SpaceMint: A Cryptocurrency Based on Proofs of Space*

Sunoo Park[1], Albert Kwon[1], Georg Fuchsbauer[2],
Peter Gaži[3], Joël Alwen[4], and Krzysztof Pietrzak[4]

[1]MIT
[2]Inria, ENS, CNRS, and PSL
[3]IOHK
[4]IST Austria

Abstract

Bitcoin has become the most successful cryptocurrency ever deployed, and its most distinctive feature is that it is decentralized. Its underlying protocol (Nakamoto consensus) achieves this by using *proof of work*, which has the drawback that it causes the consumption of vast amounts of energy to maintain the ledger. Moreover, Bitcoin mining dynamics have become less distributed over time.

Towards addressing these issues, we propose SpaceMint, a cryptocurrency based on *proofs of space* instead of proofs of work. Miners in SpaceMint dedicate *disk space* rather than computation. We argue that SpaceMint's design solves or alleviates several of Bitcoin's issues: most notably, its large energy consumption. SpaceMint also rewards smaller miners fairly according to their contribution to the network, thus incentivizing more distributed participation.

This paper adapts proof of space to enable its use in cryptocurrency, studies the attacks that can arise against a Bitcoin-like blockchain that uses proof of space, and proposes a new blockchain format and transaction types to address these attacks. Our prototype shows that initializing 1 TB for mining takes about a day (a one-off setup cost), and miners spend on average just a fraction of a second per block mined. Finally, we provide a game-theoretic analysis modeling SpaceMint as an *extensive game* (the canonical game-theoretic notion for games that take place over time) and show that this stylized game satisfies a strong equilibrium notion, thereby arguing for SpaceMint's stability and consensus.

*In an early version, our proposal was called "Spacecoin." We changed it to "SpaceMint" due to name conflicts.

1 Introduction

E-cash was first proposed by Chaum [8] in 1983, but did not see mainstream interest and deployment until the advent of Bitcoin [28] in 2009. With a market cap of over 300 trillion US dollars by December 2017, Bitcoin has given an unprecedented demonstration that the time was ripe for digital currencies.

On the flip side, Bitcoin's dramatic expansion has provoked serious questions about the currency's long-term sustainability. Bitcoin miners produce proofs of work (PoW) to add blocks to the *blockchain*, the public ledger of all transactions. For each block added, there is a reward of newly minted coins. One concern is that proofs of work deplete large amounts of natural resources: by some estimates from December 2017, the Bitcoin network consumed over 30 terawatt-hours per year, which exceeds Denmark's energy consumption. Moreover, most mining is currently done by specialized ASICs, which have no use beyond Bitcoin mining.

A related concern is the emergence of a "mining oligarchy" controlled by a handful of powerful entities.

One of the original ideas behind basing Bitcoin mining on computing power was that anyone could participate in the network by dedicating their spare CPU cycles, incurring little cost as they would be repurposing idle time of already-existing personal computers. However, modern Bitcoin mining dynamics have become starkly different [37]: the network's mining clout is overwhelmingly concentrated in large-scale mining farms using special-purpose hardware for Bitcoin mining, often in collaboration with electricity producers. As a result, mining with one's spare CPU cycles today would result in net *loss* due to electricity costs. This phenomenon undermines the stability and security intended by the original decentralized design.

In light of these issues, there has been increasing interest in cryptocurrencies based on alternatives to proofs of work. The most explored alternative is *proofs of stake* (PoStake), in which a miner's probability of successfully creating a block increases with the amount of currency he holds, rather than the amount of computation he performs. This concept has several incarnations, from ad-hoc implementations in existing cryptocurrencies [22, 36] to designs with rigorous security proofs in various models [23, 21, 9, 10]. While these are innovative proposals, the early constructions have variously suffered from attacks that arise due to the inexpensive nature of mining. On the other hand, the more recent proposals are fairly complex, usually running some kind of Byzantine agreement protocol among a sufficiently large subset of stakeholders, and thus diverge substantially from the simplicity of the original Nakamoto design. Such schemes also typically fail in case of low participation (i.e., if stakeholders are not mostly online).

In this paper, we propose SpaceMint, a cryptocurrency that uses *proofs of space* (PoSpace) [14, 34, 4] to address the aforementioned issues that occur in Bitcoin and alternatives such as PoSpace-based currencies. To mine blocks in SpaceMint, miners invest *disk space* instead of computing power, and dedicating more disk space yields a proportionally higher expectation of successfully mining a block. SpaceMint has several advantages compared to a PoW-based blockchain like Bitcoin, summarized below.

- *Ecological:* Once the dedicated space for mining is initialized, the cost of mining is marginal: a few disk accesses with minimal computation.
- *Economical:* Unused disk space is readily available on many personal computers today, and the marginal cost of dedicating it to SpaceMint mining would be small (by the previous point).[1] We thus expect that space will be dedicated towards mining even if the reward is much smaller than the cost of buying disk space for mining. In contrast, in PoW-based blockchains rational miners will stop mining if the reward does not cover the energy cost.
- *Egalitarian:* Bitcoin mining is done almost entirely on application-specific integrated circuits (ASIC) and by large "mining farms," to the point that small-scale participation (e.g., based on general-purpose hardware) is impossible. We believe SpaceMint to be less susceptible to specialized hardware than Bitcoin, as discussed in §6.

  Another cause of centralization of mining power in Bitcoin is mining pools. This paper does not address that problem directly, but an elegant and simple idea [26] to discourage mining pools in PoW-based blockchains — namely, having the mining process require the secret key to redeem the block reward – can be straightforwardly adapted for SpaceMint.[2]

1.1 Challenges and Our Contributions

In order to "replace" PoW by PoSpace to achieve consensus on the blockchain, the following problems must be addressed.

- *Interactivity:* PoSpace, as originally defined [14], is an interactive protocol. Although the same is true for the original definition of PoW [13], there the interaction was very simple (i.e., a two-message, public-coin protocol). PoSpace re-

[1] By marginal cost we mean the cost of using disk space that otherwise would just sit around unused.

[2] In a PoW this can be achieved by, e.g., not applying the hash function to a nonce directly, but to its signature. In the PoS [14] used for SpaceMint, this can be achieved by augmenting each "label" that is stored with its signature.

quires more interaction, thus it is more challenging to adapt PoSpace to the blockchain setting.

- *Determine the winner:* In a PoW-based blockchain like Bitcoin, the probability of a miner being the first to find an eligible next block increases with its hashing power. The Bitcoin protocol prescribes that once an eligible next block is announced, all miners should append that block to the blockchain and continue mining on the new longest chain. Generating a PoSpace, on the other hand, is deliberately computationally cheap. We thus need some way to determine which of many different proofs "wins". Moreover, the probability of any miner winning should be proportional to the space it dedicates, and we want a miner to learn if he is a likely winner without any interaction.
- *"Nothing-at-stake" problems:* When replacing PoW by proofs that are computationally easy to generate (such as PoStake or PoSpace), a series of problems arise known as *nothing-at-stake problems* [16].[3] The computation-intensive nature of Bitcoin mining is a key property that, informally, ensures that all miners are incentivized to concentrate their mining efforts on a single chain, which leads to consensus. When mining is computationally cheap however, miners can intuitively (1) mine on multiple chains simultaneously, not just the one the protocol specifies, and (2) try creating many different blocks with a single proof (of space or of stake) by altering the block contents slightly (e.g., by using different transaction sets) before choosing the most favorable one to announce. The latter behavior is known as "(block) grinding". Those issues are undesirable as
  1. they slow down consensus;
  2. they potentially allocate a greater reward to cheating miners
  3. they potentially enable double-spending attacks by an adversary controlling much less than 50% of the space.
- *Challenge grinding:* Yet another issue arises when the content of past blocks can influence which blocks are added to the blockchain in future. Then it may be possible for a miner to generate a long sequence of blocks whose earlier blocks might have proofs of low quality, but are generated in a biased way (by "grinding" through all the possible proofs) so that the miner can create high quality proofs later in the sequence. The problem arises when the overall sequence is of higher quality than would be expected from the miner's disk space size, due to the disproportionately high quality of later blocks. Challenge grinding may be considered a nothing-at-stake problem, but we state it separately as, unlike the other nothing-at-stake problems, we have not encountered it in other contexts.

