Embracing Hellman: A Simple Proof-of-Space Search consensus algorithm with stable block times using Logarithmic Embargo

Marijn F. Stollenga
m.stollenga@gmail.com

October 10, 2022

Abstract

Cryptocurrencies have become tremendously popular since the creation of Bitcoin. However, its central Proof-of-Work consensus mechanism is very power hungry. As an alternative, Proof-of-Space (PoS) was introduced that uses storage instead of computations to create a consensus. However, current PoS implementations are complex and sensitive to the Nothing-at-Stake problem, and use mitigations that affect their permissionless and decentralised nature.

We introduce Proof-of-Space Search (PoSS) which embraces Hellman's time-memory trade-off to create a much simpler algorithm that avoids the Nothing-at-Stake problem. Additionally, we greatly stabilise block-times using a novel dynamic Logarithmic Embargo (LE) rule. Combined, we show that PoSSLE is a simple and stable alternative to PoW with many of its properties, while being an estimated 10 times more energy efficient and sustaining consistent block times.

1 Introduction

Proof-of-Space (PoS) consensus algorithms [9, 2] are an energy-efficient alternative to the common Proof-of-Work (PoW) algorithms, that replace commitment through computations with commitment through persistent storage, greatly reducing energy requirements. Instead of doing computations during the mining process, PoS pre-computes a large set of challenges and stores the result. Then, given a challenge, the closest pre-computed challenge can be retrieved in logarithmic time. This essentially reuses the computations done before, instead of throwing them away as in PoS.

However, current PoS implementations are designed such that every miner only performs a limited retrievals per block, in an effort to be more efficient. The result is that the mining process has a negligible cost, and miners don't have to commit to one version of the blockchain and can simply mine other versions in their down time. This is known as the Nothing-at-Stake problem, and current PoS implementations avoid it by introducing complex voting mechanisms that should ensure miners are honest. Such voting mechanisms however introduce complexities and add attack vectors that need to be covered, and negatively affect the decentralised and permissionless participation to the blockchain.

In this work we introduce a new consensus algorithm called Proof-of-Space Search with Logarithmic Embargo (PoSSLE), which introduces two innovations:

- A continuous retrieval challenge that embraces the *Hellmann time-memory trade-off* [11], instead of avoiding it like other PoS algorithms, avoiding the Nothing-at-State problem. Since the algorithm is completely bottlenecked by memory-reading operations this still results in an algorithm that is orders of magnitude more efficient than traditional PoW mining in our experiments (see Section 5.2).
- A novel *Logarithmic Embargo* block-acceptance rule which stabilises block-times greatly to a very consistent time range instead of exponentially distributed block-times of other blockchains. This greatly improves stability as network throughput stays consistent and block times are predictable.

2 Related work

2.1 Proof-of-Work

The publication of the Bitcoin whitepaper [17] introduced Proof-of-Work as a consensus mechanism, currently reliably securing up to *$600 billion* at it's peak in Bitcoin alone [16], and more than *$1 Trillion* including other crypto currencies. The beauty of PoW lies in its simplicity; the collective commitment of computational power of a permissionless, decentralised network enforces a singular history of transactions, without any peer having to ask for permission to join the network. Once a transaction is accepted in the blockchain, it becomes exponentially more difficult for an adversary to change the history of the blockchain as time progresses.

However, it is known that PoW is an energy hungry mechanism [13] due to the computational mining algorithm performing as many complex computations as possible. Therefore, researchers have been looking into ways to reduce this power consumption while retaining the simplicity and security of PoW.

2.2 Proof-of-Space

Proof-of-Space (PoS) introduced the idea of using memory storage instead of CPU cycles as the main commitment mechanism [9, 2]. Instead of continuously computing challenges, as in PoW, they are only computed *once* and then *stored* on a storage medium. This set of stored computations is known as a *plot*. After plotting, a miner simply looks up the desired result instead of computing it, essentially *reusing* the computation. This process is known as *farming* to distinguish it from the energy intensive *mining* (from here on, we will refer to PoS miners as *farmers*).

