SpaceMint: A Cryptocurrency Based on Proofs of Space*

Sunoo Park† Albert Kwon† Joël Alwen‡

Georg Fuchsbauer‡ Peter Gaži‡ Krzysztof Pietrzak‡

†*MIT* ‡*IST Austria*

Abstract

Since its introduction in 2009, Bitcoin has become the most successful cryptocurrency ever deployed. However, the currency's dramatic expansion has also raised serious concerns about its long-term sustainability: (1) Bitcoin mining dynamics have shifted away from decentralization, as dedicated hardware and entry of governments and energy producers has gradually placed most mining power in the hands of a select few, and (2) the network's growth spurt has come with accompanying, vast amounts of energy constantly "wasted" solely for the purpose of sustaining the currency.

In this work, we propose SpaceMint, a cryptocurrency based on *proofs of space* instead of wasteful proofs of work. Mining in SpaceMint is designed to have low setup and overhead costs, yielding a fairer reward structure for small and large miners. Miners in SpaceMint dedicate *disk space* rather than computation. In our design, we adapt proof-of-space for the cryptocurrency setting, and propose a new block chain format and transaction types that prevent attacks that exploit the inexpensiveness of mining (from which alternative non-proof-of-work-based proposals have suffered). Our prototype shows that initializing 1 TB for mining takes about a day (a one-off cost), and miners on average spend just a fraction of a second per block mined. We also provide a formal game-theoretic analysis modeling SpaceMint as an extensive game, and prove that following the protocol is an equilibrium, thereby arguing for the currency's stability and consensus.

1 Introduction

E-cash was first proposed by Chaum [12] in 1983, but did not see mainstream interest and deployment until the advent of Bitcoin [32] in 2009. Today, with a market cap of over 6 billion US dollars, Bitcoin has given a sweeping, unprecedented demonstration that the time is ripe for cryptocurrencies. Bitcoin moves millions of dollars in transactions per day, inspired hundreds of "altcoins" and start-ups [4, 2], and receives regular popular press coverage (e.g., [7, 8]).

Widespread deployment of cryptocurrencies has the potential to transform the traditionally corporate, centralized structure of financial systems; and to facilitate secure micro-payments and micro-loans reaching rural and developing areas, to small-scale artists and content creators, and beyond [39, 21]. The block chain structure underlying Bitcoin is considered a promising tool in areas as diverse as insurance, stock markets, and land registry [29, 38]. Moreover, amid concerns about mass surveillance of individuals, cryptocurrencies may be the financial system most amenable to respecting civil rights.

However, Bitcoin's dramatic expansion has provoked serious questions about the currency's long-term sustainability, including particular concerns over the emergence of a "mining oligarchy" controlled by a handful of powerful (corporate or governmental)

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

entities, and over the steeply increasing amount of resources being consumed by the Bitcoin network.

Both of these problems spawn from Bitcoin's block chain, which provides its mechanism of keeping a public ledger of transactions. In Bitcoin, members of the network are rewarded for adding blocks containing transaction information to the block chain. An essential property of the chain is that all parties in the network are guaranteed to eventually agree on the same chain (i.e., reach *consensus*). Miners in Bitcoin add blocks by solving computationally difficult puzzles.[1] Accordingly, a Bitcoin block is considered to be a *proof of work* [15]: i.e., a proof that a certain amount of computational resources was invested.

One of the original ideas behind basing Bitcoin mining on computational power was that anyone could participate in the network by dedicating their spare CPU cycles, which incurs little cost since it uses the idle time of already-existing personal computers. However, modern Bitcoin mining dynamics have become starkly different [40]: the network's mining clout is concentrated in large-scale mining farms, often in collaboration with electricity producers. Currently, mining with your spare CPU cycles will result in net *loss*, due to electricity costs: newcomers must make a rather substantial initial investment in hardware, usually in the form of dedicated ASICs, to enter the game. This phenomenon is sometimes known as the mining oligarchy, and it undermines much of the motivation (stability and security) behind the decentralized system design.

Bitcoin also depletes large amounts of natural resources. The Bitcoin network constantly consumes electricity at a massive scale, in the order of hundreds of megawatts [34], as it mines a block every 10 minutes or so. Moreover, most mining is currently done by specialized ASICs, which have no use beyond mining Bitcoins. For these reasons, Bitcoin is considered an "environmental disaster" [22] by some.

To address these issues, we propose SpaceMint, a cryptocurrency that replaces the costly proofs of work underlying Bitcoin with *proofs of space* (PoSpace) [17]. In SpaceMint, in order to mine blocks (and thereby mint coins and confirm transactions), miners must invest *disk space* rather than computational power, and prove to the network that they are dedicating certain amounts of disk space.

[1]Currently, mining a Bitcoin block requires about 2^{65} hash computations [1].

In SpaceMint, miners who dedicate more disk space have a proportionally higher expectation of successfully mining a block and reaping the reward. It is therefore 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 three key differences:

1. In SpaceMint, the investment is in the form of capital expenditure, and the mining process after hard-drive initialization incurs negligible ongoing monetary and natural-resource cost. In contrast, in Bitcoin, the mining process requires perpetual energy expenditure.
2. In Bitcoin, resources are "used up" by mining: electricity is a *depletable* resource which once used is gone; and Bitcoin mining hardware is a 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 again and again, and *multi-purpose*, since hard drives have intrinsic value in their ability to store useful data.[2]
3. Ordinary people have (many) personal devices with unused disk space available, which can be repurposed for SpaceMint mining with very low set-up and maintenance costs. Since mining is cheap, and even small players get a proportional fair share of rewards, we argue that we can expect large amounts of space in the network and a more distributed, decentralized miner body.

[40] analyzed mining profit vs. invested resources in modern Bitcoin mining, as shown in Figure 1. We have added predicted cost and profit curves of SpaceMint mining, based on our reasoning above.

1.1 Challenges and our contributions

When replacing PoW with PoSpace, there are two important challenges.

[2]In § 6, we argue that specialized storage devices tailored for SpaceMint mining are unlikely to arise.

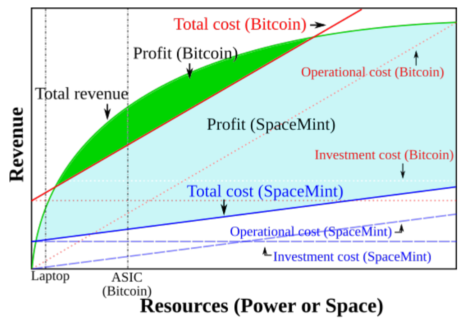

Figure 1: Mining profit vs. invested resources

- *Interactive PoSpace*: Proof-of-space, as proposed in [17], is an interactive protocol between a prover and a verifier, and thus unsuitable for cryptocurrency settings where each individual in a decentralized network should be able to behave as the verifier instead of a single entity.
- *Nothing-at-stake problems*: When replacing PoW with a different type of proof that is computationally easy to generate (such as PoSpace), a series of problems arise which are collectively known as *nothing-at-stake problems* [19]. Intuitively, because mining is cheap, miners can (1) mine on multiple chains, and (2) try multiple blocks per chain, at very little additional cost.[3] These two problems potentially allow for double-spending attacks and slow down consensus (details in §4).

SpaceMint tackles both of these issues by (1) modifying PoSpace from [17] to a non-interactive variant, and (2) introducing a new block chain structure and transaction types that prevent miners from taking advantage of the nothing-at-stake problems.

We also implement and evaluate the modified PoSpace to demonstrate the effectiveness of our scheme. Even for space larger than 1 TB, we show that (1) miners need less than a second to check if they are likely to "win" and therefore should try generating the next block, (2) block generation takes less than 30 seconds, and (3) verifying the validity of a block takes a fraction of a second. Moreover, these numbers grow very slowly (logarithmically) with larger space.

Finally, we provide a game-theoretic analysis of SpaceMint modeled as an extensive game, showing that adhering to the protocol is an equilibrium for rational miners (that is, cheating does not pay off), and we thereby argue the stability and consensus of SpaceMint. To our knowledge, this is the first analysis of a cryptocurrency formally modeled as an extensive game, with rigorous proofs of equilibrium stability.[4] Our game-theoretic model could also be adapted to cover other similar cryptocurrencies.

Summary. We believe that designing an energy-efficient cryptocurrency with a reward structure that incentivizes decentralization and participation will be instrumental to realizing the future potential of cryptocurrencies. Our contributions are:

- *Cryptocurrency from proofs of space:* We modify PoSpace [17] for the cryptocurrency setting and design SpaceMint, a cryptocurrency based purely on proofs of space, which avoids the major drawbacks of existing proof-of-work-based schemes (§3 and §4).
- *Addressing the "nothing-at-stake" problem:* We propose novel approaches to solving nothing-at-stake problems that arise in non-proof-of-work-based cryptocurrencies. Our solutions can be extended to other cryptocurrencies based on easy-to-generate proofs (§4).
- *Evaluation of proof of space:* We evaluate our modified PoSpace in terms of time to initialize space, time to generate and verify blocks, and block size (§5).
- *Game theory of SpaceMint:* We model SpaceMint as an extensive game, and prove that adhering to the protocol is an ε*-sequential Nash equilibrium.* Our analysis holds even against "selfish-mining" attacks [20] and our extended analysis also models network unreliability and clock asynchrony between players (§7).

[3]A proof-of-work-based scheme inherently disincentivizes these behaviors.

[4]Prior work on equilibria in Bitcoin has given only an informal treatment of the problem: notably, [28] presents a detailed informal discussion of equilibrium strategies in Bitcoin.

2 Related Work

Proofs of stake (PoStake). In proof-of-stake-based cryptocurrencies, the probability that a party mines the next block corresponds to the fraction of coins (out of all coins ever minted) that it holds. While such proofs require minimal physical resources to generate, PoStake-based cryptocurrency proposals to date have been susceptible to nothing-at-stake attacks. Moreover, a large fraction of coin owners must actively participate in mining in order to ensure security, and this has posed a significant challenge to PoStake currencies: even in the most popular PoStake-based currency, Peercoin [27], participation has fallen below 10%. We note that while the participation problem is inherent for PoStake, it does not apply to PoSpace.

Proofs of secure erasure. *Proofs of secure erasure* (PoSE), introduced by Perito and Tsudik [36], are another type of proof system related to PoSpace. Informally, a PoSE allows a space-restricted prover to convince a verifier that it has erased its memory of size S. PoSE with small communication complexity have been constructed by [18, 26, 9]. In [9], a weaker variant of PoSE is considered[5], where a prover "succeeds" whenever he has access to a sufficient amount of space, but must not necessarily have erased it during protocol execution. A PoSpace implies a PoSE (simply run the initialization and execution phase sequentially), but the reverse implication is unknown. The only application of PoSE of which we are aware was proposed in [36]. In particular, PoSE cannot be used for any of the applications of PoSpace put forward in [17], nor for the cryptocurrency proposed in this paper. We refer to [17] for a more detailed discussion on PoSpace vs. PoSE.

Proofs of storage/retrievability. *Proofs of storage* and *proofs of retrievability* ([23, 11, 10, 25, 14] and many more) are yet other types of proof systems, with a different purpose from PoSpace. These are 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: an important property of PoSpace is that the verifier's computation / communication are at most polylogarithmic in the prover's storage size.

Permacoin. Permacoin [31] is a cryptocurrency proposal similar to Bitcoin, that makes use of proofs of retrievability to build a novel variant of proof of work. While Bitcoin's proofs of work consist of repeatedly solving computational "puzzles" that have no intrinsic value, Permacoin's idea is to make the puzzles serve a useful purpose: the miners are incentivized to *store useful data* in order to solve puzzles, and thus the network of miners can serve as a data archive. Permacoin is, however, still fundamentally a proof-of-work-based scheme, since miners must produce many proofs of retrievability to mine each block. In contrast, in SpaceMint the dedicated storage does not store anything useful, but we avoid proofs of work altogether. Unlike Permacoin, our main goal is to avoid the necessity of *perpetual computation* for mining: the only operation required to mine SpaceMint, after initialization, is a few disk accesses every few minutes.

Burstcoin. The only cryptocurrency of which we are aware that uses disk space as the primary mining resource is Burstcoin [3].[6] The most notable security issue with Burstcoin is *time-memory tradeoffs*: a miner doing just a little extra computation can mine at the same rate as an honest miner, while using just a small fraction (e.g., 10%) of the space.[7] In terms of efficiency, a major issue with Burstcoin is that a constant fraction (0.024%) of dedicated disk space must be read every time a block is mined. In contrast, SpaceMint requires accessing only logarithmically many blocks in the size of dedicated space. For instance, at 1TB, Burstcoin reads 24GB, while

[5]Note that [9] calls this "one-stage proof of space", which has led to some confusion with the stronger "proof of space" notion proposed earlier by [16] and first realized in [17]. In this work, "proof of space" always refers to the notion of [17].

[6]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 [16].

[7]Details of the attack are given in Appendix B.

SpaceMint reads < 24 MB. Finally, verification in Burstcoin is also problematic: a miner has to hash over 8 million blocks to verify another miner's claim.

3 Proof-of-Space in SpaceMint

In this section, we first discuss a simple approach that fails to achieve the desired properties, and present our variant of PoSpace for cryptocurrencies.

Storing a function table. A straw-man solution is to have the prover store a lookup table $(1, f(1)), \ldots, (N, f(N))$ of a random-looking function f (with a short description), sorted by the output. The prover's challenge is to invert the function on a value $f(x)$ for some random $x \in [N]$: an honest prover can do this in time $\log(N)$ by binary search. Unfortunately, this approach is not a proof of space due to time/memory trade-offs [24],[8] where a cheating prover only needs to store $N^{2/3}$ input/output pairs and still invert the function in time $N^{2/3}$.

Proof-of-space for SpaceMint. The PoSpace as described in [17] instead describes schemes that penalize a cheating prover more harshly. The space is generated using a special type of directed acyclic graphs called hard-to-pebble graphs. Each node i of the graph contains a hash

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

where $p_1, \ldots, p_t$ are the parents of node i. The goal of the prover is to prove that it is storing the graph (i.e., the hashes at every node). [17] suggests two graphs that can be used for PoSpace. With the first, the prover must either dedicate $\Omega(N/\log(N))$ or incur at least $\Omega(N/\log(N))$ space complexity (thus time) to generate a proof of space. With the second, the prover must either dedicate $\Omega(N)$ space or spend $\Theta(N)$ time generating a proof.