To tackle the *interactivity* problem, SpaceMint uses the Fiat-Shamir paradigm (a standard technique to replace a public-coin challenge with a hash of the previous message, already used to adapt PoW for Bitcoin); additionally, we leverage the blockchain itself to record messages of the PoSpace protocol (concretely, we use a special type of transaction to record the commitment to its space a prover needs to send to the verifier in the initialization phase of the PoSpace).

To *determine the winner*, we define a *quality function*, which assigns a quality value to a PoSpace proof. This function can be computed by the miner locally, and is designed such that the probability of a miner having the highest-quality proof in the network is proportional to the space it dedicates.

The *nothing-at-stake* problems are more challenging to solve. To tackle these, we introduce several new ideas and leverage existing approaches. To disincentivize miners from extending multiple chains we ensure such behavior is detected and penalize it. To prevent *block grinding*, SpaceMint ensures that the PoSpace is "unique", i.e., a miner can generate exactly one valid proof for every given challenge, and this challenge itself is uniquely determined by the proofs that were used to mine a previous block. This is done by basically running two chains in parallel, a "proof chain" that contains the proofs, and a "signature chain" that contains the transactions.

[3] Although PoSpace-based currencies share some of the issues PoStake-based currencies have, they are robust to others, in particular, PoSpace does not share the tricky *participation* problem of PoStake.

Finally, to address *challenge grinding*, SpaceMint prescribes that past blocks influence the quality of *short sequences* of future blocks, thus exponentially driving down the probability that a miner could generate a sequence of blocks of disproportionately high quality by exploiting the relationship between past and future blocks.

The idea of making the challenge for a block a deterministic function of a unique credential of the resource that "won" a previous block – in combination with having a quality function by which miners can locally decide if they are likely winners – has been used in subsequent blockchain proposals like Algorand [23] or the Chia Network [3].

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 generate a candidate 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 logarithmically with larger space.

Finally, we provide a game-theoretic analysis of SpaceMint modeled as an *extensive game*. To do this, we formally specify a stylized model of SpaceMint mining and show that adhering to the protocol is a *sequential equilibrium* for rational miners in this game (i.e., deviating from the protocol does not pay off). Our analysis works in a simplified model that serves to rule out certain classes of attacks (i.e., profitable deviations based on a simplistic set of possible actions), but does not capture all possible attack vectors by real miners.[4] To our knowledge, this is the first analysis of a cryptocurrency mining as an extensive game with the corresponding game-theoretic equilibrium concepts; though the model is simplistic, we hope that this framework for rigorously ruling out certain classes of attacks will serve as a useful base upon which to build more nuanced game-theoretic models to rule out larger classes of attacks, in this and other similar cryptocurrencies.

[4]For example, "selfish mining" [17] or block withholding is not captured by our simplified model, and SpaceMint is in fact susceptible to block withholding attacks to a similar extent to Bitcoin.

1.2 Related Work

We have already discussed *proofs of stake* above. Here, we briefly mention other related proposals. A more detailed discussion can be found in the full version [31].

Proof of storage/retrievability [18, 7, 6, 19, 12, and many more] are proof systems where a verifier sends a file to a prover and later requests a proof that the prover really stored the file. Proving storage of a (random) file does show that one dedicated space, but the verifier must send the entire file first. In contrast, PoSpace requires verifier computation and communication to be polylogarithmic in the prover's storage size.

Proof of secure erasure (PoSE), one-time computable functions [33, 15, 20, 5] are proof systems where a prover convinces the verifier that is has access to some space. Additionally one can require that the proof implies that the space also was erased [33, 20], or some function can only be computed in forward direction [15]. Those protocols have only one phase, and thus cannot be used as a PoSpace, i.e., to efficiently prove space usage over time.

Permacoin [25] is a cryptocurrency proposal that uses proofs of retrievability with a novel variant of PoW. While solutions to Bitcoin's PoW puzzles carry no intrinsic value, Permacoin makes proof-of-work mining serve a useful purpose: miners are incentivized to *store useful data* and thus the network serves as a data archive. Permacoin is however still fundamentally a PoW-based scheme. In contrast, in SpaceMint the dedicated storage does not store anything useful, but we completely avoid PoW and the associated perpetual computation.

Burstcoin [1] is the only cryptocurrency we are aware of in which disk space is the primary mining resource. However, Burstcoin's design allows *time/memory trade-offs*: i.e., a miner doing 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. Moreover, Burstcoin requires a constant (albeit small) fraction (0.024%) of dedicated disk space to be read every time a block is mined, while SpaceMint requires only a logarithmic fraction. Finally, verification in Burstcoin is problematic: min-

ers must hash over 8 million blocks to verify another miner's claim. The details on this attacks can be found in Appendix B of the full version [31].

Chia Network [3] is a very recent proposal of a blockchain based on PoSpace in combination with proofs of sequential work. In a nutshell, the better the quality of the PoSpace, the faster the block can be "finalized" by a proof of sequential work, and this proof tuple then can be used to create a block. By using proofs of sequential work on top of PoSpace, Chia is even more similar to Bitcoin than SpaceMint in several respects: for example, it requires no synchronization/clocks (except, as in Bitcoin, time-stamped blocks for the occasional re-calculation of the mining difficulty), while retaining the efficiency of a pure PoSpace-based currency. The PoSpace that was developed for Chia [4] is based on ideas completely different from the PoSpace [14] we use. It has worse asymptotic security guarantees, but unlike [14], it has a non-interactive initialization phase and extremely short and efficient proofs.

Outline.

- *Cryptocurrency from proofs of space:* In (§§2–3) we modify PoSpace [14] for the blockchain setting and present SpaceMint, a cryptocurrency based purely on proofs of space.
- *Addressing the "nothing-at-stake" problems:* After describing attacks that arise from nothing-at-stake problems and challenge grinding, we describe how our design uses novel approaches to overcome them (§4). Our solutions extend to other blockchain designs based on easy-to-generate proofs.
- *Evaluation of proof of space:* We evaluate our modified PoSpace in terms of time to initialize the space, to generate and verify blocks, and block size (§6).
- *Game theory of SpaceMint:* We model SpaceMint as an extensive game, and show that adhering to the protocol is an *ε-sequential Nash equilibrium* (§7).

2 Proof of Space in SpaceMint

A PoSpace [14] is a two-phase protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. After an *initialization phase*, $\mathcal{P}$ stores some data S_γ of size N, and $\mathcal{V}$ stores a short commitment γ to S_γ. Then, in the *execution phase*, $\mathcal{V}$ sends a challenge c to $\mathcal{P}$, who returns a short answer a after reading a small fraction of S_γ.

The PoSpace from [14, 34] are specified a family of "hard-to-pebble" directed acyclic graphs of increasing size. The prover picks a graph $G = (V, E)$ from this family depending on the amount of space it wants to dedicate. $\mathcal{P}$ then stores a label l_i for each node $i \in V$, which is computed as

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

where $p_1, \ldots, p_t$ are the parents of node i and hash is a hash function (sampled by $\mathcal{V}$). In [14] two graph families are suggested, one for which any successful cheating prover must either use $\Omega(|V|/\log(|V|))$ space between the initialization and execution phase, or use $\Omega(|V|/\log(|V|))$ space during execution. The other graph family enforces either $\Theta(|V|)$ space between the phases (i.e., the same as the honest prover, up to a constant), or $\Theta(|V|)$ time during execution.