2.3 The Nothing-at-Stake problem

Current implementations of PoS explicitly reduce the amount of times a *plot* needs to be searched to a minimum, ideally once per block, to maximise efficiency of the chain. Both in the largest PoS blockchain – Chia [8] – and the Spacemint blockchain [18] a Verifiable Delay Function (VDF) is introduced. The

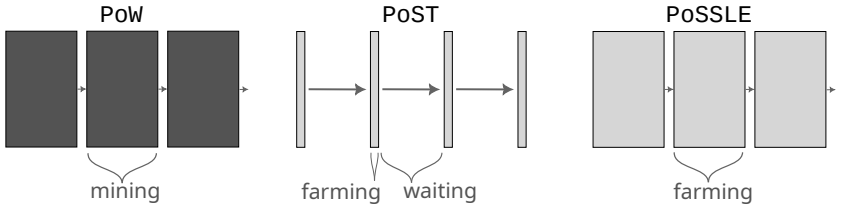

Figure 1: PoW vs PoST vs PoSSLE. PoW uses a continuous high-energy load during the mining process. PoST is the most efficient with considerable down time in which the VDF is evaluated, with causes the Nothing-at-Stake problem. PoSSLE instead uses a continuous search that doesn't have the Nothing-at-Stake problem, but relies solely on retrieval operations which are much more efficient than PoW.

VDF is a function that requires an very large amount of sequential computation, such that it can't be evaluated quickly by scaling up through parallelism.

The computation of the VDF functions as a Proof-of-Time; it ensures a certain amount of time has passed to evaluate this function. Together, this method is called Proof-of-Space-Time (PoST).

By using the output of the VDF as a challenge, the consensus mechanism only rarely performs a *plot* search and is idle most of the time (see Figure 1). However, this incentivises farmers to spend their free time on *other* chains that fork from the most secure chain, since there is essentially no downside to do so and it increases their chances to farm a block. This is known as the *Nothing-at-Stake* problem [14], which greatly de-stabilises the protocol and needs to be mitigated by an increase in protocol complexity. All methods battling the Nothing-at-Stake problem introduce some system of *validators* that build reputation and *vote* on correctly behaving nodes and punish bad ones. These mitigations greatly increase *protocol complexity* and decrease *permissionless participation* and *decentralisation*.

2.4 Hellman's Time-Memory Trade-off

Hellman showed that a deterministic random[1] permutation function can be inverted faster than simple random search, by trading off computation time for space [11]. In general, Hellman showed that a function permuting in a domain of size N can be inverted in $T * S = N$, where T is the time requirement, S the committed space; the more space is committed, the fewer computations are needed for the inversion.

In this work, we use this trade-off to create a PoS algorithm that combines the simplicity of PoW with the efficiency of PoS. The insight is that a continuous search algorithm can be embraced that uses the trade-off to focus time spent on memory retrieval instead of computation, which is inherently more efficient as we show in experiments. This is in contrast to current PoST algorithms, which actively try to avoid Hellman's trade-off [1, 8].

As part of the algorithm we relax Hellman's inversion and introduce a novel dynamic challenge, called the Logarithmic Embargo (LE), which reduces chal-

[1]Officially 'deterministic' and 'random' are mutually exclusive. We use the colloquial 'random' here for simplicity; more accurate would be a cryptographic or chaotic permutation function.

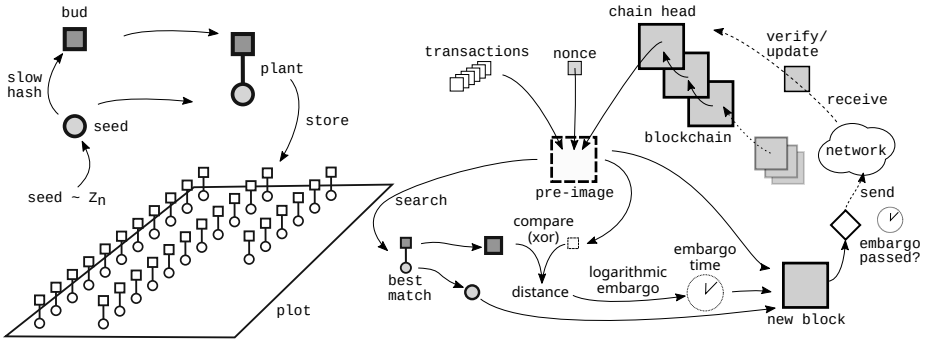

Figure 2: Diagram of Proof-of-Space Search with Logarithmic Embargo (PoSSLE).

lenge requirements as time progresses. This simple addition ties in neatly into the algorithm and greatly stabilises block-times.

3 Method

We introduce Proof-of-Space-Search with Logarithmic Embargo (PoSSLE), which combines two techniques in one stable and efficient consensus mechanism:

- Proof-of-Space-Search (PoSS), a novel consensus mechanism that embraces the Hellman trade-off and trades computational work for memory retrieval. The algorithm is running a continuous search for the best block like PoW, and unlike PoST, thus avoiding all complexities arising from the Nothing-at-Stake problem.
- A Logarithmic Embargo (LE); a dynamic block acceptance rule that reduces the challenge over waiting time; greatly increasing stability of the network by creating consistent block-times (see Section 3.3).