Formally, PoSpace described in [17] consists of four algorithms $\{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$, and is executed between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. PoSpace is carried out in two phases:

[8]Using a scheme that suffers from time-memory tradeoffs for building cryptocurrency would mean miners can increase their chance of a winning block by spending more computation, effectively yielding a proof-of-work-based scheme.

Algorithm 1 Space commit

Common input: A hard-to-pebble graph G with n nodes and a function $\mathsf{hash}\colon \{0,1\}^* \to \{0,1\}^L$.

1. $\mathcal{P}$ generates a unique nonce μ and then computes and stores $(\gamma, S_\gamma) := \mathsf{Init}(\mu, n)$, and sends the nonce[9] μ and the commitment γ to $\mathcal{V}$. S_γ contains the labels of all the nodes of G computed using Eq. (1). γ is a Merkle-tree commitment to these n labels. The total size of S_γ is $N = 2 \cdot n \cdot L$ (graph + Merkle tree).

1. Initialize
   (a) Space commit: $\mathcal{P}$ initializes its space using Init, and sends a commitment γ to $\mathcal{V}$.
   (b) Commitment verification: $\mathcal{V}$ and $\mathcal{P}$ interact to verify that γ commits to labels which satisfy Eq. (1). This is done by opening k_{cv} random labels together with the labels of their parents in the commitment γ.
2. Prove space: $\mathcal{V}$ and $\mathcal{P}$ interact to verify $\mathcal{P}$ is storing the space by opening k_p random labels.

To use PoSpace for a currency, we must make it non-interactive (since there is no designated verifier, and we want any member of the network to be able to verify). Since the verifier is public-coin, this means we just have to find a way to generate public randomness. Algorithms 1, 2, and 3 describe our non-interactive construction in detail.

We remark that proving space is much more efficient than commitment verification: for the former, it suffices to open $k_p = \Theta(1)$ labels (and $k_p = 1$ suffices), whereas for the latter, one must open at least security-parameter-many labels together with all their parents. In SpaceMint, typical miners will run only Prove space in most time-steps, and they will run the expensive Commitment verification in rare cases where they find such a high-quality block that they actually try generating a block to add to the chain.

[9]The nonce ensures that the same space cannot be used for two different proofs [17], and thus in a single-verifier setting we can let $\mathcal{P}$ generate the nounce.

Algorithm 2 Commitment verification

```
Initial state: $\mathcal{V}$ holds commitment $\gamma$ and nonce $\mu$; $\mathcal{P}$ stores $S_\gamma$ and $\mu$.
  1. $\mathcal{V}$ samples $(c_1, \dots, c_{k_{cv}}) \leftarrow \mathsf{Chal}(n, k_{cv}, \$)$ and sends these challenges to $\mathcal{P}$.
  2. $\mathcal{P}$ opens all the labels of the nodes $\{c_i\}_{i \in [k_{cv}]}$ and of all their parents and sends them to $\mathcal{V}$. This is done using $\mathsf{Ans}$ where $\mathsf{Ans}(\mu, S_\gamma, c)$ returns the Merkle inclusion proof of label $l_c$ w.r.t. $\gamma$.
  3. $\mathcal{V}$ verifies these openings using $\mathsf{Vrfy}$, where $\mathsf{Vrfy}(\mu, \gamma, c, a) = 1$ iff $a$ is a correct opening for $c$. It also checks for all $i = 1, \dots, k_{cv}$ if the label $l_{c_i}$ is correctly computed as in Eq. (1).
```

Algorithm 3 Prove space

```
Initial state: $\mathcal{V}$ holds commitment $\gamma$ and nonce $\mu$; $\mathcal{P}$ stores $S_\gamma$ and $\mu$.
  1. $\mathcal{V}$ samples $(c_1, \dots, c_{k_p}) \leftarrow \mathsf{Chal}(n, k_p, \$)$ and sends these challenges to $\mathcal{P}$.
  2. $\mathcal{P}$ opens all the labels of the nodes $\{c_i\}_{i \in [k_{cv}]}$ (using $\mathsf{Ans}$) and sends them to $\mathcal{V}$.
  3. $\mathcal{V}$ verifies these openings using $\mathsf{Vrfy}$.
```

4 SpaceMint Protocol

In this section, we describe the details of SpaceMint. We first describe some problems that arise when designing a cryptocurrency based on PoSpace, including the nothing-at-stake problems (§4.1). Next, we describe the SpaceMint protocol, and the details of block chain and transactions in SpaceMint (§§4.2–4.4). We then describe where the challenges for the miners come from (§4.5), and our solutions to the problems from §4.1 (§4.8). Finally, we explain how the quality of blocks and chains is determined (§§4.6–4.7), and discuss suggestions for concrete parameters (§4.9).

4.1 Design challenges

Nothing-at-stake: Mining multiple chains. In Bitcoin, rational miners will always work towards extending the longest known chain, as working on any other chain would lower the expected reward. When using PoSpace, however, mining is cheap, and it is thus feasible to mine on many chains in parallel. For rational miners the best strategy is to mine on all known chains to maximize their chances at getting the reward. This impedes consensus and potentially might even allow for double spending. We describe how we address this problem in §4.8.

Nothing-at-stake: Grinding blocks. If we just naively replace PoW with PoSpace in Bitcoin, a miner can simply "grind" through many different blocks: it can consider different sets of transactions to add to a block until it finds a set that is in some sense favorable for the miner. In particular, if the challenge for proofs in a block depends on previous blocks (as it does in Bitcoin), then a miner can grind blocks until it finds one that gives a challenge that will also allow him to add a future block with high probability, potentially hijacking the chain forever. We describe our solution in §4.8.

Grinding challenges. In PoSpace, an adversary who owns N bits of space can split up the space into t chunks of size N/t, and commit to each chunk separately (i.e., mimicking t different smaller miners). When generating a new block, the adversary can now choose which of the t proofs to use. The quality of each block will be typically low, but the power to chose from t different challenges might still give an advantage to the adversary. We discuss this attack in detail in Appendix C.

4.2 Protocol description

The high-level protocol of SpaceMint is similar to that of Bitcoin [32]: coin owners generate transactions to spend their coins, and the miners add these transactions to the block chain to reach consensus on the history of transactions, while getting some reward when they successfully add transactions to the chain.

4.2.1 Transactions in SpaceMint

Each coin in SpaceMint belongs to some public key (user) pk. To spend this coin, its owner signs a transaction with the secret key sk, and sends the transaction to the SpaceMint network in order to add it to the block chain. [10] Unlike Bitcoin however, there are a few special types of transactions in SpaceMint (described in §4.3) used to solve some of the challenges from §4.1 and other issues.

4.2.2 Mining in SpaceMint

We incentivize miners to add transactions to the block chain in two ways. First, 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. Second, each transaction can dedicate a small part of the transferred amount to the miner who adds the transaction to the block chain. The mining process consists of two phases: initialization and mining.

Initialization. When a miner first joins the SpaceMint network and wants to contribute N bits of space to the mining effort, it samples a public/secret key pair (pk, sk) and runs Algorithm 1 with pk as nonce μ to generate

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

The miner stores (S_γ, sk) and announces its space commitment (pk, γ) through a special transaction (details in §4.4). Once this transaction is in the block chain, the miner can start mining.

Mining. Once initialized, each miner attempts to add a block to the block chain every time period. For time period i, each miner

1. retrieves the hash value of the last block in the "best" chain so far, and a challenge c (we discuss how c is derived in §4.5). c is then expanded into sufficiently long random strings $\$_p, \$_{cv}$;
2. samples $(c_1, \ldots, c_{k_p}) \leftarrow \mathsf{Chal}(n, k_p, \$_p)$ as in Algorithm 3;
3. computes the proof $a := \{a_1, \ldots, a_p\}$ as in Algorithm 3, i.e., $a_i = \mathsf{Ans}(pk, S_\gamma, c_i)$;
4. and computes the quality $\mathsf{Quality}(pk, \gamma, c, a)$ of the proof (to be discussed in §4.6).
5. if the quality is high enough that there is a realistic chance of it being the best answer in period i, the miner creates a block and sends it out to the network in an attempt to add it to the chain.

[10]Note that for simplicity, we only consider the equivalent of Bitcoin pay-to-pubkey-hash transactions; however, more powerful scripts could be deployed in exactly the same way as for Bitcoin.

The block contains (1) a set of transactions, (2) the proof a computed above and (3) a proof that the commitment is correctly computed as defined in Algorithm 2, using $\$_{cv}$ as the randomness $\$$. Generating a full block as in Step 5 above takes about 30 seconds in our prototype implementation (§5), which could limit how fast we add the blocks to the chain. However, as discussed later in §4.5, the challenge c will be available to the miners tens of minutes before time period i, so Step 5 should not limit the frequency.

4.3 Block chain format

A block chain in SpaceMint is a sequence of blocks $\beta_0, \beta_1, \ldots$ which serve as a public ledger of all transactions. Each block $\beta_i = (\varphi_i, \sigma_i, \tau_i)$ consists of three main parts, which we call "sub-blocks". Each sub-block contains the index i that specifies the position of the block in the block chain. The structures of the sub-blocks are as follows:

- The HASH sub-block φ_i contains:
  - The current block index i.
  - The miner's signature $\zeta_\varphi = \mathsf{Sign}(sk, \varphi_{i-1})$ on a HASH sub-block with index $i-1$.
  - A "space proof" containing the miner's pk.
- The TRANSACTION sub-block τ_i contains:
  - The current block index i.
  - A list of *transactions* (§4.4).
- The SIGNATURE sub-block σ_i contains:
  - The current block index i.
  - The miner's signature $\zeta_\tau = \mathsf{Sign}(sk, \tau_i)$ on a TRANSACTION sub-block τ_i for the index i.
  - The miner's signature $\zeta_\sigma = \mathsf{Sign}(sk, \sigma_{i-1})$ on a SIGNATURE sub-block σ_{i-1} for index $i-1$.

The links between blocks in a block chain are illustrated in Figure 2. We will refer to the hash sub-blocks as the *proof chain*, and the signature sub-blocks with the transactions as the *signature chain.* In the diagram, an arrow from sub-block B' to sub-block B'' means that B' contains the miner's signature on B''. Notice that while the signature and

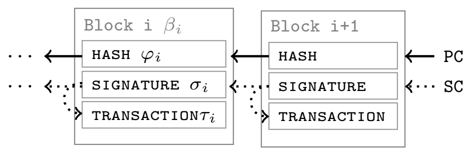

Figure 2: Our block chain consists of a proof chain PC that does not allow for grinding, and a signature chain SC that binds transactions to the proof chain.

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.

This design seems to prevent any kind of consensus, as now we can have arbitrary many signature chains containing different transactions consistent with the same proof chain. The key observation is that once an honest miner adds the ith block (honest in the sense that he will only sign one block and keep its secret key sk_i secret), the transactions corresponding to this proof chain up to block i cannot be changed any more even by an adversary who controls all secret keys from miners that added the first $i-1$ blocks (as this would require forging a signature under sk_i).

4.4 Transaction format

Transactions in SpaceMint use signature schemes that are existentially unforgeable under chosen message attacks:

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

Every transaction is signed by the user generating the transaction. We specify the three types of transactions that we allow in SpaceMint:

Payments. Coins are held and transferred by parties identified by a verification key in the support of SigKeyGen.[11] 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 block chain 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 $txId_j$,
  - sig_j is a signature of $(txId, txId_j, k_j, \vec{out})$, which verifies under key pk_{k_j} proving ownership of the the k_jth beneficiary of transaction $txId_j$ and binding the coin to the beneficiaries.

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

1. No benefactor is referenced by more than one transaction in the block chain (to prevent double-spending).
2. The sum of the input values to the transaction (i.e., the sum of the amounts provided by each benefactor) is 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 transfers to other parties.

Space commitment. A space-commitment transaction

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

consists of pk, a public key, and γ 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 require every miner to commit (pk, γ) because the PoSpace proofs from [17] have the property that by making minor changes, one can turn (pk, γ) into many other space commitments $(pk, \gamma'), (pk, \gamma''), \ldots$ that re-use parts of the space. Thus, if there were no

[11] In Bitcoin, there is a special scripting language which allows for more complex transactions. This functionality can be added to SpaceMint in striaghtforward fashion.

requirement to publish the 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

$$ctx = (\mathsf{punish}, txId, pk, \mathsf{proof}) ,$$

consists of pk, the public key of the transaction creator, and proof, evidence of a misbehavior by another miner. proof would usually contain the misbehaving miner's public key pk', index j indicating when the misbehavior happened, and a proof of misbehavior. If the proof is correct, then pk' is fined some amount of coins and pk would receive a portion of the fine as a reward for catching pk'. We describe a concrete use case of punishments in §4.8.

We require j to be within some reasonable period of time of the current block, before there is a consensus on block j. This is to ensure that the cheating miner has not yet transferred the coin associated with the misbehavior to another account.

4.5 Where the challenge comes from

One difficulty when designing a PoSpace-based block chain is the generation of the input challenge c. Our main solution derives this challenge from the chain itself (as in Bitcoin), and is described in §4.5.2, but we first explain a simple solution assuming an unpredictable beacon.

4.5.1 Unpredictable beacon

Our first solution is based on an unpredictable beacon which broadcasts a value every time period. Since the beacon is unpredictable, no party at time t has non-negligible probability of guessing the beacon value that will be announced at time $t + 1$. For instance, a beacon could be the hash of the current time and the NASDAQ chart or weather pattern. Given such a beacon, the challenge c for mining block i can be derived as a hash of the beacon value. This scheme naturally prevents block grinding, as the challenges do not depend on something under miner's control.

Unfortunately, assuming a random beacon may be impractical, especially in a decentralized setting where the source of a beacon is potentially not trusted. In the settings where a trusted source of randomness is available, however, this scheme would provide an elegant and simple solution.

4.5.2 Challenges from the past

This scheme derives the challenges from the chain itself. In Bitcoin, the PoW challenge for block i is simply the hash of block $i - 1$. We change this in three ways to prevent attacks which are possible due to the fact that we use PoSpace.