Formally, [14] specifies a PoSpace by a tuple of algorithms $\{\mathsf{Init}, \mathsf{Chal}, \mathsf{Ans}, \mathsf{Vrfy}\}$, which specify a two-phase protocol between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. Init is used to initialize the space, Chal generates a challenge, Ans computes the response to a challenge and Vrfy verifies the response. The initialization phase consists of running Algorithm 1, where $\mathcal{P}$ commits to its space, followed by Algorithm 2, where $\mathcal{P}$ proves that the commitment is computed "mostly correct". In the execution phase, given by Algorithm 3, $\mathcal{V}$ simply opens some of the committed labels to prove it has stored them.

The algorithms we give here are already made partially non-interactive for our blockchain application – in the actual PoSpace the challenges in Algorithm 2 and 3, as well as μ in Algorithm 1 are sampled by $\mathcal{V}$ and sent to $\mathcal{P}$.

[5] The nonce just ensures that the same space cannot be used for two different proofs [14]; thus in a single-verifier setting, $\mathcal{P}$ can generate the nonce.

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[5] μ and the commitment γ to $\mathcal{V}$. S_γ contains the labels of all the nodes of G computed using Eq. (1) and 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).

Algorithm 2 Prove commit

Initial state: $\mathcal{V}$ holds commitment γ and nonce μ; $\mathcal{P}$ stores S_γ and μ. Both are given the challenges $c = (c_1, \ldots, c_{k_v})$ to be used.

1. $\mathcal{P}$ computes openings $b := (b_1, b_2, \ldots)$ of all the labels of the nodes $\{c_i\}_{i\in[k_v]}$ *and of all their parents* and sends them to $\mathcal{V}$. This is done using Ans where $\mathsf{Ans}(\mu, S_\gamma, c)$ returns the Merkle inclusion proof of label l_c w.r.t. γ.
2. $\mathcal{V}$ verifies these openings using Vrfy, where $\mathsf{Vrfy}(\mu, \gamma, c, a) = 1$ iff a is a correct opening for c. It then checks for all $i = 1, \ldots, k_v$ if the label l_{c_i} is correctly computed as in Eq. (1).

3 SpaceMint Protocol

3.1 Mining

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 first generates a public/secret key pair (pk, sk) and runs Algorithm 1 as $\mathcal{P}$, with nonce μ set to pk, to generate

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

The miner stores (S_γ, sk) and announces its space commitment (pk, γ) via a special transaction. We require miners to commit (pk, γ) to prevent a type of grinding attack: the problem is that the PoSpace we use [14] have the property that by making minor changes one can turn (pk, γ) into many other space commitments that re-use most of the space.

Algorithm 3 Prove space

Initial state: $\mathcal{V}$ holds commitment γ and nonce μ; $\mathcal{P}$ stores S_γ and μ. Both are given the challenges $c = (c_1, \ldots, c_{k_p})$ to be used.

1. $\mathcal{P}$ computes openings $\{a_i := \mathsf{Ans}(\mu, S_\gamma, c_i)\}_{i\in[k_p]}$ and sends them to $\mathcal{V}$.
2. $\mathcal{V}$ verifies these openings by executing $\mathsf{Vrfy}(\mu, \gamma, c_i, a_i)$.

Once this transaction is in the blockchain, the miner can start mining.

Mining. Similar to Bitcoin, SpaceMint incentivizes mining (adding new blocks) through block rewards (freshly minted coins per block) and transaction fees. Once initialized, each miner attempts to add a block to the blockchain every time period. For time period i, a miner proceeds as follows:

1. Retrieve the hash value of the last block in the best chain so far, and a challenge c (we discuss how c is derived in §3.4), which serves as a short seed from which we derive two long random strings $\$_p, \$_v$.
2. Compute challenges $(c_1, \ldots, c_{k_p}) := \mathsf{Chal}(n, k_p, \$_p)$ for use in Algorithm 3.
3. Compute the proof of space $a = \{a_1, \ldots, a_{k_p}\}$ using Algorithm 3.
4. Compute the quality $\mathsf{Quality}(pk, \gamma, c, a)$ of the proof (details of the quality function are given in §3.5).
5. If the quality is high enough, so that there is a realistic chance of being the best answer in period i, compute the proof of correct commitment $b = \{b_1, \ldots, b_{k_v}\}$ using Algorithm 2; then create a block and send it to the network in an attempt to add it to the chain. This block contains the proofs a and b computed above and a set of transactions; the exact specification is in given §3.2 below.

Remark (Postponing Algorithm 2)**.** Note that unlike in the interactive PoSpace where one runs Algorithms 1 and 2 during initialization, we only require miners to execute Algorithm 2 if they want to add a block. This is done for efficiency reasons. For one thing, this way, the proof b (which is sig-

nificantly larger than a or γ) must only be recorded in the blockchain once the corresponding space has actually been used to mine a block. Another more subtle advantage is that now the challenge for Algorithm 2 changes with every block; thus a cheating miner (who computed some of the labels incorrectly) will only know if he was caught cheating at the same time when he generates a potentially winning proof a (and if b does not pass, he cannot use a). This allow us to tolerate a much larger soundness error in Algorithm 2, which means we can choose a smaller k_v (concretely, it's ok if he passes the proof with large probability p, as long as this requires using at least a p times the space an honest miner would use).

3.2 Blockchain Format

A blockchain 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 parts, called "sub-blocks", which contain the index i that specifies the position of the block in the blockchain. The structures of sub-blocks are as follows:

- The HASH sub-block φ_i contains
  - the current block index i,
  - the miner's signature ζ_φ on φ_{i-1}, the $(i-1)$th HASH sub-block, and
  - a "space proof" containing the miner's pk.
- The TRANSACTION sub-block τ_i contains
  - the current block index i and
  - a list of *transactions* (§3.3).
- The SIGNATURE sub-block σ_i contains
  - the current block index i,
  - the miner's signature ζ_τ on τ_i, the ith TRANSACTION sub-block, and
  - the miner's signature ζ_σ on σ_{i-1}, the $(i-1)$th SIGNATURE sub-block.

The links between blocks in a blockchain are illustrated in Fig. 1. We will refer to the hash sub-blocks as the *proof chain*, and the signature sub-blocks with the transactions as the *signature chain*. While the signature and transaction sub-blocks are all linked, the hash sub-blocks are only linked to each other and not to any signature or transaction sub-blocks.

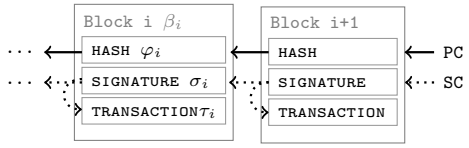


Figure 1: Our blockchain consists of a proof chain PC that does not allow for grinding, and a signature chain SC that binds transactions to the proof chain.

This design may seem 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 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.

3.3 Transactions in SpaceMint

There are three types of transactions in SpaceMint: (1) payments, (2) space commitments, and (3) penalties. Every transaction is signed by the user generating the transaction, and sent to the miners to be added to the blockchain. Here, we specify the three types of transactions.

Payments. Coins in SpaceMint are held and transferred by parties identified by public keys. A payment transaction transfers coins from m benefactors to n beneficiaries and has the form

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

- $txId$: A unique, arbitrary *transaction identifier*. That is, no two transactions in a blockchain can have the same identifier.
- $\vec{in}$: 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 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.
- $\vec{out}$: 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 specifies a *beneficiary*, and
  - v_i is the number of coins that pk_i is to be paid.

For a transaction to be valid, we require that (1) all signatures in $\vec{in}$ verify correctly; (2) no benefactor is referenced by more than one subsequent transaction in the blockchain (to prevent double-spending); (3) the sum of the input values to the transaction is at least the sum of the amounts paid to beneficiaries.