3.1 Formalisation

We define a cryptographic hash function $f : x \to y$, which deterministically maps an input $x \in \mathbb{Z}_N$ to an output $y \in \mathbb{Z}_N$ where $\mathbb{Z}_N$ is the set of non-negative integers modulo N. Typically, N is a dyadic number determined by the bit-size of the corresponding hash function.

We distinguish between two hashing functions: a fast hashing function f that is designed to calculate a cryptographic hash as fast as possible, and a slow hashing function $\overline{f}$ that is designed to take a considerable amount of computing power. In this work the Blake2 algorithm [3] for f while the Argon2D algorithm [4] is chosen for $\overline{f}$.

We define a *plant* p to be a pair of values $p = \{s, b\}$, where $s \in \mathbb{Z}_N$ is the *seed* value, and $b = \overline{f}(s)$ the *bud* value. A farmer pre-computes a set, or *plot*, of *plants* $P = \{p_0, p_1..p_M\}$. The seeds are sampled randomly from a uniform distribution $s \sim \mathcal{U}(0, N)$ and the plot is stored in a memory medium of choice.

We use $h_{\texttt{name}} \in \mathbb{Z}_N$ to define a certain hash h. All our hashes are in the domain $\mathbb{Z}_N$.

3.2 Proof-of-Space Search

Any blockchain needs to define a challenge that determines if a block is valid. The state of the blockchain is embedded into the challenge, and the entity solving the challenge inherently commits to the validity of this information by investing resources.

The information included is a block $block_t$ is:

- $h_{\texttt{prev}} = block_{t-1}$, the previous block hash.
- $h_{\texttt{new}}$, a hash based on any new information to include in the block. These can be new transactions $h_{\texttt{new}} = f(tx_0, tx_1, ...)$ or any other information, depending on the blockchain.
- $h_{\texttt{nonce}}$, a *nonce* value. This value can be changed and allows the farmer to freely affect the hash of this block without changing the other meaningful information.

This information is hashed together:

$$block'_t = f(h_{\texttt{prev}} \frown h_{\texttt{new}} \frown h_{\texttt{nonce}}) \tag{1}$$

where $block'_t$ is a pre-block hash, $\frown$ is the concatenation operation[2].

Retrieval Challenge At this point the challenge for PoW would be:

$$block' < \tau \tag{2}$$

where $\tau \in \mathbb{Z}_N$ is a fixed threshold.

However, this challenge does not allow for any pre-computations. The only way a pre-computed hash value is applicable is if input to the hash function is *exactly* the hash of the block; a coincidence that is impossibly unlikely.

Instead, we need to *loosen* the challenge. Given the pre-hash $block'$, we find the plant $p_i = \{s_i, b_i\}$ that minimises a *proof distance*:

$$\hat{p} = \min_{p_i \in P} d(b_i, block') \tag{3}$$

$$d(b_i, block') = b_i \oplus block' \tag{4}$$

where $d(\cdot, \cdot)$ is the *proof distance* and $\oplus$ is the binary *XOR* operation. The output of $d(\cdot, \cdot) \in \mathbb{Z}_N$ lies in the same domain as its inputs. The minimisation of Formula 3 can be performed in logarithmic time (see Section 4.3).

This process can be repeated by changing $h_{\texttt{nonce}}$, resulting in a new pre-hash and a new chance to find an even closer match:

$$\min_{block', h_{\texttt{nonce}}} d(b_i, block) \in \mathbb{Z}_N \tag{5}$$

It is always more beneficial to *search* for a new challenge than to *compute* new plants; a new search efficiently finds a match among a whole plot, which would cost an enormous amount of energy to compute on the fly. This ensures that

[2]For simplicity we use concatenation here, in the actual implementation on can use a Merkle-tree[15]

there is an incentive to use memory lookups instead of spending computational power.

To compute the final block hash, the prehash $block'$ and the seed value s are combined:

$$block = f(block' \frown s) \tag{6}$$

The seed value s forms the non-trivial proof that the user performed work to find a plant p that minimised the proof distance.

3.3 Dynamic Threshold

The final ingredient we need is a distance threshold τ for $d(b_i, block')$ that defines a valid block. A fixed threshold can be used like Formula 2, as is used in Bitcoin, which would result in a Poisson process with exponentially distributed block times. This variability has negative implications to the user experience and stability of the network (the longest block time recorded on Bitcoin was 141 minutes – 14 times the target 10 minutes).

