_2), \ldots$ that can reuse almost all space, while it is still possible to answer almost all challenges correctly.[16] 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.

**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 $\sigma_j$ be the $j$th block of the signature chain in the current chain, and $pk_j$ the public key of the miner who added the $j^{th}$ block. If $m \neq \sigma_{j-1}$ and $\mathsf{SigVerify}(pk_j, m, \alpha) = 1$, that is, $pk_j$ was used to sign another message than $\sigma_{j-1}$ with index $j-1$, then half the reward and transaction fees that went to $pk_j$ for adding the $j^{th}$ block are transferred 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).

## VII. Instantiation

In this section we describe the concrete steps required for setting up, mining and paying in Spacemint. We give the instantiation for the second scheme (challenge from the past), outlined in Section V-C. 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 $\kappa$ to be used for the signature and PoSpace scheme. Moreover, we must specify parameters and functions:

- $\mathsf{time} \in \mathbb{N}$ specifies the length of a timeslot in minutes. It should be sufficiently larger than the network propagation time, but otherwise as small as possible. $\mathsf{time} = 1$ seems like a reasonable choice here.
- $\delta \in \mathbb{N}$ specifies that the challenge for block $i$ is a function of block $i - \delta$. A reasonable value is $\delta = 120$.
- Reward is a function such that $\mathsf{Reward}(i)$ specifies the amount of coins a miner gets for mining the $i^{th}$ block.
- Quality is function that takes as input a space commitment $(pk, \gamma)$ for space of size $N$ together with 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 Quality function outputs $-\infty$. Otherwise the output is (with $D_N$ as defined in Section V-B):

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

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 $\phi_j$ 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 $j$th 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 $j^{th}$ block was mined. With this definition, a natural measure for the quality of the chain would be simply the sum[17] $\sum_{j=1}^{i} N(v_j)$. 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 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 is 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:

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

[16] $\gamma_0$ is a Merkle-hash of all the labels in a hard to pebble graph. We can change $\gamma_0$ 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.

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

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 $\phi_0$ contains only some random string,[18] and the signature block $\sigma_0$ 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.** In order to dedicate $N$ bits of storage for mining, a party generates an identity and a space commitment

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

It stores $S_\gamma$ (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[19] 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 $S_\gamma$ and the corresponding commitment $(pk, \gamma)$ has been added to some transcription block $\tau_j, j \leq i-1$ in this chain. Next, the miner computes the randomness for the challenge sampling by hashing the hash block that is $\delta$ blocks in the past

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

From this $c$ we then compute the challenges $c_p, c_{cv}$. The miner computes the PoSpace answer

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

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 $\phi_i = (i, \mathsf{hash}(\phi_{i-1}), p_i)$, where $p_i$ is [20] $(pk, \gamma, c, j, q, a, a_{cv})$, where

$$a_{cv} := \mathsf{Answer}(pk, S_\gamma, c_{cv})\ .$$

is the output of the commitment verification (as discussed in Section V-A, for efficiency reasons we only execute commitment verification at this point).

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 $\tau_i$. It then computes the signature block $\sigma_i = (\mathsf{Sign}(sk, \sigma_{i-1}), \mathsf{Sign}(sk, \tau_i))$ and publishes block $\beta_i = (\phi_i, \sigma_i, \tau_i)$, hoping that it will end up in the block chain, earning the miner $\mathsf{Reward}(i)$ coins, plus the transactions fees of the transactions in $\tau_i$.

**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 VI-B.

[18] 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.

[19] 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.

[20] The index $j$ of the block where the space commitment was added and the quality $q$ are redundant, but they simplify verifying the proof.

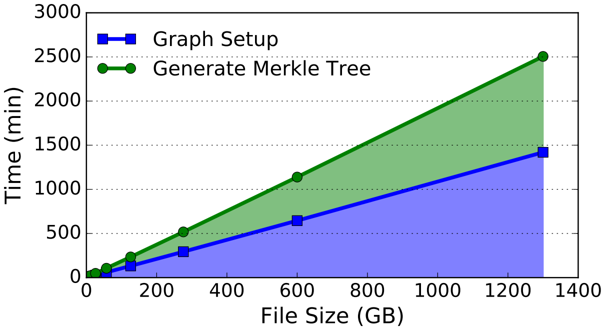

Fig. 3: Time to initialize the space.

