The Chia Network Blockchain

Bram Cohen[1] and Krzysztof Pietrzak[2]

[1]Chia Network `bram@chia.net`
[2]IST Austria `pietrzak@ist.ac.at`

July 9, 2019

Abstract

This document outlines the basic design principles of the consensus layer (the blockchain) of the Chia network. It is inspired by and similar to the Bitcoin blockchain, which achieves consensus when a majority of the computing power dedicated towards securing it is controlled by honest parties. In Chia the resource is not computing power, but disk space.

To achieve this, the *proofs of work* used in Bitcoin are replaced by *proofs of space*. To get a mining dynamic like in the Bitcoin blockchain, Chia alternates proofs of space with *verifiable delay functions*.

We provide an initial security analysis of the Chia backbone, showing that as long as at least $\approx 61.5\%$ of the space is controlled by honest parties Chia satisfies basic blockchain security properties.

Glossary

We reserve the following letters throughout this writeup:

$w \in \mathbb{Z}^+$ a security parameter that we use for various things, such as the output of H below or the size of a challenge: $w = 256$ is sufficient for all cases.

$\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^w$ a cryptographic hash function, modelled as a random oracle for proofs.

$T \in \mathbb{Z}^+$: Chia difficulty parameter (has a function similar to the difficulty parameter in Bitcoin).

$\kappa \in \mathbb{Z}^+$: honest farmers work on the κ best paths (presumably $\kappa = 3$ in Chia).

$\theta_i \in [0,1]$: speedup factor one gets by using $\kappa = i$ compared to $\kappa = 1$ (illustrated in Figure 2).

$\iota_i = 1 - \frac{1}{1+e\cdot\theta_i} \in [0,1]$ fraction of space honest farmers must hold if they use $\kappa = i$ (illustrated in Figure 2).

$\eta \in \mathbb{R}^+$: seconds required to compute one step (a squaring in a group of unknown order) of the verifiable delay function (VDF).

$\xi \in \mathbb{R}^+$: max. fluctuation of T allowed in consecutive epochs (in Bitcoin the corresponding parameter is 4, and this will be adapted in Chia).

Contents

0 Outline

In Section §1 we discuss the main **challenges** one encounters when designing a blockchain without relying of proofs of work, and how they are solved in Chia.

In §2 we introduce the main **building blocks** of Chia. Apart from standard primitives like unique signatures and cryptographic hash functions, these include new proof systems which have been developed more recently, namely, proofs of space (PoSpace) and verifiable delay functions (VDF).

In §3 we specify the **blockchain format** of Chia which differs in some crucial points from the Bitcoin blockchain. Instead of using proofs of work, Chia alternates proofs of space with verifiable delay functions. This results in a chain than in many aspects is similar to Bitcoin, in particular, as in Bitcoin no synchronisation is needed and we can prove rigorous security guarantees assuming a sufficient fraction of the resource (space in Chia, computation in Bitcoin) is controlled by honest parties.

To prevent grinding attacks (as discussed in §1) the Chia chain is split into two chains, one "ungrindable" chain called the *trunk*, which only contains the PoSpace and VDF outputs, and another chain called *foliage*, which contains everything else.

Section §4 contains the **(pseudo)code** for the algorithms used by the *farmers*, who compute the proofs of space, and the *time lords* who compute the VDF outputs.

In §5 we discuss the **difficulty adjustment** in Chia. It differs from the way difficulty adjustment work in Bitcoin as a subtle grinding attack would be possible if one would naïvely adapt it.

We also make an observation which seems not be generally known, but Nakamoto – the anonymous Bitcoin designer(s) – might have been aware of as it explains why the factor by which the difficulty can vary in consecutive epochs (an epoch refers to a sequence of blocks using the same difficulty parameter, in Bitcoin that's 2016 blocks) is set to the rather large value of 4. We show an attack that is possible if that factor was less than $e \approx 2.718$.

In §6 we provide an **initial security analysis** for Chia. We discuss this in more detail below in §1.1, but in a nutshell, for the most basic design we prove security as long as the honest farmers control at least $\approx 73.1\%$ of the total space. This can be improved by having the honest farmers extending not just the first, but the first $\kappa > 1$ chains for every depth. Which value of κ to use is a "social convention" rather than a protocol parameter. By using $\kappa = 3$ the honest farmers just need to control $\approx 61.5\%$ of the space.

1 Introduction

1.1 Bitcoin

We assume the reader has some familiarity with Bitcoin, and just mention the aspects and results that are relevant for discussing Chia. Recall that in the Bitcoin blockchain, to extend a block β, the miner must provide a proof of work for a challenge $c = \mathsf{H}(\beta, \mathsf{data})$ derived by hashing the block β to be extended and the payload data (transactions and a timestamp) of the block to be added. The proof is a nonce ν such that[1]

$$0.\mathsf{H}(c, \nu) < 1/D$$

where D is the difficulty parameter and H is a cryptographic hash function (SHA256). If H is modelled as a random function, in expectation D invocations of H are necessary and required to find such a proof.

Bitcoin specifies that miners should always work towards extending the longest chain they are aware of.[2] A malicious miner controlling *more than half* of the hashing power has full control over the chain. He can ignore blocks contributed by other parties so only his blocks appear in the chain, and thus get all the rewards and censor transactions. Even worse, such a 51% adversary can double spend.

Presumably it was assumed by Nakamoto that a malicious miner who controls *less than half* of the total hashing power can't gain anything by deviating from the honest mining strategy. This intuition was shown to be wrong due to subtle selfish mining attacks [ES14]: a miner controlling $\alpha < 1$ times the

[1] Here, for $X \in \{0,1\}^w$ we denote with $0.X$ a binary value, thus $0.X = \sum_{i=1}^{w} 2^{-X[i]} \in [0,1)$.

[2] Technically, it is the chain that accumulates proofs of work of highest total difficulty, typically (but not necessarily) that will be the chain with the largest number of blocks.

hashing power of the honest miners (i.e., less than half of the total) can add an α fraction of the blocks to the chain (compared to the $\frac{\alpha}{1+\alpha}$ fraction they would get by following the rules). On the positive side, [GKL15] show that there is no better attack. They define the chain quality as the fraction of blocks an adversarial miner can push into the chain, and show that for an adversary as above, this fraction is at most $\frac{\alpha}{1+\alpha}$.

1.2 Attacks

Grinding attacks. Unfortunately, the simple analysis of the Bitcoin backbone from [GKL15] does not extend to blockchains that use proof systems other than proofs of work, like proofs of stake (PoStake) or proofs of space (PoSpace). The reason is that in a PoW-based blockchain one can use the mining resource (the hardware and electricity required for doing the computation) only towards extending one particular block for one particular challenge. On the other hand, proof systems like proofs of space (or stake) are very cheap to compute, so a malicious miner can potentially launch a *grinding* attack by deviating from the honest mining rules and computing many more proofs than specified by the protocol (in the context of proofs of stake these attacks are called “nothing at stake” problems). We distinguish two types of grinding attacks.

digging refers to grinding attacks where the adversary tries to extend one particular block in many different ways.[3]

double dipping refers to grinding attacks where the adversary tries to extend many different blocks, not just the one(s) specified by the protocol.

We discuss digging and double dipping, and how they are handled in Chia, in more detail in §1.4 and §1.5 below.

Short and long-range attacks. Grinding attacks are not the only security problems that come up once we replace proofs of work with proofs of space (or stake) in a Bitcoin like blockchain. Another issue is *long-range attacks*, which refer to attacks where an adversary tries to generate a chain that forked from the chain the honest farmers saw a long time ago. We will discuss long range attacks and how Chia avoids these using verifiable delay functions (on top of PoSpace) in §1.7.

1.3 Replacing PoW with PoSpace

In this section we discuss what it means to replace PoW with PoSpace[4] in a blockchain, which (even ignoring security issues) is not obvious as PoW and

[3] In Bitcoin the honest mining strategy is basically digging: miners hash different nonces until a lucky miner finds a nonce whose hash is below the difficulty. In Chia a farmer should just get a single shot at extending a block, thus one can think of digging as “illegal mining”.

[4] or proofs of stake, or any other proof system where proofs can be efficiently computed and where the proof systems is a one round public-coin challenge response protocol.

PoSpace are syntactically different: In Bitcoin the miners race to find a PoW, the miner who finds it announces a new block to the network, the block gets attached, the miner gets its reward, and the miners switch to extending this new block. Using a proof system like PoSpace, *every* miner can immediately compute a PoSpace, so we somehow need to specify who "wins" and when to continue with the next block.

The PoSpace-based Spacemint [PPK$^+$15] blockchain assigns a quality to each PoSpace (this quality is simply the hash of the proof), and the protocol specifies that the PoSpace of best quality announced should be considered as the extension of the chain. In Chia this basic idea is developed further. We also assign a quality to each PoSpace, but to *finalize* the block one must augment this PoSpace with the output of a verifiable delay function. The time parameter for the VDF – which specifies how many sequential computational steps are required to compute the output – is linear in the quality of the PoSpace, which means the PoSpace with best quality (best meaning lowest value) can be finalized first. Alternating PoSpace with VDFs like this gives the Chia blockchain a farming dynamic similar to mining in Bitcoin. In particular, we don't need any synchronisation like Spacemint or PoStake-based blockchains like Ourborors.

1.4 Digging attacks

As already outlined, when a Bitcoin miner wants to extend a block β, they prepare the payload data and must then find a PoW for challenge $c = \mathsf{H}(\beta, \mathsf{data})$ and difficulty D, i.e., a nonce ν such that $0.\mathsf{H}(c, \nu) < 1/D$. As the miner chooses data, they can actually compute many different challenges $c_1, c_2, \ldots$ (e.g. by taking various subsets of the available transactions or slightly changing the timestamp). In principle, a Bitcoin miner could now choose to search for a PoW for two (or more) challenges simultaneously, but that would require splitting the resource (hardware and electricity) amongst those challenges, and thus they gain no advantage by doing so.

In sharp contrast to this, if we naïvely used a proof system like PoSpace in a Bitcoin-like design, a malicious miner could generate proofs for many different challenges $c_1, c_2, \ldots$, and then cherry pick the one that is most promising (say, the best one as outlined above). By trying out X challenges, the miner would increase his chances of winning (as compared to honest mining) by a factor of X. Thus in some sense we're back to a PoW-based scheme as rational miners would not follow the honest mining rule, but race to compute as many PoSpace as possible. We refer to such grinding attacks – where an adversary tries to extend a block in many different ways – as *digging*.

In §1.4.1 to §1.4.3 we outline how digging attacks are prevented in Chia.

1.4.1 Splitting the chain

To prevent digging by grinding through different values for data as outlined above, Chia adapts an idea from Spacemint. The chain is composed of two chains as illustrated in Figure 1; one chain is called the *trunk* and contains just

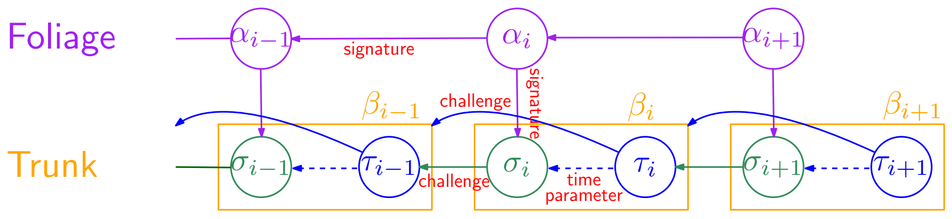


Figure 1: Illustration of the Chia blockchain that will be explained in §3. A block $\beta = (\sigma, \tau)$ in the (ungrindable) trunk chain contains a proof of space σ followed by a VDF output τ. A block α in the foliage contains the payload (transactions, timestamp), a signature of the previous block in foliage (to chain these blocks), and a signature of the proof of space (to bind the foliage to the trunk). This figure just illustrates the three deepest blocks of a chain, Figure 5 on page 20 illustrates a "richer" view that contains forks and non-finalized blocks.

the proofs (PoSpace and VDF outputs), the other chain is called the *foliage* and contains the payload data (and some signatures to bind the foliage to the trunk). All the challenges (for PoSpace and VDF) come from previous values in the trunk, and thus are not susceptible to grinding attacks where one tries out various values of data in the foliage (if we take difficulty resets into account, that's not quite true, as we'll discuss in §1.4.3).

1.4.2 Unique primitives

To prevent digging it is also crucial that the cryptographic primitives used in the trunk are *unique*. In particular, Chia uses a unique signature scheme, where uniqueness means that for every public-key/message pair there exits exactly one valid signature. The PoSpace we use are unique and a verifiable delay function is unique by definition.[5]

1.4.3 Careful with difficulty resets

Even after outsourcing the payload data to the foliage and make sure all primitives in the trunk are unique, it turns out that there is still a possible digging attack if the difficulty reset mechanism is directly adapted from Bitcoin. We discuss this in §5, but in a nutshell, the attack is possible because the difficulty parameter depends on the timestamps, which are (as part of data) in the foliage and thus can be ground. As a VDF output depends on the difficulty parameter, the VDF outputs can be indirectly ground via this connection. For this attack to work, the difficulty reset must apply to blocks immediately following the block whose timestamp was used to recompute the difficulty parameter (as is the case

[5]The definition of a VDF basically matches the definition of a *unique* proof of sequential work; the reason we cannot use simple proofs of sequential work like [CP18] in Chia is precisely because they're not uinque.

in Bitcoin). This can be easily prevented by specifying that the difficulty reset only applies after sufficiently many blocks have passed.

1.5 Double dipping attacks

In the previous section we outlined how Chia prevents digging attacks. We now discuss the other type of grinding attacks called double dipping. Recall that digging refers to attacks where an adversary tries to extend a particular block in many different ways, whereas double dipping refers to attacks where they try to extend different blocks, not just the one(s) specified by the protocol.

1.5.1 Punishment is not a solution

Penalties were suggested[6] as a means to disincentivize double dipping in 2014. Spacemint [PPK$^+$15] also specified a form of penalties[7]. Unfortunately, penalties don't really solve the issue. In particular, penalties can only be a deterrent against attacks where the potential gain is less than the potential penalty, so they might disincentivize things like selfish mining, but not double spending. Penalties are also no deterrent against long range attacks (discussed in §1.7) where an adversary tries to generate a long chain in private, simply because then there is no one around to trigger the penalty.