Using block $i - 1$ for the challenge can slow down consensus in SpaceMint: If there are many different chains, miners can get different challenges for different chains. A rational miner would thus compute answers for many different chains (since it is easy to do so), 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 (but would only mine one chain to avoid punishment). If all miners behave rationally, this will considerably slow down consensus, as bad chains get extended with blocks of comparable quality to the current best chain, and it will take longer for lower-quality chains to die off. To solve this kind of problem, Slasher [6] penalizes miners that extend chains that do not end up in the final chain, but this approach penalizes users unnecessarily frequently. Instead, we use the hash of block $i - \Delta$ for the challenge for block i for a reasonably large Δ: the probability of multiple chains surviving for more than Δ blocks decreases exponentially as Δ increases. Moreover, we only hash the block from the *proof chain*, but not the *signature chain* (Figure 2) to prevent block grinding as discussed in §4.8.

Finally, we will use the same challenge not just for one, but for δ blocks. This is done to prevent challenge-grinding attacks. We provide the intuition behind the attack and the parameter δ in §4.8, and describe in more detail in Appendix C.

4.6 Quality of a proof

The block added to the chain in a given time period is decided by the quality of the PoSpace proof included in the HASH sub-block. For valid proofs

$\pi_1 = (pk_1, \gamma_1, c_1, a_1), \ldots, \pi_m = (pk_m, \gamma_m, c_m, a_m)$, the $\mathsf{Quality}(\pi_i)$ of each proof should be assigned such that the probability (over the choice of the random oracle hash) that the ith proof has the best "quality" corresponds to its fraction of the total space:

$$\Pr_{\mathsf{hash}}[\forall j \neq i : \mathsf{Quality}(\pi_i) > \mathsf{Quality}(\pi_j)] = \frac{N_{\gamma_i}}{\sum_{j=1}^{m} N_{\gamma_j}}$$

It is sufficient to achieve this relationship for any pair of commitments for the probability above:

$$\Pr_{\mathsf{hash}}[\mathsf{Quality}(\pi_i) > \mathsf{Quality}(\pi_j)] = \frac{N_{\gamma_i}}{N_{\gamma_i} + N_{\gamma_j}} .$$

For this, we sample from 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:

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

Let $D_N(\tau)$ denote a sample of D_N using randomness τ to sample. We now define

$$\mathsf{Quality}(pk, \gamma, c, a) = D_{N_\gamma}(\mathsf{hash}(a)) \qquad (3)$$

for valid proofs. When a proof is invalid, then the $\mathsf{Quality}$ function is defined to be 0.

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 $[0, 1]$ is distributed as X. The random variable X sampled according to 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. 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 $D_{N_{\gamma_i}}$ using randomness $\mathsf{hash}(a_i)$. To do so, we normalize the hash outputs in $\{0, 1\}^L$ to a value in $[0, 1]$, and get

$$D_{N_{\gamma_i}}(\mathsf{hash}(a_i)) := \left(\mathsf{hash}(a_i)/2^L\right)^{1/N} .$$

4.7 Quality of a chain

In order to decide which of two given proof chains is the better one, we also need to define the quality of a proof chain $(\varphi_0, \ldots, \varphi_i)$, which we denote by $\mathsf{QualityPC}(\varphi_0, \ldots, \varphi_i)$. Each hash sub-block φ_j contains a proof $(pk_j, \gamma_j, c_j, a_j)$, and the quality of the block is $v_j = D_{N_j}(\mathsf{hash}(a_j))$ (§4.6). For any quality $v \in [0, 1]$, we define

$$\mathcal{N}(v) = \min\{N \in \mathbb{N} : \Pr[v < w \mid w \leftarrow D_N] \geqslant 1/2\} ,$$

the space required to get a better proof than v on a random challenge with probability 1/2. This quantity captures the amount of space required to generate a proof of this quality.

It may seem that a natural measure for the quality of the chain would be simply the sum $\sum_{j=1}^{i} \mathcal{N}(v_j)$. This definition, however, allows for challenge grinding attacks (briefly discussed in §4.8 and in detail in Appendix C). Intuitively, this attack arises when the adversary commits to his disk space in many small chunks (as opposed to one large chunk), and then tries out many potential challenges that are generated from each chunk. In this attack, the linearity of sum potentially enables an adversary to add blocks to the block chain which generate challenges that provide the adversary with unfair advantage "in the future". Hence, instead of the sum, we prefer the *product* of the $\mathcal{N}(v_j)$, or equivalently, the sum of its logarithms.

One further consideration when defining the quality of a block chain is that since the storage capacity of the network would grow over time, we would like the more recent blocks to contribute more to the quality of a chain than older blocks: this is to prevent "history rewriting" attacks where a miner mines a block far in the past, easily beats the quality of the blocks really mined at that time, and extends his own blocks until his chain takes over the current best chain. We therefore weight the contribution of the jth block for chain of length i by a discount factor Λ^{i-j}:

$$\mathsf{QualityPC}(\varphi_0, \ldots, \varphi_i) = \sum_{j=1}^{i} \log(\mathcal{N}(v_j)) \cdot \Lambda^{i-j} . \quad (4)$$

4.8 Solving design challenges

In this section, we describe how we solve the design challenges described in §4.1. Note that many of these issues do not occur in the "unpredictable beacon" scheme (§4.5.1). Thus, in this subsection, we focus our attention on the case when SpaceMint uses the (arguably more realistic) "challenge from the past" scheme to generate challenges (§4.5.2).

Mining multiple chains. We will discuss two cases, depending on whether mining is done on two (or more) chains that forked *more* or *less* than Δ blocks in the past.

Case 1: chains forked less than Δ ago. In this case, the miner will get the same challenge for both chains. SpaceMint uses punishments to disincentivize miners from extending multiple chains in this case. Note that without such punishments, it would be rational for a miner who got a very good PoSpace proof for the current challenge to extend both chains in order to prevent his block from being "orphaned". Concretely, suppose there are currently two chains whose most recent blocks are β_j and β'_j. If a miner pk' attempts to mine on both chains for index $j+1$ by announcing β_{j+1} and β'_{j+1}, using the same (pk', γ), then another miner can generate a transaction $(\mathsf{punish}, txId, pk, \{pk', \beta_j, \beta'_j, \beta_{j+1}, \beta'_{j+1}\})$ to punish them. Once this transaction completes, half of the reward money and transaction fee for the mined block will go to pk, and the other half will be destroyed. We destroy half of the reward to ensure that a cheating miner is punished even if it publishes this transaction itself.

Case 2: chains forked more than Δ ago. Here, the miner gets different challenges (and thus, two proofs of different quality) for the two chains. Now it is rational for a miner to also extend chains which are not the highest-quality chain, if it has a proof of very high quality for that particular chain. This would slow down consensus and give the miner an unfair advantage for deviating from the protocol. Because of our punishment scheme discussed in the previous case, this case is extremely unlikely to happen (the probability is exponentially small in Δ), as a fork would have to "survive" for Δ blocks despite a strong incentive for miners to only extend the chain of highest quality at any given time.

Block grinding. By decoupling proofs from transactions as shown in Figure 2, we eliminate the problem of block grinding. Since challenges are computed as a hash of a block in the *proof chain* (but does not depend on the signature chain), a miner who has found a proof has only the choice of either announcing it or not, but has no other degree of freedom to influence future challenges by changing the contents of the block.

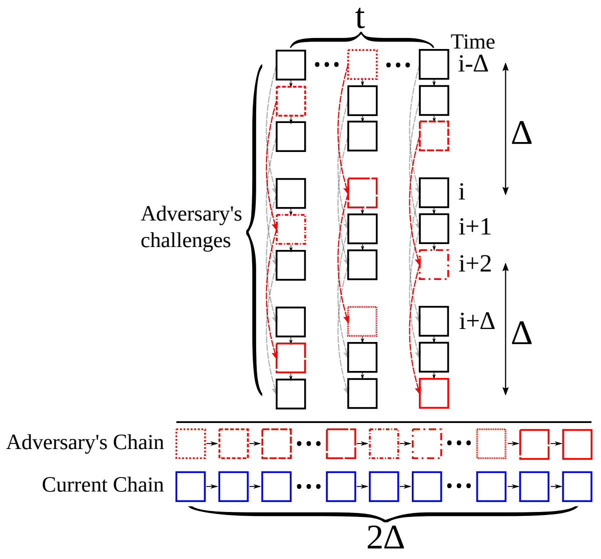

Figure 3: Challenge-grinding attack. The attack succeeds if the quality of adversary's chain is better than the current (honest) chain.

Challenge grinding. The challenge-grinding attack arises from the fact that an adversary could potentially split the space into t smaller spaces, and try multiple challenges to find the one which will be most favorable to him Δ steps in the future. As shown in Figure 3, for time-steps i through $i+\Delta-1$, the adversary could choose among t challenges in each epoch (from time $i-\Delta$ through $i-1$) and pick the ones that yield the best quality chain of length 2Δ. The adversary can then release this chain (all at once) in an attempt to overtake the longest chain. As mentioned in §4.7, this problem is exacerbated if the metric for determining the quality of a chain is a sum or any other linear function. To prevent this attack, we (1) use product instead of a sum for the quality of a chain, and (2) use the same block to derive challenges for δ blocks (i.e., use $\mathsf{hash}(\beta_i, nonce)$ for $nonce \in [1, \delta]$ as challenges for time $i+\Delta$ through $i+\Delta+\delta$). Intuitively, (1) makes it harder for the adversary to find a good chain of length 2Δ by weighing the worse blocks more, and (2) makes it exponentially harder (in δ) to find a "good" challenge that will yield δ high quality blocks, thereby reducing the quality of the chain. Detailed explanation of the attack and our defense is given in Appendix C.

4.9 Parametrization

We now discuss and justify parameter choices for SpaceMint. A more detailed discussion on parameters and their influence is in Appendix D.

Determining challenges. To minimize the probability of forks surviving for more than Δ blocks (which is necessary to prevent the "mining multiple blocks" issue from §4.8), we should choose a large Δ. On the other hand, a smaller Δ increases other security features of SpaceMint (discussed in Appendix D). We suggest $\Delta = 50$, as then it will be highly unlikely that a fork would survive for Δ steps (since the probability of a fork surviving is exponentially small in Δ), and yet the value is not large enough to introduce significant negative impacts to other aspects (see Appendix D for further discussion).

Frequency of block generation. The challenge for block i is available at least Δ blocks (which corresponds to $\Delta \cdot \mathsf{time}$ minutes) before block i is added. In terms of computation, since it takes less than 30 seconds to generate a block (§5) and we set $\Delta = 50$, we could generate blocks every few seconds given that one miner is unlikely to mine more than a few good blocks within the Δ blocks. However, we only want to generate the blocks as fast as it can propagate through the network, since the miners need to generate the signature chains using the previous block. In Bitcoin, blocks propagate to over 95% of the miners within 40 seconds [13], so we believe that $\mathsf{time} = 1$ minute would be a reasonable frequency of block generation for SpaceMint.

Quality discount factor. As discussed in §4.7, we use a discount factor Λ to determine each block's contribution to overall chain quality. The value of Λ is determined by the pace at which the total storage in the network increases. For instance, if we assume that storage stays roughly in the same order of magnitude for two-month periods, we can set Λ as large as 0.99999.[12] Such a high Λ is helpful when we argue about the hardness of generating long forks in Appendix D.

Confirmation time. To confirm a transaction, we must be sure that there is consensus regarding the transaction being on the chain, in order to prevent double spending. Bitcoin, to this end, only confirms transactions after 6 blocks are added, at which point users are reasonably confident of the consensus. Their analysis [37] assumes the adversary has less than 10% of total hashing power and gives an upper-bound of 0.001 on the probability of double-spending. Assuming a 10% adversary as in the Bitcoin analysis (with $\Lambda = 0.99999$ as discussed above), SpaceMint can confirm transactions after 6 blocks with an upper-bound of $2^{-16} \approx 0.000015$ on the probability of double-spending, and moreover this takes only 6 minutes compared to the 1 hour of Bitcoin. Even assuming a stronger adversary who controls 33% of the total space (and $\Lambda = 0.99999$ as before), SpaceMint can confirm transactions after 93 blocks with failure probability bounded by 2^{-32}. Our analysis is shown in Appendix D.

5 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 [35], 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 desktop 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.

Time to initialize. To start mining SpaceMint, the clients must first initialize their space, as de-

[12] In this case, the contribution of a block decreases by a factor $1/e \approx 0.37$ every $1/(1-\Lambda) = 100.000$ blocks, which for $\mathsf{time} = 1$ minute is roughly 69 days.

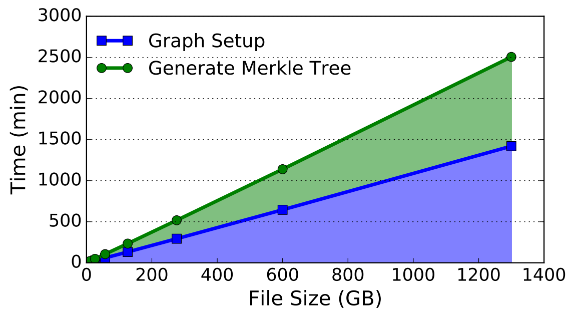

Figure 4: Time to initialize the space.

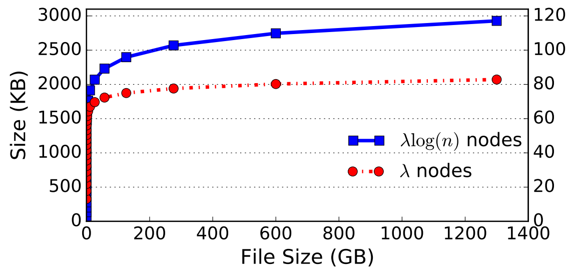

Figure 5: Size of the proof for varying space sizes for $\lambda = 30$. The y-axis on the left and right represents the size for opening $\lambda \log(n)$ and λ nodes respectively.

scribed in §4.2. Concretely, this involves computing all the hashes of the nodes, and computing the Merkle tree over the hashes. In Figure 4, 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 generate and commit the space. While expensive, this procedure is done only once when a miner first joins the SpaceMint network, and the initialized space will be used over and over again. In fact, space initialization should take non-trivial time because an extremely fast space initialization would make re-using the same space for different commitments a viable strategy.

