Contents

Green Paper **3**
Abstract . . . 3
Precursor Consensus Green Paper . . . 3

0 - Constants, Variables and Notation **4**
0.1 Important Constants . . . 4
0.2 Important Variables . . . 4
0.3 Boxes . . . 5

1 - Introduction **6**
1.1 Security . . . 7
1.2 Network Delays . . . 8
1.3 Game Theoretic Aspects . . . 9
1.4 Farmers and Timelords . . . 10
1.5 Difficulty and Chain Selection Rule . . . 11
1.6 Cryptographic Building Blocks . . . 12
1.7 A High Level View of the Protocol . . . 13
1.8 Space Oddities . . . 14
1.8.1 Sized vs. Unsized . . . 14
1.8.2 Internal vs. External . . . 15
1.8.3 Replotting . . . 15

2 - Longest-Chain Protocols from Efficient Proof Systems **16**
2.1 Grinding . . . 18
2.2 Double-Dipping . . . 18
2.3 Bootstrapping . . . 19

3 - Rational Attackers **21**
3.1 Selfish Mining in Bitcoin . . . 21
3.2 Delayed Gratification and No Slowdown . . . 21
3.3 Chain Quality . . . 22
3.4 Delayed Gratification in Chia . . . 23
3.5 No-Slowdown of Chia and other Constructions . . . 24
3.5.1 No-Slowdown in Bitcoin . . . 24
3.5.2 A Non-Example, the G-Greedy-Rule . . . 24
3.5.3 Examples of No-Slowdown. . . . 25

4 - Hash and VDF chains **27**
4.1 Hash chains . . . 27
4.2 VDF chains . . . 27
4.2.1 Notation for VDF chains . . . 28

5 - The Chia Blockchain **29**

5.1 Additional Variables and Notation for this Section . . . 29
5.1.1 Variables . . . 29
5.1.2 Step to Epoch . . . 29
5.1.3 Notation for Points of Interest . . . 30
5.2 The Challenge Chain . . . 31
5.3 Trunk Blocks . . . 32
5.4 The Reward Chain . . . 33
5.5 The Infused Challenge Chain . . . 33
5.6 The Foliage . . . 35
5.7 Fraction of Transaction Blocks . . . 36

6 - Recovering from 51% Attacks and Dynamic Availability 37
6.1 Recovery from 51% Attacks . . . 37
6.1.1 Recovering from PoW Majority in Bitcoin . . . 38
6.1.2 Recovering from PoStake Majority . . . 39
6.1.3 Recovering from PoSpace Majority . . . 39
6.1.4 Recovering from Space-Time Majority in Chia . . . 39
6.2 Dynamic Availability . . . 40
6.2.1 Dynamic Availability for PoW (Bitcoin) . . . 41
6.2.2 Dynamic Availability for PoST (Chia) . . . 41
6.2.3 Dynamic Availability from PoSpace . . . 41
6.2.4 Dynamic Availability from PoStake . . . 42

References 43
Additional Reading . . . 46

A - Building Blocks: PoSpace, VDFs and Signatures 47
A.1 (Unique) Digital Signatures . . . 47
A.2 (Unique) Proofs Of Space . . . 47
A.2.1 Algorithms for PoSpace . . . 47
A.2.2 Security of PoSpace . . . 48
A.2.3 Unique PoSpace . . . 48
A.2.4 The [AAC+17] PoSpace . . . 49
A.3 Verifiable Delay Functions . . . 49
A.3.1 The [Pie19b, Wes20] VDFs . . . 51

Green Paper

Abstract

This document outlines the rationale and main design ideas behind the consensus layer of Chia, a longest-chain blockchain akin to Bitcoin. It achieves comparable security guarantees as Bitcoin's Proof of Work (PoW) based Nakamoto consensus, while using Proofs of Space in combination with Verifiable Delay Functions (VDFs) to achieve Sybil resistance. This makes Chia much more sustainable and also more decentralized than a PoW based blockchain could be.

We outline the challenges one must solve when replacing proofs of work with an efficient proof system like proofs of space, and how they are addressed in Chia. Here *efficient* means that once the resource (like space or stake) is available, computing many proofs is basically as cheap as computing one.

This document is not a formal specification of Chia. Instead, it aims at readers who want to understand the design choices of Chia's consensus, and are interested in permissionless longest-chain blockchains from efficient proof systems in general.

Precursor Consensus Green Paper

In order to provide historical context, the Green Paper's previous version that discusses a precursor consensus which was never implemented is available here for viewing: Precursor Green Paper.

0 - Constants, Variables and Notation

0.1 Important Constants

Constants	Description
10 minutes	target **duration of a sub-slot**
32 blocks	target **number of blocks per sub-slot**
16/64 blocks	minimum/maximum number of blocks in a slot
4608 blocks	average number of **blocks per epoch**
384 blocks	average number of **blocks per sub-epoch**
64 signage points	number of **signage points per sub-slot**

The above imply the following:

Implied Constants	Description
1 day	**target time of an epoch** is $$10\ \mathsf{min} \cdot \frac{4608\ \mathsf{blocks}}{32\ \mathsf{blocks}} = 1440\ \mathsf{min} \quad (= 1\mathsf{day})$$
2 hours	**target time of sub-epoch**
18.75 seconds	**target average block arrival time** is $\frac{10\ \mathsf{min}}{32} = 18.75\ \mathsf{sec}$
9.375 seconds	**target time between signage points** is $\frac{600}{64} = 9.375\ \mathsf{sec}$

0.2 Important Variables

Variable	Description
$D \in \mathbb{N}$	difficulty parameter. Re-calibrated once per epoch to meet target of 32 blocks per slot
$T \in \mathbb{N}$	time parameter (number of VDF steps for sub-slot). Re-calibrated once per epoch to meet target of 10 minutes per sub-slot

0.3 Boxes

Objective 0:
We will use blue boxes like this one to mention key objectives we want the design of Chia to satisfy

Design Choice 0:
Green boxes like this are used to highlight important design choices, which often will refer to objectives.

Security Notice 0:
A red box stresses some important aspects required for the security of Chia, and will typically refer to some design choice.

1 - Introduction

The **Chia network** (chia.net) is a permissionless blockchain that was launched on March 19, 2021. Chia is a "longest-chain" blockchain like Bitcoin, but uses disk-space instead of computation as the main resource to achieve consensus. This holds the promise of being much more ecologically and economically sustainable and more decentralized than a proof of work (PoW) based blockchain like Bitcoin could be. Figure 1 illustrates one slot of the Chia blockchain. The main aim of this document is to explain the rationale for this rather complicated design.