Instead, we introduce a novel dynamic threshold called the Logarithmic Embargo (LE). A block is not valid before a *time-embargo* is lifted, where the embargo depends on the proof distance $d \in \mathbb{Z}_N$ (Formula 4). The shorter the distance, the earlier it becomes valid.

First, we define the *work* that went in finding the proof using the negative log-likelihood:

$$\texttt{work}(d) = -\log_2(\frac{d}{|\mathbb{Z}_N|}) \quad \text{where: } d \neq 0 \tag{7}$$

where $|\mathbb{Z}_N|$ is the maximum possible distance. We then calculate the embargo as follows:

$$t_{\texttt{embargo}}(d) = T_{\texttt{target}} \frac{C}{\texttt{work}(d)} \tag{8}$$

where $T_{\texttt{target}}$ is the target block time, and C a constant that is adjusted such that the fraction $\frac{C}{\texttt{work}(d)} \approx 1$.

The calculation of C depends on the farming speed of the network and needs to be adjusted as this speed changes over time. We recalculate C every K blocks, by observing the embargo values of the last K blocks:

$$C_t = C_{t-K} \frac{1}{K} \sum_{i=t-K}^{t-1} \frac{T_{\texttt{target}}}{t^i_{\texttt{embargo}}} \tag{9}$$

where $t^i_{\texttt{embargo}}$ is the embargo of block i.

Final block-chain rule As with any consensus algorithm, several blocks might be proposed at the same time, which could represent completely different chains with different parent blocks. Therefore we define the final block-chain rule: at any time the winning block is the block of which all blocks in its chain are valid, and the accumulated *work* is maximised:

$$\max_{block} \sum_{block_i \in \texttt{History}(block)} 2^{\texttt{work}(block_i)}. \tag{10}$$

where $b_i \in \texttt{History}(block')$ is the set of all blocks $block'_i$ in the history of block $block'$. The work is added in linear space, hence the power of 2.

Once a valid block is published, there is no incentive for a farmer to keep working on an older block, since their proof would be strictly weaker than when using the newer block.

3.4 Block Time Distribution

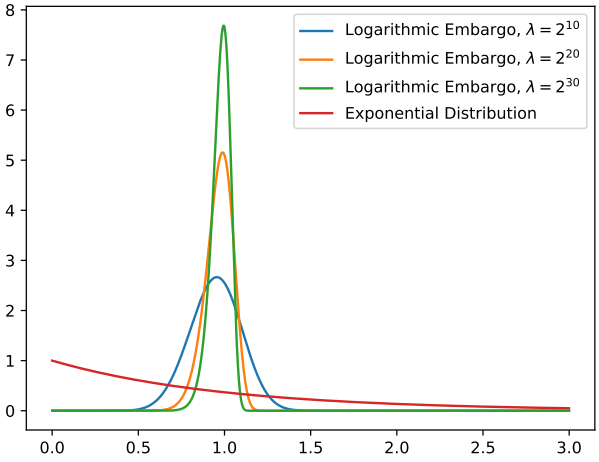

Figure 3: Probability distribution of block waiting times under Logarithmic Embargo for mining rates: $\lambda = \{2^{10}, 2^{20}, 2^{30}\}$. Also the exponential distribution is shown. The block times under Logarithmic Embargo are closely distributed around the target block time, while the exponential distribution is spread wide with a long tail.

Here we analyse the distribution of block-times under the dynamic threshold. Firstly, the distance $d(b, b')$ between a random bud value b and a pre-hash b' are randomly distributed $d \sim \mathcal{U}(0, N)$. The chance that this distance is below a given threshold $\tau \in \mathbb{Z}_N$ is:

$$p(d < \tau) = \frac{\tau}{N} \tag{11}$$

Every time a plot is searched, all plants in the plot are considered, counting as a repeated search. This happens across the network among many peers, and these searches are repeated with different $h_{\texttt{nonce}}$ values. To account for this, we assume a stable mining rate across the network of λ searches per second. In a given time of T seconds, λT searches are performed on average. This gives the following chance a distance below threshold τ is found in a given time interval

T:

$$p(\exists_i d_i < \tau; 0 < i < \lambda T) = 1 - (1 - \frac{\tau}{N})^{\lambda T} \tag{12}$$

Using the embargo formula 8 we can calculate the required τ to create a valid block in time T:

$$\texttt{work}(d) = C \frac{T_{\texttt{target}}}{t_{\texttt{embargo}}(d)} \tag{13}$$