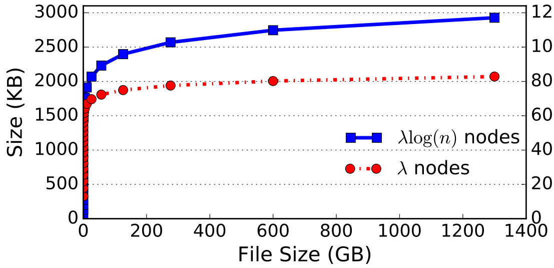

Fig. 4: Size of the proof for $\lambda = 30$ and varying size of the graph. The y-axis on the left and right represents the size for opening $\lambda \log(n)$ and $\lambda$ nodes respectively.

## VIII. Evaluation

To evaluate Spacemint, we have implemented a prototype in Go, using SHA3 in 256-bit mode as the hash function. The prototype uses the graphs from [24], and forces a cheating prover to store at least $\Omega(N/\log(N))$ bits in order to efficiently generate proofs. Given that the network infrastructure is very similar to Bitcoin, we are mainly interested in three quantities: time to initialize the space (graph), size of the proof, and time to generate and verify the proof. The experiments were conducted on a server equipped with an Intel i5-4690K Haswell CPU and 8 GB of memory. We used an off-the-shelf hard disk drive, with 2 TB of capacity and 64 MB of cache.

Our proof-of-space library and Spacemint prototype are available at: https://github.com/kwonalbert/spacemint.

**Time to Initialize.** To start mining Spacemint, the clients must first initialize their space, as described in I-B. Concretely, this involves computing all the hashes of the nodes, and computing the Merkle tree over the hashes. In Figure 3, we show the initialization time for spaces of size 8 KB to 1.3 TB. As expected the time to initialize grows linearly with the size of the space; at 1.3 TB, it takes approximately 41 hours to commit the graph. While expensive, we note that this procedure is done only once when the miner first joins the Spacemint network, and will use the initialized space over and over again. In fact, we require space initialization to non-trivial time, because an extremely fast space initialization would make re-using the same space for different commitments a viable strategy (Section V-C).

**Size of the Proof.** A proof (i.e., a full solution to the puzzle) in Spacemint consists of the hashes of the challenge nodes in the graph and their parents, and the Merkle inclusion proofs

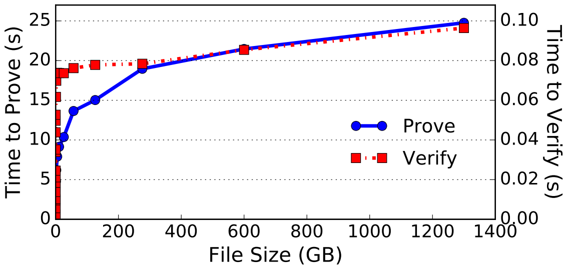

Fig. 5: Time it takes for a potential winner to prove and verify proof-of-space when $\lambda \log(n)$ nodes are opened for $\lambda = 30$.

for those nodes. For the simpler PoSpace that we implemented, the number of nodes we have to open is $\lambda \cdot \log(n)$, where $\lambda$ is a statistical security parameter. Since each opening is of size $\log(n) \cdot 32$ bytes, the overall proof size is upper-bounded $3 \cdot \lambda \cdot \log^2(n) \cdot 32$ bytes as each node has at most 2 parents in this PoSpace graph. We note that the proof sizes shown here are known to be sufficient by the proofs in [13], but we are unaware of any concrete attacks. In practice, we believe that opening fewer nodes may already be sufficient. Figure 4 demonstrates the size of the proof when we open $\lambda \log(n)$ nodes vs. just $\lambda$ nodes for $\lambda = 30$. We believe that the size of a sufficiently secure proof will lie somewhere in between, closer to opening $\lambda$ nodes.

**Time to Generate/Verify the Proof.** Unlike Bitcoin, generating an answer for a puzzle (i.e., generating a proof-of-space) takes little time. In Bitcoin, the miner is expected to work through most of the epoch in an attempt to find a preimage of a hash with sufficient difficulty. In Spacemint, assuming a miner is storing the space correctly, the miner needs to only perform $O(1)$ look-ups in the disk to find their solution (Section V-A), which takes fraction of a second. For instance, it takes $< 1$ ms to read a single hash from the disk. Only if the miner believes its answer is of very good quality will it generate the full proof, but even this takes seconds, not minutes.