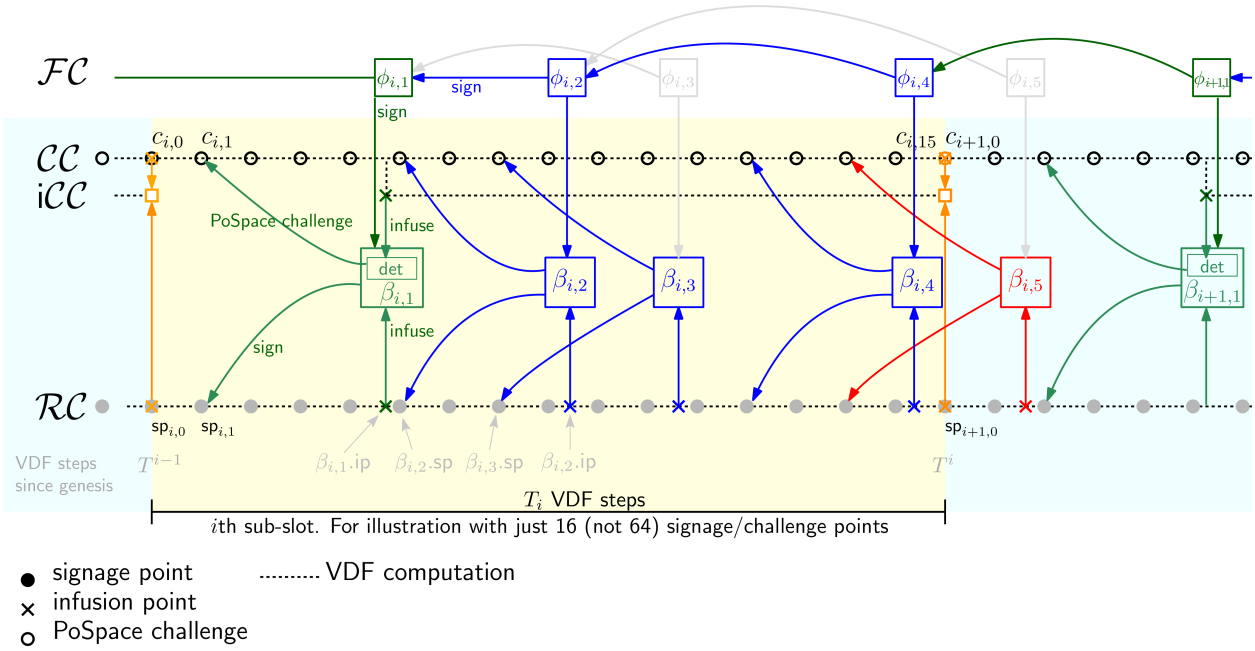

Figure 1: Illustration of one slot (taking around 10 minutes) of the Chia blockchain. For illustration the slot has just 16 (instead 64) signage points and only 4 blocks (the actual chain has a target of 32).

As mentioned, Chia is basically what's called a *longest-chain* protocol in the literature [BNPW19; BDK+19]. This notion captures blockchain protocols that borrow the main ideas from the Bitcoin blockchain: the parties (called miners in Bitcoin and farmers in Chia) that dedicate resources (hashing power in Bitcoin, disk space in Chia) towards securing the blockchain just need to

1. listen to the (P2P) network to learn about progress of the chain and to collect transactions.
2. locally use the resource (via proofs of work in Bitcoin or proofs of space in Chia) trying to create a block which extends the current chain.
3. if a winning block is found, gossip the new block to the network.

No other coordination or communication amongst the parties is required. In particular, as the miners in Bitcoin, the farmers in Chia only need to speak up once they find a block and want it to be included in the chain.

Constructing a secure permissionless blockchain using proofs of space is much more challenging than using proofs of work. In particular, a secure (under dynamic availability) longest-chain protocol

based on proofs of space alone does not exist [BP22], so Chia's *proofs of space and time* (PoST) consensus protocol, apart from farmers providing disk space, additionally relies on so called timelords who evaluate verifiable delay functions (VDFs). Figure 2 gives an overview of the formal security proofs and more informal arguments outlined in this document.

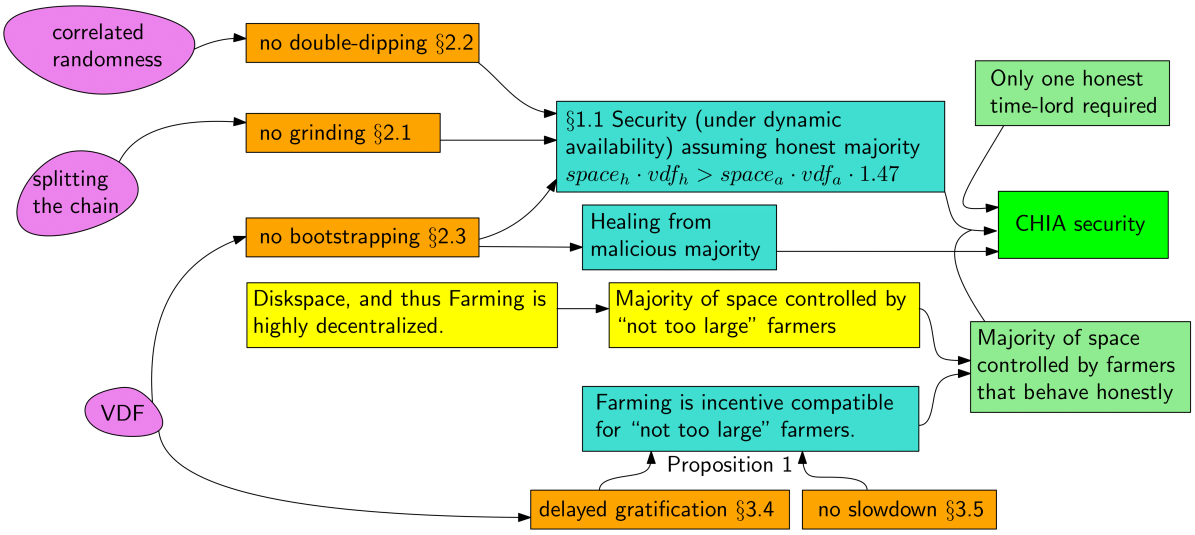

Figure 2: An illustration of the main security proofs and arguments for the Chia consensus layer.

1.1 Security

The Bitcoin blockchain is secure [GKL15] as long as the hashing power $hash_h$ (measured in hashes per second) contributed by honest parties is larger than the hashing power $hash_a$ available to an adversary, i.e.,

$$hash_h > hash_a \tag{1}$$

Similarly, the security of Chia depends on the amount of space $space_h$ and $space_a$ controlled by the honest parties and the adversary, respectively. Additionally, the speed vdf_h and vdf_a (measured in steps per second) of the VDFs run by the fastest honest timelord and the adversary are relevant. With these definitions,

$$\text{Chia is provably secure if : } space_h \cdot vdf_h > space_a \cdot vdf_a \cdot 1.47 \tag{2}$$

Let us stress that Chia only requires a single timelord (which runs 3 VDFs) to be active at any time, in particular, vdf_h in eq.(2) refers to the speed of the fastest VDFs controlled by an active and honest timelord, it doesn't matter if one or a billion timelords are active. In practice we'd still expect a small number – not just one – timelords to be available to have a backup should the currently fastest timelord become unavailable.