1.5.2 Smoothing out grinding advantage

One idea to countering double dipping and also digging attacks that was introduced in Spacemint (and later used e.g. in Ourborors Praos) is to deviate from a blockchain-like design by using the same challenge for several consecutive blocks. The reason this makes grinding attacks less attractive is that it is unlikely that a challenge will be good for many consecutive blocks. To get back to the cherry picking analogy: you can choose the best cherries going through them one by one, but if you must pick one entire bucket out of many buckets full of cherries, then the quality of the cherries in the best bucket will only be slightly better than the average.

1.5.3 How Chia adresses double dipping

The specification of the Chia blockchain has no built-in mechanism to prevent double dipping at all, instead we first quantify the advantage one gets from double dipping. Then we show how to decrease this advantage by changing the farming rules and having the honest farmers extend more than just one block at every depth.

We explain the result in more detail in §1.6 below, but in a nutshell, we show that the basic design is secure (in the sense of having non-zero chain quality) as

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

[7] In Spacemint, whenever a space miner publishes two blocks for the same block position but extending different blocks, this pair of blocks can be used (within a special type of transactions) to steal half the block reward of the cheating miner while the other half is destroyed.

long as at least 73.1% of the space[8] is controlled by honest farmers. As proofs of work are not susceptible to double dipping, Bitcoin only requires the honest miners to control $> 50\%$ of the hashing power to be secure. To narrow this gap, we also let honest farmers double dip to some extent: a parameter $\kappa \in \mathbb{Z}^+$ specifies that the honest farmers should compute a PoSpace for the first κ (not just the first) blocks at every depth. Setting $\kappa = 3$ seems like a good choice; this doesn't increase the work for farmers by too much, and now the honest farmers just need to control $\approx 61.5\%$. Figure 2 shows how this fraction drops as κ increases.

Remark 1 (Social convention of choosing κ)**.** *Let us stress that the value of κ is not part of the specification of* Chia*. Rather, it is a social convention by the honest farmers and thus can easily be adapted the future, in particular, this will not need a fork. One might chose to lower κ to speed up consensus, or increase it to improve security (this not only means raising the fraction of space required for double spending, but also e.g. making selfish mining less profitable).*

1.6 Chain Quality of Chia.

We provide an initial security analysis for Chia in an idealized setting so we can focus on the core problems. Concretely, we assume that

i. we have no network delays,

ii. an adversary can compute the VDF no faster than the honest parties (but they can compute an unlimited amount of VDF outputs in parallel),

iii. the proofs of space are ideal (i.e., allow for absolutely no time-memory trade-offs) and

iv. we have no difficulty resets.

1.6.1 Chain quality

Chain quality for $\kappa = 1$. In this idealized setting we prove for the most basic $\kappa = 1$ case – where the honest farmers compute a PoSpace only for the first block they see at each depth – the following

> (Corollary of Lemma 2) An adversary who controls less than a $\frac{1}{1+e} \approx 0.269$ fraction of the total space cannot (in private) grow a chain faster than the honest parties.

As illustrated in Figure 1, in Chia the challenge to compute the PoSpace σ_i is derived from the last VDF output τ_{i-1}. This challenge is not simply the hash of τ_{i-1}, but the hash of a signature of τ_{i-1}. This means a potential adversary will not be able to predict the challenge for a farmer, which in turn allows us to prove the following

[8]This number is $1 - (1/e)(1 + 1/e) \approx 0.731$ where $e \approx 2.718$ is Euler's number.

> (Informal statement of the "no slowdown" Lemma 4) An adversary who cannot break the security of the signature scheme (but otherwise can be unbounded, in particular, has unlimited space and even can compute VDF outputs in zero time) cannot slow down the rate at which the honest farmers grow the chain.

As a corollary of Lemmas 2 and 4 we get the following statement

> As long as the honest farmers control at least a $1 - \frac{1}{1+e} \approx 0.731$ fraction of the total space, the blockchain is secure in the sense that a non-zero fraction of the blocks in the chain were provided by honest farmers (chain quality property), and a block that is sufficiently deep in the chain will almost certainly remain there forever (common prefix property).

It is clear that as the fraction of the space controlled by the honest farmers increases beyond the minimal 0.731 fraction, the chain quality – i.e., the fraction of blocks in the chain contributed by honest farmers – will also increase, but at this point we can't prove any non-trivial quantitive bounds for this.

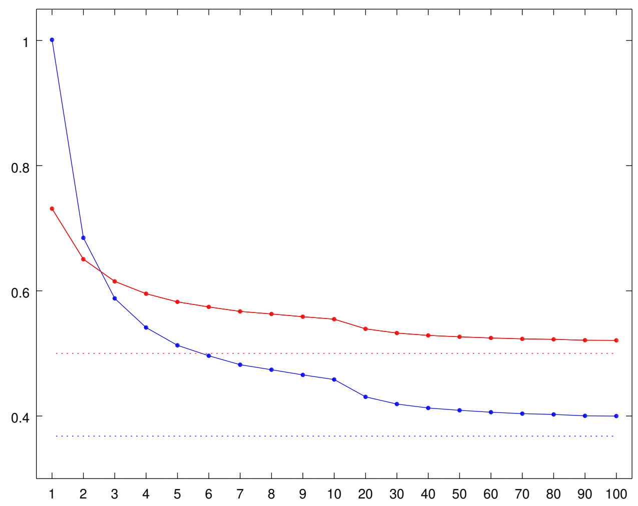
Chain quality for $\kappa > 1$. The reason for the gap between the ≈ 0.731 fraction (of space) we need to be under honest control in Chia and the 0.5 fraction (of computing power) in Bitcoin is due to double dipping attacks. If we specify that honest farmers must extend *every* block they see, then we'd also only require a 0.5 fraction, as the most extreme double dipping attack would then coincide with the honest farming strategy. Of course that would be completely impractical.

We will consider settings in between the two extreme cases where a parameter $\kappa \in \mathbb{Z}^+$ specifies that the honest farmers should compute a PoSpace for the first κ blocks they see at every depth. At the same time, every time lord should work towards finalizing κ blocks at every depth (so the extreme cases above correspond to $\kappa = 1$ and $\kappa = \infty$).

Setting a value for κ is a trade-off: higher values of κ give better security guarantees, but increase the work required of the farmers as they must compute κ proofs for every depth. To set a value for κ in Chia it helps to know how fast the fraction of space required to be controlled by honest farmers approaches 0.5 as we increase κ, and we denote these values by ι_κ. We can only analytically compute this fraction for $\kappa = 1$, where it is $\iota_1 = 1-(1/e)(1+1/e) \approx 0.731$, and it follows from the definition that ι_∞ is 0.5. To determine the other values we ran simulations (which will be discussed in §6.2). As one would expect the ι_i converge very quickly towards 0.5 as illustrated in Figure 2. Chia will use $\kappa = 3$, which corresponds to $\iota_3 \approx 0.615$.

1.6.2 The underlying probabilistic experiment

The technical statement of Lemma 2 considers the following simple probabilistic experiment. For positive integers h, ℓ, consider an h-ary tree of depth ℓ. Assign each edge of the tree a weight sampled uniformly iid in $[0, 1]$. We then consider the following random variables (also illustrated in Figure 4).



κ :	1	2	3	4	5	6	7	8	9	10
θ_κ :	1.00107	0.68468	0.58789	0.54136	0.51296	0.49626	0.48197	0.47395	0.46567	0.45830
ι_κ :	0.73127	0.65049	0.61510	0.59540	0.58235	0.57428	0.56712	0.56300	0.55866	0.55472

κ :	10	20	30	40	50	60	70	80	90	100
θ_κ :	0.45830	0.43062	0.41910	0.41282	0.40913	0.40607	0.40377	0.40252	0.40030	0.39992
ι_κ :	0.55472	0.53929	0.53254	0.52878	0.52655	0.52468	0.52326	0.52249	0.52110	0.52087

Figure 2: The red values shows (simulated values for) $\iota_1 \ldots \iota_{100}$, the necessary fraction of space that honest farmers must hold to outpace a malicious farmer who has an infinite number of VDF servers. It approaches 0.5 (red dotted line) as κ (on x-axis, note the step size increases from 1 to 10 at $\kappa = 10$) increases. The blue values show (simulated values for) $\theta_1 \ldots \theta_{100}$, where $\theta_i = \lim_{\ell\to\infty, h\to\infty} \mathrm{E}[C^\ell_{\kappa,h}]/\mathrm{E}[C^\ell_{1,h}]$ specifies what speedup can be achieved by extending the κ (as opposed to just one) blocks at every depth; it approaches $1/e$ as $\kappa \to \infty$. To simulate θ_i we sampled $C^\ell_{\kappa,h}$ (cf. §1.6.2) for $h = 999, \ell = 100000$ and approximated $\theta_\kappa \approx C^\ell_{\kappa,h}/\mathrm{E}[C^\ell_{1,h}]$. The ι_κ were computed from the θ_κ as $\iota_\kappa = 1 - \frac{1}{1+e\cdot\theta_\kappa}$.

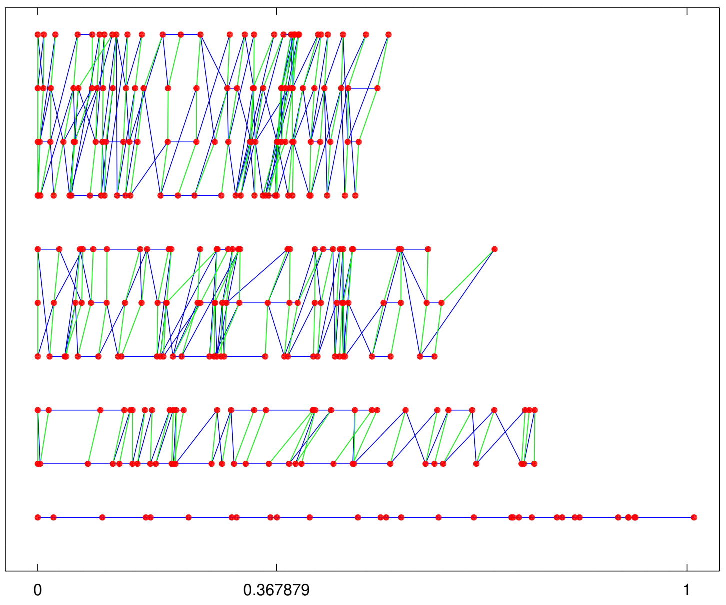


Figure 3: Visualization of chain growths for $\kappa = 1$ (bottom) to $\kappa = 4$ (top), simulated with $h = 29, \ell = 30$. x-axis is time, the green line connects the κ blocks (red dots) at the same level. The chain (or for $\kappa > 1$ tree) is in blue.

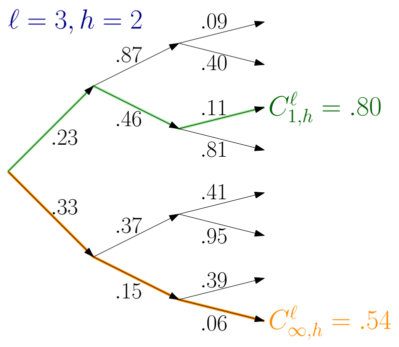


Figure 4: Illustration of the probabilistic experiment outlined in §1.6.2. The globally shortest path has weight 0.54; the path we get by greedily following the best edge at every node has weight 0.80.

- (greedy path) Let $C^{\ell}_{1,h}$ be the total edge weight of the path we get by starting at the root, following the edge with lowest weight at every node, and ending at a leaf.
- (globally shortest path) Let $C^{\ell}_{\infty,h}$ be the minimum of the total edge weight of all h^{ℓ} paths from the root to the leaves.

$C^{\ell}_{1,h}$ basically captures the time honest miners with total space h need to grow a path of length ℓ, whereas $C^{\ell}_{\infty,h}$ captures the time an adversary who tries to extend all possible paths needs (i.e., an adversary launching the most extreme double dipping attack). It follows from simple order statistics (cf. Lemma 1) that

$$\mathrm{E}[C^{\ell}_{1,h}] = \frac{\ell}{1+h}$$

and Lemma 2 states that

$$\mathrm{E}[C^{\ell}_{1,h}] \leq \mathrm{E}[C^{\ell}_{\infty,h}] \cdot e \ . \tag{1}$$

We derive eq.(1) analytically, and simulations indicate that it is tight as h, ℓ go to infinity.

We also define random variables $C^{\ell}_{\kappa,h}$ for any $\kappa \in \mathbb{Z}^{+}$, where $C^{\ell}_{\kappa,h}$ captures the time h honest farmers need to grow a path of length ℓ assuming they try to extend the first κ blocks they see at every depth.

We define θ_κ as the factor by which one can speed up chain growth by extending κ chains instead just one chain

$$\theta_\kappa = \lim_{\ell\to\infty, h\to\infty} \mathrm{E}[C^{\ell}_{\kappa,h}]/\mathrm{E}[C^{\ell}_{1,h}] = \frac{1+h}{\ell} \cdot \lim_{\ell\to\infty, h\to\infty} \mathrm{E}[C^{\ell}_{\kappa,h}]$$

Although we define this value in the limit for large ℓ and h, this value converges very fast so we can get good approximations by sampling $C^{\ell}_{\kappa,h}$ for fairly small finite ℓ, h. We will show that the fraction ι_κ discussed in the previous section can be computed from θ_κ as

$$\iota_\kappa = 1 - \frac{1}{1 + e \cdot \theta_\kappa} \ .$$

θ_κ and ι_κ are illustrated in Figure 2.

1.7 Long range attacks

Most attacks on blockchains can be roughly classified as *short range* or *long range* attacks. Short range attacks refer to attacks which only involve the last few blocks, whereas long range attacks are attacks where an adversary tries to generate a chain that forked (from the chain seen by the honest parties) many blocks in the past.

Short range attacks against Bitcoin include selfish mining [ES14] – where the adversary strategically withholds blocks to get an unfair share of the block rewards – or double spending – where the adversary tries to generate a chain in private that is sufficiently long to double spend. As Bitcoin specifies that a block at depth 6 can be considered confirmed, even an adversary with significantly less than 50% of the hashing power has a good chance is succeeding in this attack (which requires growing a chain of length 6 faster than the honest miners).