Size of the proof. A proof in SpaceMint consists of the Merkle inclusion proof for a set of node labels. For the PoSpace that we implemented, the number of nodes we have to open is $\lambda \cdot \log(n) + 1$ (as $k_{cv} = \lambda \cdot \log(n)$ in Algorithm 2 and $k_p = 1$ in Algorithm 3), where λ is a statistical security parameter. Every node in this graph has at most two parents, and each opening of a node is $\log(n) \cdot 32$ bytes. Thus, the overall proof size is upper-bounded $3 \cdot \lambda \cdot \log^2(n) \cdot 32$ bytes. Though opening fewer than $\lambda \log(n)$ nodes is not shown to be secure, we are unaware of any concrete attacks even for opening λ nodes. We believe that the size of a sufficiently secure proof will lie somewhere in between, closer to opening λ nodes. Figure 5 demonstrates the size of the proof when we open $\lambda \log(n)$ nodes vs. just λ nodes for $\lambda = 30$.

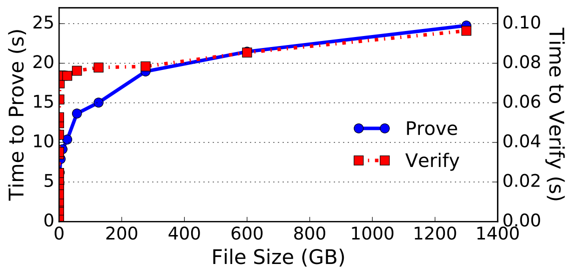

Figure 6: Time it takes for a miner to prove and verify the space when $\lambda \log(n)$ nodes are opened for $\lambda = 30$.

Time to generate/verify the proof. In SpaceMint, assuming a miner is storing the space correctly, the miner needs to only open $k_p = 1$ node in the Merkle tree to check the quality of its solution (§4.2), which takes just a fraction of a second; it takes < 1 ms to read a single hash from the disk. Only in the rare case where the miner believes its answer is of very good quality will it generate the full proof, which still takes less than 30 seconds.

Our proofs are substantially bigger than Bitcoin's, and require more than just one hash evaluation to verify. However, for an active currency, we still expect the size and verification time for the proofs added with every block to be marginal compared to the size of the transactions added with every block or the time required to verify that the transactions are consistent. Figure 6 indeed shows that even though it takes seconds to generate the proof, verification takes only a fraction of a second.

Energy estimates. Though our prototype was evaluated using a full CPU which wastes a lot of energy, a cost-conscious miner could mine on a much more energy-efficient device (e.g., Raspberry Pi [5]). An efficient microcontroller consumes less than 10 W of power, and most miners will only open a few nodes per time step since the quality of their answers will usually be bad. To get an upper bound on the power requirement, suppose there are 100 000 miners, each with 1 TB of space, and about 1% of the miners mine "good" answers for which they will generate a full answer. Then we have

$$10\text{W}\cdot 100\,000\cdot 0.01s + 10\text{W}\cdot 1000\cdot 20s = 210\,000\text{J/block}$$

which translates to 210 kJ/min if we add one block every 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 the 1% figure is a very conservative bound, so the difference could be even larger in practice.

6 Discussion

In this section we shortly address two potential issues with SpaceMint, DoS attacks and the use of very cheap storage like tapes.

DoS. A party who wants to mine must have its space commitment (pk, γ) added to the hash chain. A malicious party could flood the network with countless requests of fake commitments to be added to the chain. One simple way to counter this 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 that one has to provide a full commitment verification proof (as in Algorithm 2) for the commitment (pk, γ) to be added. This proof is only provided to show that the space for this commitment has really been instantiated, which is fairly expensive, but the proof will not be added to the chain.

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 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.

7 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: i.e., it is important that the protocol rules are designed such that miners are never in a situation where deviating from the protocol yields more expected profit than behaving 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 blockchain. At each time step, each player (miner) must choose a strategy, specified by:

- which blocks to extend (if any),
- which extended blocks to publish (if any).

Showing that adhering to the protocol is an *equilibrium* of such a game means that rational miners are not incentivized to deviate from the protocol when playing the game. From this, by definition of the SpaceMint protocol, it follows that rational miners will reach consensus on a single chain, and will not be able to get an unfair mining advantage by using a "cheating" strategy.

We remark that game-theoretic analyses inherently start by defining a game which *models* reality, and prove properties of the game in this model. It is almost never possible for a model to capture *all aspects* of a real-world situation, and it is moreover desirable to have a model which is simple enough to allow for a rigorous analysis of incentives, while still being close to reality.

7.1 Game-theoretic preliminaries

The standard game-theoretic notion for a strategic game which occurs over multiple time steps (rather than in "one shot") is 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*, which 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 blockchain, 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 7.1 (Extensive game)**.** *An* extensive game $\Gamma = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$ *is defined by:*

- $[N]$*, a finite set of players.*
- H*, the set of all possible histories, which must satisfy the following two properties:*
  - *the empty sequence* $()$ *is in* H*, and*
  - *if* $(a_1, \ldots, a_K) \in H$ *then for all* $L \leqslant K$*, it holds that* $(a_1, \ldots, a_L) \in H$*.*

  We write $Z \subseteq H$ *to denote the subset consisting of all* terminal histories*. For any history* h*,*
  $$A(h) = \{a : (h, a) \in H\} = \bigtimes_{i \in [N]} A_i(h)$$
  denotes the set of action profiles that can occur at that history, and $A_i(h)$ *denotes the set of actions that are available to player* i *at history* h*.*
- $f(\cdot, h)$ *is a probability measure on* $A_{\mathcal{C}}(h)$*, where* $h \in H$ *and* $\mathcal{C}$ *denotes the Chance player.*
- $\vec{\mathcal{I}} = (\mathcal{I}_1, \ldots, \mathcal{I}_N)$*, where each* $\mathcal{I}_i$ *is a partition of* H *into disjoint* information sets*, such that* $A_i(h) = A_i(h')$ *whenever* h *and* h' *are in the same information set* $I \in \mathcal{I}_i$*. Let* $A_i(I)$ *denote the set of actions that are available to player* i *at any history in information set* I*.*
- $\vec{u} = (u_1, \ldots, u_N)$*, where each* $u_i : Z \to \mathbb{R}$ *is the utility function of player* i*.*

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, Bob does not know about the other blocks that Alice tried. Thus, SpaceMint is a game of *imperfect information.*

The information that players do not know about other players' actions is modeled by the partitions $\vec{\mathcal{I}} = (\mathcal{I}_1, \ldots, \mathcal{I}_N)$ in Definition 7.1. 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 7.2 ("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 7 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.

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.

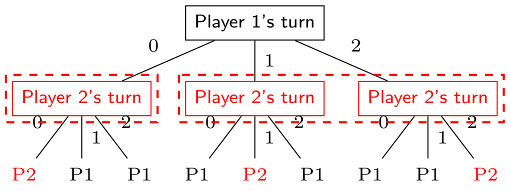

Figure 7: Game tree for the "Match my number" game. Leaves are labelled with the winning player.

Definition 7.3 (Strategy profile)**.** *A* strategy profile $\vec{\alpha} = (\alpha_1, \ldots, \alpha_N)$ *of an extensive game* $\Gamma = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u}\rangle$ *specifies for each player* $i \in [N]$ *and each information set* $I \in \mathcal{I}_i$ *a probability distribution* $\alpha_i(I)$ *over the action set* $A_i(I)$*. We say that* α_i *is the* strategy *of player* i*.*

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

Recall that the utility functions $u_1, \ldots, u_N$ were defined on 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 $\vec{\alpha}$.

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

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,

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

7.2 Equilibria of extensive games

The most widely known equilibrium concept for a strategic game is the Nash equilibrium [33], given in Definition 7.12. Intuitively, in a Nash equilibrium, each player's strategy is a *best response* to the strategies of the other players.

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

Definition 7.4 (Nash equilibrium of an extensive game)**.** *Let* $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u}\rangle$ *be an extensive game. A strategy profile* $\vec{\alpha}$ *is a* Nash equilibrium of Γ *if for each player* $i \in [N]$ *and each strategy* α'_i *of player* i*,*

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

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

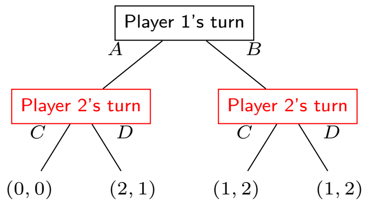

Figure 8: Game tree for the "Unstable" game.

It is a Nash equilibrium of this game for player 1 to choose B, and player 2 to always choose C[13] However, the strategy profile (B, C) seems "unstable"[14], 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

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

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

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

In Definition 7.6, $\mu_i(I)(h)$ 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 $u_i((\vec{\alpha}, \vec{\mu})|\{()\})$, i.e. the expected utility for player i at the start of the game.

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

Definition 7.7 (Sequentially rational assessment)**.** *Let $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ be an extensive game. An assessment $(\vec{\alpha}, \vec{\mu})$ is* sequentially rational *if for every $i \in [N]$ and every strategy α'_i of player i, for every information set $I \in \mathcal{I}_i$, it holds that*

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

Definition 7.10 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 *sequential 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 players also *believe* it will occur with zero probability.

Definition 7.8 (Consistent assessment)**.** *Let $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ be an extensive game. A strategy profile $\vec{\alpha}$ is said to be* completely mixed *if it assigns positive probability to every action at every information set. An assessment $(\vec{\alpha}, \vec{\mu})$ is* consistent *if there is a sequence $((\vec{\alpha}^n, \vec{\mu}^n))_{n \in \mathbb{N}}$ of assignments that converges to $(\vec{\alpha}, \vec{\mu})$ in Euclidean space, where each $\vec{\alpha}^n$ is completely mixed and each belief system $\vec{\mu}^n$ is derived from $\vec{\alpha}^n$ using Bayes' rule.*

Finally, we arrive at the definition of a sequential Nash equilibrium, which is the canonical solution concept in the game theory literature for extensive games.

Definition 7.9 (Sequential Nash equilibrium)**.** *An assessment is an* sequential Nash equilibrium *if it is sequentially rational and consistent.*

Relaxing to epsilon-optimality. In our analysis, we will focus on a slight relaxation of sequential rationality, which requires an assessment to be *ε-close to optimal* rather than optimal. The definition of *ε-sequentially rational assessment*, given below, is a direct application of the standard game-theoretic notion of *ε-optimality* to Definition 7.7. This relaxed notion is required for settings where we assume computationally bounded players.

Definition 7.10 (ε-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* ε-sequentially rational *if for every $i \in [N]$ and every strategy α'_i of player i, for every information set $I \in \mathcal{I}_i$, it holds that*

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

Definition 7.11 (ε-sequential Nash equilibrium)**.** *An assessment is an* ε-sequential Nash equilibrium *if it is ε-sequentially rational and consistent.*

There is an analogous, ε-optimal variant of Nash equilibrium (i.e. without sequential rationality) which is standard in the game theory literature:

Definition 7.12 (ε-Nash equilibrium of extensive game). *Let $\Gamma = \langle N, H, f, \vec{\mathcal{I}}, \vec{u} \rangle$ be an extensive game. A strategy profile $\vec{\alpha}$ is an* ε-Nash equilibrium *of* Γ *if for each $i \in [N]$ and each strategy α_i' of player i,*

$$u_i(\vec{\alpha}) \geqslant u_i(\alpha_i', \vec{\alpha}_{-i}) - \varepsilon \ .$$

7.3 Game-theoretic analysis

In order to analyze the game-theoretic properties of SpaceMint mining, we define an extensive game, SpaceMint, which models the actions that miners can take, and the associated payoffs.

7.3.1 Simplifying the action space

To facilitate analysis, we have simplified the action space as much as possible while still accurately capturing the important features that influence the incentives of SpaceMint miners. Concretely:

- 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 model the proof-of-space challenge according to the "unpredictable beacon" scheme rather than the "challenge from the past" scheme. We remark that this simplification (unlike the other simplifications listed here) does mean that our game-theoretic analysis does not model certain attacks: namely, the attacks to which "unpredictable beacon" is robust but "challenge from the past" is vulnerable. The only example of such an attack of which we are currently aware is challenge grinding (Appendix C).
- We do not include the action of *creating a space commitment* because (as discussed in §4.2, "Mining") we can assume rational miners commit to all the space they have, and not more.[15]

Remark. To simplify exposition, we do not explicitly model the amount of the space that each player has in the game defined below. A standard way to model this would be to assign each player a *type* t_i, representing player i's amount of space. Our exposition keeps the types implicit; our theorems require that no player has more than 50% of the space committed by active miners.

7.3.2 The SpaceMint mining game

Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ be a proof of space. Let $\mathcal{B}$ denote set of all *blocks* as defined in §4.3, and for any $\ell \in \mathbb{N}$, let $\mathcal{B}_\ell$ denote the set of all blocks with index ℓ.[16] Let $\mathcal{B}^\ell$ denote the set of blocks with index at most ℓ, i.e. $\mathcal{B}^\ell = \bigcup_{\ell'=0}^{\ell} \mathcal{B}_{\ell'}$. Let B_{gen} be the genesis block; note that $\mathcal{B}_0 = \{B_{\text{gen}}\}$.

For a block $B \in \mathcal{B}$ and a challenge $c \leftarrow \mathsf{Chal}$, we define $\mathsf{Ext}_i(B, c)$ to be the block generated by player i when mining the next block after B using the PoSpace challenge c (see §4.3 for exact block format). For $\ell \in \mathbb{N}$ and challenge c, define:

$$\tilde{\mathcal{B}}_{\ell,i,c} = \{(B, B') \in \mathcal{B}_{\ell-1} \times \mathcal{B}_\ell : B' = \mathsf{Ext}_i(B, c)\}$$

and let $\tilde{\mathcal{B}}^{\ell,i,c} = \bigcup_{\ell' \in \{0,\ldots,\ell\}} \tilde{\mathcal{B}}_{\ell',i,c}$.

Definition 7.13 (The SpaceMint Game). *Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ be a proof of space. For any number of players $N \in \mathbb{N}$, any number of time steps*

[15] Note that this argument does not quite apply when challenge grinding attacks are possible. However, the argument holds here since our analysis is in the unpredictable beacon setting.

[16] The index denotes the block's position in the block chain. In §4.3, i is used to refer to the block index, but in this section we use ℓ to avoid confusion with the player indices.