As outlined above, our proofs are substantially bigger than Bitcoin's, and require more than just one hash evaluation to verify. However, for an active currency, we can still expect the size and verification time for the proofs added with every block to be marginal compared to the size of the transaction added with every block, and the time required to verify that the transactions are consistent. Figure 5 indeed shows that though it may take seconds to generate the proof, verification takes a fraction of a second.

**Energy.** Though our prototype was evaluated using a full CPU which wastes a lot of energy, one could in principal run the prover and the verifier on a energy-efficient devices such as Raspberry Pi [3]. An efficient microcontroller consumes less than 10 W of power, and most miners will only open one node per time-step since the quality of their answers will likely be bad. To get an upper bound on the power requirement, let us assume that there are 100,000 miners, each with 1 TB of space, and about 1% of the miners mine "good" answers which they will want to generate a full answer. Then we have

$$10\text{W} \cdot 100000 \cdot 0.01s + 10\text{W} \cdot 1000 \cdot 20s = 210000\text{J/block}$$

which translates to 210 kJ/min if we add one block a minute. In contrast, Bitcoin on average uses 100 MW, so it consumes 6 GJ/min, which is several orders of magnitude larger. We note that this 1% figure is a very conservative bound, so the difference could be even larger in practice.

## IX. Discussion

We now discuss some minor issues and how to resolve them.

**DoS.** A party who wants to mine must have its space commitment $(pk, \gamma)$ 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 $(pk, \gamma)$ 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 $c = \mathsf{hash}(pk, \gamma)$. 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 $(pk, \gamma)$ will not be added to the chain if some commitment $(pk, \gamma')$ is already in the chain. As the PoSpace scheme from [13] 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 $(pk, \gamma)$ is independent from the random oracles used for any other commitments. This implies that space cannot be re-used for 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.

## X. Game theory of Spacemint

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, Spacemint mining is modeled by the following $n$-player strategic game. Game-play 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.

### A. Game-theoretic preliminaries

The standard game-theoretic notion for a strategic game which occurs over multiple time steps (rather than in "one shot") is the *extensive game*. In order to accurately model the probabilistic aspects of the Spacemint 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 which behaves according to a known probability distribution.

In the Spacemint 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.

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 X.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 $(a_1, \dots, a_K) \in H$ then for all $L \le K$, it holds that $(a_1, \dots, a_L) \in H$.*

  *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.*
- *$\vec{\mathcal{I}} = (\mathcal{I}_1, \dots, \mathcal{I}_N)$, 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 $I$.*
- *$\vec{u} = (u_1, \dots, u_N)$, where each $u_i : Z \to \mathbb{R}$ is the utility function of player $i$.*

**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 Spacemint, 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, Spacemint is a game of *imperfect information*.

The information that players do not know about other players' actions is modeled by the partitions $\vec{\mathcal{I}} = (\mathcal{I}_1, \dots, \mathcal{I}_N)$ in Definition X.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 X.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 6 shows the game tree, with player 2's information sets as dashed red boxes: within each dotted box, player 2 cannot tell which history he is at.

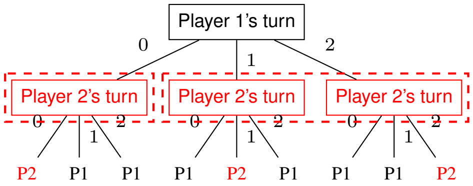

Fig. 6: Game tree for the "Match my number" game. Leaves are labelled with the winning player.

**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 X.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 $\alpha_i$ 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 $\alpha$ 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,

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

**Equilibrium notions.** The most widely known equilibrium concept for a strategic game is the Nash equilibrium [23], given in Definition X.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 $\alpha'_i$ and all other players' actions are as in $\vec{\alpha}$.

**Definition X.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 $\Gamma$ *if for every player $i \in [N]$ and every strategy $\alpha'_i$ 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 X.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.

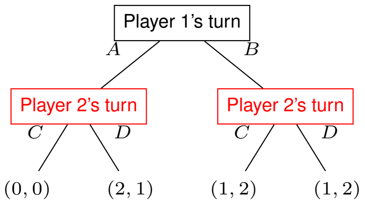

Fig. 7: Game tree for the "Unstable" game.