Long range attacks are not an issue for Bitcoin,[9] but they are a major problem for blockchains based on efficient proof systems like proofs of space or proofs of stake.

1.7.1 Long range attacks on PoStake using old money

A particularly annoying longe range attack *against proof of stake* blockchains uses the observation that it is sufficient to hold a large stake (i.e., the secret keys allowing to spend coins) that was valid at some point *in the past* (but given the current chain can be worthless). Such old keys can be used to fork the chain at the point where the coins where valid, and then one can bootstrap a chain that looks valid in the present. An adversary could potentially acquire such a stake at low cost, e.g. by buying a large fraction of the coins and then immediately selling them. There doesn't seem to be a solution to this attack unless one assumes trusted parties or requires parties to come online sufficiently frequently so one can acquire check-points at sufficiently short intervals.

1.7.2 Long range attack on PoSpace using resource variation

Recall that in Bitcoin the "longest chain" does not really mean the chain with most blocks, but rather it is the "heaviest chain", the one that required most hashing power to generate (this can be computed by summing up the difficulty parameter over each block). We can adapt this rule to a PoSpace-based chain by saying that the "heaviest chain" is the one whose sum of qualities of the PoSpace (the quality of a block reflects the amount of space that was used to generate the block) over all blocks is maximal. This is delicate for various reasons, but the most serious is that now it is possible to generate a chain that is heavier than the honestly generated chain *using space that is only the average of the space that was used for farming the honest chain*, which can be much lower than the current amount of space dedicated towards mining (e.g. because mining became much more lucrative over time or disk space became much cheaper). Spacemint [PPK+15] addresses this problem in an ad-hoc way by putting more weight on more recent blocks when calculating the weight of a chain.

1.7.3 Solving long range attacks using VDFs

The two long range attacks outlined above make crucial use of the property that PoStake and PoSpace can be computed quickly, so one can bootstrap a

[9] A 51% adversary can generate an arbitrary long fork, but against this adversary everything is lost anyways.

heavy chain in very short time once one has sufficient (old) stake or space. For PoW-based blockchains like Bitcoin this is not possible: one can generate a chain that is heavier than the current Bitcoin blockchain using hashing power that is slightly higher than the average hashing power that was dedicated since genesis, but this will (in 2019) require a decade, at which point it is useless as the honest chain will also have advanced by a decade.

Chia achieves this desirable property (i.e., that a chain cannot be bootstrapped quickly) without relying on wasting massive computing power as required by PoW. Instead, Chia alternates PoSpace with VDFs. This way, the main resource is space, while the VDFs guarantee that one cannot bootstrap a chain much faster than honestly farming it.

2 Building Blocks: PoSpace, VDFs and Signatures

In this section we sketch the main building blocks used in the Chia blockchain: unique digital signatures, proofs of space [DFKP15] and verifiable delay functions [BBBF18, Wes18, Pie18, BBF18]. The definitions are not fully general, but instead tailored to the particular constructions of PoSpace from [AAC+17] and the VDFs [Wes18, Pie18, BBF18] based on sequential squaring.

2.1 (Unique) Digital Signatures

A digital signature scheme is specified by three algorithms; a (probabilistic) key-generation algorithm Sig.keygen, a signing algorithm $\mu \leftarrow \mathsf{Sig.sign}(sk, m)$ and a verification algorithm Sig.verify. We assume the standard security notion (unforgeability under chosen message attacks) and perfect completeness, that is, a correctly generated signature will always verify:

$$\begin{array}{rl} \forall m, & \Pr[\mathsf{Sig.verify}(pk, m, \mu) = \mathsf{accept}] = 1 \\ \text{where} & (pk, sk) \leftarrow \mathsf{Sig.keygen} \ ; \ \mu \leftarrow \mathsf{Sig.sign}(sk, m) \ . \end{array}$$

Chia uses signatures in the foliage (to chain foliage blocks and to bind them to the trunk) and also in the trunk (so only the farmer can compute the challenge). To avoid grinding attacks, the signatures used in the trunk must be unique, that is for every pk (this includes maliciously generated public keys) and message m there can be at most one accepting signature

$$\forall pk, m, \ (\mathsf{Sig.verify}(pk, m, \mu) = \mathsf{accept}) \wedge (\mathsf{Sig.verify}(pk, m, \mu') = \mathsf{accept}) \Rightarrow (\mu = \mu') \ .$$

2.2 (Unique) Proofs Of Space

2.2.1 Algorithms for PoSpace

A proof of space is specified by the four algorithms given below

$\mathsf{PoSpace.init}$ on input a space parameter $N \in \mathcal{N}$ (where $\mathcal{N} \subset \mathbb{Z}^+$ is some set of valid parameters) and a unique identifier pk (we use pk to denote the identifier as in Chia it will be the public key of a signature scheme) outputs[10]

$$S = (\ S.\Lambda\ ,\ S.N = N\ ,\ S.pk = pk) \leftarrow \mathsf{PoSpace.init}(N, pk)$$

Here $S.\Lambda$ is the large file of size $|S.\Lambda| \approx N$ the prover needs to store. We also keep N, pk as part of S as it will be convenient.

$\mathsf{PoSpace.prove}$ on input S and a challenge $c \in \{0,1\}^w$ outputs a proof

$$\sigma = (\ \sigma.\pi\ , \sigma\ =\ \sigma.N = S.N\ ,\ \sigma.pk = S.pk\ ,\ \sigma.c = c\ )\ \leftarrow \mathsf{PoSpace.prove}(S, c)$$

Here $\sigma.\pi$ is the actual proof, the other entries in σ are just convenient to keep around.

$\mathsf{PoSpace.verify}$ on input a proof σ outputs accept or reject

$$\mathsf{PoSpace.verify}(\sigma) \in \{\mathsf{reject}, \mathsf{accept}\}\ .$$

We assume perfect completeness

$$\forall N \in \mathcal{N}, c \in \{0,1\}^w,\ \Pr[\mathsf{PoSpace.verify}(\sigma) = \mathsf{accept}] = 1 \text{ where}$$
$$S \leftarrow \mathsf{PoSpace.init}(N, pk) \text{ and } \sigma \leftarrow \mathsf{PoSpace.prove}(S, c)$$

2.2.2 Security of PoSpace

We will not give the formal security definition for PoSpace here, but informally it states that an adversary who stores a file of size significantly less than N bits should not be able to produce a valid proof for a random challenge unless he invests a significant amount of computation (ideally close to what it costs to run the full initialization $\mathsf{PoSpace.init}(N, pk)$). Moreover it must be impossible to amortize space, that is, initalizing space for $m > 1$ different identities must require m times as much space.

To prevent grinding attacks, we need our PoSpace to be unique as defined below.

[10]The first constructions of PoSpace from [DFKP15] were based on depth-robust graphs. The initialization phase in these PoSpace was not just a function as it is here, but an interactive protocol. The definition we give here captures the [AAC+17] PoSpace (which was developed for Chia) where the initialization phase is non-interactive, this makes its use in a blockchain design much simpler. The Spacemint [PPK+15] proposal is using graph-based PoSpace and because of that must bootstrap the blockchain itself to make initialization non-interactive: farmers must post a commitment to their space to the blockchain via a special type of transaction before it can be used for farming. Without this, Spacemint would succumb to grinding attacks (on the message send to the verifier during the initialization phase).

2.2.3 Unique PoSpace

A PoSpace is unique if for any identity pk and any challenge c there is exactly one proof, i.e.,

$$\forall N, pk, c,$$
$$|\{\sigma \ : \ (\mathsf{PoSpace.verify}(\sigma) = \mathsf{accept}) \wedge ((\sigma.N, \sigma.pk, \sigma.c) = (N, pk, c))\}| = 1$$

We call a PoSpace *weakly* unique if the *expected* number of proofs is close to 1, i.e.,

$$\forall N, pk, c,$$
$$\mathrm{E}_{c \leftarrow \{0,1\}^w} [|\{\sigma : (\mathsf{PoSpace.verify}(\sigma) = \mathsf{accept}\}) \wedge ((\sigma.N, \sigma.pk, \sigma.c) = (N, pk, c))\,|]$$
$$\approx 1$$

For weakly unique PoSpace we assume that whenever there is more than one proof for a given challenge which passes verification, $\mathsf{PoSpace.prove}(S, c)$ outputs all of them.

The [AAC+17] PoSpace used in Chia is only *weakly unique*. To be able to focus on the main challenges, we will nonetheless assume a *unique* PoSpace when analyzing Chia but our analysis can be extended without major difficulties to handle weakly unique PoSpace, things just get a bit more messy.

2.2.4 The [AAC+17] PoSpace

We give a very high level outline of the PoSpace from [AAC+17]. The space parameter is given implicitly by a value $\ell \in \mathbb{Z}^+$, the actual space required is approximately $N \approx \ell \cdot 2 \cdot 2^\ell$ bits (e.g. for $\ell = 40$ that's 10 terabytes). Let $L := \{0,1\}^\ell$ denote the set of ℓ bit strings. Below we denote with $X_{|\ell}$ the ℓ bit prefix of a string X.

The identity $id := pk$ together with a hash function H defines two functions $f : L \to L, g : L \times L \to L$ as

$$f(x) = \mathsf{H}(id, x)_{|\ell} \quad \text{and} \quad g(x, x') = \mathsf{H}(id, x, x')_{|\ell} \ .$$

Note that if we model H as a random function, then f, g are also random functions. On a challenge $y \in L$ the prover must answer with a tuple

$$id, (x, x') \qquad \text{where} \qquad x \neq x', f(x) = f(x'), g(x, x') = y$$

if it exists. In this construction, for roughly a $(1 - 1/e) \approx 0.632$ fraction of the challenges $y \in L$ there will be at least one proof, and the expected number of proofs is 1 (so it is a weakly unique PoSpace).

The prover will generate and store two tables so they can efficiently generate proofs. They first compute and store a table with the values $(x, f(x))$ sorted by the 2nd entry. With this table, the prover can now efficiently enumerate all tuples (x, x') where $x \neq x'$ and $f(x) = f(x')$ to generate a table containing all triples $(x, x', y = g(x, x'))$; the expected number of such triples is $|L| = 2^\ell$. This table is then sorted by the thrid value. Now given a challenge y one can efficiently look up proofs in the second table as it is sorted by the y values. Storing the second table requires $\approx 3|L|\log(|L|) = 2^{\ell+1}\ell$ bits, and this can be brought down to $\approx 2|L|\log(|L|)$ bits by encoding it in a more clever way.

Chia is based on this PoSpace, but to further minimize the effect of time/space trade-offs (where a malicious farmer tries to save on space at the cost of doing more computations), a nested version of this construction is used. We omit the details in this writeup.

2.3 Verifiable Delay Functions

A VDF is specified by the two algorithms given below.

$\mathsf{VDF.solve}$ on input a challenge $c \in \{0,1\}^w$ and time parameter $t \in \mathbb{Z}^+$ outputs a proof

$$\tau = (\ \tau.y\ ,\ \tau.\pi\ ,\ \tau.c = c\ ,\ \tau.t = t\ ) \leftarrow \mathsf{VDF.solve}(c, t)$$

and runs in (not much more than) t sequential steps (what a step is depends on the particular VDF). Here $\tau.y$ is the output and $\tau.\pi$ is a proof that $\tau.y$ has been correctly computed. For convenience we also keep (c, t) as part of τ.

$\mathsf{VDF.verify}$ on input τ outputs accept or reject.

$$\mathsf{VDF.verify}(\tau) \in \{\mathsf{reject}, \mathsf{accept}\}$$

Verifying must be possible in $\ll t$ steps, for existing VDFs verification just takes $\log(t)$ [Pie18] or even constant [Wes18] time.

We have perfect completeness

$$\forall t, c\ :\ \mathsf{VDF.verify}(\mathsf{VDF.solve}(c, t)) = \mathsf{accept}$$

The two security properties we require are

uniqueness: It is hard to come up with any statement and an accepting proof for a wrong output. More precisely, it is computationally difficult to find any τ' where for $\tau \leftarrow \mathsf{VDF.solve}(\tau'.c, \tau'.t)$ we have

$$\mathsf{VDF.verify}(\tau') = \mathsf{accept} \quad \text{and} \quad \tau.y \neq \tau'.y\ .$$

Note that we only need $\tau.y$ (but not $\tau.\pi$) to be unique, i.e., the proof $\tau.\pi$ showing that $\tau.y$ is the correct value can be malleable. This seems sufficient for all applications of VDFs, but let us mention that in the [Pie18, Wes18] VDFs discussed below also $\tau.\pi$ is unique.

sequentialitiy: Informally, sequentiality states that for any t, an adversary $\mathcal{A}$ who makes less than t sequential steps will notfind an accepting proof on a random challenge. I.e., for some tiny ϵ

$$\Pr[\mathsf{VDF.verify}(\tau) = \mathsf{accept} \ \wedge \ \tau.c = c \ \wedge \ \tau.t = t \ : \ c \overset{rand}{\leftarrow} \{0,1\}^w, \tau \leftarrow \mathcal{A}(c,t)] \leq \epsilon$$

Let us stress that $\mathcal{A}$ is only bounded by the number of *sequential* steps, but they can use high parallelism. Thus the VDF output cannot be computed faster by adding parallelism beyond what can be used to speed up a single step of the VDF computation.

2.3.1 The [Wes18, Pie18] VDFs

The VDFs proposed in [Wes18, Pie18] (see [BBF18] for an overview of those constructions) are both based on squaring in a group of unknown order, for concreteness let the group be $\mathbb{Z}_N^*$ where $N = pq$ is the product of two large primes p, q. On input $(c \in \mathbb{Z}_N^*, t)$ the ouput of $\mathsf{VDF.solve}(c,t)$ is (y, π) with $y = c^{2^t} \bmod N$. This y can be computed by squaring c sequentially t times $c \to c^2 \to c^{2^2} \to \cdots \to c^{2^t}$, and it is conjectured that there is no shortcut to this computation if one doesn't know the factorization of N.

The VDFs from [Wes18, Pie18] differ in how the proof π that certifies that $y = c^{2^t} \bmod N$ is defined. The proof in [Wes18] is shorter (1 vs. $\log(T)$ elements), but soundness of the proof requires an additional assumption (that taking random roots is hard).