$K \in \mathbb{N}$, any consensus-delay $\Psi \in \mathbb{N}$, and any reward function $\rho \colon \mathbb{N} \to \mathbb{N}$, we define the extensive game

$$\mathsf{SpaceMint}_{\Pi,K,\rho} = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u} \rangle$$

as follows:

- *The set H of histories is defined inductively.*
  - *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 $i \in [N]$.*
  - *For any non-terminal history h and any $i \in [N]$, the action set $A_i(h)$ of player i at h is:*

    $$A_i(h) = \mathcal{P}\left(\mathcal{B}^{|h|} \times \mathcal{B}^{|h|+1}\right) \times \mathcal{P}\left(\mathcal{B}^{|h|} \times \mathcal{B}^{|h|+1}\right).$$

    An action $a_i \in A_i(h)$ is a pair of sets $a_i = (\mathfrak{T}, \mathcal{A})$. $\mathfrak{T}$ is the set of blocks that player i tries *extending in this time step, and $\mathcal{A} \subseteq \mathfrak{T}$ is the set of blocks that player i* announces *in this time step. An element in $\mathfrak{T}$ (or $\mathcal{A}$) is a pair of blocks $(B', B) \in \mathcal{B}^{|h|} \times \mathcal{B}^{|h|+1}$ where B' is the existing block which player i wishes to extend, and $B \in \mathcal{B}$ is the extended block.*
- *The probability measure $f(\cdot, h)$ is uniform on $\{0,1\}^m$.*
- *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.*
  - $$[(h, ((\mathfrak{T}_1, \mathcal{A}_1), \ldots, (\mathfrak{T}_N, \mathcal{A}_N), a_{\mathcal{C}}))]_i = \big\{(h', ((\mathfrak{T}'_1, \mathcal{A}'_1), \ldots, (\mathfrak{T}'_N, \mathcal{A}'_N), a'_{\mathcal{C}})) \in H : h \sim_i h' \wedge \mathfrak{T}_i = \mathfrak{T}'_i \wedge \mathcal{A}_i = \mathcal{A}'_i \wedge a_{\mathcal{C}} = a'_{\mathcal{C}} \wedge \forall i' \neq i,\ \mathcal{A}_{i'} = \mathcal{A}'_{i'}\big\},$$

    where h and h' are histories of equal length, and the pairs $(\mathfrak{T}_{i'}, \mathcal{A}_{i'})$ and $(\mathfrak{T}'_{i'}, \mathcal{A}'_{i'})$ are actions of player i'. That is, two histories are equivalent under $\sim_i$ if they are identical except in the "first components" $\mathfrak{T}_{i'}$ of the actions $(\mathfrak{T}_{i'}, \mathcal{A}_{i'})$ taken by players other than i.
- *$\vec{u} = (u_1, \ldots, u_N)$, where each $u_i : Z \to \mathbb{R}$ is defined as described below. For a history h, let $\mathfrak{C}(h)$ denote the sequence of actions taken by the Chance player in h. Let $B.\mathrm{chal}$ denote the challenge c within the proof of space of a block B.*

  Recall that the functions $\mathsf{Quality}(B)$ and $\mathsf{QualityPC}(\vec{B})$ were defined in §4.6. We define a new function

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

  Also, let $\mathsf{QualityPC}((B_1, \ldots, B_L), (c_1, \ldots, c_L))$ be equal to $\mathsf{QualityPC}((B_1, \ldots, B_L))$ whenever

  $$\forall \ell \in [L],\ B_\ell.\mathrm{chal} = c_\ell \textit{ and } B_\ell \in \mathcal{B}_\ell$$

  and

  $$\forall \ell \in [L],\ \exists i \in [N] \textit{ s.t. } (B_{\ell-1}, B_\ell) \in \tilde{\mathcal{B}}_{\ell,i,c_\ell}$$

  and equal to 0 otherwise.

  For any history h, let $\mathcal{A}(h)$ be the set of all blocks announced by any player in history h:

  $$\mathcal{A}(h) = \left\{ B : \begin{array}{l} \exists i \in [N], \mathcal{A}' \textit{ s.t. } (\cdot, B) \in \mathcal{A}' \textit{ and} \\ \textit{player } i \textit{ took action } (\cdot, \mathcal{A}') \textit{ in } h \end{array} \right\}$$

  Let $\mathsf{blocks}(h)$ denote the sequence of "winning blocks" at any given history h:

  $$\mathsf{blocks}(h) = \operatorname*{arg\,max}_{\vec{B} \in (\mathcal{A}(h))^{|h|}} (\mathsf{QualityPC}(\vec{B}, \mathfrak{C}(h))).$$

  Let $\mathsf{blocks}_\ell(h)$ denote the ℓth block in the chain. Let $\mathsf{win}_\ell(h)$ be the player who announced the winning block $\mathsf{blocks}_\ell(h)$ for index ℓ.[17]

  Recall that a history $h = (\vec{a}_1, \ldots, \vec{a}_J)$ is a sequence of $J \leqslant K$ action profiles. For $j \in [J]$, let $(\mathfrak{T}_{i,j}, \mathcal{A}_{i,j})$ be the action of player i in $\vec{a}_j$. Let $\mathsf{one}_\ell(i, h)$ be an indicator variable for the event that player i announces at most one block with index ℓ, i.e.

  $$\left| \left\{ B : B \in \mathcal{B}_\ell \textit{ and } (\cdot, B) \in \bigcup_{j \in [J]} \mathcal{A}_{i,j} \right\} \right| \leqslant 1.$$

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

  $$u_i(h) = \sum_{\ell \in [K-\Psi]} \delta_{i,\mathsf{win}_\ell(h)} \cdot \mathsf{one}_\ell(i, h) \cdot \rho(\mathsf{blocks}_\ell(h)),$$

  where $\delta_{i,j}$ is the Kronecker delta function. That

[17] We can assume that the winning block is unique at each time-step, and $\mathsf{Quality}$ imposes a total order on blocks.

is, a player's utility is the sum of the rewards he has received for announcing a winning block, up to index $K - \Psi$.

By Definition 7.13, 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, $\mathfrak{C}(h) = \mathfrak{C}(h')$ for any $h, h' \in I$ in the same information set I, so we define $\mathfrak{C}(I)$ to be equal to $\mathfrak{C}(h)$ for any $h \in I$.

For a block $B \in \mathcal{B}$ and a challenge $c \leftarrow \mathsf{Chal}$, we define $\mathsf{Ext}_i(B, c)$ to be the block generated by player i when mining the next block after B using the PoSpace challenge c (see §**??** for exact block format).

Theorem 7.14. *Let*

$$\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$$

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 $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ be a pure strategy profile of $\mathsf{SpaceMint}_{\Pi,K,\rho}$, defined as follows: for each $i \in [N]$, for any information set $I \in \mathcal{I}_i$ such that $I \neq \{()\}$,

$$\alpha_i(I)\left(\left(\{\mathsf{blocks}_j(I)\}, \{\mathsf{Ext}_i(\mathsf{blocks}_j(I), \mathfrak{C}_j(I))\}\right)\right) = 1,$$

where $j \geqslant 1$ is the length of the histories in information set I[18]*. That is, player i's next action at information set I is*

$$\hat{\alpha}_i = \left(\{\mathsf{blocks}_j(I)\}, \{\mathsf{Ext}_i(\mathsf{blocks}_j(I), \mathfrak{C}_j(I))\}\right).$$

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

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

$$v = \mathsf{QualityPC}((\mathsf{blocks}(I), \mathsf{Ext}_i(B, \mathfrak{C}_j(I))), \mathfrak{C}(I))$$

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, $u_i(\vec{\alpha}) \geqslant u_i(\alpha'_i, \vec{\alpha}_{-i})$ for any strategy α'_i which distributes probability over actions of the form (S, T) where $|S| = 1$.

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

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

7.3.3 Analyzing the SpaceMint game

In this section, we prove that honest mining is an ε-sequential Nash equilibrium of the SpaceMint game.

By Definition 7.13, 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, $\mathfrak{C}(h) = \mathfrak{C}(h')$ for any $h, h' \in I$ in the same information set I, so we define $\mathfrak{C}(I)$ to be equal to $\mathfrak{C}(h)$ for any $h \in I$.

First, Theorem 7.15 defines an ε-Nash equilibrium of the SpaceMint game, and then Theorem 7.16 shows that this Nash equilibrium is, moreover, an ε-sequential equilibrium.

Theorem 7.15. *Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ 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 $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ be a pure strategy profile of $\mathsf{SpaceMint}_{\Pi,K,\rho}$, defined as follows: for each $i \in [N]$, for any information set $I \in \mathcal{I}_i$ such that $I \neq \{()\}$,*

$$\alpha_i(I)\left(\left(\{\mathsf{blocks}_\ell(I)\}, \{\mathsf{Ext}_i(\mathsf{blocks}_\ell(I), \mathfrak{C}_\ell(I))\}\right)\right) = 1,$$

where $\ell \geqslant 1$ is the length of the histories in information set I.[19] *That is, player i's next action at information set I is*

$$\hat{\alpha}_i = \left(\{\mathsf{blocks}_\ell(I)\}, \{\mathsf{Ext}_i(\mathsf{blocks}_\ell(I), \mathfrak{C}_\ell(I))\}\right).$$

Let

$$\xi = \frac{\max_{i \in [N]} t_i}{\sum_{i \in [N]} t_i}$$

be the maximum fraction of space possessed by a sin-

[18] All histories in an information set are the same length.

[19] All histories in an information set are the same length.

gle player,[20] *and suppose $\xi < 0.5$. Then $\vec{\alpha}$ is an ε-Nash equilibrium of* $\mathsf{SpaceMint}_{\Pi,K,\rho}$*, where*

$$\varepsilon = \exp\left(-\frac{1}{2K}\cdot \mathbb{E}\left[\mathsf{diff}_1\right]^2 \cdot \left(\sum_{j=0}^{K-1}\Lambda^{2j}\right)^2\right),$$

Λ is the discount factor defined in §4.7 and diff_1 *is defined as in §D.*

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

$$v = \mathsf{QualityPC}((\mathsf{blocks}(I), \mathsf{Ext}_i(B, \mathfrak{C}_\ell(I))), \mathfrak{C}(I))$$

is the same for any block B which was announced for block index ℓ. Therefore, no utility can be gained by choosing any block B over any other block B' to extend. That is, $u_i(\vec{\alpha}) \geqslant u_i(\alpha'_i, \vec{\alpha}_{-i})$ for any strategy α'_i which distributes probability only over action sequences $((\mathcal{T}_{i,1}, \mathcal{A}_{i,1}), \dots, (\mathcal{T}_{i,K}, \mathcal{A}_{i,K}))$ such that

$$\forall \ell \in [K-\Psi],\ \ |\mathcal{A}_i \cap \mathcal{B}_\ell| = 1,$$

where $\mathcal{A}_i = \bigcup_{j\in[K]} \mathcal{A}_{i,j}$.

Moreover, for any given block index ℓ, not announcing any block *or* announcing multiple blocks precludes a player from being the "winner" and receiving the reward at index ℓ, so announcing exactly one block per index is preferable to announcing any other number of blocks. Hence, for any strategies α''_i, α'_i such that α''_i announces exactly one block per index with probability 1, and α'_i assigns non-zero probability to action sequences $((\mathcal{T}_{i,1}, \mathcal{A}_{i,1}), \dots, (\mathcal{T}_{i,K}, \mathcal{A}_{i,K}))$ such that

$$\forall \ell \in [K-\Psi],\ \ |\mathcal{A}_i \cap \mathcal{B}_\ell| \neq 1,$$

(where $\mathcal{A}_i = \bigcup_{j\in[K]} \mathcal{A}_{i,j}$ as above), it holds that

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

We can now restrict our attention to strategies which announce exactly one block per index. Fix any time-step $j \in [K]$. Let α'_i be any strategy in which the probability that player i announces a block $B_j \in \mathcal{B}_j$ at time-step j is less than 1.

Suppose player i does not announce a block $B \in \mathcal{B}_j$ at time-step j. Since we are assuming that i announces exactly one block per index, we know i announces a block $B_{i,j} \in \mathcal{B}_j$ at some time-step $j' > j$. If the other players use strategies $\vec{\alpha}_{-i}$ (i.e. they announce exactly one block with index j at each time-step j), then no player (other than i) will extend player i's block $B_{i,j}$. Let $\vec{B}' = (B_{\text{gen}}, B'_1, \dots, B'_{j-1}, B_{i,j})$ be the unique (length-j) blockchain induced by $B_{i,j}$. If player i does not extend his own block $B_{i,j}$, then he will gain no utility after time-step j. Thus, the only way player i can gain any utility in time-steps after j is if he extends his own blocks all the way up to time-step K:

$$B_{i,j+1} = \mathsf{Ext}_i(B_{i,j}, \mathfrak{C}_j(I_{i,j}))$$
$$\cdots$$
$$B_{i,K} = \mathsf{Ext}_i(B_{i,K-1}, \mathfrak{C}_j(I_{i,K-1}))$$

where $I_{i,j}$ denotes player i's information set at time-step j, and moreover his self-extended chain has higher quality than any chain produced by the other players: that is, at the terminal history h,

$$\mathsf{QualityPC}((\vec{B}', B_{i,j+1}, \dots, B_{i,K}), \mathfrak{C}(I_{i,K})) = \underset{\vec{B}\in(\mathcal{A}(h))^{|h|}}{\arg\max}\ (\mathsf{QualityPC}(\vec{B}, \mathfrak{C}(h))). \quad (5)$$

By Theorem D.2, the probability that (5) holds is at most

$$\exp\left(-\frac{1}{2K}\cdot \mathbb{E}\left[\mathsf{diff}_1\right]^2 \cdot \left(\sum_{j=0}^{K-1}\Lambda^{2j}\right)^2\right).$$

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

We now show that honestly following the SpaceMint protocol is an ε-sequential Nash equilibrium of the SpaceMint game.

Theorem 7.16. *Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ 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* $\mathsf{SpaceMint}_{\Pi,K,\rho}$ *where:*

- *$\vec{\alpha}$ and $\hat{\alpha}_i$ are defined as in Theorem 7.15, 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 $\hat{\alpha}_i$.*

[20] Recall that t_i is the amount of space that player i has (defined in the remark just before Definition 7.13).