Space commitments. A space-commitment transaction

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

consists of pk, a public key, and γ which was computed as $(\gamma, S_\gamma) := \mathsf{Init}(pk, N)$. Thus, ctx is a space commitment to a space of size N.

Penalties. A penalty transaction

$$ctx = (\mathsf{penalty}, txId, pk, prf)$$

consists of pk, the public key of the transaction creator, and *prf*, a proof of penalty-worthy behavior by another miner. These transactions serve to penalize miners that engage in malicious behavior. The primary usage of penalties in SpaceMint is to disincentivize mining on multiple chains (e.g., the proof would contain two blocks of the same index signed by the same miner), but penalty transactions can be used to discourage other types of (detectable) behavior in blockchain-based currencies.

3.4 Where the Challenge Comes From

In Bitcoin, the PoW challenge for block i is simply the hash of block $i-1$. For SpaceMint, using block $i-1$ for the challenge can slow down consensus: 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), and if one of them is very good, try to add a block to the corresponding chain, even if this chain is not the best chain seen so far. If all miners behave rationally, this will considerably slow down consensus, as bad chains get extended with blocks of quality similar to the current best chain, and it will take longer for lower-quality chains to die off.

Instead, we derive the challenge for block i from the hash of block $i-\Delta$, for a reasonably large Δ: the probability of multiple chains surviving for more than Δ blocks decreases exponentially as Δ increases. Moreover, in contrast to Bitcoin, we only hash the block from the *proof chain*, but not the *signature chain* (Fig. 1): this serves to prevent block-grinding attacks, since there is nothing to grind on (the proof chain is fixed regardless of the set of transactions in the block). Finally, we will use the same challenge not just for one, but for δ consecutive blocks. This is done to prevent challenge-grinding attacks, as we explain in §4.

3.5 Quality of Proofs and Chains

Quality of a proof. The block to be added to the chain at each time step is decided by a *quality* measure on the PoSpace proof included in each proposed block. For a set of valid proofs $\pi_1 = (pk_1, \gamma_1, c_1, a_1), \ldots, \pi_m = (pk_m, \gamma_m, c_m, a_m)$, we require $\mathsf{Quality}(\pi_i)$ to be such that the probability that π_i has the *best* quality among $\pi_1, \ldots, \pi_m$ corresponds to the ith miner's fraction of the total space in the network. The probability is over the choice of the random oracle hash, which we use to hash answer a. We require:

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

where N_{γ_i} is the space committed to by γ_i.

Let D_N be a distribution that samples N values in $[0,1]$ at random and outputs the largest of them:

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

Let $D_N(\tau)$ denote a sample from D_N with sampling

randomness τ. For valid proofs we now define

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

The Quality of an invalid proof is set to 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 $[0,1]$. If 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 $F_X(z) = \Pr[X \leq z] = z^N$, since this is the probability that all N values r_i considered in (2) are below z. Therefore, if we want to sample from 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

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

Quality of a chain. In order to decide which of two given proof-chain branches 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 $v_j = D_{N_j}(\mathsf{hash}(a_j))$. For any quality $v \in [0,1]$, we define

$$\mathcal{N}(v) = \min\left\{N \in \mathbb{N} : \Pr_{w \leftarrow D_N}[v < w] \geq 1/2\right\} ,$$

the space required to obtain a proof with quality better 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.

In order to prevent challenge-grinding attacks, it is desirable for the chain quality to depend *multiplicatively* on constituent block qualities (described in more detail in §4), and moreover it is useful to weight the contribution of the jth block for a chain of length i by a *discount factor* Λ^{i-j}. From these motivations we derive the following quality function. Note that we have used a sum of logarithms, rather than a product, to achieve the multiplicativity.

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

4 Nothing-at-Stake Problems and Solutions

In this section we discuss the "nothing-at-stake" issues, which were already mentioned in the introduction. We describe them here in more detail, and outline how SpaceMint defends against them.

Recall that the difficulty arises due to the ease of computing multiple candidate blocks: in a PoSpace (or PoStake) based currency, a miner can compute *many* proofs (either extending different chains, or computing different proofs for the same chain) at little extra cost. Deviating from the protocol like this can be rational for a miner as it might lead to higher expected rewards. PoW-based blockchains also suffer from such "selfish mining" attacks [17], and basing the blockchain on efficiently computable proofs like PoSpace or PoStake can further aggravate this problem. Such behavior can significantly slow down consensus as well as push the scheme to follow energy expenditure trends similar to PoW-based schemes, which arise whenever there is an advantage to be gained by doing extra computation.

An even more serious issue is double-spending attacks, which become possible if a miner can create a sufficiently long chain in private which has better quality than the honestly mined chain. In all known blockchain proposals, a miner controlling more than half of the mining resources (hashing power, stake or space) can do this. But it is considered problematic if a blockchain is susceptible to double spending by adversaries with significantly less than half of the network resources.

I. Grinding blocks. *The problem:* In Nakamoto-style blockchains, the challenge for the proof computed by the miners (like PoW in Bitcoin or PoSpace in SpaceMint) is somehow derived from previous blocks. If it is computationally easy to generate proofs, a miner can try out many different blocks (for example by including different transactions) until it finds an advantageous one that will allow him to generate good proofs for future blocks. This is an issue for selfish-mining and double-spending attacks.

The solution: We decouple proofs from transactions as shown in Fig. 1. This eliminates the problem of

block grinding, as now challenges depend only on the *proof chain*. Moreover, our PoSpace are "unique" in that a prover can generate at most one valid proof per challenge. Hence, the only degree of freedom that a miner has in influencing future challenges is to either publish its proof (so it might end up in the chain), or to withhold it.

II. Mining on multiple chains. *The problem:* In Bitcoin, rational miners will always work towards extending the longest known chain. However, when mining is computationally easy, it can be rational to mine on *all* (or at least many) known chains in parallel, to "hedge one's bets" across all chains that might eventually become part of the public ledger. Again, this is an issue for selfish-mining and double-spending attacks.

The solution: To address this problem in the context of selfish-mining attacks (we discuss double-spending later), we derive the challenge for block i from block $i-\Delta$ for some parameter Δ (§3.4). Note that for any given challenge, there is a single proof (i.e., the proof is deterministic given the challenge). Then, we impose a *penalty*, via the penalty transactions of §3.3, for any pair of identical proofs published in two candidate blocks: half of the block reward for such a "misbehaving" block is allocated to the creator of the penalty transaction, and the other half simply disappears.[6] Let us consider 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 penalties (§3.3) to disincentivize miners from extending multiple chains in this case; without the penalties, a rational miner with a good-quality PoSpace proof could announce blocks on multiple chains to maximize his chances of winning. Concretely, suppose a miner pk' attempts to mine concurrently on two chains whose most recent blocks are β_j and β'_j, by announcing β_{j+1} and β'_{j+1} (which have the same quality and were mined using the same space). Then anyone who observes this can generate a transaction $(\mathsf{penalty}, txId, pk, \{pk', \beta_{j+1}, \beta'_{j+1}\})$ to penalize pk'. This transaction can be added to a chain extending β_{j+1} (or β'_{j+1}), and its meaning is that half of the reward (block reward and transaction fees) that should go to the miner who announced β_{j+1}, is now going to pk (the "accuser") instead, and the other half of the reward is destroyed, i.e., cannot be redeemed by any party. We destroy half of the reward so the penalty hurts even if the cheating miner can be reasonably sure to be able to accuse itself. For this to work, mining rewards can only be transferred by a miner some time after the block was added, so that there is enough time for other miners to claim the penalty.[7]

Case 2: chains forked more than Δ ago. In this case the miner receives different challenges for different chains, leading to proofs of different quality for the two chains. In this case, even with our penalty scheme in place, a rational miner can still get an advantage by deviating: instead of only trying to extending the highest-quality chain, it also generates proofs for the lesser chain. As the challenges differ, so will the two proofs, and if the proof on the lesser chain has very high quality, the rational miner would publish it, hoping that this chain will become the best chain and survive.