It is a Nash equilibrium of this game for player 1 to choose $B$, and player 2 to always choose $C$[21] However, the strategy profile $(B, C)$ seems "unstable"[22], 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 X.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 $\mu_i$ is a function that assigns to every information set in $\mathcal{I}_i$ a probability measure on histories in the information set.*

In Definition X.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 $(\alpha, \mu)$ 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 X.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 $\alpha'_i$ of player $i$, for every information set $I \in \mathcal{I}_i$, it holds that*

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

Definition X.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 X.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.*

[21] It is straightforward to verify that this is an equilibrium, by considering the payoff matrix of the game.

[22] In this example, we assume that the game is with perfect information.

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

**Definition X.9** (Sequentially rational Nash equilibrium)**.** *An assessment is a* sequentially rational Nash equilibrium *if it is sequentially rational and consistent.*

### *B. Game-theoretic analysis of Spacemint*

In order to analyze the game-theoretic properties of Spacemint mining, we define an extensive game, $\mathsf{SpacemintGame}$, 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 Spacemint miners. Concretely:

- We do not include the action of *creating a space commitment* because (as discussed in Section V-A under "Mining") we can assume that rational miners will commit to all the space they have, and nothing else.[23]
- 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 Spacemint 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[24]. 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 X.10** (The Spacemint Game)**.** *Let $\mathcal{B}$ denote set of all* blocks *as defined in Section VI. 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* $\mathsf{SpacemintGame}_{\Pi,K,\rho} = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$ *as follows:*

- *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 $a = (a_1, \dots, a_N, a_{\mathcal{C}})$ 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.*
  - $[(h, ((S_1, T_1), \dots, (S_N, T_N), a_{\mathcal{C}}))]_i =$

    $$\{(h', ((S_1', T_1'), \dots, (S_N', T_N'), a_{\mathcal{C}}')) \in H : \\ h \sim_i h' \wedge S_i = S_i' \wedge T_i = T_i' \wedge a_{\mathcal{C}} = a_{\mathcal{C}}' \\ \wedge \forall j \neq i,\ T_j = T_j' \} ,$$

    *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.*
- *$\vec{u} = (u_1, \dots, u_N)$, 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 $j$th 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 VII. We define*

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

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

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

  *Let* $\mathsf{blocks}(h)$ *denote the sequence of "winning blocks" at each time step in the game, defined inductively:*
  - $\mathsf{blocks}(()) = ()$
  - $\mathsf{blocks}(h = (h', ((S_1, T_1), \dots, (S_N, T_N), a_{\mathcal{C}}))) =$ $\arg\max_{B \in T} (\mathsf{Quality}(B, \mathsf{beac}_{|h|}(h)))$, *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*[25].

  *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:*

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

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

[25] 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.

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

*Let* $\mathsf{winners}_j(h)$ *denote the jth winner in the sequence* $\mathsf{winners}(h)$*. Let* $\mathsf{onlyone}_j(i,h)$ *be an indicator variable for the event that player i's jth 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*[26]*. 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 X.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 $c$ (see Section VII for exact block format).

**Theorem X.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* $\mathsf{SpacemintGame}_{\Pi,K,\rho}$*, 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* $I$[27]*. 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* $\mathsf{SpacemintGame}_{\Pi,K,\rho}$*.*

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

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

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 $\alpha_i'$ 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 $\alpha_i'$ 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 $\alpha_i'$ of player $i$. The theorem follows. □

**Theorem X.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* $\mathsf{SpacemintGame}_{\Pi,K,\rho}$ *where:*

- $\vec{\alpha}$ *and* $\hat{\alpha}_i$ *are defined as in Theorem X.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* $\mathsf{SpacemintGame}_{\Pi,K,\rho}$*.*

*Proof.* Let $I \in \mathcal{I}_i$ be any information set of player $i$ in $\mathsf{SpacemintGame}_{\Pi,K,\rho}$, and let $L$ be the length of histories in $I$. It follows from Definition X.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 $\mathsf{SpacemintGame}_{\Pi,K-L,\rho}$. Since winners, 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 X.11, $\vec{\alpha}|_{K-L}$ is a Nash equilibrium of $\mathsf{SpacemintGame}_{\Pi,K-L,\rho}$. 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 $\alpha_i'$ of player $i$. Applying this argument for every $I$, we conclude that $(\vec{\alpha}, \vec{\mu})$ is sequentially rational in $\mathsf{SpacemintGame}_{\Pi,K,\rho}$.