By using $d = \tau$ as the required distance, $t_{\texttt{embargo}}(d) = T$ as the required embargo, and expanding $\texttt{work}(d)$, we get:

$$-\log_2 \frac{\tau}{N} = C \frac{T_{\texttt{target}}}{T} \tag{14}$$

$$\tau = N 2^{-C \frac{T_{\texttt{target}}}{T}} \tag{15}$$

We can plug this back in formula 12:

$$p(\exists_i d_i < \tau(T); 0 < i < \lambda T) = 1 - (1 - 2^{-C \frac{T_{\texttt{target}}}{T}})^{\lambda T} \tag{16}$$

This function is expresses the cumulative chance that a valid block is found within a time period T. Since this is a cumulative distribution we can take the derivative of T to get the probability that the valid block happens at time T. This results in a complex function which we omit, but we plot it for different values in Figure 3. We can see that the block times are concentrated around a small period, resulting in stable and predictable block times, while the exponentially distributed times of other block chains are spread out over a large range. We verify these block times in experiments in Section 5.1.

4 Implementation

To validate PoSSLE we create an implementation in the Zig language [12]. We use the Argon2D algorithm [4] for our slow hash function $\overline{f}$, with an input and output domain of 256 bits: $\mathbb{Z}_{2^{256}}$, and parameters *time* = 1, *memory* = 128, *parallelism* = 1. There parameters were determined to be a good trade-off for the desired plotting speed, however other parameters could be used to make plotting faster or slower. We use the Blake2 algorithm [3] for our fast hash function f, again with an input and output domain of 256 bits: $\mathbb{Z}_{2^{256}}$.

The source code is maintained at `https://github.com/marijnfs/zig-possle`.

4.1 Plotting

Before we can farm, we need to create a plot. For each plant, the seed and bud need to be stored. Both the *seed* and *bud* consist of 256 bits, or 32 bytes. Thus one *plant* consists of 64 bytes of information. For our implementation we use a plot size of 128GB, which can contain 2^{31} plants. This is perfect for indexing, since we can use 32 bit integers with one bit to spare which we need to differentiate between leafs and nodes.

To efficiently plot this number of plants, we use the following strategy:

1. $N_{\texttt{base}}$ plants $p_i = (s_i, b_i)$ are created by sampling seeds from $s_i \sim \mathcal{U}(0, 2^{256})$ and calculating bud $b_i = \overline{f}(s_i)$.
2. Plants p_i are sorted by b_i and stored in memory in a set S.
3. Steps 1-2 are repeated creating sets of size $N_{\texttt{base}}$; consecutive sets are merged recursively by *merge sort* [5].
4. Once the set is too large to fit in memory, it is stored to disk, creating a set of size $N_{\texttt{persistent}}$.
5. Steps 3-4 are repeated, creating plots on disk. Consecutive sets are merged recursively on disk by *merge sort*.
6. This process is repeated until the required plot size N is achieved.

Algorithm 1 Find first changing bit in a plot

Require: $p = (s_0, b_0), (s_1, b_1)..(s_N, b_N)$ is a plot sorted by b_i

```
function FindChangingBit(l, r, bit)
    while l ≠ r do
        m ← (l+r)/2
        if b_m[bit] then
            r ← m
        else
            l ← m + 1
    return l
```

4.2 Indexing

The plotting algorithm results in a set of plants, sorted by their bud value. To perform lookups in this set in $\log N$ time, we need to index the set using a *binary trie* [10]. This process is defined in algorithm 2. For our set of 2^{31} plants, we can create a *trie* index using 32 bit indices. We can index the nodes and plants in the *trie* using 31 bits, and use the 32th bit to distinguish between them. We need as many indices as there are nodes, and 32 bit index uses 4 bytes, thus we need 4×2^{31} bytes, or 8GB, to store the indices. The result is a $\frac{1}{16}$ additional use in space; well worth the $\log N$ lookup time.

4.3 Farming

The *farming* process consists of creating a block pre-hash $block'$ using Formula 1 and finding the closest bud using the algorithm described in Algorithm 3. Starting from the first node, the trie structure is descended following the current search bit to choose between the *left* and *right* node. In the case a reference is empty it means there are no plants in that range with this bit, and the other reference needs to be taken. Finally a leaf will be found, and the corresponding plant is returned.

The farmer repeats this process and keeps track of their best block that builds upon the blockchain, with the shortest embargo (Formula 5), waiting for the embargo to lift.