On the other hand, we make no assumptions about the number of VDFs controlled by the adversary.

Security as in eq.(2) holds even when assuming the adversary controls an unbounded number of VDFs of speed vdf_a.

This assumption comes at a price: there's a 1.47 factor by which the adversarial resources are multiplied in eq.(2). This factor is there due to an attack we call "double dipping". This and other attacks will be discussed in §2. For now let us just mention that there's nothing special about the constant 1.47, it can be lowered to $1 + \epsilon$ for any $\epsilon > 0$ by increasing the number of blocks that depend on the same challenge (in Chia this is set to at least 16).

The bound in eq.(2) is not tight in the sense that we don't have an attack that works if we replace ">" with "<". We have an attack assuming giving the adversary a slightly lower boosting factor of 1.34

$$\text{double spending in Chia possible if : } space_h \cdot vdf_h < space_a \cdot vdf_a \cdot 1.34 \tag{3}$$

More concretely, if $vdf_h = vdf_a$, i.e., if the adversary has (an unbounded number of) VDFs of the same speed as the fastest honest timelord, then double spending is possible controlling $\frac{100\%}{1+1.34} \approx 43\%$ of the total space.

A contribution of this writeup is a modular approach towards achieving secure longest-chain blockchains from efficient proof systems. In §2 we outline three attack vectors (illustrated in Figure 3) that emerge if we naïvely replace proof of work with an efficient proof systems.

1.2 Network Delays

In Bitcoin each block contains the hash of the previous block. If two blocks are found at roughly the same time, so there was no time for the block that was found first to propagate to the miner that found the second, they will refer to the same block, and only one can be added to the chain. The other will be "orphaned" and does not contribute towards securing the blockchain. The fraction of orphaned blocks depends on the network delay (the smaller the delay the fewer orphans) and the block-arrival time (fewer blocks per minute decrease the probability of orphans). Taking this into account, the security statement for Bitcoin from eq.(1) should be augmented to:

$$hash_h \cdot \left(1 - \frac{network\ delay}{block\ arrival\ time}\right) > hash_a \tag{4}$$

Even with its very slow 10 minutes block arrival time, Bitcoin's orphan rate was measured to be around 1.6% [DW13]. As the Chia chain is not a typical hash chain, but an ongoing VDF computation where blocks are infused, there's an elegant way to avoid orphans: the "infusion point" of a block is around 30 seconds (more precisely, between 28.125 and 37.5 seconds) worth of VDF computations after the "signage point" it must refer to, and as long as the network delay is small enough so the block creating/gossiping process takes less than 30 seconds no orphans will occur. In particular, the bound from eq.(2) holds under this very weak network assumption independent of the block arrival time.

The target block arrival time in Chia is set to 18.75 seconds (32 blocks per 10 Minutes slot), and

while each of those blocks contributes to security, only a subset of these blocks actually carry transactions (roughly 36%, that's a block every 51.2 seconds) in order to ensure that transaction blocks sequentially refer to each other. This prevents issues with inconsistent transactions, as each block producer knows the entire history.

1.3 Game Theoretic Aspects

Apart from proving security assuming the honest parties control a sufficient majority of the resources, to argue that a longest-chain protocol will be secure in the real world we need to justify why rational parties would behave honestly in the first place. In particular, it should not be possible to get more rewards by deviating from the honest mining/farming behavior. While Bitcoin is not fair in this sense due to selfish mining attacks [ES18], these attacks are not really practical and have not been observed in the wild for reasons we'll sketch below and discuss in more detail in §3.

Fairness in Chia

Achieving fairness that is comparable to what Bitcoin achieves is a main design goal of Chia. While arguing about fairness directly is rather subtle, we identify two clean properties called "no slowdown" and "delayed gratification" a longest-chain can satisfy. Delayed gratification by itself already is a deterrent against selfish farming, and we show (Proposition 1 in §3) that these two properties jointly imply a chain-quality (i.e., fraction of honest blocks in the longest chain) no worse that what Bitcoin achieves.

No-Slowdown

The no-slowdown property was identified as a desirable property for longest-chain blockchains in [CP19]. It holds if (even an unbounded) adversary cannot slow down the growth of the chain by participating. We discuss the no-slowdown in Chia and various other chains in §3.5.

Delayed Gratification

The Chia design ensures that proof of space challenges are only revealed once they are needed, and once they're revealed they cannot be influenced any more. This then implies that it's impossible for a selfish farmer to create more blocks by deviating from honest farming, and thus – like in Bitcoin – the only thing a selfish farmer can do is prevent other farmers from adding their fair share of blocks in the current epoch (potentially even losing out on blocks themself). The reason a selfish farmer would do this is in order to enforce a lower difficulty, and thus more rewards for themselves, in the future [ES18]. We denote chains with this property as having "delayed gratification".[1] While delayed gratification doesn't prevent selfish mining, it severely limits the type of selfish mining possible, and we don't expect to observe selfish farming in Chia for the same reasons we don't observe selfish mining in Bitcoin. As mentioned above, in combination with the no-slowdown property it even implies a chain-quality as in Bitcoin.

[1] According to Wikipedia, *delayed gratification* is the resistance to the temptation of an immediate pleasure in the hope of obtaining a valuable and long-lasting reward in the long-term.

1.4 Farmers and Timelords

Constructing a secure blockchain based on proofs of space is significantly more challenging than with proof of work. So the Chia design, as illustrated in Figure 1 is (arguably necessarily) more sophisticated than Bitcoin or other PoW based blockchains, which are basically just hash chains. Apart from proofs of space and standard cryptographic building blocks like hash functions and signature schemes, the security of Chia crucially relies on verifiable delay functions (VDFs) [BBBF18; Pie19b; Wes20]. Informally, VDFs are functions whose computation is inherently sequential and verifiable and thus serve as a "proof of time".

We will now shortly sketch how the Chia blockchain is maintained by farmers and timelords.

Farmers

Farmers are the analog of miners in Bitcoin, but instead of hashing power, farmers contribute disk-space towards securing the Chia blockchain. As in Bitcoin, they are incentivized by block-rewards and transaction fees. As in Bitcoin, the block-rewards (i.e., some freshly minted coins that go to the block creator) decrease over time, but unlike in Bitcoin they will never go to zero for reasons outlined in [CKWN16].

To participate in farming a farmer must first initalize its disk-space, this process is called plotting and the files created and stored during this process are called plots. The smallest allowed plot in Chia is slightly larger than 100GB, though for plotting one temporarily needs more than this. Once the plot(s) are in place, a farmer just listens to the network for proof of space challenges. There's a new challenge roughly every 9.375 seconds and they are computed by a timelord as discussed below. For efficiency reasons there's a "plot filter" which for each plot dismisses all but (in expectation) one in 512 challenges immediately, so a plot is only accessed once every 80 minutes. The reason to not increase this time even further are so called replotting attacks which we'll discuss in §1.8.3.