If one uses an RSA group as above, a trusted setup or a multiparty computation is needed to sample the modulus N in a way that nobody learns its factorization. [Wes18, BBF18] suggest using the class group of an imaginary quadratic field as the underlying group of unknown order. These groups can be obliviously sampled – this means given random bits one can sample a group without learning its order – and thus there is no need for a trusted setup.

3 The Blockchain

3.1 Trunk and Foliage

In this section we specify the blockchain format using the building blocks we introduced in §2. As outlined in the introduction, Chia uses two separate chains called the *trunk* and the *foliage* to prevent grinding attacks. Blocks in the trunk $\beta_0, \beta_1, \ldots$ contain the proofs (the PoSpace and the VDF outputs required to finalize the block). The foliage contains a block α_i for every block β_i in the trunk. This block contains the payload (transactions and timestamps) of the blockchain and a signature to bind the foliage to the trunk and at the same time chain the blocks in the foliage.

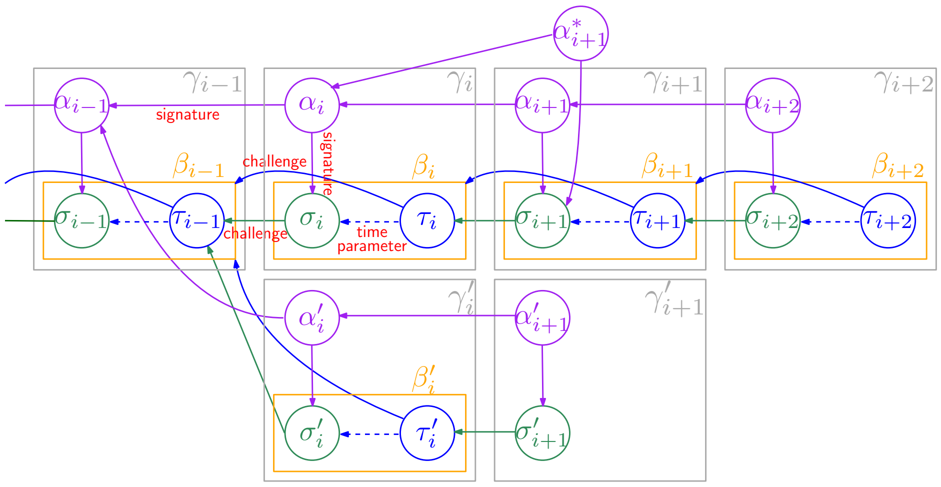


Figure 5: Illustration of a possible view of the blockchain (as stored in $\mathcal{C}$ in §4.1). The longest path has $i+2$ blocks and its head is the finalized block γ_{i+2}. There is a fork at block γ_{i-1}; the block γ'_{i+1} is non-finalized. The (presumably malicious) farmer who generated the PoSpace σ_{i+1} signed off two distinct foliage blocks α_{i+1} and α^*_{i+1}. The farmer of σ_{i+2} extended the foliage block α_{i+1}.

3.2 Block Format

A block $\gamma_i = (\beta_i, \alpha_i)$ contains a block from the trunk

$$\beta_i = (i, \sigma_i, \tau_i)$$

where $\sigma_i = (\sigma_i.\pi, \sigma_i.N, \sigma_i.pk, \sigma_i.c, \sigma_i.\mu)$ is a proof of space with an extra entry $\sigma_i.\mu$ (explained below) and $\tau_i = (\tau_i.y, \tau_i.\pi, \tau.c, \tau.t)$ is a VDF output. Moreover, γ_i must contain a block from the foliage

$$\alpha_i = (\phi_i, \mathsf{data}_i) \ .$$

We call a block γ_i as just described *finalized*, and if the VDF output τ_i is missing we say the block is *non-finalized*.

A (finalized or non-finalized) block $\gamma_i = (\beta_i, \alpha_i)$ *extends* a previous block $\gamma_{i-1} = (\beta_{i-1}, \alpha_{i-1})$ if the following conditions are satisfied (if the block is non-finalized point 2 below is omitted).

1. σ_i is a valid PoSpace

$$\mathsf{PoSpace.verify}(\sigma_i) = \mathsf{accept}$$

where the challenge is the hash $\sigma_i.c = \mathsf{H}(\sigma_i.\mu)$ of the unique signature $\sigma_i.\mu$ of the previous VDF output τ_{i-1}, so we also check (see also Remark 2 below)

$$\mathsf{Sig.verify}(\sigma_i.pk, \sigma_i.\mu, \tau_{i-1}) = \mathsf{accept} \quad \text{and} \quad \sigma_i.c = \mathsf{H}(\sigma_i.\mu) \ . \qquad (2)$$

2. τ_i is a valid VDF output
$$\mathsf{VDF.verify}(\tau_i) = \mathsf{accept}$$
where the challenge is the (hash of the) previous trunk block (see also Remark 3 below) and the time parameter is the hash of the previous PoSpace multiplied by the current difficulty parameter T.
$$\tau_i.c = \mathsf{H}(\beta_{i-1}) \quad \text{and} \quad \tau_i.t = 0.\mathsf{H}(\sigma_i) \cdot T \tag{3}$$
3. ϕ_i is a signature (under the pk used in the current PoSpace) of (the hash of) the previous block in the foliage, the proof of space in the trunk and the data.
$$\mathsf{Sig.verify}(\sigma_i.pk, \mathsf{H}(\alpha_{i-1}, \sigma_i, \mathsf{data}_i), \phi_i) \quad = \quad \mathsf{accept}$$
4. data_i is the actual data contained in the block. In this document we just discuss the consensus layer of Chia, and for this it is not relevant how this is specified exactly. But, like in Bitcion, it is essentially a list of transactions that must be consistent with the transactions $\mathsf{data}_0, \ldots, \mathsf{data}_{i-1}$ in the previous blocks.

Remark 2 (Why the PoSpace challenge is a signature)**.** *The reason the PoSpace challenge in eq.(2) is defined as the hash of a signature instead simply the hash of the VDF output, i.e., $\sigma_i.c = \mathsf{H}(\tau_{i-1})$, is so that only the farmer (who knows the secret key for $\sigma_i.pk$) can compute the challenge. This will be crucial later to prove the "no slowdown" Lemma 4.*

Remark 3 (Why the VDF challenge comes from the previous block)**.** *The reason the VDF challenge in eq.(3) is defined as the hash of the previous trunk block instead of simply using the previous PoSpace to derive the challenge, i.e., $\tau_i.c = \mathsf{H}(\sigma_i)$, is so a time lord who has finished computing τ_{i-1} can release it and immediately start computing τ_i. The farmers can now compute the PoSpace σ_i for this block, and if the time lord receives a σ_i before it made $\tau_i.t = 0.\mathsf{H}(\sigma_i) \cdot T$ sequential steps it can finalize this block and continue with τ_{i+1} right away. This way the honest parties will notget slowed down because there is a network delay. This is important as an adversary who has his VDF servers and storage physically close would not have this disadvantage. In rare cases where the best PoSpace has extremely good quality (and thus can be finalized in just a few seconds) there is a chance the time lord will not receive the PoSpace in time. To avoid this the time parameter in* Chia *will have an additive term to enforce a lower bound on the running time of the VDF so the network delay should never slow down the honest parties.*

4 Algorithms for Farmers and Time Lords

In this section we provide the pseudocode for the farmers, who dedicate space towards securing the chain, and the time lords, who compute the VDF outputs to finalize blocks.

4.1 Chain Management

Space farmers and time lords keep their local view on the chain stored in a data structure $\mathcal{C}$, Figure 5 illustrates the highest blocks in a possible view.

Chain.init takes no input and outputs a current view $\mathcal{C}$ of the chain as received from the network.

Chain.update is invoked whenever new blocks are received from the network or a proof is locally generated (this proof is a VDF output for time lords and a PoSpace for farmers). It expects as inputs either a finalized or non-finalized block γ (it is convenient to also allow an index/VDF output pair (i, τ) as input, in which case it checks if this input finalizes some known non-finalized block, and continues as if its input were this finalized block).

It updates the local view $\mathcal{C}$ and sometimes further disseminates these blocks – e.g. using a gossip protocol like Bitcoin – to make sure all honest farmers will ultimately get it.

The output are two (possibly both empty) sets (Γ_f, Γ_n), where Γ_f (Γ_n) contains all the valid, fresh and finalized (non-finalized) blocks (as defined below) that were just attached to $\mathcal{C}$. For every block in Γ_n we also add the hash of the trunk block before it as it will be required to compute the next VDF challenge.

We will not fully specify the pseudocode of the Chain.update functionalitiy as it is not particular to Chia. It will be convenient to define the following three predicates that a block (which is given as input to Chain.update) can have

(non-)finalized: (as already defined in §3.2) a block $\gamma = (\beta = (i, \sigma, \tau)), \alpha = (\phi, \mathsf{data}))$ is called **finalized**; if the VDF output τ is missing it is called **non-finalized**.

valid: a (finalized or non-finalized) block is valid if it *extends* (as defined in §3.2) a block already in $\mathcal{C}$.

fresh: a block is fresh if it is valid and not already in $\mathcal{C}$ (we also consider it fresh if it is finalized, but only its non-finalized version was already in $\mathcal{C}$).

Remark 4 (Multiple foliage blocks)**.** *An honest farmer will sign exactly one foliage block for each PoSpace they find (cf. line 13 in Algorithm 3 below), but a malicious farmer might sign and gossip more than one foliage block for the same PoSpace. We specify that* Chain.update *will usually consider the first foliage block it sees for any given PoSpace as the valid one, but one must deviate from this rule if the PoSpace gets finalized and extended with a new block, and this new block extends a different foliage block than the one we saw first. E.g. in the view in Figure 5 we'd consider α_{i+1} to be the correct foliage block even if we saw α^*_{i+1} first.*

4.2 Space Farmer Algorithms

To simplify the exposition, we assume that all parties who want to dedicate space dedicate the same amount of N bits, but this is not necessary, as we'll discuss in Remark 8. A party that wants to act as a space farmer first initializes their space S (and some other variables) running the SpaceFarmer.init algorithm below.

Algorithm 1 SpaceFarmer.init

```
1: **Global Parameters:** $N$
2: $\mathcal{C} \leftarrow$ Chain.init                          ▷ extract view from network
3: $(pk, sk) \leftarrow$ Sig.keygen              ▷ sample public/secret signature key pair
4: $S \leftarrow$ PoSpace.init$(N, pk)$        ▷ initialize space of size $N$ with identity $pk$
5: $S.sk := sk$                                        ▷ add $sk$ to the stored file
6: Initalize a vector pos_count to all 0                          ▷ see Remark 5
7: **Output:**
8: $S = (S.\Lambda, S.N, S.pk, S.sk)$, pos_count          ▷ State for SpaceFarmer.farm
9: $\mathcal{C}$                                                ▷ State for Chain.update
```

Remark 5 (Storing "infinite" vectors)**.** *The i'th entry of the vector* pos_count *stores how many proofs were generated at depth i. We think of* pos_count *as infinite, but it will always only have a tiny active window $j, \ldots, j+\delta$ such that $\forall i < j$,* pos_count$[j] = \kappa$*, and $\forall i > j$,* pos_count$[j] = 0$*, so it can be efficiently stored. The same holds for the* finalized *and* running *vectors used by time lords in the pseudocode below.*

After initializing, the space farmer runs the infinite loop SpaceFarmer.loop. In this loop, on every valid, fresh and finalized block that was received from the network, SpaceFarmer.farm is invoked, which decides whether to compute a proof of space to extend this block. The parts of the SpaceFarmer.farm pseudocode that only affect the foliage are shown in purple.

Algorithm 2 SpaceFarmer.loop

```
1: **loop**
2:     Wait for block(s) $\Gamma$ to be received from the network
3:     $(\Gamma_f, \Gamma_n) \leftarrow$ Chain.update$(\Gamma)$
4:     $\forall \gamma \in \Gamma_f$ : SpaceFarmer.farm$(\gamma)$                    ▷ Algorithm 3
5: **end loop**
```

Remark 6 (Taking difficulty resets into account)**.** *The* SpaceFarmer.farm *Algorithm 3 as well as the* TimeLord.finalize *Algorithm 6 below become more complicated if we take difficulty resets into account. Concretely, these algorithms might then have to extend/finalize a block at depth i even if they have already extended/seen κ blocks at this depth if this new block is the head of a chain that has a higher "cumulative work" than the κ chains we've seen so far at this*

Algorithm 3 SpaceFarmer.farm

```
1: Global Parameters: $\kappa$    ▷ # of proofs per slot ($\kappa = 3$ in Chia)
2: Input: $\gamma_i = (\beta_i = (i, \sigma_i, \tau_i), \alpha_i)$.    ▷ finalized, fresh & valid block at depth $i$
3: State: $S = (S.\Lambda, S.N, S.pk, S.sk)$, pos_count
4: if $\mathsf{pos\_count}(i+1) = \kappa$ then    ▷ if already generated $\kappa$ PoSpace
5:     return without output    ▷ at depth $i+1$ ignore this block
6: end if
7: $\mathsf{pos\_count}(i+1) \leftarrow \mathsf{pos\_count}(i+1) + 1$
8: $\mu \leftarrow \mathsf{Sig.sign}(S.sk, \tau_i)$    ▷ compute signature of last VDF output
9: $c \leftarrow \mathsf{H}(\mu)$    ▷ challenge is hash of this signature
10: $\sigma_{i+1} \leftarrow \mathsf{PoSpace.prove}(S, c)$    ▷ compute PoSpace
11: $\sigma_{i+1}.\mu := \mu$    ▷ keep signature as part of the proof
12: Generate $\mathsf{data}_{i+1}$    ▷ create payload for this block
13: $\phi_{i+1} \leftarrow \mathsf{Sig.sign}(S.sk, (\alpha_i, \sigma_{i+1}, \mathsf{data}_{i+1}))$    ▷ signature for foliage
14: $\mathsf{Chain.update}(i+1, \sigma_{i+1}, \alpha_{i+1} = (\phi_{i+1}, \mathsf{data}_{i+1}))$    ▷ Cf. Remark 7
```

depth. This could happen if, for example, we had a network split, the farmer landed in the part of the network controlling a smaller fraction of the space, and as the split is over we want them to switch back to the heavier chain generated in the other partition. We omit the details in this writeup.