We address this problem by arguing that it is extremely unlikely (the probability is exponentially small in Δ) that this case occurs, as a weaker branch of the chain would have to "survive" for Δ blocks despite a strong incentive (via our punishment scheme) for miners to only extend the chain of highest quality.

III. Grinding challenges. *The problem:* Challenge grinding is a type of attack that can be used for double-spending, by generating a long chain in private that is of higher quality than what would result when using one's resources honestly. It arises from the fact that an adversary can split its space into m smaller chunks. As discussed in §3.5, the quality of a block is purposely designed such that splitting a fixed amount of space into smaller chunks and choosing the

[6] This is to disincentivize penalizing oneself.

[7] The idea of penalizing miners for extending multiple chains goes back at least to slasher `https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm`. Unlike previous penalty-based proposals, we do not need the miners to make a deposit up-front; instead, they will simply lose their mining reward if they cheat.

highest-quality block among them does *not* affect the expected quality of the block generated. However, a miner can examine all possible chains of some given length, and then pick the chain that gives it the most favorable challenges for future blocks.

Concretely, consider our setting, where the challenge for block i is determined by block $i - \Delta$. An adversary can generate a sequence of length 2Δ where the first half of the blocks is chosen to provide the most favorable challenges for the later half of the sequence.[8] Note that the first half of this sequence would be of even poorer quality than the expected quality from honest mining given the adversary's total amount of space; however, the benefit gained in the second half of the sequence can outweigh this loss in quality in the first half. The adversary can then release this high-quality chain (all at once) in an attempt to overtake the current best chain.

Note that in this attack the adversary explores multiple chains in parallel, which we have addressed already using a penalizing scheme. But penalizing does not protect against double-spending attacks in which the adversary never actually published two proofs for the same slot. And even he would, a double-spending attack can be profitable even if one loses some mining rewards due to the penalizing scheme.

The solution: As mentioned in §3.5, the problem with this attack is exacerbated if the metric for determining the quality of a chain is a sum or any other linear function. Thus, to prevent this attack, (1) we define the quality of a chain as the *product* of the amounts of space needed for the proofs in it, rather than their *sum*; and (2) we use the same block to derive challenges for δ future 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, as worse blocks are weighted more; and (2) is helpful because it means that a challenge-grinding adversary would have to choose "early" blocks to optimize their chances over sequences of δ future challenges rather than just a single future challenge, thus making it exponentially harder (in δ) to find a "good" challenge that will yield δ high-quality blocks at once. Another way to see this is that by the Chernoff bound, the average of δ independent random variables deviates less from its expectation as δ grows. So for large δ, even the ability to select between multiple challenges (each giving a sample of the average of δ i.i.d. variables) is not very useful to find one where this value deviates by a lot from its expectation. A more detailed discussion of this attack and our defense is given in the full version [31].

[8] As for each of the Δ blocks there are m distinct challenges, the search space here is of huge size m^{Δ}. Consequently, this attack might seem artificial, but by pruning and just considering the most promising sub-chains at every level, one will probably not miss the best one.

5 Parametrization

We now discuss and justify some suggested parameter choices for SpaceMint. A more detailed discussion on parameters, their interplay, and their impact on various attacks is given in §D due to space constraints.

Determining challenges. To minimize the probability of forks surviving for more than Δ blocks (which is necessary to prevent the "mining on multiple chains" issue described in §4), we should choose a large Δ. On the other hand, a smaller Δ increases other security features of SpaceMint (detailed in §D). We suggest $\Delta = 50$, which makes it highly unlikely that a fork survives 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 §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 (§6) 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 they can propagate through the network, since the miners need to gener-

ate the signature chains using the previous block. In Bitcoin, blocks propagate to over 95% of the miners within 40 seconds [11], so we believe that $\mathsf{time} = 1$ minute would be a reasonable frequency of block generation for SpaceMint.

Quality discount factor. As discussed in §3.5, we use a discount factor Λ to define 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.[9] Such a high Λ is helpful when we argue about the hardness of generating long forks. Detailed analysis is given in §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 consensus. Their analysis [35] 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), in SpaceMint after 6 blocks we have on upper-bound of $2^{-16} \approx 0.000015$ on the probability that a block will remain in the chain. This also takes only 6 minutes compared to the 1 hour of Bitcoin. This analysis only applies to the proof chain, but to avoid double spending we must be sure that the transaction also remains in the signature chain, for this reason one should wait for a few extra blocks, so one can be reasonably sure that at least one of those blocks was added by an honest miner (who will not sign another list of transactions for this block).

Even assuming a stronger adversary who controls 33% of the total space (and $\Lambda = 0.99999$ as before), after 93 blocks a block in the proof chain will be safe with failure probability bounded by 2^{-32}. For further details see §D.

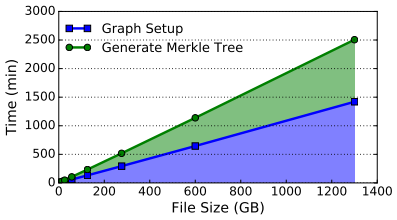
[9] In this case, the contribution of a block decreases by a factor $1/e \approx 0.37$ every $1/(1-\Lambda) = 100.000$ blocks, which for $\mathsf{time} = 1$ minute is roughly 69 days.

6 Evaluation

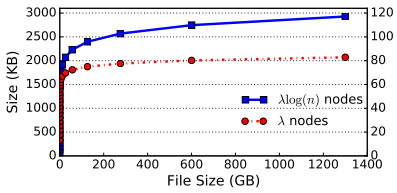
To evaluate SpaceMint, we have implemented a prototype in Go, using SHA3 in 256-bit mode as the hash function. The prototype uses the graphs from [32], and forces a cheating prover to store at least $\Omega(N/\log(N))$ bits in order to efficiently generate proofs. Given that the network infrastructure 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 described in §3.1. Concretely, this involves computing all the hashes of the nodes, and computing the Merkle tree over the hashes. In Figure 2a, 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.

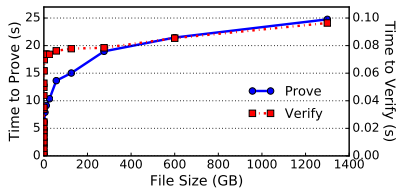
Size of the proof. A proof in SpaceMint consists of the Merkle inclusion proof for a set of node labels. For the PoSpace that we implemented, the number of nodes we have to open is $\lambda \cdot \log(n) + 1$ (as $k_v = \lambda \cdot \log(n)$ in Algorithm 2 and $k_p \ll k_v$ in Algorithm 3), where λ 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. Fig-



(a) Time to initialize space.



(b) Proof size for varying space sizes with $\lambda = 30$. The left and right vertical axes represent proof size when opening $\lambda \log(n)$ and λ nodes (respectively).



(c) Time for a miner to prove and verify when $\lambda \log(n)$ nodes are opened for $\lambda = 30$.

Figure 2: Results of evaluation

ure 2b demonstrates the size of the proof when we open $\lambda \log(n)$ nodes vs. just λ nodes 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 a small k_p number of nodes in the Merkle tree to check the quality of its solution (§3.1), 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. As for every space-commitment, a miner must open k_p nodes for every time-slot, we want this value to be as small as possible, in practice $k_p = 1$ seems secure, though one might set it to some small constant to be on the safe side.

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 2c 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 [2]). 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.