- *$\vec{\mu}$ 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.*

Let

$$\xi = \frac{\max_{i\in[N]} t_i}{\sum_{i\in[N]} t_i}$$

be the maximum fraction of space possessed by a single player, and suppose $\xi < 0.5$. Then $(\vec{\alpha}, \vec{\mu})$ is an ε-sequential Nash equilibrium of $\mathsf{SpaceMint}_{\Pi,K,\rho}$ where

$$\varepsilon = \exp\left(-\frac{1}{2K}\cdot \mathbb{E}\left[\mathsf{diff}_1\right]^2 \cdot \left(\sum_{j=0}^{K-1} \Lambda^{2j}\right)^2\right),$$

Λ is the discount factor defined in §4.7 and diff_1 is defined as in §D.

Proof. Fix any player $i \in [N]$. Let $I \in \mathcal{I}_i$ be any information set of player i in $\mathsf{SpaceMint}_{\Pi,K,\rho}$, and let L be the length of histories in I. It follows from Definition 7.13 that the expected utility of player i at I is $u_i((\vec{\alpha},\vec{\mu})|I) =$

$$\sum_{j\in[L]} \delta_{i,\mathsf{win}_j(h)} \cdot \mathsf{one}_j(i,h) \cdot \rho(\mathsf{blocks}_j(h)) + u_i'((\vec{\alpha},\vec{\mu})),$$

where u_i' is the utility function of player i in the game $\mathsf{SpaceMint}_{\Pi,K-L,\rho}$. Since win, one, 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 7.15, $\vec{\alpha}|_{K-L}$ is an ε-Nash equilibrium of $\mathsf{SpaceMint}_{\Pi,K-L,\rho}$. Since $\vec{\mu}$ is derived from $\vec{\alpha}$ by Bayes' rule, it follows that $u_i((\vec{\alpha},\vec{\mu})|I) \geqslant u_i(((\alpha_i', \vec{\alpha}_{-i}),\vec{\mu})|I)$ for any strategy α_i' of player i. Applying this argument for every I, we conclude that $(\vec{\alpha},\vec{\mu})$ is ε-sequentially rational in $\mathsf{SpaceMint}_{\Pi,K,\rho}$.

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

7.3.4 Concluding game theory remarks

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 blockchain will end after a certain number K of time steps? The answer is *no*. Theorem 7.16 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[21]. The crucial observation is that even if different players have different values of K "in their heads", their equilibrium strategies are still the same.

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

Synchrony and network delays. We have also considered a generalized game that additionally models network delays and clock asynchrony, and performed an analysis to characterize parameters for which equilibrium can be achieved in this setting too. As the main ideas are similar to the analysis of the SpaceMint game, we refer to Appendix E for details of of the generalized analysis.

7.3.5 Remarks on selfish mining

The "selfish mining" attack against Bitcoin (proposed by [20]) is as follows: a miner extends his own blocks to make a parallel chain to the main block chain, without announcing his blocks to the network.

[21]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 7.16.

The attack is successful if the parallel chain eventually becomes higher-quality than the main chain: then, the miner announces the entire parallel chain, and honest miners will be incentivized to start mining on the newly announced chain. [20] showed that even for miners with significantly less than 50% of the network's resources, this attack pays off in expectation. This overturned the prior, widely held belief that Bitcoin was robust to attacks by adversaries with a minority of network computing power.

It turns out that SpaceMint is far more robust than Bitcoin against selfish mining attacks. The Bitcoin protocol prescribes that miners attempt to extend the longest chain they have seen so far, at any point in time. In contrast, SpaceMint prescribes that in each time-step, miners attempt to extend the best chain seen during the preceding time-step. The discrete nature of SpaceMint mining is a key difference from Bitcoin, for robustness against selfish mining. The success probability of selfish mining in Bitcoin depends on propagating "selfish" blocks across the network faster than "honestly mined" blocks, since which one is seen first will determine which one is extended by more miners. In SpaceMint, most honestly mined blocks announced in a time-step will be seen by most miners by the start of the following time-step,[22] so intuitively, this "race" condition is irrelevant and this makes selfish mining harder.

The SpaceMint game (Definition 7.13) explicitly models the possibility of selfish mining, and our game-theoretic analysis shows SpaceMint's robustness against such attacks.

7.3.6 Remarks on 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 blockchain 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 (or, perhaps more realistically, to leave the game). Moreover, P can prevent certain transactions from ever getting into the blockchain, by refusing to extend blocks which contain these transactions – thus, P can mine multiple blocks per time step without ever being punished. This attack was first analyzed by [28] 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. Moreover, once such an attack is detected by the community, the value of the currency will plummet, so the expected monetary benefit to attackers would be arguably low or even negative. The improbableness of a 51% attack by a mining pool is supported by Bitcoin's history: whenever mining pools (e.g. `ghash.io`) have approached 50% of Bitcoin computing power, self-interested miners quickly started leaving the mining pool in order to avoid destabilizing the currency.

8 Conclusion

We have presented SpaceMint, a cryptocurrency that uses efficient proofs of space instead of energy-intensive proofs of work to maintain a public transaction ledger. We have constructed a variant of proofs of space that is suitable for the cryptocurrency setting, and proposed a novel block chain structure and new transaction types to address some of the issues of other existing cryptocurrencies. We have also shown that maintaining a public ledger could be much more

[22]Assuming a reasonably reliable network. Honest miners mine and send out their blocks at the start of each time-step.

efficient using proofs of space using our prototype. Finally, we have presented the first formal modeling of cryptocurrency as an extensive game, and our analysis shows that honest mining satisfies strong equilibrium properties in said model, including resistance to selfish mining.

Acknowledgements

We would like to thank Andrew Miller and Bram Cohen for bringing the challenge grinding attack to our attention. We thank Srini Devadas for useful feedback on draft versions of this paper, and Ethan Heilman for an interesting discussion about costs of modern Bitcoin mining.

References

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

[2] Bitcoin startups. `https://angel.co/bitcoin`.

[3] Burstcoin. `http://burstcoin.info`.

[4] Crypto-currency market capitalizations. `http://coinmarketcap.com`.

[5] Raspberry Pi. `www.raspberrypi.org`.

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

[7] The New York Times: News about Bitcoin. `http://topics.nytimes.com/top/reference/timestopics/subjects/b/bitcoin`.

[8] Wired: Bitcoin. `http://www.wired.com/tag/bitcoin`.

[9] 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.

[10] 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.

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

[12] D. Chaum. Advances in cryptology: Proceedings of crypto 82. pages 199–203, Boston, MA, 1983. Springer US.

[13] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10, Sept 2013.

[14] 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.

[15] 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.

[16] 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`.

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

[18] 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.

[19] Ethereum. Problems. https://github.com/ethereum/wiki/wiki/Problems.

[20] I. Eyal and E. G. Sirer. Financial cryptography and data security: 18th international conference, fc 2014, christ church, barbados, march 3-7, 2014, revised selected papers. pages 436–454, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[21] B. Gates and M. Gates. Mobile banking will help the poor transform their lives, Jan 2015. 2015 Gates Annual Letter. https://www.gatesnotes.com/2015-annual-letter?page=3.

[22] M. Gimein. Virtual Bitcoin Mining Is a Real-World Environmental Disaster, April 2013. http://www.bloomberg.com/news/articles/2013-04-12/virtual-bitcoin-mining-is-a-real-world-environmental-disaster.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] M. Mainelli and C. von Gunten. Chain of a lifetime: How blockchain technology might transform personal insurance, Dec 2014. Z/Yen Group, Long Finance.

[30] A. Miller, December 2015. Personal communication.

[31] 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.

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

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

[34] K. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In *Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET*, pages 280–285, June 2014.

[35] 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.

[36] 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.

[37] M. Rosenfeld. Analysis of hashrate-based double spending. *CoRR*, abs/1402.2009, 2014.

[38] The Economist. The great chain of being sure about things, Oct 2015. http://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable.

[39] The World Bank Group. Crowdfunding in emerging markets: Lessons from east african startups, 2015. License: Creative Commons Attribution CC BY 3.0.

[40] 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`.

A Proof-of-Space parameters

The two PoSpace constructed in [17] have the following efficiency/security properties. Below t_{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 number n of nodes of the underlying graph, an honest prover $\mathcal{P}$ must dedicate

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

bits of storage ($n \cdot L$ for the labels, and almost the same for the values required to efficiently open the Merkle tree commitment). A typical value for L is 256, then $N = 512 \cdot n$.

Proposition A.1 ([17] 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 the proof execution phase with probability $\geqslant 2^{-\Theta(k_p)}$, then $\mathcal{P}$ either stores $\Theta(N)$ bits (i.e., almost as much as an honest prover) or runs in time $\Theta(n \cdot \log\log(n) \cdot t_{\mathsf{hash}})$ (i.e., the time required for initialization).*

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

Proposition A.2 ([17] 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 $\geqslant 2^{-\Theta(k_p)}$, 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 have to set $k_{cv} = \lambda \cdot \log(n)$ where λ 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 §2. We not only discuss Burstcoin because it is relevant related work; looking at its design also 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 initializes his disk space as follows: he picks a nonce μ and an account identifier (which is a hash of a public key) Id, and then computes iteratively 4097 values $x_0, x_1, \ldots \in \{0,1\}^{256}$ as

$$x_0 = H(Id, \mu) \quad \text{and} \tag{6}$$

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

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[23]. 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 blockchain[24]. 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, μ), 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, $i \leqslant 10$). This would require hashing only at most 50 blocks[25]. 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 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[26], such time-memory trade-offs seem practical[27]. We remark that 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.*
>
> —`http://burstcoin.info/intro`

Grinding and extending multiple chains. The two main challenges we had to overcome when designing SpaceMint were attacks based on grinding

[23]Note that in equation (6), a freshly computed block x_i is *pre*pended 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.

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

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

[26]`https://en.bitcoin.it/wiki/Mining_hardware_comparison`

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

and mining multiple chains. (The problem with time-memory trade-offs was solved in the Proofs of Space [17] 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 PrevGenSig and PrevBlkGenerator. The naming indicates that PrevGenSig corresponds to the value 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 PrevBlkGenerator, so we 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).

C Challenge Grinding Attacks

In this section we describe the challenge grinding attack (which was communicated to us by Andrew Miller [30]), and our solution.

Recall (from §4.7) that the quality of a block chain in SpaceMint is defined by:

$$\mathsf{QualityPC}(\varphi_0,\ldots,\varphi_i) = \sum_{j=1}^{i} \log(\mathcal{N}(v_j)) \cdot \Lambda^{i-j} \ . \quad (7)$$

Intuitively, this is the sum of logarithms of qualities of blocks in the chain, with the discount factor Λ ensuring that more recent blocks weighted slightly more. For the purpose of this section, the factor Λ is unimportant, so we omit it. Then, notice that an equivalent measure is the *product* of block qualities:

$$\mathsf{QualityPC}^{\times}(\varphi_0,\ldots,\varphi_i) = \prod_{j=1}^{i} \mathcal{N}(v_j) \ . \quad (8)$$

A natural question is: *why take the product, rather than the sum?* It turns out that there is a possible attack in the case that QualityPC takes a sum, i.e.

$$\mathsf{QualityPC}^{+}(\varphi_0,\ldots,\varphi_i) = \sum_{j=1}^{i} \mathcal{N}(v_j) \ , \quad (9)$$

which is mitigated by instead taking a product. In a nutshell, the basic intuition for this is that the geometric mean is more robust against outliers than the arithmetic mean.

We now describe the attack against the sum-based quality function.

Challenge grinding attack. To use a space commitment (pk,γ) for computing a proof for block i, we must use the challenge computed as a hash of block $i-\Delta$ as $c := \mathsf{hash}(pk,\varphi_{i-\Delta})$. It is important that Δ is at least the number of time-steps required to reach consensus with overwhelming probability, so that each miner (i.e., each public key) gets exactly one chance at mining a block at time-step i. Thus, it is not possible to get an unfair advantage by spending computational power to "try many different challenges and pick the best one". In a *Challenge grinding attack*, the adversary does exactly this, by producing long enough ($>\Delta$) sequences of blocks that he controls his own future challenges.

Let $\mathcal{A}$ be a challenge grinding adversary who controls space of size N. $\mathcal{A}$ splits up his space into as many separate space commitments as possible:[28] let $(pk_1,\gamma_1),\ldots,(pk_m,\gamma_m)$ be his space commitments, which together comprise space $N=\sum_{j=1}^{m} N_{\gamma_j}$.

If this adversary "honestly" mined t consecutive blocks (by taking the highest-quality proof φ_i among all his m space commitments, at each time-step i), then the expected quality of the resulting chain is

$$\mathbb{E}[\mathsf{QualityPC}^{+}(\varphi_0,\ldots,\varphi_t)] = \sum_{i=1}^{t} \mathbb{E}[\mathcal{N}(v_i)] = t\cdot N$$

according to the *sum-based* quality function (9). (Re-

[28]Subject to the minimum size requirement for a space commitment.

call that by construction, the expectation of $\mathcal{N}(v_i)$ is N, where v_i is the quality of proof φ_i.)

In fact, for a sum-based quality function, $\mathcal{A}$ can do significantly better than this for all sufficiently long chains. First, he partitions the block indices $\{1,\ldots,t\}$ into disjoint pairs $(i, i+\Delta)$. For simplicity, suppose $t = 2\Delta$ and let the pairs be

$$(1, 1+\Delta),\ \ldots,\ (\Delta, 2\Delta)\ .$$

Then, for each pair $(i, i+\Delta)$ and every space commitment (pk_j, γ_j), $\mathcal{A}$ computes a challenge $c_{i,j} := \mathsf{hash}(pk_j, \varphi'_{i,j})$, where $\varphi'_{i,j}$ is the proof corresponding to commitment (pk_j, γ_j) at time-step i. At this point, $\mathcal{A}$ has computed m possible challenges $c_{i,1},\ldots,c_{i,m}$ for each time-step $i+\Delta$. He can choose the best, $c_i^* \in \{c_{i,1},\ldots,c_{i,m}\}$, such that the quality of his block at position $i+\Delta$ is maximized. Informally, this is like having just one challenge, but m times more space.

Now, for a pair $(i, i+\Delta)$, the expected quality of $\mathcal{A}$'s proof in time-step $i+\Delta$ is increased to $N \cdot m$. Note that this strategy actually *decreases* the expected quality in the earlier time-step i compared to the honest strategy, since instead of optimizing for quality at position i, $\mathcal{A}$ optimizes for quality at position $i+\Delta$: the expected quality $\mathcal{A}$'s proof in time-step i is decreased to N/m.