In Chia only roughly 36% of the blocks will carry transactions, but as a farmer doesn't know whether their block will be a transaction block when creating the block, farmers must always include transactions to the blocks they create.

Timelords

A timelord runs three VDFs, once every 9.375 seconds they gossip a "signage point" that serves as a PoSpace challenge for the farmers. They also listen to the network for blocks created by farmers. If a valid block is received in time it gets "infused" into the chain (the infusion is always somewhere from 28.125 to 37.5 seconds after the signage point).

The above only holds for the the timelord which runs the fastest VDFs. Timelords with slower VDFs can basically just recompute values that were already gossiped, so there's seemingly no point for them to participate. We still want a small number of timelords to participate (or at least be ready to take over) should the fastest timelord fail or misbehave.

Unlike farmers, timelords do not receive any rewards in form of block-rewards or transaction fees. One reason is technical, unlike for farmers whose PoSpace contained in the blocks are linked to a signature public-key to which a reward can be given, the computation of the timelords is (and to prevent grinding attacks must be) canonical, they cannot attach a public-key to the values

they computed. A second reason is the fact that it's not clear at all how such a reward would be distributed. If the fastest timelord gets the entire reward only they would be incentivized, but not the slower ones we'd like to have as back-ups. If also the slower ones get something then we'd get a PoW type lottery which we want to avoid in the first place. Chia thus relies on a small number of timelords to run fast VDFs without being incentivised by on-chain rewards.

1.5 Difficulty and Chain Selection Rule

Difficulty

In Bitcoin a difficulty parameter D controls how many hashes are required in expectation to find a block. This parameter is re-calibrated every 2016 blocks (called an epoch and taking roughly 2 weeks) so blocks arrive roughly every 10 Minutes.

Chia has two parameters, a difficulty parameter D and a time parameter T, these are re-calibrated once every 4608 blocks (this epoch takes around 1 day). The time parameter T is reset to fit the target time of 10 minutes per slot, while the difficulty is reset to target an average of 32 blocks per slot.

For example if in an epoch the amount of space is 10% higher than anticipated the difficulty for the next epoch would get up $D_{new} := D_{old} \cdot 1.1$ while the time parameter remains unchanged $T_{new} := T_{old}$. If the VDF speed in the epoch is 10% higher than anticipated (i.e., the epoch only takes 24/1.1 instead of 24 hours) the time parameter goes up $T_{new} := T_{old} \cdot 1.1$, and even though the space didn't change, the difficulty needs also to go up $D_{new} := D_{old} \cdot 1.1$ account for the fact that now an epoch has more VDF steps.

Chain Selection Rule

Bitcoin has a very simple chain selection rule (aka. fork choice rule) which specifies which fork a miner should work on: a miner should always try to extend the "heaviest" chain they are aware of. The weight of a chain is the sum of the blocks, each multiplied by the difficulty parameter used while it was mined. Unless we consider forks which pass an epoch boundary, the heaviest chain is also the chain with the larger number of blocks, hence the name "longest chain" protocol.

We can define the weight of a chain in Chia analogously to Bitcoin, and currently the default Chia farmer code follows basically the same "follow the heaviest chain" rule as Bitcoin miners. But let us stress that it's not clear whether for Chia this simple rule is the best choice. For example, one could consider a rule for farmers where in case of a fork where both chains have the same weight they would work on both chains (note that in a PoW based chain this is not possible). While such a rule can slow down consensus, the observed fork could be due to an attack (double spending or selfish mining) trying to "split" the contribution of the honest space in two different chains, letting the farmers work on both forks would thwart this.

In Chia we also must specify a chain selection rule for the timelords. A timelord who does not control the fastest VDF will constantly fall behind and thus intuitively should just constantly adapt the chain with the most VDF steps in them. But if all timelords naïvely do this a malicious timelord controlling the fastest VDF could simply skip infusing any blocks they want, allowing for all kinds of attacks. Thus the rule for timelords has to be more nuanced, taking into account the chains they

observe, and also blocks that were created by farmers but not infused in any of the chains.

Determining the best rules for Chia farmers and timelords is ongoing research. Fortunately, the rules for farmers and timelords are more of a social convention rather than a specification of the chain. As our understanding improves, new rules can be implemented in the code base and there's no need for a (soft) fork.

1.6 Cryptographic Building Blocks

The Chia blockchain uses standard cryptographic building blocks, in particular hash functions and signature schemes. More interestingly, it relies on two (non-interactive) proof systems which were especially developed for constructing sustainable blockchains: proofs of space and verifiable delay functions. We shortly discuss the requirements Chia has to these building blocks.

Hash Functions.

Chia uses SHA256 for hashing, but any collision resistant hash function would do. For efficiency reasons, we also use the round function of CHACHA8 and BLAKE3 within the proof of space construction where we just need some scrambling but no cryptographic hardness (not even one-wayness).

Signatures.

Chia uses deterministic BLS signatures for signing. In principle any signature scheme could be used as long as the signatures are unique, i.e., it's impossible (or at least computationally hard) to create two different valid signatures for the same message. Uniqueness will be crucial to prevent so called grinding attacks.

Verifiable Delay Functions.

A VDF is specified by some inherently sequential function, and a proof system for showing the output of the function is correct. The sequential function used in the VDF deployed in Chia is repeated squaring in class groups of unknown order. The group is not fixed, but a fresh group is sampled every time a value is infused. The proof system is Wesolowski's [Wes20] proof of exponentiation which has proof of size only one group element. Only the VDF output, but not the proofs, are committed on-chain. This has the advantage that one can replace the proofs. In the current implementation one first computes a much larger but faster to compute proof of 64 group elements, which later is replaced by a normal (one element) Wesolowski proof. It also means one can easily replace Wesolowski's proof with another proof system should a weakness with this proof system (which relies on new number theoretic assumptions) be discovered. We discuss VDFs in detail in §A.3.

Proofs of Space.

The notion of proofs of space was introduced, and a first construction proposed, in [DFKP15] (a security proof for their construction in the random oracle model was given in [Pie19a]). This construction, which is combinatorial and based on pebbling lower bounds for particular graphs, has the major drawback that the initialization phase is *interactive*. A consequence of this is that if one wants to use this PoSpace in a blockchain, the farmers must first commit to their plots before they can be used for farming (say by recording this commitment on-chain via a special transaction as

suggested in Spacemint [PKF+18]). A new PoSpace with a *non-interactive* initialization had to be developed for Chia [AAC+17]. This construction basically just specifies some function f, and then stores its function table $(x, f(x))$ sorted by the outputs $f(x)$. On challenge some value y, the prover looks up the entry (x, y) (which is efficient as the list is sorted) and replies with the proof x, which can be easily verified checking that $y \stackrel{?}{=} f(x)$. Unfortunately this simple construction miserably fails to be secure: the prover can store much less than the full function table, while still being able to efficiently find proofs. The reason are Hellman's time-memory trade-offs, a technique proposed in 1980 to break symmetric cryptographic schemes [Hel80]. In [AAC+17] it is shown how this simple construction can be "salvaged" to overcome any time-memory trade-offs.[2] We'll discuss definition of a PoSpace, and the construction used in Chia in particular, in §A.2.