Remark 7 (Keeping the network load small)**.** *If* SpaceFarmer.farm *generated a fresh (non-finalized) block, it will invoke* Chain.update *to update $\mathcal{C}$ and disseminate this fresh block. Disseminating every block created by a space farmer would generate substantial network load.*

To avoid this, we can specify Chain.update *so it does not disseminate blocks (finalized or not, locally generated or received from the network) which have basically no chance of ending up in the blockchain. This brings down the network load from quadratic (in the number of farmers) to linear (and the communication for each individual farmer from linear to constant).*

Remark 8 (Dealing with non-equal space)**.** *We assume that each space farmer dedicates exactly N bits of space, but in practice farmers will have vastly different amounts of space that they want to contribute. A simple solution to deal with this is by letting N be some arbitrary unit of space (1 bit, or 100GB) and now a farmer with $k \cdot N, k > 1$ bits of space can emulate k space farmers, each having N space. This solution incurs a computational cost during farming which is linear in the dedicated space.*[11] *A better solution to allow space farmers to initialize just one proof of space of size $N \cdot k$ (that is, replace line 4 in* SpaceFarmer.init *"$S \leftarrow \mathsf{PoSpace.init}(N)$" with "$S \leftarrow \mathsf{PoSpace.init}(N \cdot k)$").*

If the space farmer now generates a proof of space σ, the time to finalize the block is not just given by a value t that is uniform in $[0, T]$ (computed as $t := 0.\mathsf{H}(\mu) \cdot T$ in line 8 of Algorithm 6 below), but instead we let the farmer sample k random values $t_1 := 0.\mathsf{H}(\sigma, 1) \cdot T, \ldots, t_k := 0.\mathsf{H}(\sigma, k) \cdot T$ and they can

[11]But note that using Remark 7, the communication cost remains unchanged.

then use the smallest of these values. From the farmer's view, having k proofs and getting one random value for each is as good as having one proof that gives k random values, but the latter is more efficient, as in only requires to compute one – not k – proofs of space.

The cost of the above solution is still in some sense linear in k, as one has to evaluate H *k times to find the smallest* t_i*. In Spacemint [PPK+15] a solution is described whose asymptotic cost is really just ploylogarithmic. In a nutshell, one uses* $\mathsf{H}(\sigma)$ *as random coins to sample from a distribution which corresponds to the minimum of k iid values sampled uniformly from* $[0,1]$*. This way the farmer is efficient even if the granularity N of the space is just a bit, or even goes to* 0*.*

4.3 Time Lord Algorithms

Besides space farmers, sustaining the blockchain requires at least one time lord participating to finalize blocks. Recall that to finalize a block with PoSpace σ requires computing a VDF on challenge $\mathsf{H}(\sigma)$ running for sequential time $0.\mathsf{H}(\sigma)\cdot T$. To participate as a time lord, the party first runs the simple initialization below which retrieves the current blockchain from the network and initializes two vectors used to keep track of how many VDF outputs have already been computed and are currently being computed at every depth.

Algorithm 4 TimeLord.init

```
1: $\mathcal{C} \leftarrow$ Chain.init                                  ▷ extract view from network
2: Initalize vectors running and finalized where for every $i$ running[$i$] = 0 and
   finalized[$i$] is the number of finalized blocks in $\mathcal{C}$ at depth $i$.
3: Output:
4: finalized, running                  ▷ State for TimeLord.finalize/startVDF/restore
5: $\mathcal{C}$                                             ▷ State for Chain.update
```

After initializing, the time lord runs the infinite loop TimeLord.loop.

Algorithm 5 TimeLord.loop

```
1: loop
2:     Wait for block(s) $\Gamma$ to be received from the network
3:     $(\Gamma_f, \Gamma_n) \leftarrow$ Chain.update($\Gamma$)
4:     $\forall((i,\sigma),\alpha,c) \in \Gamma_n$ : TimeLord.finalize($i,\sigma,c$)        ▷ Algorithm 6
5:     $\forall((i,\sigma,\tau),\alpha) \in \Gamma_f$ : TimeLord.restore($i$)              ▷ Algorithm 8
6: end loop
```

In this loop, whenever a valid, fresh and *non-finalized* block is received, TimeLord.finalize is invoked. This algorithm then decides whether to start a thread to compute a VDF output to finalize this block. In addition, for every valid, fresh and *finalized* block received, TimeLord.restore is invoked, which adapts the local view to take this block into account.

The time lord's state contains two vectors, finalize and running, where at any timepoint, for any $i \in \mathbb{Z}$

finalized[i] $\in \{0, 1, \ldots, \kappa\}$ contains the number of finalized (either by the time lord itself, or received from the network) blocks at depth i (once this counter reaches κ it is no longer increased).

running[i] $\in \{0, 1, \ldots, \kappa\}$ contains the number of threads currently running that compute a VDF which will finalize a block at depth i.

In a nutshell, the goal of the time lord algorithms is to generate κ finalized blocks at every depth i as fast as possible. As we just want κ blocks, for any depth i at any timepoint it holds that

$$0 \leq \mathsf{finalized}[i] + \mathsf{running}[i] \leq \kappa$$

and moreover for every i, the running$[i]$ VDF threads are the ones which – given the view of the time lord – are the ones which can be finalized earliest. Concretely, this means that after receiving a *finalized* block from the network, the time lord might have abort the VDF thread scheduled to finish last because this thread would create the $(\kappa+1)$th block at this depth. After receiving a *non-finalized* block, the time lord might have to start finalizing this block, and in this case might also have to abort the VDF thread scheduled to finish last.

Algorithm 6 TimeLord.finalize

```
1:  Global Parameters: $T$, $\kappa$
2:  Input: $\beta_i = (i, \sigma_i)$ ▷ non-finalized, fresh & valid block for depth $i$ received
3:  $c$ $(= \mathsf{H}(\beta_{i-1}))$                       ▷ hash of previous trunk block
4:  State: finalized, running
5:  if finalize$[i] = \kappa$ then                ▷ already finalized $\kappa$ blocks for this slot
6:      return with no output
7:  end if
8:  $t := 0.\mathsf{H}(\sigma_i) \cdot T$        ▷ VDF time parameter required to finalize this block
9:  if finalize$[i]$ + running$[i] < \kappa$ then  ▷ less than $\kappa$ proofs finalized or running
10:     start thread TimeLord.startVDF$(i, c, t)$       ▷ to finish at time now $+ t$
11:     running$[i]$ := running$[i] + 1$
12: end if
13: if finalize$[i]$ + running$[i] = \kappa$ then   ▷ exactly $\kappa$ proofs finalized or running
14:     if the slowest VDF thread for slot $i$ will finish at time $> t +$ now then
15:         abort the thread of this VDF
16:         start thread TimeLord.startVDF$(i, c, t)$
17:     end if
18: end if
```

Remark 9 (Time lord parallelism). *If $\kappa = 1$, a time lord will never have more than one VDF thread running at once. If $\kappa > 1$, then in theory there is no upper*

Algorithm 7 TimeLord.startVDF

```
1: State: finalized, running
2: Input: $i, (c, t)$
3: $\tau_i \leftarrow$ VDF.solve$(c, t)$ ▷ start thread computing VDF, if this thread does not
   get aborted it will output $\tau_i$ in time required for $t$ sequential steps
4: finalized$[i]$ := finalized$[i] + 1$
5: running$[i]$ := running$[i] - 1$
6: Chain.update$(i, \tau_i)$
```

Algorithm 8 TimeLord.restore

```
1: State: finalized, running
2: Input: $i$            ▷ fresh, valid & finalized block for depth $i$ was received
3: if running$[i] > 0$ and running$[i]$+finalized$[i] = \kappa$ then
4:     abort the thread TimeLord.startVDF for slot $i$ scheduled to finish last
5:     running$[i]$ := running$[i] - 1$
6: end if
7: finalized$[i]$ := $\min\{$finalized$[i] + 1, \kappa\}$
```

bound on the number of threads. If $\kappa = 2$, then we can have many threads, if at many consecutive depths the second best proof of space is much worse than the best one, as illustrated in Figure 6. Such a constellation with $\mathrm{p} \gg \tau$ parallel threads is unlikely (its probability drops exponentially in p), and in practice being able to run, say, 2κ VDFs in parallel will be sufficient most of the time. In the rare cases where it is not, one just pauses the VDF scheduled to finish last (this will decrease the chance of this block ending up in the final chain, but in this particular configuration this block would have an extremely low chance of catching up and not getting orphaned anyways).

Remark 10 (Same κ for space and time)**.** *Note that we use the same parameter κ for the number of PoSpace generated by space farmers at every depth and the number of VDF outputs generated by time lords at every depth. This way we're guaranteed that – if there is at least one honest space farmer – a time lord will always have enough (i.e., κ) blocks to finalize at every depth. And – if there is at least one honest time lord – every space farmer will always get enough (i.e., κ) finalized blocks to extend. But it is of course possible (due to network latency or adversarial behavior) to see more than κ finalized blocks at some depth.*

5 Difficulty Adjustment

In this section we will discuss how the difficulty parameter T is adjusted to ensure blocks appear at roughly the same intervals even as the space dedicated towards securing Chia or the speed of the VDF computation varies over time. Recall that this parameter is used in the TimeLord.finalize Algorithm 6 where the time parameter for the VDF required to finalize a block with a PoSpace σ

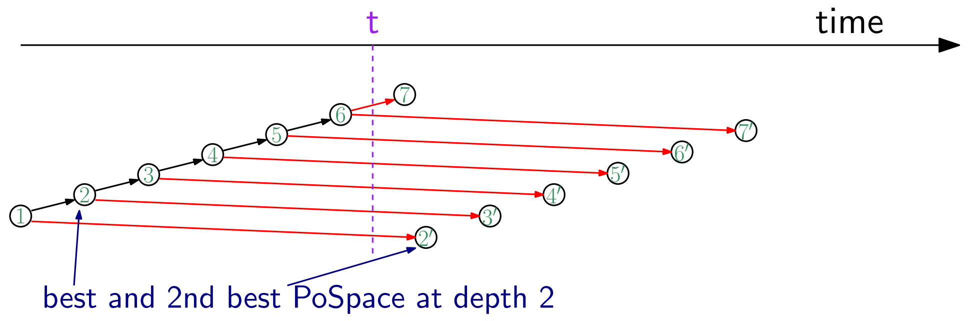


Figure 6: Illustration of a configuration as discussed in Remark 9 where at time t as many as 7 VDF threads (shown in red) are running in parallel.

is computed as $t := 0.\mathsf{H}(\sigma) \cdot T$.

5.1 Difficulty adjustment in Bitcoin

The parameter T plays a role similar to the difficulty parameter D in Bitcoin. There D specifies how hard it is to solve the proof of work required to extend a block: to attach a block β, one needs to find a nonce ν such that $0.\mathsf{H}(\beta, \nu) < 1/D$. Assuming H behaves like a random function, an expected D evaluations of H are necessary and sufficient to find such a proof. To keep the rate at which new blocks are attached to the blockchain at around 600 seconds (= 10 minutes) on average – despite the fact that the hashing power dedicated towards mining varies greatly over time – the parameter D is adjusted every 2016 blocks ($\approx$ every two weeks) as follows: Every block in Bitcoin contains a timestamp (an integer stating the number of seconds since January 1, 1970). Consider a chain of length i, where i is a multiple of 2016. Let χ_j denote the timestamp in the jth block and D_c the current difficulty (which was used for blocks $i - 2015$ to i). The difficulty D_n for the next 2016 blocks is computed as follows. Let

$$\begin{aligned} D' &:= \frac{\text{target time for 2016 blocks (10min per block)}}{\text{time required for the last 2016 blocks (according to timestamps)}} \cdot D_c \\ &= \frac{2016 \cdot 600}{\chi_i - \chi_{i-2016}} \cdot D_c \end{aligned}$$

If we set the new difficulty to $D_n := D'$, this would change the block arrival time for the next blocks to 10 minutes (in expectation) assuming the hashing power is the same as it was (on average) for the last 2016 blocks. That's basically how Bitcoin difficulty adjustment works,[12] except that there is an additional rule that does not allow the difficulty to vary by more than a factor of 4 in every

[12]Bitcoin only takes the time used for the last 2015 blocks (not 2016 as above) into account. This seems to be a bug in the original implementation.

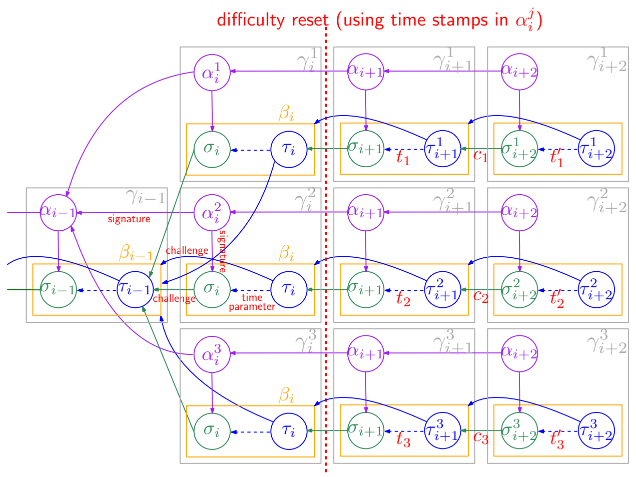


Figure 7: Illustration of the grinding attack on the difficulty parameter discussed in §5.2, where slightly different timestamps in the foliage blocks $\alpha_i^1, \alpha_i^2, \alpha_i^3$ right before the difficulty reset at depth i lead to different PoSpace challenges c_1, c_2, c_3 for the blocks at depth $i+2$.

adjustment, so the actual difficulty is set to

$$D_n := \begin{cases} \min\{D', 4 \cdot D_c\} & \text{if } D' \geq D_c \\ \max\{D', D_c/4\} & \text{if } D' < D_c \end{cases} \tag{4}$$

As we'll explain in §5.4 below setting this value to 4 seems to have a specific reason that is not generally known.