Impact of storage medium. Almost all modern Bitcoin mining is done by clusters of application-specific integrated circuits (ASICs), which can compute hashes for a tiny fraction of the hardware and energy cost of a general-purpose processor. We believe that SpaceMint mining would not be as susceptible to advantages from specialized hardware as Bitcoin, and that regular hard disk drives are well-suited to serve as SpaceMint mining equipment. Let us consider existing categories of storage devices. Although hard disks are expensive compared to other storage devices, most notably tapes, devices like tapes are not adequate for mining as we require frequent random accesses to answer the PoSpace challenges. Solid state drives do allow for (fast) random accesses, but are more expensive than hard disks and do not provide any benefit since the rate of lookups required for mining is very low. Notably, SpaceMint mining hinges on doing a few random lookups every minute.

The required frequency is so low that speed is a non-issue: cheap, *slow* random access is what SpaceMint miners are after.

7 Game Theory of SpaceMint

Intuitively, SpaceMint mining is modeled by the following n-player strategic game. Game-play occurs over a series of discrete time steps, in each of which a block is added to the blockchain. At each time step, each player (miner) must choose a strategy, specified by: (1) which blocks to extend (if any), and (2) 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, it follows that rational miners will reach consensus on a single chain, and will not be able to get an 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.

Let us stress that our analysis herein is intended as a basic framework to model blockchain-based cryptocurrencies using the standard game-theoretic notion of an *extended game*, and does not claim to comprise an exhaustive modeling of all possible attack vectors. In particular, our stylized model does not capture some important aspects, most notably block withholding, which is used in "selfish mining." Nevertheless, we believe that our simple modeling framework for cryptocurrency as an extended game may be of value as a base upon which to build more nuanced game-theoretic models, and thus we have chosen to include it in this exposition.

7.1 Informal overview of results

The standard game-theoretic notion for a strategic game which occurs over multiple time steps (rather than in "one shot") is the *extensive game*. An extensive game takes place over discrete time steps. In each time step, one or more players take an action from a well-defined set of possible "moves". At any point, the sequence of all moves made by all players so far is called a *history*. In some games, players do not necessarily know all the moves made by other players. For any history, an extensive game defines a set of possible actions that each player can take at that history. If all of these sets are empty, then the game is considered to have ended, and such a history is called a *terminal history*.

Each player has a *utility function* that assigns real-valued utilities to each terminal history. For example, in a simple two-player game like rock-paper-scissors, each player's utility function might assign utility 1 to histories in which they won, and utility -1 to histories in which they lost. When modeling SpaceMint, the utility that a player (i.e., a miner) assigns to a history depends on the amount of currency he has earned by successfully adding blocks to the chain.

In order to 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. Essentially, we model the beacon as an additional player, called Chance, which makes random moves.

Equilibrium concepts. The most widely known equilibrium concept for a strategic game is the Nash equilibrium [29]. Intuitively, in a Nash equilibrium, each player's strategy is a *best response* to the strategies of the other players.

The Nash equilibrium concept was originally formulated for *one-shot* games (in which all players make a move simultaneously, then the game is over), 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 adaptively changing strategy partway through a game: in particular, there exist Nash equilibria that

are not "stable" in the sense that given the ability to adaptively switch strategies during the game, no rational player would stick with his Nash-equilibrium strategy all the way to the end of the game.

Thus, for extensive games, the alternative (stronger) notion of *sequential* equilibrium is the standard equilibrium notion in game theory. This stronger concept ensures players are making the best decision possible *at each history* during game-play.

We remark that while informal analyses have been presented that argue that Bitcoin mining constitutes a Nash equilibrium, we are not aware of any prior analyses that model a cryptocurrency as an extensive game or consider sequential equilibria. Since protocols inherently occur over time, we strongly believe that extensive games are the appropriate game-theoretic formalism for analyzing stability of cryptocurrencies, and accordingly that sequential equilibrium is the right equilibrium concept for cryptocurrencies.

7.1.1 SpaceMint as an extensive game

In the "SpaceMint Game", 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. Players do not necessarily know all actions taken by other players: the information known to each player at any point comprises all the moves that he himself has taken, together with the information in the blockchain. Based on this information, each player decides his action in each time step, aiming to maximize his utility, i.e., his expected reward from adding blocks to the chain.

Due to the rather extensive definitional preliminaries required to state our results formally, we now give an informal theorem statement together with a proof sketch. Rigorous theorems and proofs are given in §7.2.

Theorem 7.1. *It is a sequential equilibrium of the SpaceMint game (Definition 7.2, §7.2) for all computationally bounded players to adhere to the mining protocol, provided that no player holds more than* 50% *of all space.*

Proof sketch. Our proof proceeds in two main steps, showing the following.

1. Adhering to the protocol is a Nash equilibrium of the SpaceMint game.
2. Adhering to the protocol is moreover *sequentially rational* at each history during game-play.

To prove item 1, we consider the information available to an arbitrary player at any given time step. At the start of his turn, the player has an expected utility based on all the information he knows (e.g., if he has mined some blocks that have been added to the blockchain, his utility is high). Since a miner's utility function is simply his expectation of reward from adding blocks to the chain, a utility-maximizing miner can choose an action to take in each time step as a function just of the blockchain (i.e., he need not separately take into account the full sequence of moves he has made in the past, as these only affect his expected utility if they have impacted the blockchain). In a given turn, the player will choose the action that yields the highest expectation of future reward: his action consists of mining a set of blocks (locally), and then announcing a set of blocks to the network. Based on SpaceMint's penalty transactions and the fact that any given miner's block quality is fixed in each time step, we are able to argue that mining and announcing exactly one block is an optimal strategy. Moreover, this strategy adheres to the protocol.

To prove item 2, it is necessary to show that there exist a system of *consistent beliefs* of the players over the entire duration of the game, under which each player is not incentivized to deviate from the protocol at any point during game-play. We show that such a belief system can be derived by applying Bayes' rule to the Nash equilibrium strategy which consists of adhering to the protocol. This concludes the proof sketch; for the full proof see §7.2. □

7.2 Formally modeling SpaceMint mining as an extensive game

For standard game-theoretic terminology and preliminaries (such as definitions of extensive games, Nash equilibria, and sequential equilibria), we refer the reader to a standard textbook such as [30].

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

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 §3.2, and for any $\ell \in \mathbb{N}$, let $\mathcal{B}_\ell$ denote the set of all blocks with index ℓ.[10] 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 PoSpace challenge c (see §3.2 for exact block format). For $\ell \in \mathbb{N}$ and challenge c, define:

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

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

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.

Definition 7.2 (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 $K \in \mathbb{N}$, 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 $a_i = (\mathcal{T}, \mathcal{A})$. $\mathcal{T}$ is the set of blocks that player i tries *extending in this time step, and $\mathcal{A} \subseteq \mathcal{T}$ is the set of blocks that player i* announces *in this time step. An element in $\mathcal{T}$ (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.*
  - *$[(h, ((\mathcal{T}_1, \mathcal{A}_1), \ldots, (\mathcal{T}_N, \mathcal{A}_N), a_{\mathcal{C}}))]_i =$*

    $$\{(h', ((\mathcal{T}'_1, \mathcal{A}'_1), \ldots, (\mathcal{T}'_N, \mathcal{A}'_N), a'_{\mathcal{C}})) \in H : h \sim_i h' \wedge \mathcal{T}_i = \mathcal{T}'_i \wedge \mathcal{A}_i = \mathcal{A}'_i \wedge a_{\mathcal{C}} = a'_{\mathcal{C}} \wedge \forall i' \neq i,\ \mathcal{A}_{i'} = \mathcal{A}'_{i'}\},$$

    where h and h' are histories of equal length, and the pairs $(\mathcal{T}_{i'}, \mathcal{A}_{i'})$ and $(\mathcal{T}'_{i'}, \mathcal{A}'_{i'})$ are actions of player i'. That is, two histories are equivalent under $\sim_i$ if they are identical except in the "first components" $\mathcal{T}_{i'}$ of the actions $(\mathcal{T}_{i'}, \mathcal{A}_{i'})$ taken by players other than i.