1.7 A High Level View of the Protocol

The design and rationale of the Chia blockchain is explained in the following sections, here we'll just give a very high level view of the chain as illustrated in Figure 1. The chain itself consists of four chains, one hash chain and three VDF chains.

Hash and VDF chains

While hash chains are a classical cryptographic construction, VDF chains were first used in Chia. A VDF chain alternates VDF computations with *infused* values. It provides the security properties present in hash-chains, that is, the head of a chain commits its entire past (technically, given the head of a hash or VDF chain, it's computationally infeasible to come up with two different chains that end in that value). In addition, VDF chains come with a sequentiality property: the number of sequential steps to compute the VDF chain is the sum of the steps required for all the VDFs in that chain, i.e., the VDFs must be computed sequentially. Hash and VDF chains are discussed in more detail in §4.

The four chains which constitute the Chia blockchain are the (1) foliage chain FC, which is a normal hash-chain and contains the transactions (2) the reward chain RC which records all blocks (3) the challenge chain CC used to create PoSpace challenges and (4) the infused challenge chain $\mathsf{i}CC$ for some extra security properties. While RC and CC are normal VDF chains, $\mathsf{i}CC$ is more of a sequence of forks from CC.

Blocks

A block $\beta = \{\beta_F, \beta_T\}$ is made of two parts, the foliage block β_F, which contains the payload (transactions and a time-stamp) and the trunk block $\beta_T = \{\sigma, \mu_{\mathsf{rc\_sp}}\}$ which contains a PoSpace σ and a signature $\mu_{\mathsf{rc\_sp}}$.

Building the Chains

- A timelord computes the $RC, CC, \mathsf{i}CC, FC$ chains and broadcasts relevant values to the network. This includes signage points rc_sp and cc_sp which are the values of the RC and CC chains (together with a proof that these values are on the VDF chains) once every 9.375

[2] The crucial observation that makes this possible is the fact that Hellman's attack assumes that $f(.)$ can be efficiently computed in forward direction, while for a PoSpace we just require that the entire function table of $f(.)$ can be computed in time linear in the size of the table.

seconds.

- A farmer who receives these *signage points* $\mathsf{rc\_sp}$ and $\mathsf{cc\_sp}$ checks whether these points are of interest (i.e., on the heaviest known chain) and all the VDF proofs verify. Next, for each of their plots, they use $\mathsf{cc\_sp}$ as a challenge to compute a PoSpace σ. Then they check whether σ satisfies a winning condition that allows to produce a block.

  If a winning PoSpace σ is found, the farmer creates a signature $\mu_{\mathsf{rc\_sp}}$ of $\mathsf{rc\_sp}$ (that verifies under the pk associated with σ) and a foliage block β_F and then gossips the block $\beta = \{\beta_F, \beta_T = \{\sigma, \mu_{\mathsf{rc\_sp}}\}\}$.

- When a timelord receives this block, they check whether the block satisfies all conditions to be infused into the chain.

  (RC) If yes, the trunk block β_T is infused into RC once its *infusion point* is reached, which is somewhere between 3 and 4 signage points (28.125 to 37.5 seconds worth of VDF computations) past the signage point of that block.

  (CC) If this happens to be the first block whose signage points are in the current slot, then $\mu_{\mathsf{rc\_sp}}$ (but not σ) is infused into the challenge chain CC at the end of the current slot. This way the challenge chain depends only on one block per slot.

  ($\mathsf{i}CC$) For some extra security, the timelord doesn't simply wait till the end of the slot to infuse the signature, but a third VDF is used to fork from CC at the infusion point by infusing $\mu_{\mathsf{rc\_sp}}$ into CC, and this fork, called the infused challenge chain $\mathsf{i}CC$, is then infused back to CC at the end of the slot.

  (FC) If the signage point of this block is later than the infusion point of the last *transaction block*, then this block is also a transaction block. Only in this case its foliage β_F is appended to the foliage (hash) chain FC, and this block becomes a "transaction block".

1.8 Space Oddities

When constructing a proof of stake or proof or a space based longest-chain protocol one faces similar challenges due to "nothing at stake" (aka. costless simulation) issues, we'll discuss these in §2. But there also aspects in which Space and Stake differ, and we'll shortly discuss three of them below. The first difference is the fact that space, unlike stake, is an unsized resource, which for example means that we can't have "certificates" [LR21]. The second difference is the fact that stake is an internal resource, while space is an external resource, one of the consequences of this is that a space based protocol can recover from malicious majority, while a stake based cannot. The third are replotting attacks against space which have no analogue in the stake setting.

1.8.1 Sized vs. Unsized

A key difference between stake and work is the fact that in a stake based chain we know the amount of the resource available for mining, while for an external resource like work or space this is no longer the case. Lewis-Pye and Roughgarden [Lew21; LR21] formalize this as the *sized* vs. *unsized* setting and prove some fundamental differences between them. The main result in [LR21] shows that *certificates* which "provide incontrovertible proof of block confirmation", only exist in the sized

setting, i.e., for PoStake but not PoWork blockchains.

In their framework *space* and also *space and time* (i.e., the available space multiplied with the speed of the available VDFs) as used in Chia are an unsized resource, so we can't hope to get certificates.

1.8.2 Internal vs. External

Work or space are actual resources and we can unambiguous talk about some party holding some amount of the resource at some given point in time. Stake on the other hand is an *internal* resource defined relative to some chain on which it is recorded and "holding some stake" usually refers to the stake a party controls on the chain that currently is considered the valid one by honest parties.

The main advantage on using stake to secure a longest-chain protocol is the fact that it's extremely sustainable as no external resource is required to secure the chain. But this comes at a price, one common argument against stake is that the chain is not really permissionless as participating in mining requires acquiring stake from the parties currently controlling it. Also from a security perspective an internal resource is delicate as keys controlling stake *not* on the current chain can be used to attack the chain. A simple example would be an attack by which a party acquires keys that were valid at some block B_i in the past, but which are no longer valid at the current block and thus are "cheap" (e.g., the party can lend a large amount of stake for a short time, or offer to buy outdated keys), and then uses these keys to fork at block B_i and bootstrap a chain to the present.