5.2 A grinding attack using difficulty resets

As we discussed in §1.4, by splitting the chain into trunk and foliage and only using unique primitives in the trunk we prevent any kind of digging attacks (recall these refer to grinding attacks where an adversary tries to extend a block in many different ways) as the blocks in the trunk cannot be ground in any way. This simple argument ignores the difficulty parameter T, and we observe that using the same difficulty reset mechanism as in Bitcoin would allow for a subtle digging attack on Chia. On a high level, the reason is that the (ungrindable) trunk is not completely independent from the (grindable) foliage because the time parameter for the VDF depends on the difficulty parameter T, which itself is computed from the timestamps in the foliage.

Using this connection one can launch a digging attack as illustrated in Figure 7. Concretely, consider a chain of depth $i-1$ (with head block $\gamma_{i-1} = (\beta_{i-1}, \alpha_{i-1})$) for an i that is a multiple of 2016, so after the ith block we'll have a difficulty reset. A malicious farmer with space S now

1. Generates a trunk block $\beta_i = (\sigma_i, \tau_i)$ at depth i that extends γ_{i-1}.

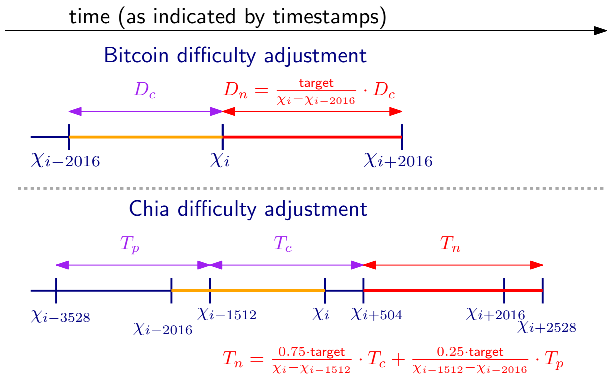

time (as indicated by timestamps)
Bitcoin difficulty adjustment
D_c
$D_n = \frac{\text{target}}{\chi_i - \chi_{i-2016}} \cdot D_c$
χ_{i-2016}
χ_i
χ_{i+2016}
Chia difficulty adjustment
T_p
T_c
T_n
χ_{i-3528}
χ_{i-2016}
χ_{i-1512}
χ_i
χ_{i+504}
χ_{i+2016}
χ_{i+2528}
$T_n = \frac{0.75 \cdot \text{target}}{\chi_i - \chi_{i-1512}} \cdot T_c + \frac{0.25 \cdot \text{target}}{\chi_{i-1512} - \chi_{i-2016}} \cdot T_p$

Figure 8: Illustration of the difficulty adjustment in Bitcoin and Chia. In Bitcoin the difficulty for the new epoch D_n (in red) is computed using the timestamps of the epoch D_c right before it (in orange). In Chia the period to compute the new difficulty T_n is shifted one fourth of an epoch backwards.

2. Generates many foliage blocks (say 3, as illustrated in Figure 7) $\alpha_i^1, \alpha_i^2, \alpha_i^3$ which differ by containing slightly different timestamps. The farmer now has three chains with head blocks $\gamma_i^j = (\beta_i, \alpha_i^j), j \in \{1,2,3\}$ at depth i.
3. Each chain defines a slightly different difficulty parameter $T_1, \ldots, T_3$ to be used for blocks at depth $i+1, \ldots, i+2016$ as each T_j is computed using a different timestamp.
4. Extends each block $\gamma_i^j, j \in \{1,2,3\}$ with a block $\gamma_{i+1}^j = ((\sigma_{i+1}, \tau_{i+1}^j), \alpha_{i+1})$. As we use different time parameters $t_j = 0.\mathsf{H}(\sigma_{i+1}) \cdot T_j$, the VDF outputs τ_{i+1}^j will be different, and then also the challenges $c_j = \mathsf{H}(\tau_{i+1}^j)$ for the PoSpace at depth $i+2$ will be distinct.

The above constitutes a grinding attack where the malicious farmer can get arbitrary many challenges (the number is only limited by the number of VDFs he can compute in parallel) for blocks that are one block after a difficulty adjustment. Being able to grind one block every 2016 blocks might not seem like a serious threat, but a malicious farmer with a small fraction of the space who can compute many VDFs in parallel could create fairly long forks that outrun the honest chain and thus allow for double spending.[13] In the next section we outline Chia's adjustment procedure, which prevents this grinding attack.

5.3 Difficulty adjustment in Chia

The adjustment for the difficulty T in Chia works similarly to the adjustment in Bitcoin discussed in §5.1, but to avoid the grinding attack outlined in §5.2 the

[13]Consider a malicious farmer who controls 10% of the space and can compute 1000 VDFs in parallel. Then with probability $\approx (0.1)^2 = 0.01$ he'll be able to finalize the two blocks at depth i and $i+1$ required for the grinding attack as quickly as the honest farmers, but he now has 1000 challenges for the block at depth $i+2$. The malicious farmer can now compute the blocks at depth $i+2, i+3, \ldots$ as quickly as the honest chain if at every depth he discards the $\approx 90\%$ of the blocks that need longest to finalize. This way he can outrun the honest chain for $\approx \log_{10}(1000) = 4$ blocks before no blocks are left. At this point he has generated a fork of depth 6 that is longer than the honest chain.

adjustment does not kick in right after the timestamp used to compute it has been published, but is delayed by a few blocks.

For concreteness, we discuss below the setting where an epoch in Chia has 2016 blocks (i.e., as in Bitcoin), the target arrival time is 200 seconds (in Bitcoin it is 600), and we set the delay before the difficulty reset applies to one fourth of an epoch, or 504 blocks.

Assume we have a chain of length i where 2016 divides i, and let T_c be the current difficulty parameter (used for blocks $i-1511$ to $i+504$), let T_p be the previous difficulty parameter (used for blocks $i-3527$ to $i-1512$). Recall that χ_j denotes the timestamp in the jth block, let

$$T' := \frac{1512 \cdot 200}{\chi_i - \chi_{i-1512}} \cdot T_c + \frac{504 \cdot 200}{\chi_{i-1512} - \chi_{i-2016}} \cdot T_p$$

Setting the next difficulty parameter T_n (for blocks $i+505$ to $i+2520$) to T' would result in an (expected) block-arrival time of 200 seconds assuming the space dedicated towards farming and the speed of the VDF is on average what it was for the last 2016 blocks. We basically set T_n to T', but as in Bitcoin we don't allow for a fluctuation larger than a factor of 4 in consecutive difficulty parameters

$$T_n := \begin{cases} \min\{T', 4 \cdot T_c\} & \text{if } T' \geq T_c \\ \max\{T', T_c/4\} & \text{if } T' < T_c \end{cases}$$

Let us give the intuition why this adjustment prevents the attack from §5.2. A malicious farmer can theoretically still launch a grinding attack as in §5.2, but now it is not sufficient to compute a (trunk) chain that is two blocks deep (i.e., the blocks at depth i and $i+1$ in Figure 7) before they get distinct challenges for the PoSpace (at depth $i+2$). But now, they need to compute a chain of length two plus the 504 blocks one must wait before the difficulty adjustment kicks in, which is up to depth $i+505$. If this 506-block fork lags way behind the honest chain, then the fact that the adversary gets many challenges at depth $i+505$ will not be enough for the adversary to catch up. If, on the other hand, the adversary is powerful enough to compute a very long (506 blocks) fork at a speed close to the honest chain then this adversary will already be able to generate shorter chains (but still long enough for double spending) faster than the honest parties with good probability due to variance in block generation times.

5.4 On the importance of 4

As outlined in §5.1, Bitcoin does not allow the difficulty parameter in adjacent epochs to vary by more than a factor of 4, and we adapt this rule in Chia.

The choice of 4 might seem arbitrary, but in this section we'll outline an attack against Bitcoin that would be possible if Nakamoto had bounded the fluctuation in-between epochs not by 4, but some value ξ less than $e \approx 2.718$.

The attack is mostly theoretical for several reasons: it requires a 51% adversary, the gain to the adversary is rather marginal and the attack generates a chain with very unbalanced timestamps and is thus easy to detect. We nonetheless explain the attack in more detail as it is surprising that the value of the fluctuation bound opens an attack vector and it should guide the choice of this seemingly rather irrelevant parameter.

As mentioned, in this attack we consider an adversary who controls the majority of the hashing power, called a 51% adversary. Such an adversary can double spend, but they might choose not to do so in order to not undermine the trust in the currency, but instead use their resources to squeeze as much profit from block rewards and transaction fees as possible. A 51% adversary can make sure 100% of the blocks in the longest chain are his (and thus get all the rewards) by simply ignoring all blocks generated by other miners, this way generating a block every 10 minutes in expectation. The adversary can increase this frequency by adding more hashing power, but this will only increase the frequency for one epoch as the next difficulty adjustment will bring the time down to the target frequency of 10 minutes again. If the adversary later decreases the hashing power again to the original level, they will pay for the previous increase by creating blocks at a rate slower than 10 minutes for an entire epoch. A simple calculation shows that the best strategy is to just keep the hashing power constant.[14] This argument ignores the fluctuation bound ξ. In the next section we show that it is possible for a 51% adversary to publish blocks at a rate faster than the target time of 10 min while keeping the difficulty constant if and only if the fluctuation bound ξ is less than $e \approx 2.718$.

5.4.1 An attack if $\xi < e$

Assume Bitcoin used a fluctuation bound $\xi < e \approx 2.718$ instead of $\xi = 4$. Then for a positive constant $\delta = \delta(\xi) > 0$ and some $c = c(\xi) \in \mathbb{Z}^+$, there exists a chain of epochs (each of length c blocks) where the difficulty for the first epoch is D_{start} and after the last epoch the difficulty (as computed from the timestamps contained in the chain) is D_{end} such that

1) The difficulty drops by a constant factor $D_{end} = D_{start} \cdot (1 - \delta)$.
2) The average block arrival time (as indicated by the timestamps) of the chain is at most 10 minutes.

By the two points above a 51% adversary can generate a chain with an average block arrival time strictly below 10 minutes while not increasing the difficulty parameter as follows: They generate and publish the chain as above, and then attach an epoch with an average block-arrival time

$$\frac{D_{end}}{D_{start}} \cdot 10 \text{ minutes} = (1 - \delta) \cdot 10 \text{ minutes}$$

[14]E.g. consider an adversary who first generates blocks every $10/2 = 5$ min for one epoch. This will double the hardness parameter, and to get it down to the original value they will have to generate blocks at a $10 \cdot 2 = 20$ minute frequency for one epoch. This gives an average time of $\frac{5+20}{2} = 12.5$ minutes over those two epochs and thus is slower than just sticking with the 10 minute frequency.

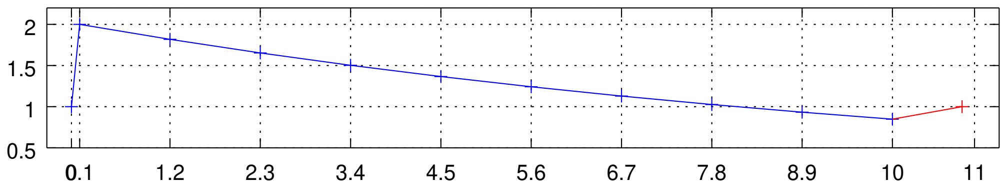


Figure 9: Illustration of the attack from §5.4.1 for $\xi = 2, \epsilon = 0.1$. One unit on the x-axis is the target time for one epoch, the y-axis shows the difficulty (normalized so it starts with 1). We have 10 epochs in blue, the first of length $\epsilon = 0.1$ which increases the difficulty by a factor of $\xi = \max\{1/\epsilon, \xi\}$, followed by 9 epochs of length $(1+\epsilon) = 1.1$, each decreasing the difficulty by a factor $(1+\epsilon)$. These 10 epochs take exactly 10 units of time (one unit being the target time for an epoch), and the difficulty at the end is strictly below the difficulty at the start (in this example $D_{end} = D_{start} \cdot 0.84820$). We then add an eleventh epoch shown in red which takes time $D_{end}/D_{start} = 0.84820$ and thus brings the difficulty back to D_{start}. So we fit 11 epochs in 10.84820 target times without changing the difficulty.

which is strictly less than 10 minutes. After this epoch the difficulty becomes $D_{end} \cdot \frac{D_{start}}{D_{end}} = D_{start}$, so it is back to the initial difficulty, while the block-arrival time of the last epoch – and thus also the entire chain – is strictly below 10 minutes.