Algorithm 2 Building a binary trie for farming

```
function BuildTrie(p)
    trie ← List(∅)
    stack ← Stack(∅)
    Push(stack, {0, {0, N}, ∅})
    while {bit, {L, R}, parentref} ← Pop(stack) do
        node ← {L = ∅, R = ∅}
        idx ← FindChangingBit(bit, L, R)
        if idx = L then
            node.L ← ∅
        else if idx = L + 1 then
            node.L ← {L, leaf = True}
        else
            Push(stack, {bit + 1, {L, idx}, &node.L})
        if idx = R then
            node.R ← ∅
        else if idx + 1 = R then
            node.R ← {idx, leaf = True}
        else
            Push(stack, {bit + 1, {idx, R}, &node.R})
        node_idx ← Insert(trie, node)
        if parentref ≠ ∅ then
            ⋆parentref ← {node_idx, leaf = False}
    return trie
```

Algorithm 3 Find closest plant in a plot in $\log(N)$ time

```
function Search(pattern, trie, plot)
    idx ← 0
    bit_idx ← 0
    while True do
        if trie[idx].L = ∅ then
            idx ← trie[idx].R
        else if trie[idx].R = ∅ then
            idx ← trie[idx].L
        else if pattern[bit_idx] then
            idx ← trie[idx].R
        else
            idx ← trie[idx].L
        bit_idx ++
        if IsLeaf(idx) then return plot[idx]
```

5 Experiments

The experiments are carried out on an 8-core AMD Ryzen 9 4900H with 64G memory and a WD Blue SN570 500GB Solid State Disk (SSD) for storage.

5.1 Block-times

We test if the block-times behave as predicted in Section 3.4. We ran an experiment with 32 peers with target block times of 5 seconds for one day. We plotted all block times of the experiment in the histogram in Figure 4. For comparison, all block times of the Bitcoin network until August 2022 are plotted as well. The graph is normalised to the target block time for comparison.

The block-times clearly follow the predicted distributions in Figure 3, with PoSSLE showing block times close to the target, while Bitcoin's PoW shows a spread out exponential distribution. Bitcoin's block-times show a small dip in the very short block-times that don't follow the exponential distribution. This is possibly caused by synchronisation issues where a second block is found so quickly that the previous block is not spread over the network yet and gets rejected.

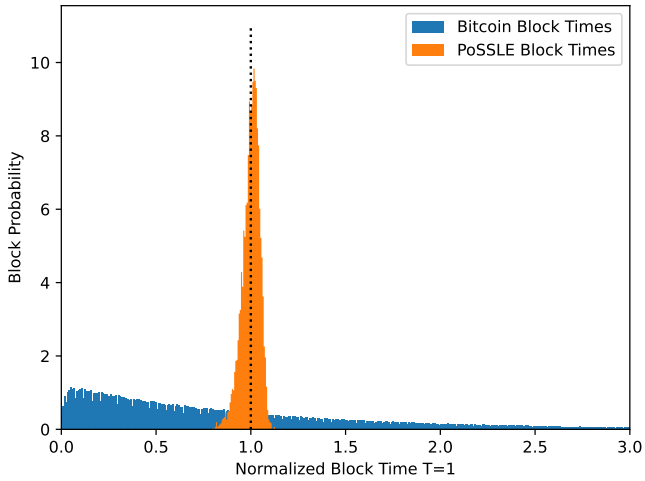

Figure 4: Block time distribution of PoSSLE vs Bitcoin's Proof-of-Work (PoW). The block time distribution of all Bitcoin blocks until August 2022 are plotted, versus the block times of PoSSLE in our experiment. The block times are normalised such that the target block time is 1, denoted by a black vertical line. The block times of PoSSLE are closely distributed around the target, whereas the Bitcoin block times are spread exponentially with a long tail that extends beyond the graph end.

5.2 Performance

We test individual parts of the algorithm in isolation on a single core, to see how they contribute to the bottleneck:

Calculating the pre-hash *block′* On a single core we can compute about 440,000 pre-hashes per second.

Descending the binary trie We index a 128GB plot, and randomly pick a pattern to search in the resulting *trie*. We only look at the index lookup and don't actually retrieve the corresponding item. This can be done 1,500,000 per second.

Retrieving a plant from memory Loading a plant from memory can be done quickly. We loaded 32MB of plants into memory and could perform $400 \cdot 10^6$ random reads per second.

Retrieving a plant from SSD Loading random plants from a 128GB plot directly results in 1,100,000 plants per second.