To prevent such attacks some chain require parties to delete old keys, but it's irrational for a party to delete old keys if they can be valuable in the future, say because one can sell them to an attacker (and this is rational if one holds just little stake, so not selling is unlikely to prevent the attack) or because there's a deep reorg and the old keys suddenly become valuable again. Combining stake with VDFs would make such attacks harder, but not prevent them as we'll discuss in §6

1.8.3 Replotting

A subtle but important difference between stake and space is the fact that space allows for *replotting* which has no analogue in the stake setting: Given a challenge c, a space farmer controlling a plot S of size N can *efficiently* compute *one* proof $\sigma \leftarrow \mathsf{PoSpace.prove}(S, c)$. This is analogous to the stake based setting, but unlike in the stake setting, the farmer can *inefficiently* compute *multiple* proofs for challenge c by repeatedly creating fresh plots and computing one proof with each of them.

We refer to attacks exploiting this fact as *replotting attacks*. The most basic design choice to harden a chain against replotting attacks is to make sure that challenges arrive at a sufficiently high rate so that substantial replotting in-between two challenges is not feasible. Moreover the plot filter (which dictates what fraction of plots must be accessed with every challenge) cannot be chosen too aggressively as more aggressive filters makes potential replotting attacks easier.

A fundamental fact about PoSpace that crucially relies on replotting is that *no PoSpace based longest-chain protocol secure under dynamic availability exists* [BP22], we'll discuss their result in more detail in §6.2.3. Chia overcomes this no-go theorem by using VDFs, we discuss security under dynamic availability and healing from malicious majority in the work, stake and space setting in §6.

2 - Longest-Chain Protocols from Efficient Proof Systems

Before we can outline the specification of the Chia blockchain and its rationale in more detail, we first must understand the general challenges one faces when constructing a PoSpace based blockchain and some of the relevant literature on how to address these challenges.

Ultimately, we want to argue security assuming only that sufficient fraction of the resource (space, fast VDFs) is controlled by *rational* parties. Towards this, in this Section we first discuss how to achieve security assuming sufficiently many parties are *honest*, and then in §3 how *rational* behavior is incentivized by ensuring that deviating from the protocol will not give any (or very little) extra reward to parties.

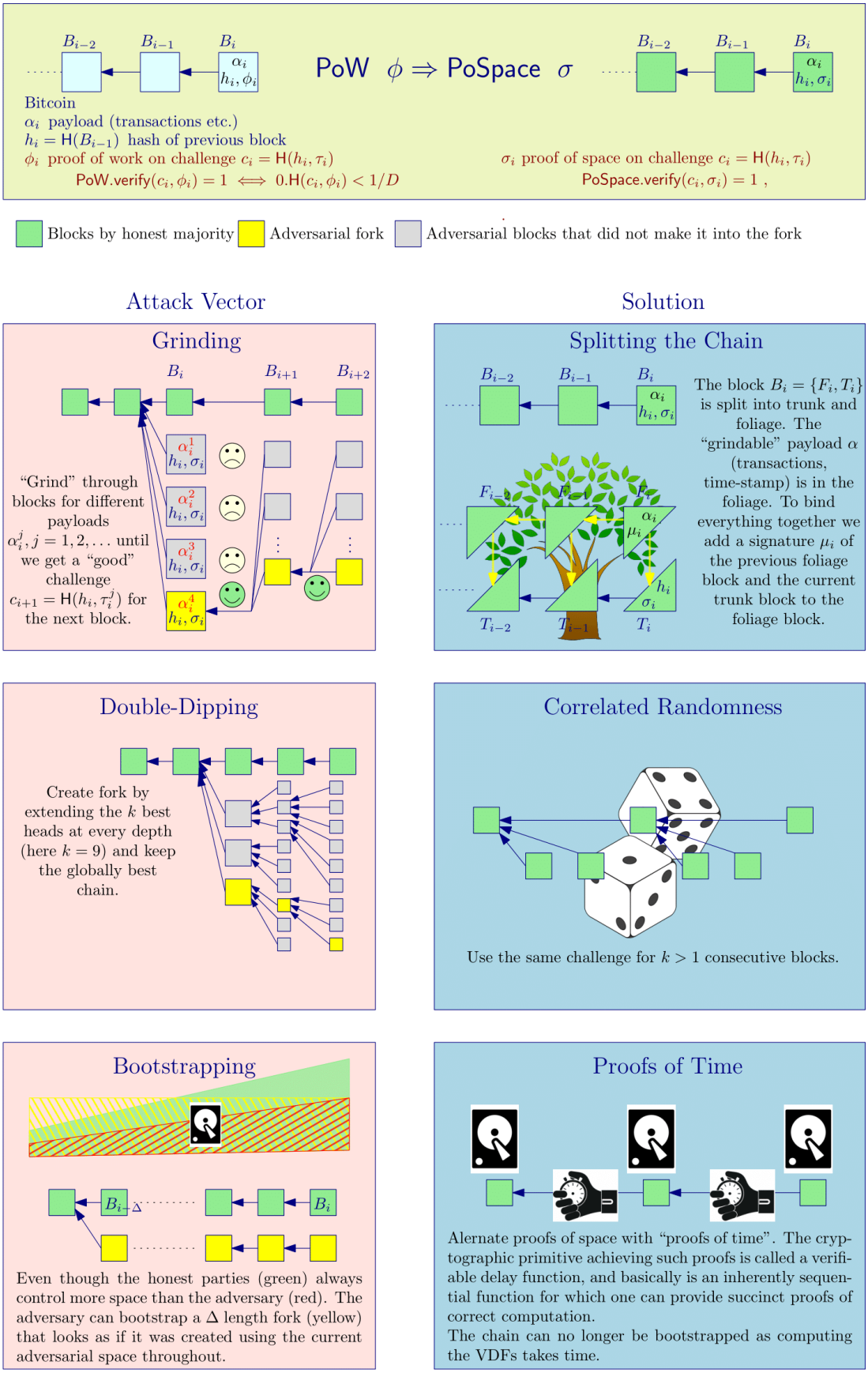

Figure 3: Illustration of the three main attack vectors that arise if we replace PoW with proofs of space (or any other efficient proof systems) in Bitcoin, and how they are addressed in Chia.

As mentioned in the introduction, just replacing proofs of work in Bitcoin with proofs of space does not work. For one thing, there are syntactic differences between proofs of work and proofs of space. But more importantly, security breaks down if one replaces PoW with PoSpace in any straight forward way. The reason for this is the fact that for PoSpace (after plotting) it's extremely cheap to compute a proof for a given challenge. The analogue issue with proof of stake is sometimes called "nothing at stake". We'll refer to proof systems where proofs can be efficiently computed (like proofs of space or stake) as *efficient*, and describe three attack vectors that arise because of this issue: grinding, double-dipping and bootstrapping. Those are illustrated in Figure 3 and described below. In §5 we'll describe how those attacks are addressed in Chia in more detail.