We will now explain how to construct a chain that satisfies points 1) and 2) above. An illustration is given in Figure 9. Let $\xi < e \approx 2.718$ and $\epsilon > 0$ be some small constant. Let $Z := 2016 \cdot 600$ denote the target time (in seconds) for an epoch in Bitcoin. We will consider a chain that contains $1/\epsilon$ epochs, and let X_i denote the time for the ith epoch divided by Z (here time means the difference between the timestamp in the last block of this and the previous epoch), so e.g. $X_i = 1$ means the average block arrival time is exactly 10 minutes in the ith epoch. We set the timestamps in the blocks so the first epoch passes extremely quickly, while the others need slightly longer than the target. Concretely, we set the X_i to

$$X_1 = \epsilon \quad , \quad X_i = (1+\epsilon) \text{ for } i = 2, \ldots, 1/\epsilon$$

The total time for this chain of $1/\epsilon$ epochs is

$$\epsilon + (\frac{1}{\epsilon} - 1)(1+\epsilon) = \frac{1}{\epsilon}$$

We assume $1/\epsilon > \xi$, then the difficulty grows by (the maximum allowed factor of) ξ after the first epoch. After that it decreases by a factor $1/(1+\epsilon)$ for $1/\epsilon - 1$ epochs, and thus at the end is

$$D_{end} = D_{start} \cdot \xi \cdot \left(\frac{1}{1+\epsilon}\right)^{1/\epsilon - 1}$$

As ϵ goes to 0, the last term above goes to $1/e$, i.e.,

$$\lim_{\epsilon \to 0} D_{end} = \lim_{\epsilon \to 0} D_{start} \cdot \xi \cdot \left(\frac{1}{1+\epsilon}\right)^{1/\epsilon - 1} = D_{start} \cdot \xi / e$$

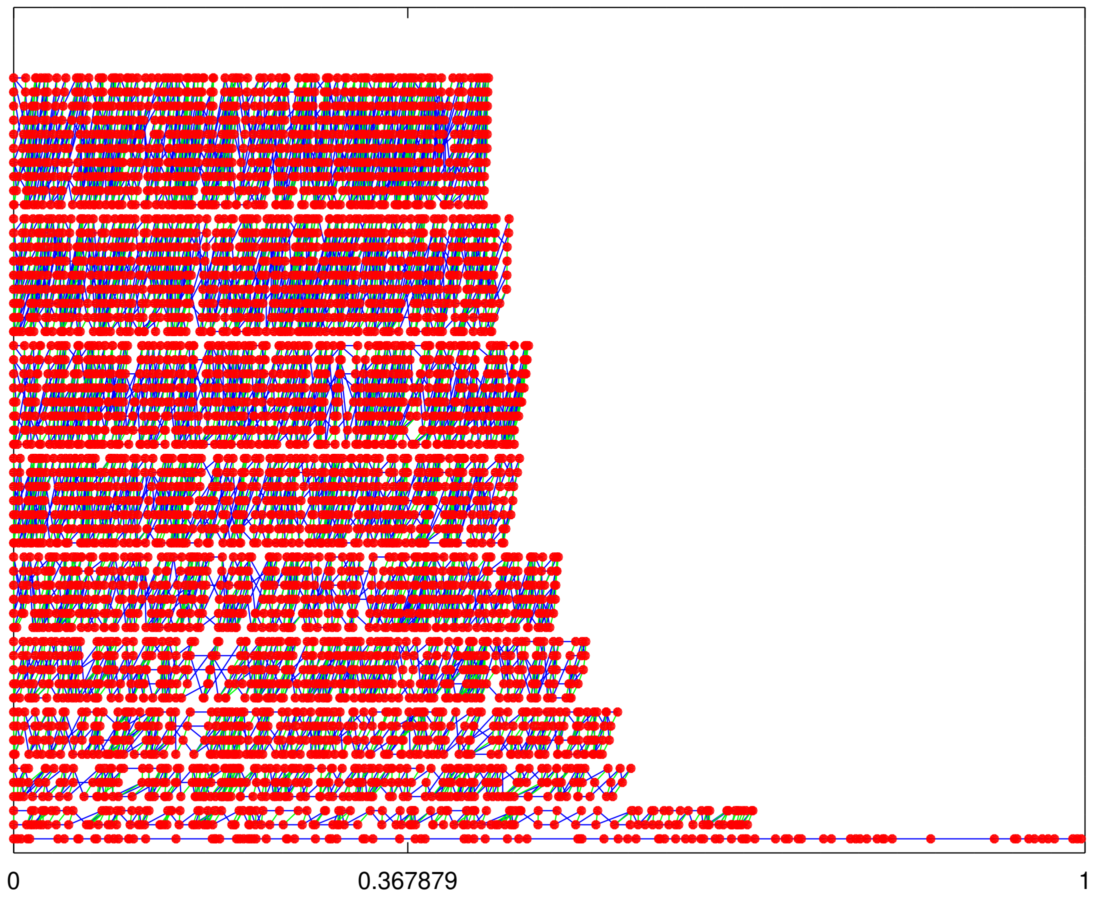


Figure 10: Illustration of simulated samples of $C^{\ell}_{\kappa,h}$ as computed by Algorithm 9 for $\ell = 100, h = 99$ and $\kappa = 1$ to 10 (bottom $\kappa = 1$, top $\kappa = 10$). We see how the time to grow a chain (of length ℓ by h honest farmers) drops from 1 ($= \ell/(h+1)$) towards $1/e \approx 0.3678$ as κ increases.

Thus, for any $\xi < e$, we can choose an $\epsilon > 0$ such that D_{end} is strictly smaller than D_{start} as claimed.

5.4.2 Optimality of the attack

The particular attack we described only works if $\xi < e$, but it is not hard to see that this is not only sufficient, but also necessary. That is, a chain with average block arrival times below the target and where the difficulty at the end is no larger than at the beginning exists if and only if $\xi < e$.

6 Chain Growth and κ

In this section we provide an initial security analysis of the Chia blockchain. We take inspiration from [GKL15] who introduce and analyze the *Bitcoin backbone*.

6.1 Our idealized setting

As discussed in the introduction our analysis is in an idealized setting where

i. we have no network delays,

ii. we have no difficulty resets,

iii. an adversary can compute the VDF no faster than the honest parties,

iv. and the PoSpace are ideal, i.e., they allow for absolutely no time-memory trade-offs.

We consider this idealized setting so we can focus on the actual challenges. The analysis of Chia is non-trivial even with the above restrictions. This is in contrast to Bitcoin, where showing basic security properties (non-zero chain quality, liveliness and persistence) is trivial once we make assumptions i., ii. and the analog of assumption iv. which for Bitcoin means the PoW are ideal.

We are confident – but haven't worked out the details – that removing the above restrictions in our current analysis can be done using existing or straightforward techniques (at the price of achieving weaker quantitative bounds and having more technical proofs).

More concretely, the Bitcoin backbone paper [GKL15] takes network delays into account, and their follow-up work [GKL17] also considers difficulty resets. Removing restrictions i. and ii. should be possible along the same lines for Chia. As outlined in Remark 3, in Chia we can actually tolerate significant network delays without any degradation in security. As for iii., allowing the adversary to compute the VDF a factor $\zeta > 1$ faster than the honest parties has the same effect as increasing the adversarial space by a factor of ζ, so this can easily be taken into account.

Let us also recall that we assume that every farmer dedicates exactly the same amount (N bits) of space, and we discussed in Remark 8 how to drop this assumption.

Apart from the assumption discussed above, we of course always assume that the cryptographic building blocks are secure. In particular, we assume that the signature scheme satisfies the standard security notion of unforgeability under chosen message attacks, and that the hash function is collision resistant (whenever used for chaining) or (the stronger assumption) that it behaves like a random function (when used to map unpredictable values to uniform challenges).

6.2 The random variable $C^{\ell}_{\kappa,h}$ modelling chain growth

To formally argue and simulate the security of Chia we define – for $\kappa, h, \ell \in \mathbb{Z}^+$ – a random variable $C^{\ell}_{\kappa,h}$ whose distribution is the time required for the first path of length ℓ to appear in our idealized setting where h honest farmers grow a chain using parameters κ, $T = 1, \eta = 1$ (and there is at least one honest time lord to finalize blocks).

We fix the difficulty T and the time η required to compute one step of the VDF to 1 simply because the time to grow a chain is linear in T and η, so those parameters are uninteresting and we can keep the number of parameters low by fixing them (though, as outlined in Remark 6, the fact that difficulty can be reset complicates things).

We will denote the expected value of this variable by

$$c^{\ell}_{\kappa,h} \overset{\mathsf{def}}{=} \mathrm{E}[C^{\ell}_{\kappa,h}] \ .$$

The pseudocode of sampling $C^{\ell}_{\kappa,h}$ is given in Algorithm 9.

Remark 11 (How to start sampling $C^{\ell}_{\kappa,h}$ if $\kappa > 1$?)**.** *One difficulty we encounter when defining $C^{\ell}_{\kappa,h}$ for $\kappa > 1$ is how to define the "starting configuration" for sampling. We could assume that there is just one block to extend at the beginning, but (except for the genesis block) the honest farmers have $\kappa > 1$ blocks to extend at every depth, so this would overestimate the time required to grow the chain. We can assume there are κ blocks at the starting time, but this would underestimate the time, as in an actual chain those κ blocks at a given depth will be finalized at different times. When defining $C^{\ell}_{\kappa,h}$ we nevertheless opt for this option (i.e., in Algorithm 9 at line 2 we start with κ blocks at time* 0*), but as the chain starts to behave in a "typical" way almost immediately after we start sampling it (this can be seen in Figure 3) $C^{\ell}_{\kappa,h}$ is a good approximation for the actual time required to grow a chain no matter what the actual starting configuration is. Looking ahead, the reason we define θ_κ in §6.4 for large ℓ (in fact for $\lim_{\ell\to\infty}$) is just so the exact starting configuration doesn't matter.*

Algorithm 9 sample $C^{\ell}_{\kappa,h}$

```
1: Input: $\kappa, \ell, h$
2: $s[1,\dots,\kappa] = 0$                              ▷ initially we have $\kappa$ paths of length 0
3: for $i = 1$ to $\ell$ do                                             ▷ sample $\ell$ steps
4:     for $j = 1$ to $\kappa$ do                           ▷ extend each of the $\kappa$ states...
5:         for $k = 1$ to $h$ do                                         ▷ by $h$ values...
6:             $p[j,k] = s[j] + \mathsf{rand}([0,1])$    ▷ each chosen uniformly from $[0,1]$
7:         end for
8:     end for
9:     $z = \mathsf{sort}(p[1,1],\dots,p[\kappa,h])$                  ▷ sort the $\kappa \cdot h$ values
10:    $s = z[1,\dots,\kappa]$                         ▷ new state in the $\kappa$ shortest paths
11: end for
12: Return min(s)
```

6.3 Analytical bounds on $C^{\ell}_{\kappa,h}$

For $\kappa = 1$ it is not hard to determine the exact expectation of $C^{\ell}_{1,h}$ using the following fact

Proposition 1 (kth order statistics for uniform random variables). *Let $X_1, \ldots, X_n$ be iid each uniform in $[0,1]$. Then the expectation of the kth smallest value of those X_i is*

$$\frac{k}{n+1}$$

Lemma 1 ($\kappa = 1$ chain growth).

$$c^{\ell}_{1,h} \overset{\mathsf{def}}{=} \mathrm{E}[C^{\ell}_{\kappa,h}] = \frac{\ell}{1+h} \tag{5}$$

Proof. Let Z_i be sampled by sampling h random variables $X_1, \ldots, X_h$ iid with uniform distribution in $[0,1]$, then $C^{\ell}_{1,h} = \sum_i^{\ell} Z_i$. By Proposition 1 we have $\mathrm{E}[Z_i] = \frac{1}{h+1}$, and by linearity of expectation

$$c^{\ell}_{1,h} = \mathrm{E}[C^{\ell}_{1,h}] = \mathrm{E}\left[\sum_i^{\ell} Z_i\right] = \sum_{i=1}^{\ell} \mathrm{E}[Z_i] = \frac{\ell}{1+h} \ . \qquad \square$$

As following a strictly larger number paths will always increase the growth speed, for any ℓ, κ, h we have

$$c^{\ell}_{\kappa+1,h} > c^{\ell}_{\kappa,h} \ . \tag{6}$$

Fortunately, this speedup is not unbounded. Even as κ goes to infinity, it can speed up chain growth by at most a factor $e \approx 2.718$

Lemma 2 (upper bound on chain growth). *For any ℓ, h*

$$\lim_{\kappa \to \infty} c^{\ell}_{\kappa,h} \geq e^{-1} \frac{\ell}{1+h} = \underbrace{e^{-1}}_{\approx 0.3678} c^{\ell}_{1,h} \ .$$

The proof is in Appendix A

6.4 ι_i and θ_i

It is crucial to understand how $c^{\ell}_{\kappa,h}$ behaves as a function of κ for two reasons. First, we need to know what happens in the limit $\kappa \to \infty$ to understand how fast a potential adversary (who can run an unlimited number of VDF servers) can grow the chain. Second, we would like to understand what happens for small κ so we can set a good value for the honest farmers. Looking ahead, setting $\kappa = 3$ seems to offer the best trade-off.

As we only have analytical bounds for the extreme cases $\kappa = 1$ and $\kappa = \infty$, we'll rely on simulations for the values in between. Figure 3 on page 12 illustrates simulations for chain growth for $1 \leq \kappa \leq 4$. This figure might lead us to believe that, in contrast to the case $\kappa = 1$, the chain does grow faster by an unbounded factor as κ goes to ∞. Fortunately, by Lemma 2 this is not the case, and the maximum speedup one can get by using a larger κ is $e \approx 2.718$. In Figure 10 we show a simulation up to a larger $\kappa = 10$ where this convergence becomes visible.

We define $\theta_\kappa \in [0,1]$ as the factor by which one can speed up chain growth by extending κ instead of just one chain:

$$\theta_\kappa = \lim_{\ell\to\infty, h\to\infty} \mathrm{E}[C^\ell_{\kappa,h}]/\mathrm{E}[C^\ell_{1,h}] = \frac{1+h}{\ell} \cdot \lim_{\ell\to\infty, h\to\infty} \mathrm{E}[C^\ell_{\kappa,h}]$$

We define this value in the limit for large ℓ and h, but it converges very fast so we can get good approximations by sampling $C^\ell_{\kappa,h}$ for fairly small finite ℓ, h. The reason we consider the limit in ℓ is the issue with the starting configuration for sampling outlined in Remark 11. We have

- $\theta_1 = 1$ by definition.
- $\theta_\infty \geq 1/e \approx 0.367$ (by Lemma 1 and 2).
- $\forall i \,:\, \theta_{i+1} < \theta_i$ (as following more paths always gives strictly more advantage).

Simulated values for $\theta_i, i = 1, \ldots, 100$ are shown (by the blue line) in Figure 2 on page 11.