From this we can conclude that the bottle neck is the random read operations per second of the SSD. This can be expected due to the farming algorithm; the CPU is performing hashes and doing memory retrieval to descend the *binary trie* and has very little work to do, the memory is in theory continuously used to retrieve nodes of the binary trie, but it outpaces the lookups of the SSD and has to wait, resulting in a lower load. Finally, the SSD is the bottleneck because it can perform only a certain number of random access operations per second.

Note that the throughput of the SSD is still far below it's maximum load since only very small 64 byte *plants* need to be retrieved, and not larger blocks (a typical SSD can load 4MB blocks at a time). The result is that this bottle-neck results in none of the components being loaded heavily, making the algorithm efficient.

Using memory instead of SSD Since memory is much faster than an SSD, one might think of storing a plot directly in memory. However, the memory capacity of computer memory is much lower than an SSD for the same price. A 128GB memory kit costs about the same as a 4TB SSD with 32 times the capacity. Additionally, the bottleneck of the *binary trie* means that the speed of the memory can't be used fully anyway. Therefore, an SSD is a better choice for storage medium.

6 Discussion

6.1 Energy Considerations

Mechanism	Energy Usage	Protocol Complexity	Decentralisation
Centralised Server	Negligible	None	None
Proof-of-Work	High	Low	Full
Proof-of-Space-Time	Low	High	Moderate
Proof-of-Space-Search	Moderate	Low	Full

Table 1: Comparison of Consensus Mechanism: Efficiency versus Decentralisation.

One of the main selling points of Proof-of-Space is their lower energy usage. Energy usage estimation is not as simple are recording the energy usage of a

mining/farmer server, but greatly depends on the farming/mining rewards and thus the popularity of a crypto currency. To compare fairly, we compare the energy usage of PoW, PoST, and PoSSLE by looking at how much energy is used for every dollar invested in the mining/farming process. We can assume a miner/farmer wants the most return for their investment and we assume a time horizon of two years.

For PoW we can look up the best mining hardware, that has been optimised over years. We use the Antminer S19 Pro for the comparison; one of the most efficient PoW miners. It uses 3250W and costs $4300. Over two years we would use about 57mWH, and with a energy cost of 0.20 $/kWH, we would spend about $11400 in energy, or about 73% of the budget.

For PoST, we look at specialised mining hardware for the Chia blockchain. A dedicated mining rig with 32 harddisks can be found for $2499 [7]. It is advertised as using 50W, or 876kWH over two years, costing about $175 or 6.5% in electricity. However, it could be expected that power usage is even lower given the sparsity of work within PoST.

For PoSS we have to make an estimate as no existing blockchain uses it. Since the efficiency of the PoSS algorithm is maximised by using the fastest memory that can search the largest amount of space, the hardware of choice are solid state disks (SSD) with a high throughput. SSDs have been optimised for random memory access over many years and are used in essentially any computer younger than a few years, thus we can treat this as relatively optimised hardware.

A fast SSD (Western Digital SN750 M.2 2000 GB) costs about $190 and uses 3W at peak load. As shown in the experiments, we are unlikely to use this full load. However, we also have to account for overhead of the hardware around the SSDs, including computer memory to store the index. Therefore we use the full 3W peak load, and estimate an optional 100% overhead. Thus we estimate about 3W-6W power usage for every $190 invested, or 52-105KWh and $11-$21 in two years; about 5-10% in electricity.

This would make PoSS and PoST approximately 10 times more efficient than PoW. For perspective, the Bitcoin network used about 80.91TWh in electricity in 2021 [6]. If PoSSLE would be used as an replacement for PoW in Bitcoin, it would save about 73TWh in energy; around the electricity consumption of Belgium.

We summarise the comparison in Table 1. We estimate a lower energy usage for PoST than our estimates since this is reported for the Chia ,

It is also worth noting that the hardware for PoSSLE and PoST are general storage devices that are repurposable for other uses, essentially recycling the hardware. This is not true for the specialised hardware in PoW which can only perform one type of computation very fast.

6.2 Embracing Hellman

The PoSS algorithm follows Hellman's time-memory trade-off [11]. This can be seen by considering a farmer who allocates M plants to the search, representing our memory commitment. This user can search this space in $\log_2(M)$ steps, but we use a domain of size $N = 2^{256}$ so we can use the constant upper-bound $\log_2(2^{256}) = 256$, since our domain is finite by design.

The chance that the closest distance between a random element and set M is less than threshold τ is $p = \frac{\tau}{2N/M}$. Given the Bernoulli distribution, the expected number of operations T required is $T = \frac{2N}{\tau M}$, and we get the relation $\frac{2N}{\tau} = TM$; up to a constant factor of the Hellman equation $N = TM$ due to our relaxation of the challenge.