2.1 Grinding

In longest-chain blockchains the challenge used to determine the miner/farmer who can add a block is derived from the chain itself. In Bitcoin, where the challenge for a block is simply the hash of the previous block, a miner can influence the PoW challenge by trying out different transaction sets or time stamps. While such "grinding" through different challenges gives no advantage in PoW based cryptocurrencies, it's a problem once we use an efficient proof system.

To prevent such grinding we adopt an approach from Spacemint [PKF+18] and *split the chain* in two parts which we'll call trunk and foliage. The trunk contains only canonical proofs, and the challenges depend only on values contained in the trunk. This way the only choice a farmer has to influence the challenge is by withholding a winning block. The foliage contains all the remaining "grindeable" content, in particular transactions and time-stamps.

2.2 Double-Dipping

Even once grinding is no longer an option, an adversary can in private create an entire "block-tree" by forking at each level. While each path in such a tree will have an exponentially small (in the depth) probability of overtaking the honest chain if the adversary controls less than half the resources, there's also an exponential number of paths, so it's not clear how much of an advantage this strategy gives. For constructions where each challenge depends on the previous block (as in Bitcoin), it was shown [CP19] that this strategy "boosts" the resource by a factor $e \approx 2.718$, in particular, with this strategy an adversary (having an unlimited number of VDFs whose speed matches the fastest honest time lord) can create a chain that is longer than the honest one with only a $1/(1+e) \approx 0.269$ fraction of the total space, and thus significantly less than the 0.5 fraction (of hashing power) required in Bitcoin.

To limit the impact of double-dipping an early version [CP19] of Chia consensus specified that also the honest parties do a very limited form of double-dipping and try to extend the best 3 blocks they see at every depth, this rule was (by simulations) shown to increase the space required by an adversary from the 26.9% mentioned above, to 38.5%.

The deployed Chia protocol uses *correlated randomness* to limit the impact of double dipping. This elegant idea was introduced in [BDK+19], and basically suggest to only use every kth block to

compute the challenges.[3] The authors of [BDK+19] determine the exact fraction of the *resource* the adversary must control to break security as a function of k (as mentioned, it's 2.718 for $k = 1$, and goes to 1 as k increases). Chia uses a variant where a challenge depends on one out of *at least* (not exactly) $k = 16$ blocks. Their analysis also applies to this setting, and with $k = 16$ states that by double dipping the adversary can boost their resource by a factor of 1.47, which means they must control at least a $\frac{1}{1+1.47} = 0.405$ fraction of the resource for an attack.

The resource considered in [BDK+19] is simply stake, while in Chia it's the product of space and VDF speed. Concerning VDFs, while for the honest parties the only thing that matters is the *speed* of the three VDFs controlled by the fastest honest time lord, for an adversary the speed as well as the number of VDFs available to them matter. In the security analysis we can simply assume the adversary controls an unbounded number of VDFs, as that's when the analysis from [BDK+19] applies. This is how eq.(2) $space_h \cdot vdf_h > space_a \cdot vdf_a \cdot 1.47$ in §1.1 was derived.

2.3 Bootstrapping

A major issue with longest-chain blockchains based on efficient proof systems is bootstrapping (aka. costless simulation) by which an adversary can use its resource to create a chain at basically no cost. Such bootstrapping can be used for short range attacks like selfish mining, but also long range attacks where an adversary forks the chain at a point in the past and then "bootstraps" it into the present. Such long range attacks make it hard to achieve security under dynamic availability, where we assume that the honest parties control a majority of the resource at any point in time, but the total resource can vary over time. The amount of hashing power contributed towards securing Bitcoin has varied by many orders of magnitude in the past, and the same already happened to Chia in the first weeks after launch.

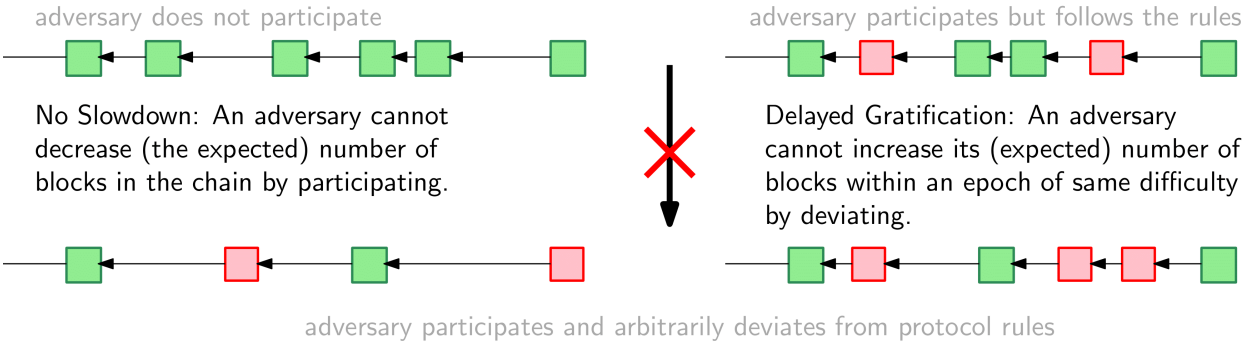

Figure 4: Illustration of the no slowdown and delayed gratification properties. A longest-chain blockchain satisfying these properties is no more susceptible to selfish mining than Bitcoin.

Existing proposals to achieve security under such *dynamic availability* include check-pointing, which is problematic as parties that join for the first time or have not followed the chain for a longer period need additional trust assumptions to decide which chain to follow. Unlike for grinding and double-dipping, the attacks that become possible due to bootstrapping are quite different for proofs

[3] Using the same challenge for $k > 1$ blocks has already been suggested in [PKF+18], but this is a different requirement (as the challenge can still depend on all blocks) and adds much less security than correlated randomness for small k.

of space and proofs of stake. In the latter one must consider old keys that hold no stake in the current chain, but still can be used to bootstrap from a past block. To address this it was suggested to have honest parties use key-evolution schemes [BGK+18] so the current keys cannot be used to create blocks in the past. Key-evolution is problematic as it's clearly not rational for honest parties to do; they could sell their keys or lose their stake in case of a deep reorg.

The essence of the bootstrapping problem is the fact that one cannot ensure that time has passed in-between the creation of subsequent blocks. Chia solves this problem by combining proofs of space with *proofs of time*, concretely, verifiable delay functions (VDFs), which enforce that some inherently sequential computation (which requires time linear in the length of the computation) was performed in-between the creation of blocks. Chia combines those three countermeasures (splitting the chain, correlated randomness and proofs of time) into a single design which is secure if the resources controlled by the honest parties satisfy the bound from Eq.

3 - Rational Attackers

In §2 we discussed how costless simulation opens attack vectors for double spending in longest-chain blockchains and how these are addressed in Chia. To show security we assumed that a sufficient fraction of the resource is controlled by *honest* parties who follow the protocol rules. In reality it's unrealistic to assume that parties will behave altruistically, instead we need to argue that it's *rational* for parties to follow the protocol rules. Unfortunately costless simulation also makes this task much more challenging than in a PoW based system.