We define ι_i as the fraction of the space honest farmers following a $\kappa = i$ strategy must control so they can grow a chain faster than a malicious farmer who controls the remaining $1 - \iota_i$ fraction of the space and follows a $\kappa = \infty$ strategy. We will use this value in the next section to state interesting security properties of Chia. We claim that

$$\iota_i \leq 1 - \frac{1}{1 + e \cdot \theta_i} \ . \tag{7}$$

Before we prove this simple claim, let us observe as a sanity check that for $i = \infty$ we get $\iota_\infty \leq 1/2$ (using $\theta_\infty \geq 1/e$) which makes sense as here the honest and malicious farmer do exactly the same thing, so having half the space (i.e., as much as the malicious farmer) is sufficient. In the remainder of this section we will derive eq.(7). Farmers following a $\kappa = i$ strategy with space s can grow a chain at the same speed as an adversary (who follows a $\kappa = \infty$ strategy) with space s' if $s/\theta_i = s'/\theta_\infty$. Using $\theta_\infty \geq 1/e$ this gives

$$s/\theta_i = s'/\theta_\infty \leq e \cdot s' \ .$$

Using this in the last step below, we can give a lower bound of the fraction $\frac{s'}{s'+s}$ of the total space the malicious farmer can control as

$$1 - \iota_i = \frac{s'}{s'+s} = \frac{1}{1 + s/s'} \geq \frac{1}{1 + e \cdot \theta_i} \ .$$

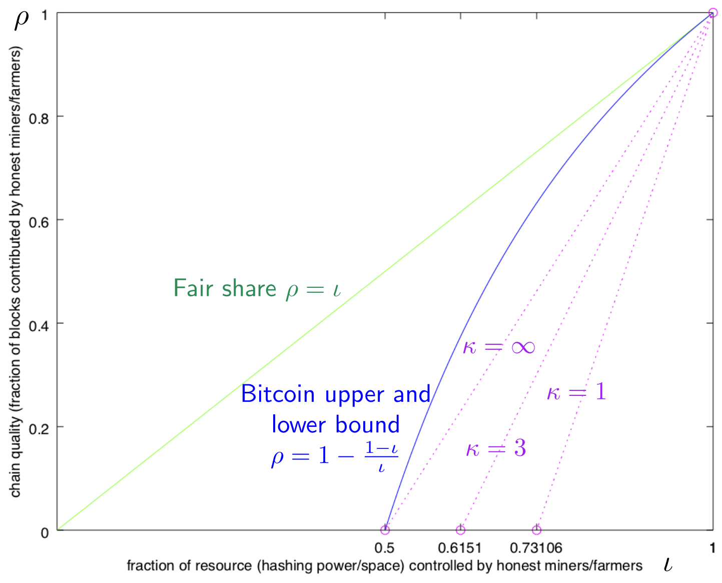


Figure 11: This figure illustrates the achieved chain quality – i.e., the fraction ρ of blocks (y-axis) that are guaranteed to come from honest miners/farmers – assuming they hold a ι fraction (x-axis) of the total hashing power/space. The green line shows the wishful ideal bound $\rho = \iota$ where no strategy is better than the honest mining strategy. The blue line illustrates the Bitcoin chain quality $\rho = 1 - \frac{1-\iota}{\iota}$, which is matched by selfish mining attacks [ES14]. For Chia we show that $\rho > 0$ if $\iota > \iota_i$ whenever the honest farmers follow a $\kappa = i$ strategy. The figure shows $\iota_1 = e/(1+e) \approx 0.73106$ (proven analytically), $\iota_3 \approx 0.6151$ (by simulations, Chia will use $\kappa = 3$) and $\iota_\infty = 0.5$ (trivially). In all cases we don't know exactly how ρ goes from 0 to 1 as ι increases from ι_i to 1 (the dotted lines).

6.5 The Chia Backbone

In §6.4 we established that honest farmers following a $\kappa = i$ strategy can grow a chain faster than a malicious farmer who can compute an unlimited number of VDFs in parallel (and thus follow a $\kappa = \infty$ strategy) if the honest farmers control at least a ι_i fraction of the total space. We proved analytically that $\iota_1 \approx 0.731$ and, using simulations, that $\iota_3 \approx 0.615$. In Chia the honest farmers will follow a $\kappa = 3$ strategy for farming.

Thus, it is *necessary* to assume the honest farmers following a κ strategy control at least a ι_κ fraction of the space, as otherwise attacks like double spending would be possible. In this section we show that assuming this is also *sufficient* to guarantee a meaningful security property for Chia.

Concretely, we consider the notion of *chain quality* which was introduced for Bitcoin in [GKL15] : the chain quality is $\rho \in [0, 1]$ if in any sufficiently long part of the chain (as seen by honest parties) at least a ρ fraction of the blocks must have been contributed by honest miners. If the miners control a $\iota \in [0,1]$ fraction of the hashing power, we ideally want them to contribute a $\rho = \iota$ fraction of the blocks so their share of block rewards corresponds to their share of the contributed resource, this is illustrated by the green line in Figure 11. Chain quality might not seem like the most important property of a blockchain, but it implies natural properties like persistence (once a block is sufficiently deep in the chain it will stay there forever) and liveness (all transactions will ultimately be added to the chain and end up sufficiently deep so they can be considered confirmed).

[GKL15] show that Bitcoin achieves a chain quality of $\rho = 1 - \frac{1-\iota}{\iota}$, as illustrated by the blue line in Figure 11. Although this is much worse than the "fair" $\rho = \iota$, it is the best one can hope for as this upper bound is matched by selfish mining attacks [ES14]. For Chia we prove that if the honest farmers use a $\kappa = i$ strategy, then the chain quality is non-zero whenever $\iota > \iota_i$ (where ι_i is the fraction of space required to grow a chain faster than the malicious farmers introduced in the previous section). Unlike for Bitcoin, we currently don't know how exactly the chain quality ρ improves as the fraction ι of space controlled by the honest farmers increases from ι_i to 1, that is, how exactly the dotted lines between the $(\iota_i, 0)$ and $(1,1)$ points behave in Figure 11.[15]

Lemma 3 (Chain Quality of Chia)**.** *In the idealized setting from §6.1, the following holds for any $i \in \mathbb{Z}^+$: if the honest farmers follow a $\kappa = i$ strategy and control more than a ι_i fraction of the space (that is, we have h honest farmers each controlling N bits of space, and one malicious farmer with $m \cdot N$ bits of space where $\frac{h}{m+h} > \iota_i$) then the chain quality is non-zero.*

Before we can prove this lemma we first need to prove the following

Lemma 4 (No Slowdown Lemma)**.** *In the idealized setting from §6.1, an adversary cannot slow down chain growth. That is, no matter what the adversary does, the honest parties will see a chain that (in expectation) is at least as long as if the adversary were not present. This even holds if the adversary has a much larger amount of space and can compute the VDFs much faster than the honest farmers (but we need to assume the signature scheme is secure).*

Note that this lemma by itself gives no meaningful security guarantee, it only says there always will be a chain in the view of the honest parties that is at least as long as if the adversary was not present, but this chain could contain only adversarially generated blocks (0 chain quality) and the adversary might even replace the entire chain (no persistence).

Proof of Lemma 4. We do assume that the adversary knows the identities, that is the public keys pk of all the honest farmers, but does not know the corresponding secret keys. As already discussed in Remark 2, because of this an adversary will not be able to predict the quality of the PoSpace of the honest farmers.

[15] $(1,1)$ is on this curve as for $\iota = 1$ we trivially have $\rho = 1$ because the honest miners/farmers will trivially get all the blocks if the malicious miner/farmer has no resources whatsoever.

We claim that because of this, whenever the adversary publishes a block, this can (in expectation) only increase the speed at which the chain grows. If this block is ignored by the honest farmers (because it is not within the first κ blocks at this depth) this is clear. Otherwise the honest farmers will extend this block instead of some other block they would have extended. But from the viewpoint of the adversary, who does not know the secret keys, the distribution of qualities those blocks will have is exactly the same for the new and the kicked-out block. Because the new block must be finalized before the kicked-out block, this can only increase the speed at which the chain grows. □

Proof of Lemma 3. Assume for contradiction that the chain quality is zero. This means the entire chain as viewed by the honest parties only contains blocks farmed by the adversary. By Lemma 4, this chain is at least as long as the chain the honest farmers would have mined if the adversary were not there. This is a contradiction as by definition, an adversary controlling less than a $1 - \iota_i$ fraction of the space cannot grow a chain faster than honest farmers following a $i = \kappa$ strategy. □

References

[AAC$^+$17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 357–379. Springer, Heidelberg, December 2017.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://eprint.iacr.org/2018/712`.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467. Springer, Heidelberg, April / May 2018.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, August 2015.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, March 2014.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, August 2017.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. `https://eprint.iacr.org/2018/627`.

[PPK+15] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gaži. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. `http://eprint.iacr.org/2015/528`.

[Wes18] Benjamin Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. `https://eprint.iacr.org/2018/623`.

A Missing Proofs for Chain Quality

In this section we prove state and prove Theorem 1, Lemma 2 follows as a corollary of this lemma.

For $h, \ell \in \mathbb{Z}^+$, let $T_{h,\ell}$ denote the full h-ary tree of depth ℓ where each edge is labelled with a value iid chosen from $[0, 1]$. The weight of a path in this tree is the sum of the labels of the edges on this path. We consider the following random variables:

$C^\ell_{\kappa,h}$: As sampled by Algorithm 9. That is, given $T_{h,\ell}$, we consider the paths from the root to the leaves, where we start at the root, and in every level keep track of the κ paths with lowest weight (if $\kappa \leq h$, there are h^ℓ of them). For the important special cases where we just keep track of one or all paths we introduce the following variables.

$H^\ell_h \overset{\mathsf{def}}{=} C^\ell_{1,h}$: is the weight of the path (of length ℓ) starting at the root of $T_{h,\ell}$, where at each node we follow the edge with smallest value (until we reach a leaf).

$A^\ell_h \overset{\mathsf{def}}{=} C^\ell_{\infty,h}$: is the minimum weight of any path from the root to a leaf in $T_{h,\ell}$.

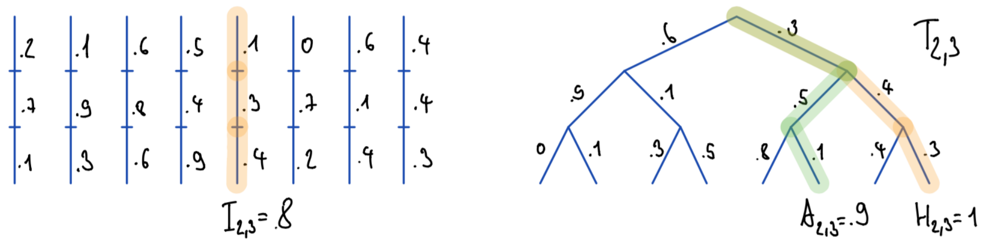


Figure 12: Illustration of outcomes of the variables $T^\ell_h, H^\ell_h, A^\ell_h, Z^\ell_h$ for $h = 2$ and $\ell = 3$.

Y^ℓ **:** is the sum of ℓ iid values chosen uniformly from $[0, 1]$.

Z^ℓ_h **:** is $\min\{Y^\ell[1], \ldots, Y^\ell[h^\ell]\}$ where each $Y^\ell[i]$ is iid as above.

We use the corresponding small letters to denote the expected values of those variables, e.g., h^ℓ_h denotes $\mathrm{E}[H^\ell_h]$.

Note that for $\ell = 1$ we have[16] $H^1_h \sim A^1_h \sim Z^1_h$, and in particular $h^1_h = a^1_h = z^1_h$.

H^ℓ_h corresponds to the time required to farm a path of length ℓ using h space by the honest farmers, who always extend the best proof found. A^ℓ_h corresponds to adversarial (or rational) farming where one extends every possible chain. Z^ℓ_h is just introduced for technical reasons, it is much easier to analyze than A^ℓ_h, and by Lemma 5 it is a lower bound for A^ℓ_h.

Lemma 5. *For any h, ℓ, the following holds for all $x \in [0, \ell]$*

$$\Pr[Z^\ell_h \leq x] \geq \Pr[A^\ell_h \leq x]$$

and thus also

$$z^\ell_h \leq a^\ell_h \ .$$

Proof sketch. $\Pr[Z^\ell_h \leq x]$ as well as $\Pr[A^\ell_h \leq x]$ both consider the event that at least one of h^ℓ paths, each of length ℓ and each edge assigned a weight that is a uniform value in $[0, 1]$, has weight at most x. The only difference is that the in the latter, the edge weights are correlated (as many paths in the tree share edges). The *expected* number of paths of length $\leq x$ (or any other predicate) is exactly the same in Z^ℓ_h as in A^ℓ_h. But intuitively in Z^ℓ_h the probability of *at least one* path being short can be larger because the lengths are positively correlated in A^ℓ_h. We omit making this argument formal. $\square$

We proved Lemma 1 which states

$$h^\ell_h = c^\ell_{1,h} \overset{\mathsf{def}}{=} \mathrm{E}[C^\ell_{\kappa,h}] = \frac{\ell}{1+h} \ .$$

[16] $A \sim B$ means the random variables are identically distributed.

By definition $c^\ell_{\infty,h} = a^\ell_h \leq h^\ell_h = c^\ell_{1,h}$, and to prove Lemma 2 we have to bound how much smaller a^ℓ_h can be than h^ℓ_h.

The concrete question we ask is, if honest farmers have space h, what's an upper bound on the space $M = M(h)$ we can give an adversary (that expands all paths) such that they can't grow a chain faster than the honest guys. By the following theorem, setting $M = h/e$ is sufficient.

Theorem 1. *With $h = eM$ (where $e \approx 2.718$) and $\ell/h < 1$*

$$\Pr\left[A^\ell_M > h^\ell_h\right] > 1 - 1/e > 0.632$$

Proof. The cumulative distribution function of Y^ℓ is[17]

$$\Pr[Y^\ell \leq x] = \frac{1}{\ell!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k\binom{\ell}{k}(x-k)^\ell \ .$$

We will consider the case $x = \ell/eM = \ell/h$, as $\ell/h < 1$, the above sum just has one term. Using $\ell^\ell/e^{\ell-1} \leq \ell!$, we get

$$\Pr\left[Y^\ell \leq \frac{\ell}{eM}\right] = \frac{1}{\ell!}\sum_{k=0}^{0}(-1)^k\binom{\ell}{k}(x-k)^\ell = \frac{x^\ell}{\ell!} = \frac{\ell^\ell}{\ell! e^\ell M^\ell} < \frac{1}{e \cdot M^\ell} \ .$$

Thus, by the union bound

$$\Pr\left[Z^\ell_M \leq \frac{\ell}{eM}\right] \leq \Pr\left[Y^\ell \leq \frac{\ell}{eM}\right] M^\ell \leq 1/e$$

Using Lemma 5 in the first step and $h^\ell_h = \ell/(eM+1)$ together with the above equation in the second step

$$\Pr\left[A^\ell_M \leq h^\ell_h\right] \leq \Pr\left[Z^\ell_M \leq h^\ell_h\right] \leq 1/e \ .$$

Equivalently,

$$\Pr\left[A^\ell_M > h^\ell_h\right] > 1 - 1/e \geq 0.63 \ .$$

□

[17] `https://en.wikipedia.org/wiki/Irwin-Hall_distribution`