7 Future Work

Plot Sharing An interesting property of POSS is that peers do not need to build unique plots. In fact, peers could share their plots such that a large plot is computed only once and shared after that. This greatly reduces the amount of energy spent for plotting, which is significant for other POS algorithms. However, to be fully self reliant, a peer might still want to calculate a plot on its own to avoid malicious sharing of false plots.

Early Syncing To avoid the bursts of network traffic around the target block-time, blocks could be shared before their embargo, but without accepting them. Peers could keep track of the best successor block that is still under embargo, while mining their own competing block. Then when the embargo is lifted, the peer can switch to the new chain head, without having to download the block again, spreading out the network traffic.

Trie Extension Our binary *trie* datastructure uses 1/16th of the space of the target plot. This factor can be extended by not just storing a single *plants* at the leafs, but groups of them. Since a (small) group can be retrieved and searched in linear time, we can address a much larger amount of data while keeping the index small enough to keep it entirely in computer memory for efficiency.

8 Conclusion

We have introduced PoSSLE, a simple and energy efficient proof-of-space consensus algorithm with predictable stable block-times through a novel Logarithmic Embargo rule. PoSSLE implements a continuous farming algorithm that combines the simplicity of PoW with the efficiency of PoS by embracing the Hellmann time-memory trade-off instead of avoiding it like other PoS algorithms. This avoids the Nothing-at-Stake problem and the many complexities that come with it.

We believe PoSSLE provides the security and simplicity of a PoW, with the energy efficiency of a PoS. Through our implementation and currently available hardware, we estimate that PoSSLE would uses a 10th of the energy of a PoW algorithm – if PoSSLE was used in Bitcoin this would be a savings of 73TWh in energy; around the electricity consumption of Belgium.

More work is needed to validate PoSSLE at the scale of a typical cryptocurrency.

References

[1] Hamza Abusalah et al. “Beyond Hellman’s time-memory trade-offs with applications to proofs of space”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 357–379.

[2] Giuseppe Ateniese et al. “Proofs of space: When space is of the essence”. In: *International Conference on Security and Cryptography for Networks*. Springer. 2014, pp. 538–557.

[3] Jean-Philippe Aumasson et al. “BLAKE2: simpler, smaller, fast as MD5”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2013, pp. 119–135.

[4] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. “Argon2: new generation of memory-hard functions for password hashing and other applications”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 292–302.

[5] Coenraad Bron. “Merge sort algorithm”. In: *Communications of the ACM* 15.5 (1972), pp. 357–358.

[6] Cambridge. *Cambridge Bitcoin Electricity Index*. 2022. URL: `https://ccaf.io/cbeci/index`.

[7] *CHIA (XCH) Mining – Ploting / Farming Rig 2TB/256TB*. 2022. URL: `https://xprimeshop.net/chia_mining_farming_hardware/chia_XCH_mining%E2%80%93ploting-farming_rig_2TB-256TB`.

[8] Bram Cohen and Krzysztof Pietrzak. *The chia network blockchain*. 2019.

[9] Stefan Dziembowski et al. “Proofs of space”. In: *Annual Cryptology Conference*. Springer. 2015, pp. 585–605.

[10] Edward Fredkin. “Trie memory”. In: *Communications of the ACM* 3.9 (1960), pp. 490–499.

[11] Martin Hellman. “A cryptanalytic time-memory trade-off”. In: *IEEE transactions on Information Theory* 26.4 (1980), pp. 401–406.

[12] Andrew Kelley. *Zig Programming Language*. 2022. URL: `https://ziglang.org/`.

[13] Varun Kohli et al. “An Analysis of Energy Consumption and Carbon Footprints of Cryptocurrencies and Possible Solutions”. In: *arXiv preprint arXiv:2203.03717* (2022).

[14] Wenting Li et al. “Securing proof-of-stake blockchain protocols”. In: *Data privacy management, cryptocurrencies and blockchain technology*. Springer, 2017, pp. 297–315.

[15] Ralph C Merkle. “A digital signature based on a conventional encryption function”. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 369–378.

[16] Binance Capital Mgmt. *CoinMarketCap*. 2022. URL: `https://coinmarketcap.com/`.

[17] Satoshi Nakamoto. “Bitcoin whitepaper”. In: *URL: https://bitcoin.org/bitcoin.pdf (: 17.07. 2019)* (2008).

[18] Sunoo Park et al. “Spacemint: A cryptocurrency based on proofs of space”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2018, pp. 480–499.