- *$\vec{u} = (u_1, \ldots, u_N)$, where each $u_i\colon Z \to \mathbb{R}$ 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 $B.\text{chal}$ 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 §3.5. We define a new function*

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

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

[10] The index denotes the block's position in the blockchain. In §3.2, i is used to refer to the block index, but in this section we use ℓ to avoid confusion with the player indices.

$\mathsf{QualityPC}((B_1, \ldots, B_L))$ *whenever*

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

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

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*:*

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

Let $\mathsf{blocks}(h)$ *be the "winning block sequence" at any* $h \in H$*:*

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

Let $\mathsf{blocks}_\ell(h)$ *denote the* ℓ*th block in the chain. Let* $\mathsf{win}_\ell(h)$ *be the player who announced the winning block* $\mathsf{blocks}_\ell(h)$ *for index* ℓ*.*[11] *Recall that a history* $h = (\vec{a}_1, \ldots, \vec{a}_J)$ *is a sequence of* $J \le K$ *action profiles. For* $j \in [J]$*, let* $(\mathcal{T}_{i,j}, \mathcal{A}_{i,j})$ *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.*

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

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

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

where $\delta_{i,j}$ *is the Kronecker delta function.That 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 7.2, 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 §3.2 for exact block format).

Theorem 7.3. *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* $\rho \colon \mathbb{N} \to \mathbb{N}$*, 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* $I \neq \{()\}$*,*

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

where $j \ge 1$ *is the length of the histories in information set* I[12]*. 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

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

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, $u_i(\vec{\alpha}) \ge u_i(\alpha_i', \vec{\alpha}_{-i})$ 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, $u_i(\vec{\alpha}) \ge u_i(\alpha_i', \vec{\alpha}_{-i})$ 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 $u_i(\vec{\alpha}) \ge u_i(\alpha_i', \vec{\alpha}_{-i})$ for all strategies α_i' of player i. The theorem follows. □

7.3 Analyzing the SpaceMint game

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

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

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

By Definition 7.2, 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 7.4 defines an ε-Nash equilibrium of the SpaceMint game, and then Theorem 7.5 shows that this Nash equilibrium is, moreover, an ε-sequential equilibrium.

Theorem 7.4. *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 $\rho\colon \mathbb{N} \to \mathbb{N}$, 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 $I \neq \{()\}$,*

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

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

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

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

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

Λ is the discount factor defined in §3.5 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

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

is the same for any block B which was announced for block index ℓ. Therefore, no utility is gained by choosing any block B over any other block B' to extend: i.e., $u_i(\vec{\alpha}) \geq u_i(\alpha_i', \vec{\alpha}_{-i})$ for any strategy α_i' which distributes probability only over action sequences $((\mathcal{T}_{i,1}, \mathcal{A}_{i,1}), \ldots, (\mathcal{T}_{i,K}, \mathcal{A}_{i,K}))$ 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 $((\mathcal{T}_{i,1}, \mathcal{A}_{i,1}), \ldots, (\mathcal{T}_{i,K}, \mathcal{A}_{i,K}))$ 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

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

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 $\vec{B}' = (B_{\text{gen}}, B_1', \ldots, B_{j-1}', B_{i,j})$ 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:

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

where $I_{i,j}$ denotes player i's information set at time step j, and moreover his self-extended chain has

[13] Recall that t_i is the amount of space that player i has (defined in the remark just before Definition 7.2).

higher quality than any chain produced by others: i.e., at the terminal history h,

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

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

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

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

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

Theorem 7.5 (formal version of Theorem 7.1, §7.1)**.** *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 $\rho\colon \mathbb{N} \to \mathbb{N}$, 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 7.4, and for each $n \in \mathbb{N}$, we define $\vec{\alpha}^n$ to be the completely mixed strategy profile which (at history h) assigns probability $1/|A_i(h)|^n$ to every action except $\hat{\alpha}_i$, and assigns all remaining probability to $\hat{\alpha}_i$.*
- *$\vec{\mu}$ is derived from $\vec{\alpha}$ using Bayes' rule in the following way: $\vec{\mu} = \lim_{n\to\infty} \vec{\mu}^n$, where for each $n \in \mathbb{N}$, $\vec{\mu}^n$ is derived from $\vec{\alpha}^n$ using Bayes' rule.*

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

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

Λ is the discount factor (§3.5) 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 7.2 that the expected utility of player i at I is $u_i((\vec{\alpha}, \vec{\mu})|I) =$

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

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 7.4, $\vec{\alpha}|_{K-L}$ 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 $u_i((\vec{\alpha}, \vec{\mu})|I) \geq u_i(((\alpha_i', \vec{\alpha}_{-i}), \vec{\mu})|I)$ for any strategy α_i' of player i. Applying this argument for every I, we conclude that $(\vec{\alpha}, \vec{\mu})$ is ε-sequentially rational in $\mathsf{SpaceMint}_{\Pi,K,\rho}$.

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

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.

Sunoo's research is supported by the following grants: NSF MACS (CNS-1413920), DARPA IBM (W911NF-15-C-0236), and SIMONS Investigator Award Agreement Dated June 5th, 2012. Georg is supported by the French ANR EfTrEC project (ANR-16-CE39-0002). Joël and Krzysztof are supported by the European Research Council (ERC) consolidator grant 682815 - TOCNeT.

References

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

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

[3] Chia Network. https://chia.network/, 2017.

[4] H. Abusalah, J. Alwen, B. Cohen, D. Khilko, K. Pietrzak, and L. Reyzin. Beyond Hellman's time-memory trade-offs with applications to proofs of space. In *ASIACRYPT (2)*, volume 10625 of *LNCS*, pages 357–379. Springer, 2017.

[5] 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, Heidelberg, Sept. 2014.

[6] 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.

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

[8] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO 82*, pages 199–203. Springer US, 1983.

[9] P. Daian, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[10] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. http://eprint.iacr.org/2017/573.

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

[12] 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*, pages 397–406, 2003.

[13] 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, Heidelberg, Aug. 1993.

[14] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In R. Gennaro and M. Robshaw, editors, *CRYPTO 2015*, volume 9216 of *LNCS*, pages 585–605. Springer, 2015.

[15] 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, Heidelberg, Mar. 2011.

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

[17] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014.

[18] 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, Heidelberg, Mar. 2003.

[19] 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.

[20] N. P. Karvelas and A. Kiayias. Efficient proofs of secure erasure. In M. Abdalla and R. D. Prisco, editors, *Security and Cryptography for Networks - SCN 2014*, volume 8642 of *LNCS*, pages 520–537. Springer, 2014.

[21] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *CRYPTO 2017,*

Part I, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, Aug. 2017.

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

[23] S. Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

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

[25] 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.

[26] A. Miller, A. E. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 680–691. ACM Press, Oct. 2015.

[27] T. Moran and I. Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. `http://eprint.iacr.org/2016/035`.

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

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

[30] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[31] S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. `https://eprint.iacr.org/2015/528`.

[32] 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.

[33] 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, Heidelberg, Sept. 2010.

[34] L. Ren and S. Devadas. Proof of space from stacked expanders. In M. Hirt and A. D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285. Springer, Heidelberg, Oct. / Nov. 2016.

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

[36] The NXT Community. Nxt whitepaper. `https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf`, July 2014.