With this approach, $\mathcal{A}$ generates a chain where half the blocks have quality around $N \cdot m$ and the other half N/m, so the expected chain quality is

$$\begin{aligned}\mathbb{E}[\mathsf{QualityPC}^{+}(\varphi'_0,\ldots,\varphi'_t)] &= \sum_{i=1}^{t} \mathbb{E}[\mathcal{N}(v_i)] \\ &\approx t \cdot \frac{(N/m) + N \cdot m}{2} \\ &> tmN/2\ .\end{aligned}$$

Summing up, $\mathcal{A}$ just using space N that was initialized once, generated a chain of quality that would require total space over $mN/2$ if generated by honest mining. This $m/2$ factor can be even further improved by optimizing over blocks separated not just by Δ positions, but by $k \cdot \Delta$ positions: e.g. blocks i and $i+\Delta$ can be used to generate t^2 challenges and pick the best proof for block $i+2\Delta$, yielding a factor $m^2/3$ improvement.[29]

If the $\mathsf{QualityPC}$ function is product-based (or equivalently, based on the sum of logarithms), as in (7), the attack outlined above no longer works. The reason is that the (expected) quality of our two blocks j and $j+\Delta$ is N/t and $N \cdot t$, respectively, and thus the product is N^2 (which is the same we get in the honest mining process, where each block has expected quality N).

Although we have eliminated the specific space-grinding attack described above, we remark that it is still possible to get some minor advantage by challenge grinding even with a product-based measure of quality. Recall that the attack generated m challenges at block i (using the m space commitments), and then picked the challenge which gave the best quality for block $i+\Delta$. Instead of only doing this, an adversary could check which challenge (among the m candidates) gives the highest value for the product of block qualities at position j and $j+\Delta$. Using this strategy, the expected quality of these blocks is $N^2 \cdot \log(m)$, which is a factor $\log(m)$ higher than the N^2 we get by honest mining (but still much smaller than the $m/2$ factor in the original attack).

Before we explain how to solve the above problem, let us observe that the reason that challenge grinding is possible in the first place is the variance in the quality of a proof: for a space commitment (pk, γ), the expected quality of a proof is N_γ, but for any $\alpha > 1$, the quality will be higher than $\alpha \cdot N_\gamma$ with probability roughly $1/\alpha$. This variance is necessary as we need the expected quality of the best proof found amongst many commitments $(pk_1, \gamma_1), \ldots, (pk_m, \gamma_m)$ to be the sum $\sum_{i=1}^{m} N_{\gamma_i}$ of all the spaces.

We can decrease the advantage of challenge grinding (over honestly mining) by lowering the variance of the quality of the proofs. As mentioned above, the variance of proof quality is an important feature that we cannot simply remove; however, we can cluster proofs together so that the advantage that challenge grinding gives "amortizes" over many proofs. One way to use the same challenge for several consecutive blocks. Concretely, we introduce a new parameter δ which specifies how many blocks are generated using the same challenge. The challenge for block i is no

[29] More generally, we can get $m^k/(k+1)$ for any $k \in \mathbb{N}$. The computational cost of the attack grows as t^k.

longer computed as

$$c := \mathsf{hash}(pk, \varphi_{i-\Delta})$$

but as

$$c := \mathsf{hash}(pk, \varphi_{i-\Delta-(i \bmod \delta)})$$

Now, a challenge grinding adversary must try to "optimize" δ proofs at once, which will give a much lower advantage than being able to "grind" the proof for every block individually. We suggest to set $\delta = 10$, which seems more than sufficient to prevent challenge grinding (we do not recommend using much larger δ, since that can make generating long forks easier, as we discuss in Appendix D).

[AK]minspace should be maybe 100?

D Parameter setting and interplay

We have defined several parameters which control the efficiency and security of SpaceMint. Some of those parameters cannot simply be seen as security parameters (where increasing the parameter means more security at the price of decreased efficiency) as there's a delicate interplay between them, where changing some parameter increases some security property, at the prize of decreasing another. We discuss the importance of the most important parameters on the most relevant attacks below, a summarized view is given in Table 1. The parameters are

$\mathsf{time}[1]$ which specifies the time (in minutes) between blocks.

$\delta[10]$ which specifies for how many blocks the same challenge is used.

$\Delta[50]$ which specifies that the challenge for a block is computed as the hash of the block at least Δ blocks in the past (and at most $\Delta + \delta$).

$\Lambda[0.99999]$ specifies how the weight of blocks – when computing the quality of a chain – degrades for older blocks.[30]

[30]With this Λ the weight drops by 50% every 69314 blocks, or 48 days (as $\Lambda^{69314} \approx 0.5$).

$\mathsf{minspace}[100]$ specifies (in GB) the minimal size of space one must dedicate to start mining.

where the number in the $[\cdot]$ brackets indicates the value for the parameter as set in our suggested instantiation.

Challenge grinding. We discussed challenge grinding in detail in Appendix C and how setting the parameter δ sufficiently high prevents this attack. We also discussed how challenge grinding becomes more successful the more space commitments one can generate given some fixed space, thus setting the $\mathsf{minspace}$ parameter up also makes the attack harder.

Extending multiple chains. We must ensure that for a miner it's rational to only announce blocks extending one chain, and this chain should be the chain of highest quality known to the miner. Ensuring that a miner only announces one block is done by penaltizing miners otherwise §4.5.2. If we have fork, we assume that with high probability this penalising is sufficient to make sure that one of them will "die" within at most Δ blocks, as otherwise miners will get different challenges for the two chains, which (assuming the miners are rational) will slow down consensus. Clearly, increasing Δ makes this event less likely (at an exponential rate).

Short forks by space reuse. The security of SpaceMint relies on the assumption that it's not possible to reuse space for mining. As we can compute the challenges for up to $\Delta + \delta$ blocks in advance, for reusing space even just twice, one would have to instantiate the space in less than

$$\mathsf{time}(\Delta + \delta) = 1(50 + 10)/2 = 30 \text{ minutes}$$

This is far off from the ≈ 200 minutes required to instantiate 100GB of space, which is the minimum we suggest (cf. Figure 6).

Long forks by space reuse. The situation is different if we consider an adversary who wants to come up with long range fork (and not extend the current chain) because (as specified in Equation (4)) more recent blocks contribute more to the quality of a chain. We shortly sketch how this attacks works: Let cur

Table 1: A summary on how different parameters influence different security properties of SpaceMint as discussed in Appendix D. An arrow ↑ (↓) means increasing (decreasing) this parameter will increase the security against the corresponding attack, ⇑ means that increasing this parameter has a major influence on the security against this attack. The ↑' arrows do not refer directly to minspace, but rather the time required to initialise the minimal allowed space, which scales linear in minspace as shown in Figure 4. Decreasing time makes the scheme only more secure, but setting it very low will force miners to dedicate more computation. Also setting minspace up will make the scheme more secure, but a high minspace will lower its usability, as parties with small space will not be able to participate.

parameters	time	Δ	δ	Λ	minspace
range/unit	$\mathbb{N}^+/min$	$\mathbb{N}^+$	$\mathbb{N}^+$	$[0,1]$	$\mathbb{N}^+/GB$
suggested	1	50	10	0.99999	100
attacks					
Challenge Grinding	-	-	⇑	-	↑
Extending Multiple Chains	-	⇑	-	-	-
Short Forks by Space Reuse	↓	↓	↓	-	↑'
Long Forks by Space Reuse	↓	↓	↓	⇑	↑'
Long Forks by Space Decrease	-	-	-	⇓	-

denote the index of the current block. An adversary first extends the current chain up to some block cur + low by simply using the space he has available (so, the low new blocks will be of low quality compared to the actual chain). Then the adversary extends this chain to block cur + low + high with high quality blocks, i.e., somewhat better than the blocks of the actual chain. As the adversary has less space than the total space contributed by all miners, he must re-instantiate the space many times while generating these blocks. This will take a lot of time, but the advertsry has time(low + high) minutes to generate these high blocks, so it will be possible by setting low high enough.

How large high must be depends on how fast the influence of blocks degrades as specified by the parameter Λ, concretely, it will be in the order of $1/(1-\Lambda)$.[31] Every time the adversary re-initalizes its space, it can generate challenges for the next $\Delta + \delta$ blocks. Assuming re-initialization takes about Tinit minutes, generating a chain while re-sampling even just once for every block (which will allow to generate blocks that look as if they had been mined with twice the space that is available to the adversary, this will only be sufficient if the adversary has more than half the space of the honest miners available) is by a factor

$$\beta = \frac{\mathsf{Tinit}}{(\Delta+\delta)\mathsf{time}}$$

slower than the speed at which the actual chain grows, which means we must set

$$\mathsf{low} \approx \mathsf{high}/\beta \approx \frac{1}{(1-\Lambda)} \cdot \frac{\mathsf{Tinit}}{(\Delta+\delta)\mathsf{time}}$$

to finish the fork on time. For minspace = 100(GB) we have Tinit ≈ 200 so low $\approx 100000 \cdot 200/60$ minutes. Thus, even with our rough analysis, this implies that a fork would have to go back way over half a year. Of course even such a long fork constitutes an attack, and thus some mechanisms to handle very long forks must be in place. This could either be some type of checkpoints, but we believe that for such long forks relying on *weak subjectivity*[32] should be sufficient.

Long forks by space decrease. The above attack assumes $\Lambda < 1$. If $\Lambda = 1$, then we have no

[31] As the contribution of blocks that far in the past is just a small fraction ($\approx 1/e$) of the most recent blocks.

[32] `https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/`

degradation to the contribution of chain quality as blocks are further in the past. This is problematic as it allows to generate a chain, stating from the genesis block, using space that is only as large as the *average* amount space that has been available by the miners over the entire lifetime of the currency, which can be much lower than the currently used space. But then, also in this case we could rely on weak subjectivity.

Overtaking the chain. The adversary may try to only extend his own blocks, and attempt to take overtake the main chain. In this case, the adversary will only get rewarded for those blocks if the quality of his chain eventually becomes greater than the quality of the chain mined by the rest of the network. We say that the attack is *successful* if the blocks thus mined by the adversary eventually become part of the highest-quality chain.

A successful overtake would enable an adversary to do a *double-spending* attack (by putting a transaction transferring money to someone in the "main" block chain, and later overwriting the transaction when his self-mined block chain overtakes the main one). A successful long-fork attack is also a necessary component of a *selfish mining* attack (described in §7.3.5).

Recall the quality of a block (from §4.6):

$$\mathsf{Quality}(pk, \gamma, c, a) = D_{N_\gamma}(\mathsf{hash}(a))$$

where $D_N(\mathsf{hash}(a))$ is defined as

$$D_N(\mathsf{hash}(a)) := \left(\mathsf{hash}(a)/2^L\right)^{1/N}$$

We model the hash function as a random oracle, so $\mathsf{hash}(a)/2^L$ is distributed as $r'/2^L$ for random $r' \leftarrow \{0,1\}^L$. This distribution is statistically close to randomly sampling $r \leftarrow [0,1]$: that is,

$$\Delta\left(\{r'/2^L\}_{r'\leftarrow\{0,1\}^L},\ \{r\}_{r\leftarrow[0,1]}\right) = 2^{-L}$$

where Δ denotes statistical distance. Henceforth, our analysis considers only the latter distribution, which we denote by D^*_N.

$$D^*_N \sim \{r^{1/N}\}_{r\leftarrow[0,1]}$$

Let $(\varphi_0, \ldots, \varphi_M)$ be a proof chain, where each proof sub-block φ_j contains a proof $(pk_j, \gamma_j, c_j, a_j)$ and the quality of the jth block is $v_j \leftarrow D^*_{N_j}$. The quality of a block chain (cf. §4.7) is given by

$$\mathsf{QualityPC}(\varphi_0, \ldots, \varphi_M) = \sum_{j=1}^{M} \log(\mathcal{N}^*(v_j)) \cdot \Lambda^{M-j}$$

where $\Lambda \in [0,1]$ and $\mathcal{N}^*$ is defined as

$$\mathcal{N}^*(v) = \min\{N \in \mathbb{N} : \Pr_{w\leftarrow D^*_N}[v < w] \geqslant 1/2\}. \quad (10)$$

Lemma D.1. *$\mathcal{N}^*(v) = -1/\log(v)$.*

Proof. By definition of D^*_N, increasing N means $\Pr_{w\leftarrow D^*_N}[v < w]$ increases. Therefore, (10) implies

$$N^*(v) = N \quad \text{s.t.} \quad \Pr_{w\leftarrow D^*_N}[v < w] = 1/2.$$

Also by definition of D^*_N, it holds that

$$\begin{aligned}\Pr_{w\leftarrow D^*_N}[v < w] &= \Pr_{r\leftarrow[0,1]}[v < r^{1/N}] \\ &= \Pr_{r\leftarrow[0,1]}[v^N < r]\end{aligned}$$

Setting the above probability to 1/2 and solving for N gives $N = -1/\log(v)$. The claim follows. □

Suppose, without loss of generality, that the adversary begins his long-fork attack at time 0. Let N_{adv} be the amount of space the adversary has, and let N_{honest} be the amount of space the rest of the network has. For any $M \in \mathbb{N}$, let $\mathcal{E}_M$ denote the event that after M blocks, the adversary's block chain is of higher quality than the honest miners' block chain. Then, by definition of $\mathsf{QualityPC}$, $\Pr[\mathcal{E}_M]$ equals

$$\Pr\left[\sum_{j=1}^{M}\Big(\log\left(\mathcal{N}^*(\hat{v}_j)\right) - \log\left(\mathcal{N}^*(v_j)\right)\Big)\cdot\Lambda^{M-j} > 0\right], \quad (11)$$

where $\hat{v}_j, v_j$ are random variables representing the quality of the jth block of the adversary and the network respectively, and the probability is taken over $\hat{v}_j$ and v_j. Using Claim D.1 to substitute for $N^*(\cdot)$ and rearranging, we obtain that $\Pr[\mathcal{E}_M]$ equals

$$\Pr\left[\sum_{j=1}^{M}(\log(-\log(v_j)) - \log(-\log(\hat{v}_j)))\cdot\Lambda^{M-j} > 0\right] \quad (12)$$

For $j \in [M]$, we define new random variables diff_j and $\mathsf{diff}_j^{\Lambda,M}$ as follows:

$$\mathsf{diff}_j = \log(-\log(v_j)) - \log(-\log\hat{v}_j))$$

Table 2: **Bounding the probability of a successful overtake of the chain:** p is the probability of a successful overtake, ξ is the adversary's proportion of the network disk space, and the tabulated values are fork length (in blocks).