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 Spacemint 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 X.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[28]. The crucial observation is that even

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

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

[28] 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 X.12.

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 [20] 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.

## XI. Conclusion

We have presented Spacemint, a cryptocurrency that uses efficient proofs of space instead of energy-intensive proofs of work to maintain a public ledger of all transactions. We have described a variant of a proof-of-space protocol that is more suitable for cryptocurrencies, and modified the structure of the hash chain and transactions to address some of the issues of other cryptocurrencies. We have also demonstrated the feasibility of Spacemint through a prototype, and show that maintaining a public ledger could be much more efficient with proof-of-space. Finally, we do a game-theoretic analysis of Spacemint modeled as an extensive game, and prove that it satisfies strong equilibrium properties.

## Acknowledgements

We thank Ethan Heilman for an interesting discussion about costs of modern Bitcoin mining.

## References

[1] Bitcoin network graphs. http://bitcoin.sipa.be/index.html.

[2] Burstcoin. http://burstcoin.info.

[3] Raspberry pi. www.raspberrypi.org.

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

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

[6] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi. Proofs of space: When space is of the essence. In M. Abdalla and R. D. Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 538–557. Springer, Sept. 2014.

[7] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 598–609. ACM Press, Oct. 2007.

[8] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In *CCSW*, pages 43–54, 2009.

[9] H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 857–866, 2014.

[10] R. Di Pietro, L. Mancini, Y. W. 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.

[11] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Aug. 1993.

[12] S. Dziembowski. Proofs of space and a greener bitcoin, June 2013. Presentation at Workshop on Leakage, Tampering and Viruses, Warsaw. https://sites.google.com/site/warsawcryptoworkshop2013/abstracts.

[13] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In *CRYPTO 2015*, 2015.

[14] S. Dziembowski, T. Kazana, and D. Wichs. One-time computable self-erasing functions. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 125–143. Springer, Mar. 2011.

[15] P. Golle, S. Jarecki, and I. Mironov. Cryptographic primitives enforcing communication and storage complexity. In M. Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135. Springer, Mar. 2003.

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

[17] A. Juels and B. S. Kaliski Jr. Pors: proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 584–597. ACM Press, Oct. 2007.

[18] N. P. Karvelas and A. 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.

[19] S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.

[20] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Workshop on the Economics of Information Security*, June 2013.

[21] A. Miller, A. Juels, E. Shi, B. Parno, and J. 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.

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

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

[24] W. J. Paul, R. E. Tarjan, and J. R. Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, 1976–1977.

[25] D. Perito and G. Tsudik. Secure code update for embedded devices via proofs of secure erasure. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 643–662. Springer, Sept. 2010.

[26] S. Valfells and J. H. Egilsson. Minting money with megawatts: How to mine bitcoin profitably, September 2015. http://www.researchgate.net/publication/278027487_Minting_Money_With_Megawatts_-_How_to_Mine_Bitcoin_Profitably.

## Appendix

### A. Proof-of-Space Parameters

The two PoSpace constructed in [13] have the following efficiency/security properties. Below $t_{\mathsf{hash}}$ denotes the time required to evaluate the underlying hash function $\mathsf{hash} : \{0,1\}^* \to \{0,1\}^L$ on inputs of length $2L$ (to hash an input of length $m \cdot L$ takes time $m \cdot t_{\mathsf{hash}}$ by using Merkle-Damgård), For a given $n$, the number of nodes of the underlying graph, an honest prover $\mathcal{P}$ must dedicate

$$N = 2 \cdot n \cdot L$$

bits of storage ($L \cdot n$ for the labels, and almost the same for the values required to efficiently open the Merkle tree commitment).