[37] 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 [14] have the following efficiency/security properties. Below, t_{hash} denotes the time required to evaluate the underlying hash function $\mathsf{hash}\colon \{0,1\}^* \to \{0,1\}^L$ 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 ([14] 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_v, k_p denote the parameter k we set for the commitment verification and the proof execution phase. If a (potentially cheating) prover $\mathcal{P}$ passes the commitment verification phase, then with probability $1 - 2^{\Theta(k_v)}$ the following holds: If $\mathcal{P}$ can make $\mathcal{V}$ accept in the proof execution phase with probability $\geq 2^{-\Theta(k_p)}$, then $\mathcal{P}$ either stores $\Theta(N)$ bits (i.e., almost as much as an honest prover) or runs in time $\Theta(n \cdot \log\log(n) \cdot t_{\mathsf{hash}})$ (i.e., the time required for initialization).*

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

Proposition A.2 ([14] 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_v, k_p be as above. If a (potentially cheating) prover passes the commitment verification phase, then with probability $1 - 2^{\Theta(-k_v/\log(n))}$ the following holds: If $\mathcal{P}$ can make $\mathcal{V}$ accept in in the proof execution phase with probability $\geq 2^{-\Theta(k_p)}$, then $\mathcal{P}$ either stores $\Omega(nL/\log(n)) = \Omega(N/\log(n))$ bits or requires $\Omega(N/\log(n))$ space and $\Omega(t_{\mathsf{hash}} \cdot n/\log(n))$ time during execution.*

To use the above PoSpace in our construction, we must set $k_v = \lambda \cdot \log(n)$, where λ is a statistical security parameter, and $k_p = \Theta(1)$ can be a constant.

B Burstcoin

Here we give some more details on the efficiency and security issues of Burstcoin as outlined in §1.2.

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 following 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 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[14]. 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. (How this is done is irrelevant for this discussion, so we omit it.) 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. 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. More-

[14] Note that in equation (6), a freshly computed block x_i is *pre*pended to the previous input. This is because 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.

over, 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, $i \leq 10$). This would require hashing only at most 50 blocks (as the miner computes $x_0, \ldots, x_i$ and sets $s_i = x_i \oplus x_{4096}$). 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 disk.

Given that application-specific integrated circuits (ASICs) can compute in the order of millions of hashes per second per dollar invested[15], such time/memory trade-offs seem practical[16]. 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`

Block grinding and extending multiple chains. The two main challenges we had to overcome when designing SpaceMint were attacks based on grinding and mining multiple chains. (The problem with time/memory trade-offs was solved in the Proofs of Space paper [14] 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 is thus "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 be included in a block).

C Challenge-grinding attacks

In this section we describe the challenge-grinding attack (which was communicated to us by Andrew Miller [24]), and our solution. Recall (from §3.5) that the quality of a blockchain in SpaceMint is defined by:

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

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

[16] 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).

where v_j is the quality of the proof in φ_j and $\mathcal{N}(v_j)$ is the space required for a proof of quality v_j. The discount factor Λ ensures that more-recent blocks weigh slightly more. For the purpose of this section, the factor Λ is not important, so we omit it. Then, notice that an equivalent measure is the *product* of block qualities:

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

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 $\mathsf{QualityPC}$ takes a sum, i.e.

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

which is mitigated by instead taking a product. 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. When using a space commitment (pk, γ) to compute 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 so 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 (subject to the minimum size allowed for space commitments): 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). (Recall 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 $N \cdot m$. 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 position $i+\Delta$: the expected quality of $\mathcal{A}$'s proof in time step i is only 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, $\mathcal{A}$, 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. (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.)

If the 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 as obtained by honest mining, 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 counter this attack, let us observe that what makes challenge grinding 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 proofs. As mentioned above, this variance is an important feature that we cannot simply remove; however, we can cluster proofs together so that the advantage from challenge grinding "amortizes" over many proofs. One way is 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 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 when 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 §D).

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 leads to more security but less efficiency) as there is a delicate interplay between them, where changing some parameter increases one security property at the price 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 the following, where the number in $[\cdot]$ brackets indicates the parameter value in our suggested instantiation.

- time [1] which specifies the time (in minutes) between blocks.
- δ [10] which specifies for how many blocks the same challenge is used.
- Δ [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 $\Delta + \delta$).
- Λ [0.99999] specifies how the weight of blocks – when computing the quality of a chain – degrades for older blocks.[17]
- minspace [100] specifies (in GB) the minimal size of space one must dedicate to start mining.

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

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 initialize the minimal allowed space, which scales linear in minspace as shown in Figure 2a. Decreasing time makes the scheme only more secure, but setting it very low will force miners to dedicate more computation. Also setting minspace high 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	-	-	-	⇓	-

Challenge grinding. We discussed challenge grinding in detail in §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 increasing minspace also makes the attack harder.

Extending multiple chains. We must ensure that for a miner it is 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 penalizing miners otherwise (§4). If there is a fork, we assume that with high probability this penalizing is sufficient to ensure that one branch will "die" within at most Δ blocks. 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 is 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 initialize the space in less than

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

This is far from the $\approx$ 200 minutes required to instantiate 100 GB of space, which is the minimum we suggest (Figure 2c on p. 13).

A promising approach to further harden Spacemint against space reuse is to use "proofs of space-time" as suggested in [27], which will make the initialisation of the space more expensive, but as a consequence also reusing the space comes at a much higher cost.

Long forks by space reuse. The situation is different if we consider an adversary who tries to create a long-range fork (and not extend the current chain) because (as specified in Eq. (4), p. 9) more recent blocks contribute more to the quality of a chain. We shortly sketch how this attacks works: Let cur 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 $\mathsf{cur} + \mathsf{low} + \mathsf{high}$ 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 adversary has $\mathsf{time}(\mathsf{low} + \mathsf{high})$ 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 $1/(1-\Lambda)$ (as the contribution of blocks from far in the past is just a small fraction ($\approx 1/e$) of the most recent blocks.)

Every time the adversary re-initalizes its space, it can generate challenges for the next $\Delta + \delta$ blocks. Let Tinit denote the time in minutes required for re-initialization. Consider an adversary that generates a chain while re-sampling even only 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.) Then the adversary is by a factor

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

slower than the speed at which the actual chain grows. This means it 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 $\mathsf{minspace} = 100(\mathrm{GB})$ we have $\mathsf{Tinit} \approx 200$ so $\mathsf{low} \approx 100\,000 \cdot 200/60$ minutes. Thus, even with our rough analysis, this implies that a fork would have to go back at the very least 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*[18] should be sufficient.

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

Long forks by space decrease. The above attack assumes $\Lambda < 1$. If $\Lambda = 1$ then there is no degradation of the contribution to chain quality of blocks 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 of 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. Another attack involves the adversary extending only his own blocks, and attempting to overtake the main chain. In this case, the adversary will only get rewarded for those blocks if the quality of his chain eventually exceeds that 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" blockchain, and later overwriting the transaction when his self-mined blockchain overtakes the main one.

Recall the quality of a block (from §3.5):

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

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,

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

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

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

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

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

quality of a blockchain (cf. §3.5) 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

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

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 that of the rest of the network. For any $M \in \mathbb{N}$, let $\mathcal{E}_M$ denote the event that after M blocks, the adversary's blockchain is of higher quality than the honest miners' blockchain. Then, by definition of $\mathsf{QualityPC}$, $\Pr[\mathcal{E}_M]$ equals

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

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 find that $\Pr[\mathcal{E}_M] =$

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

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

$$\begin{aligned} \mathsf{diff}_j &:= \log(-\log(v_j)) - \log(-\log \hat{v}_j)) \ , \\ \mathsf{diff}_j^{\Lambda,M} &:= \mathsf{diff}_j \cdot \Lambda^{M-j} \ . \end{aligned}$$

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

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

Theorem D.2.

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

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

$$\Pr[\mathcal{E}_M] \leq \exp\Big(-\frac{1}{2M} \cdot \Big(\sum_{j=1}^{M} \mathbb{E}\big[\mathsf{diff}_j^{\Lambda,M}\big]\Big)^2\Big) \ . \quad (14)$$

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

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

where the second equality follows from the fact that the diff_j are identically distributed for all j. Substituting this expression in Eq. (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 change slightly 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 adversarial parties is more than 50% of total space.*