	$\Lambda = 0.99999$					$\Lambda = 0.99998$					$\Lambda = 0.99997$				
$\xi \setminus p$	2^{-8}	2^{-16}	2^{-32}	2^{-64}	2^{-128}	2^{-8}	2^{-16}	2^{-32}	2^{-64}	2^{-128}	2^{-8}	2^{-16}	2^{-32}	2^{-64}	2^{-128}
0.1	3	5	10	19	37	3	5	10	19	37	3	5	10	19	37
0.25	10	19	37	74	148	10	19	37	74	148	10	19	37	74	148
0.33	24	47	93	186	371	24	47	93	186	373	24	47	93	186	374
0.4	68	136	271	543	1092	68	136	272	546	1104	68	136	273	549	1116
0.45	277	554	1114	2254	4614	277	557	1127	2307	4852	278	561	1140	2365	5130

$$\mathsf{diff}_j^{\Lambda,M} = \mathsf{diff}_j \cdot \Lambda^{M-j}$$

Note that both diff_j and $\mathsf{diff}_j^{\Lambda,M}$ have support $[-1,1]$. We can now write

$$\Pr[\mathcal{E}_M] = \Pr\left[\sum_{j=1}^{M} \mathsf{diff}_j^{\Lambda,M} > 0\right]. \qquad (13)$$

Theorem D.2.

$$\Pr[\mathcal{E}_M] \leqslant \exp\left(-\frac{1}{2M}\cdot \mathbb{E}\left[\mathsf{diff}_1\right]^2 \cdot \left(\sum_{j=0}^{M-1} \Lambda^{2j}\right)^2\right).$$

Proof. Applying a Hoeffding bound to the right-hand side of (13), we obtain:

$$\Pr[\mathcal{E}_M] \leqslant \exp\left(-\frac{1}{2M}\cdot \left(\sum_{j=1}^{M} \mathbb{E}\left[\mathsf{diff}_j^{\Lambda,M}\right]\right)^2\right) \qquad (14)$$

By definition of diff_j and $\mathsf{diff}_j^{\Lambda,M}$,

$$\begin{aligned}\mathbb{E}\left[\mathsf{diff}_j^{\Lambda,M}\right] &= \Lambda^{M-j}\cdot \mathbb{E}\left[\mathsf{diff}_j\right] \\ &= \Lambda^{M-j}\cdot \mathbb{E}\left[\mathsf{diff}_1\right]\end{aligned}$$

where the second equality follows from the fact that the diff_j are identically distributed for all j. Substituting this expression into (14) and using linearity of expectation, we obtain the inequality given in the theorem statement. □

The values in Table 2 were calculated by using Mathematica to solve (for M) the expression given in Theorem D.2.

Remark. The dynamics of long-fork attacks become slightly different if there is more than one (independent) adversarial party. In this case, the probabilities shown in Table 2 are still accurate as long as no adversarial party owns more space than all the honest parties combined, *even if the sum of the space owned by the adversarial parties is greater than 50% of total space.*

E Modeling synchrony delays

For simplicity, the SpaceMint game, as presented above, models the mining process as if players had perfectly synchronized clocks and there were no network delays or errors. However, our analysis can be extended to model network unreliability and asynchrony of clocks. We can prove that honest mining is an ε-sequential Nash equilibrium where the value of ε depends on a *network reliability parameter* and a *synchrony parameter*. As long as network delays are small and players' clocks are "approximately" in sync, ε is also small. A more formal statement and proof sketch are given below.

Definition E.1 (Augmented SpaceMint game). *Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ be a proof of space. For any number of players $N \in \mathbb{N}$, any number of time steps $K \in \mathbb{N}$, any consensus-delay $\Psi \in \mathbb{N}$, and any reward function $\rho\colon \mathbb{N} \to \mathbb{N}$, the extensive game*

$$\mathsf{AugSpaceMint}_{\Pi,K,\rho}^{\eta,\upsilon} = \langle N, H, f_{\mathcal{C}}, \vec{\mathcal{I}}, \vec{u}\rangle$$

is defined almost exactly like $\mathsf{SpaceMint}_{\Pi,K,\rho}$, except with two additional parameters:

1. *$\eta \in [0,1]$, the* network reliability parameter*, which models the probability of network failure: specifically, the probability that a player tries to announce a block in a time-step j, but some or*

all other players do not receive it in time-step j, due to network unreliability;

2. υ, *the* synchrony parameter, *which models the probability (for any distinct $i, i' \in [N]$) that a block announced by player i in time-step j will* not *be seen by player i' in the same time-step, due to clock asynchrony.*

For a probability p, let Ber_p be the Bernoulli distribution with success probability p. Let Ber_p^N be the distribution of N-tuples where each element is sampled independently from Ber_p. The game $\mathsf{AugSpaceMint}^{\eta,\upsilon}_{\Pi,K,\rho}$ differs from $\mathsf{AugSpaceMint}_{\Pi,K,\rho}$ (only) as follows:

- *At each time-step $j \in [K]$, in addition to announcing the beacon value, the Chance player samples $2N$ bits as follows:*
$$\vec{b}^\eta = (b^\eta_{1,j}, \ldots, b^\eta_{N,j}) \leftarrow \mathsf{Ber}^N_{1-\eta}$$
$$\vec{b}^\upsilon = (b^\upsilon_{1,j}, \ldots, b^\upsilon_{N,j}) \leftarrow \mathsf{Ber}^N_{1-\upsilon}$$
- *The information sets are oblivious to the bits $b^\eta_{i,j}, b^\upsilon_{i,j}$. Formally, 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.*
  - $$\left[\left((\mathcal{T}_1, \mathcal{A}_1), \ldots, (\mathcal{T}_N, \mathcal{A}_N), (a_\mathcal{C}, \vec{b}^\eta, \vec{b}^\upsilon)\right)\right]_i = \Big\{\left((\mathcal{T}'_1, \mathcal{A}'_1), \ldots, (\mathcal{T}'_N, \mathcal{A}'_N), (a'_\mathcal{C}, \vec{b}'^\eta, \vec{b}'^\upsilon)\right) \in H : \mathcal{T}_i = \mathcal{T}'_i \wedge \mathcal{A}_i = \mathcal{A}'_i \wedge a_\mathcal{C} = a'_\mathcal{C} \wedge \forall i' \neq i,\ \left(\mathcal{A}_{i'} = \mathcal{A}'_{i'} \vee b^\eta_{i',j} \cdot b^\upsilon_{i',j} = 0\right)\Big\}$$
  - $$\Big[\Big(h, \left((\mathcal{T}_1, \mathcal{A}_1), \ldots, (\mathcal{T}_N, \mathcal{A}_N), (a_\mathcal{C}, \vec{b}^\eta, \vec{b}^\upsilon)\right), \left((\mathcal{T}'_1, \mathcal{A}'_1), \ldots, (\mathcal{T}'_N, \mathcal{A}'_N), (a'_\mathcal{C}, \vec{b}'^\eta, \vec{b}'^\upsilon)\right)\Big)\Big]_i = \Big\{\Big(h'', \left((\mathcal{T}''_1, \mathcal{A}''_1), \ldots, (\mathcal{T}''_N, \mathcal{A}''_N), (a''_\mathcal{C}, \vec{b}''^\eta, \vec{b}''^\upsilon)\right), \left((\mathcal{T}'''_1, \mathcal{A}'''_1), \ldots, (\mathcal{T}'''_N, \mathcal{A}'''_N), (a'''_\mathcal{C}, \vec{b}'''^\eta, \vec{b}'''^\upsilon)\right)\Big) \in H : h \sim_i h'' \wedge \mathcal{T}_i = \mathcal{T}''_i \wedge \mathcal{T}'_i = \mathcal{T}'''_i \wedge \mathcal{A}_i = \mathcal{A}'_i \wedge \mathcal{A}'_i = \mathcal{A}'''_i \wedge a_\mathcal{C} = a''_\mathcal{C} \wedge a'_\mathcal{C} = a'''_\mathcal{C} \wedge \forall i' \neq i, \mathcal{A}_{i'} = \mathcal{A}''_{i'} \wedge \left(\mathcal{A}'_{i'} = \mathcal{A}'''_{i'} \vee b^\eta_{i',j} \cdot b^\upsilon_{i',j} = 0\right)\Big\},$$

  where h and h'' are histories of equal length and $j = |h| + 1$.
- *For any history h, let $\mathcal{A}^*(h)$ be the set of all blocks announced by any player i in history h* in a time-step j such that $b^\eta_{i,j} = b^\upsilon_{i,j} = 1$:
$$\mathcal{A}^*(h) = \left\{ B : \begin{array}{c} \exists i \in [N], \mathcal{A}' \text{ s.t. } (\cdot, B) \in \mathcal{A}' \text{ and} \\ \text{player } i \text{ took action } (\cdot, \mathcal{A}') \text{ at} \\ \text{time-step } j, \text{ and } b^\eta_{i,j} = b^\upsilon_{i,j} = 1 \end{array} \right\}$$
Let $\mathsf{blocks}^*(h)$ denote the sequence of "winning blocks" at any given history h:
$$\mathsf{blocks}^*(h) = \operatorname*{arg\,max}_{\vec{B} \in (\mathcal{A}^*(h))^{|h|}} \left(\mathsf{QualityPC}(\vec{B}, \mathfrak{C}(h))\right).$$
Let $\mathsf{blocks}^*_\ell(h)$ denote the ℓth block in the chain. Let $\mathsf{win}^*_\ell(h)$ be the player who announced the winning block $\mathsf{blocks}^*_\ell(h)$ for index ℓ.[33]
- *The players' utility functions are defined as follows: for a terminal history h of length K, the utility of player i is given by $u_i(h) =$*
$$\sum_{\ell \in [K-\Psi]} \delta_{i,\mathsf{win}^*_\ell(h)} \cdot \mathsf{one}_\ell(i,h) \cdot \rho(\mathsf{blocks}^*_\ell(h)),$$
where $\delta_{i,j}$, one_ℓ are defined as in Definition 7.13.

Theorem E.2. *Let $\Pi = \{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$ be a proof of space. Recall the definition of $(\vec{\alpha}, \vec{\mu})$ from Theorem 7.16. In the original formulation of the SpaceMint game, it was possible to associate a unique blockchain with each information set, and the definition of $(\vec{\alpha}, \vec{\mu})$ depended on this fact. In the augmented SpaceMint game, this property no longer holds, because sometimes a player will be unaware of a block which was announced by another player in the preceding time-step. Nonetheless, it is a well-defined strategy for the players to compute their strategies α_i according to $(\vec{\alpha}, \vec{\mu})$* as if they were playing in the original SpaceMint game instead of the augmented game. *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 the assessment of $\mathsf{AugSpaceMint}^{\eta,\upsilon}_{\Pi,K,\rho}$ defined in the above-described way.*

[33]We assume that the winning block is unique at each time-step: that is, $\mathsf{Quality}$ imposes a total order on blocks. 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.

Let

$$\xi = \frac{\max_{i \in [N]} t_i}{\sum_{i \in [N]} t_i}$$

be the maximum fraction of space possessed by a single player, and suppose $\xi < 0.5$. *Then* $(\vec{\alpha}^*, \vec{\mu}^*)$ *is an* ε*-sequential Nash equilibrium of* $\mathsf{AugSpaceMint}^{\eta,\upsilon}_{\Pi,K,\rho}$ *where*

$$\varepsilon = \eta + \upsilon + \exp\left(-\frac{1}{2K} \cdot \mathbb{E}\left[\mathsf{diff}_1\right]^2 \cdot \left(\sum_{j=0}^{K-1} \Lambda^{2j} \right)^2 \right),$$

Λ *is the discount factor defined in §4.7 and* diff_1 *is defined as in §D.*

Proof sketch. Consider the bits $b^\eta_{1,j}, \dots, b^\eta_{N,j}$ and $b^\upsilon_{1,j}, \dots, b^\upsilon_{N,j}$ that are sampled by the Chance player in each time-step j. In any given time-step j, by definition, player i's utility is the same as in the original (i.e. not augmented) SpaceMint game if $b^\eta_{i,j} = b^\upsilon_{i,j} = 1$, and zero otherwise. For $i' \neq i$, player i's utility is the same as in the original SpaceMint game if $b^\eta_{i,j} = b^\upsilon_{i,j} = 1$, and *increased* on expectation otherwise. Since we are seeking to *upper-bound* the expected utility of deviations from the equilibrium strategy, we will disregard the impact of $b^\eta_{i',j}, b^\upsilon_{i',j}$ when analyzing the strategy of player i, and simply assume $b^\eta_{i',j} = b^\upsilon_{i',j} = 1$ for all $i' \neq i$ and all j.

The values $b^\eta_{i,j}, b^\upsilon_{i,j}$ are sampled independently for different time-steps j, and only affect the utility of the player in the current time-step j. Hence, player i's strategy in time-step j need not depend on the values $b^\eta_{i,j'}, b^\upsilon_{i,j'}$ for $j' \neq j$. We conclude that when analyzing the strategy of any given player i at any given time-step j, we need only consider the two variables $b^\eta_{i,j}, b^\upsilon_{i,j}$. The "bad cases" are $b^\eta_{i,j} = 0$ and $b^\upsilon_{i,j} = 0$, as these events cause a decrease in player i's utility (compared to the original SpaceMint game). Recall that $\Pr[b^\eta_{i,j} = 0] = \eta$ and $\Pr[b^\upsilon_{i,j} = 0] = \upsilon$ by definition. Applying a union bound over the probabilities of these bad events, and using Theorem 7.16, we obtain the required result. □

Remark. Definition E.1 models the case when network delays and synchrony issues only cause block announcements to be delayed by one time-step. We remark that our analysis extends straightforwardly to the cases when the delays are longer than one time-step, or have variable length, or even cause block announcements to be permanently suppressed rather than delayed.