**Proposition A.1** ([13] first construction)**.** *There exists a PoSpace in the random oracle model with the following properties:*

- **Efficiency:** *The verifier runs in time $O(L)$ during initialization (it just has to send a nonce and store a commitment) and $O(k \cdot \log(n) \cdot \log\log(n) \cdot t_{\mathsf{hash}})$ during execution (it must check $O(k \cdot \log\log(n))$ openings of the Merkle tree commitment, the parameter $k$ is discussed below). The (honest) prover runs in time $O(n \cdot \log\log(n) \cdot t_{\mathsf{hash}})$ during initialization and in $O(k \cdot \log(n) \cdot \log\log(n) \cdot t_{\mathsf{hash}})$ during execution.*
- **Security:** *Let $k_{cv}, k_p$ denote the parameter $k$ we set for the proof execution and commitment verification phase. If a (potentially cheating) prover $\mathcal{P}$ passes the commitment verification phase, then with probability $1 - 2^{\Theta(k_{cv})}$ the following holds: If $\mathcal{P}$ can make $\mathcal{V}$ accept in in the proof execution phase with probability $\geq 2^{-\Theta(k_p)}$, then $\mathcal{P}$ either stores $\Theta(N)$ bits (i.e., almost as much as an honest prover) or runs in time $\Theta(t_{\mathsf{hash}} \cdot n \cdot \log\log(n))$ (i.e., the time required for initialization).*

To use the above PoSpace in our construction, we'll have to set $k_{cv} = \lambda$ where $\lambda$ is a statistical security parameter, and $k_p = \Theta(1)$ can be a constant.

**Proposition A.2** ([13] second construction)**.** *There exists a PoSpace in the random oracle model with the following properties:*

- **Efficiency:** *The verifier runs in time $O(L)$ during initialization and in $O(k \cdot \log(n) \cdot t_{\mathsf{hash}})$ during execution. The (honest) prover runs in time $O(n \cdot t_{\mathsf{hash}})$ during initialization and in $O(k \cdot \log(n) \cdot t_{\mathsf{hash}})$ during execution.*
- **Security:** *Let $k_{cv}, k_p$ denote the parameter $k$ we set for the proof execution and commitment verification phase. If a (potentially cheating) prover passes the commitment verification phase, then with probability $1 - 2^{\Theta(-k_{cv}/\log(n))}$ the following holds: If $\mathcal{P}$ can make $\mathcal{V}$ accept in in the proof execution phase with probability $\geq 2^{-\Theta(k_p)}$, then $\mathcal{P}$ either stores $\Omega(nL/\log(n)) = \Omega(N/\log(n))$ bits or requires $\Omega(N/\log(n))$ space and $\Omega(t_{\mathsf{hash}} \cdot n/\log(n))$ time during execution.*

To use the above PoSpace in our construction, we'll have to set $k_{cv} = \lambda \cdot \log(n)$ where $\lambda$ is a statistical security parameter, and $k_p = \Theta(1)$ can be a constant.

### B. Burstcoin

In this section we give some more details on the efficiency and security issues of Burstcoin as outlined in Section II. We not only discuss Burstcoin because it is relevant related work, but also, looking at its design illustrates some of the challenges that we had to solve when designing a Proof of Space based cryptocurrency.

The only specification of the Burstcoin mining process that we were able to find is the webpage http://burstcoin.info/intro, which unfortunately is rather informal. The description below is thus only our best guess on how exactly the mining process in Burstcoin works, mostly based on the figure http://burstcoin.info/assets/img/flow.png.

Burstcoin uses the Shabal256 hash function, which below we will denote with $H(\cdot)$. To mine Burstcoin, a miner first initialises his disk space as follows: he picks a nonce $\mu$ and an account identifier (which is a hash of a public key) $Id$, and then computes iteratively $4096$ values $x_0, x_1, \ldots \in \{0,1\}^{256}$ as

$$x_0 = H(Id, \mu) \quad \text{and} \tag{4}$$

$$x_{i+1} = H(x_i \| x_{i-1} \| \ldots \| x_0) \quad \text{for} \quad i = 0, \ldots, 4095\,. \tag{5}$$

The miner then stores $s_0, \ldots, s_{4095}$ where $s_i = x_i \oplus x_{4096}$. Each block $s_i$ is called a "scoop", and the $4096$ scoops together are called a "plot". The miner is supposed to store as many plots as he can (using different nonces) until all the dedicated space is filled. To compute a plot, one must hash $4096 \cdot \frac{1+4096}{2} \approx 8$ million 256-bit blocks[29]. In the following we assume for simplicity that there is just one plot $s_0, \ldots, s_{4095}$.

**Efficiency.** Once every few minutes, a new block gets added to the hash-chain. At this point the miner can compute a designated (public) index $i \in \{0, \ldots, 4095\}$ and must look up the value $s_i$. This $s_i$ then determines if the miner "wins" and thus can add the next block to the block chain[30]. Note that this requires accessing a constant fraction of the entire dedicated disk space (i.e. one block per plot, or $0.024\%$), every time a new block gets mined. Moreover, in order to verify that a miner "won" and can add a block, it is necessary to recompute the entire plot from the initial inputs $(Id, \mu)$, which, as mentioned above, involves hashing over $8 \cdot 10^6$ blocks. In comparison, in Spacemint, the number of bits read from the disk is only logarithmic in the size of the dedicated space, and verification also just requires a logarithmic number of hashes. (In Bitcoin, verification requires just a single hash.)

**Time-memory trade-offs.** We observe that Burstcoin allows for a simple time-memory trade-off: instead of storing an entire plot $s_0, \ldots, s_{4095}$, a miner can initially compute and store only the value $x_{4096}$. The miner then re-computes the required scoop $s_i$ at a given time-step, but only if $i$ is sufficiently small (say, $i \leq 10$). This would require hashing only at most 50 blocks[31]. Thus, the miner will get a shot at adding a block only at $10/4095 \approx 0.25\%$ of the time slots, but now also only requires a $1/4095 \approx 0.025\%$ fraction of the

[29] Note that in equation (4), a freshly computed block $x_i$ is prepended to the previous input. This is important as Shabal256 is an iterated hash function: appending instead of prepending would bring the number of hashes required to compute a plot down to linear (instead of quadratic) in the length of the plot, but at the same time would allow for much more dramatic time-memory trade-offs than the ones outlined below.

[30] The details of how to add a block to the chain are irrelevant for this discussion, and hence we omit them.

[31] To be precise, the miner computes $x_0, \ldots, x_i$ and sets $s_i = x_i \oplus x_{4096}$.

space that would be needed to store an entire plot. Using this strategy, given some fixed amount of disk-space, it is possible to mine $0.25/0.025 = 10$ times faster than the honest mining algorithm, at the price of having to compute a modest number of extra hashes. More generally, using this type of mining strategy, it is possible to mine $t$ times faster at the price of having to hash $t^2/2$ blocks with every block read from the disk.

Given that application-specific integrated circuits (ASICs) can compute in the order of millions of hashes per second per dollar invested[32], such time-memory trade-offs seem practical[33]. We remark that in http://burstcoin.info/intro, the creators of Burstcoin discuss the possibility of mining their currency in a pure proof-of-work style, though they come to a different conclusion from ours:

> *Technically, this mining process can be mined POW-style, however mining it as intended will yield thousands of times the hashrate, and your hardware will sit idle most of the time. Continuously hashing until a block is found is unnecessary, as waiting long enough will cause any nonce to eventually become valid.*

**Grinding and Extending Multiple Chains.** The two main challenges we had to overcome when designing Spacemint were attacks based on grinding and mining multiple chains. (The problem with time-memory trade-offs was solved in the Proofs of Space [13] paper upon which this work builds.)

Due to lack of documentation of the Burstcoin mining process, we do not know to what extent Burstcoin can be attacked using grinding or by extending multiple chains. From our understanding of the Burstcoin mining process, it seems especially crucial to avoid grinding of the index of the scoop to be used in a given round: otherwise, a malicious miner could "hijack" the chain forever (i.e. mine all future blocks) using only a very small fraction of the total dedicated space, as follows. The figure http://burstcoin.info/assets/img/flow.png indicates that this scoop index is computed from two values $\mathsf{PrevGenSig}$ and $\mathsf{PrevBlkGenerator}$. The naming indicates that $\mathsf{PrevGenSig}$ corresponds to the value $\mathsf{NewGenSig}$ used in the previous block. This value is computed deterministically and thus is "ungrindable". We were not able to find details on the functionality of $\mathsf{PrevBlkGenerator}$ so do not know whether it can be grinded; however, it seems possible that this value serves to bind transactions to proofs within a given block, and thus can be grinded (by trying different sets of transactions to include in a block).

[32] https://en.bitcoin.it/wiki/Mining_hardware_comparison

[33] However, we remark that currently, ASICs exist primarily for the SHA256 hash function used in Bitcoin (and not for the more unconventional Shabal256 hash used in Burstcoin).