In analogy to *selfish mining* in Bitcoin, we refer to strategies by which a party gets more rewards than they would by following the protocol rules as *selfish farming*. To argue that rational parties will behave honestly it's necessary to bound the efficacy of selfish mining/farming strategies.

In §3.1 below we first discuss selfish mining and why we don't observe it in Bitcoin even though it's possible in principle. As directly analyzing the security of a longest-chain protocol against selfish mining/farming is very challenging we take a modular approach. In §3.2 we first identify two properties – *no slowdown* and *delayed gratification* illustrated in Figure 5 – which are satisfied by Bitcoin, and then show in §3.3 that they imply robustness against selfish mining (through the notion of chain quality) of the level as achieved by Bitcoin. In §3.4 and §3.5 we then sketch how those notions are achieved in Chia.

3.1 Selfish Mining in Bitcoin

While Bitcoin prevents double spending assuming a majority of the hashing power is controlled by miners who altruistically follow the protocol, it allows for *selfish mining* [ES18] by which a miner with a $\alpha < 0.5$ fraction of the hashing power can create more than an α fraction of the blocks and thus gets an unfair share of the block rewards. In some settings this fraction can be as large as $\alpha/(1-\alpha)$ (e.g. a 0.33 fraction for $\alpha = 0.25$).[4] Selfish mining has not been observed in Bitcoin, and there are various reasons why this is the case

1. selfish mining requires either a fairly large fraction of the hashing power or very good control of the network (cf. Footnote [7]) to be profitable
2. the attack would be easily detected and
3. delayed gratification as defined below.

3.2 Delayed Gratification and No Slowdown

The Bitcoin blockchain is split in epochs, each with a targeted duration of two weeks, and only at the end of an epoch the difficulty is reset to accommodate for the variation of the hashing power. Assuming the network is reliable, within an epoch, a selfish miner cannot create more blocks than

[4] To achieve such a large fraction we must assume that (1) honest miners follow the (original Bitcoin) rule and in case they learn of two longest chains they always try to extend the one they saw first and (2) that once the selfish miner learns about a block mined by the honest miners, they can release a withheld block such that their block reaches most of the honest miners faster than this honest block. If either of these conditions is not met, selfish mining is much less profitable, and only becomes profitable at all for selfish miners who control a fairly large fraction of the resource [SSZ15].

they would get by honest mining. This follows from a crucial property of proofs of work: there's no way to find more proofs of a given difficulty (and thus blocks) in a given time window than simply following the protocol and always working on the known longest chain. The only thing selfish mining does in Bitcoin is to make honest parties waste their hashing power, so after the next difficulty reset (which only happens every 2 weeks) the difficulty is lower than it should be, and only at this point the selfish miner makes some extra profit. Another property of PoW based chains like Bitcoin is that an adversary cannot slow down chain growth. We capture these two desirable properties separately below.

Delayed Gratification: A chain where an adversary cannot increase the number of blocks they find in expectation within an epoch of same difficulty by deviating from the honest strategy is said to have the *delayed gratification* property. In Chia, by "not deviating" we mean that the adversary simply runs an honest farmer using its available space, and additionally, should the adversary control VDFs that are faster than the fastest honest time lord, they are also assumed to run a time lord. Intuitively, delayed gratification is a good deterrent to selfish mining by itself as it limits selfish mining to adversaries who follow a "long term" agenda.

No Slowdown: A chain where an adversary (no matter what fraction of the resource they control) cannot slow down the expected block arrival time by interacting with the chain is said to have the *no slowdown* property.

3.3 Chain Quality

A longest-chain blockchain is said to have *chain quality* ρ if the fraction of blocks mined by honest miners is at least ρ (with high probability and considering a sufficiently large number of blocks). Chain quality was introduced in [GKL15] as a metric to quantify how susceptible a chain is to selfish mining. Ideally, assuming an adversarial miner who controls an α fraction of the resource, the chain quality should be $\rho = 1 - \alpha$ as this means that the adversary cannot increase its fraction of blocks by deviating.

By the Proposition below delayed gratification and the no slowdown property imply a bound on *chain quality* which matches the bound proven for Bitcoin (when ignoring network delays).

Proposition 1 (Delayed Gratification and No Slowdown implies Chain Quality). *Consider a longest-chain protocol which has the delayed gratification and no slowdown property against an adversary who controls an α fraction of the global resource, then the chain quality is $1 - \frac{\alpha}{1-\alpha}$ (compared to the ideal $1 - \alpha$).*

Proof. Consider an adversarial miner A with an α fraction of the resource and let ℓ denote the (expected) number of blocks to be found if everyone would mine honestly. By the no slowdown property, no matter what A does the number of blocks found is at least $\ell' \geq (1 - \alpha) \cdot \ell$. By delayed gratification, at most $\alpha \cdot \ell$ of those blocks were created by A, we get a chain quality of

$$\begin{aligned} \textit{chain quality} &= \frac{\text{honest blocks}}{\text{total blocks}} \\ &= \frac{\ell' - \alpha \cdot \ell}{\ell'} \\ &= 1 - \frac{\alpha \cdot \ell}{\ell'} \\ &\geq 1 - \frac{\alpha \cdot \ell}{(1-\alpha) \cdot \ell} \\ &= 1 - \frac{\alpha}{1-\alpha} \end{aligned}$$ □

3.4 Delayed Gratification in Chia

Having motivated why the no slowdown and delayed gratification properties are useful, in this and the next section we will sketch how they are achieved in Chia. Recall that delayed gratification means a selfish farmer cannot add more blocks into the chain than he could by honestly following the protocol. To achieve this in Chia we ensure that

Objective 1: Unpredictable and Immutable Challenges
(a) a challenge is revealed as late as possible.
(b)) once it's revealed, it's almost certainly too late for a selfish farmer to influence it in any way.
(c)) whether a plot can produce a block for a challenge only depends on the plot and the challenge (and not say, on what other plots exist).

These properties imply delayed gratification as a selfish farmer cannot do anything to influence challenges in a controlled way due to (a) & (b), and cannot do anything to increase its number of winning blocks for a given challenge due to (c).

We will sketch how properties (a)-(c) are achieved in Chia next. To follow the arguments the reader might want to recap the high level outline in §1.7 and illustration in Figure 1

(a) The only reason for the infused challenge chain $\mathsf{i}CC$ is to make sure that the challenge becomes known as late as possible, in particular when considering an adversary with a faster VDF than the fastest honest time lord.

(b) We infuse the *first* block of each slot into the challenge chain CC, this way making sure that this block is buried deep in the chain (by 31 blocks on average) once revealed, and thus almost impossible to revert.

(c) We use a variation on the correlated randomness technique from [BDK+19], where we let the challenge depend on every kth challenge on average, rather than exactly. This way only the challenge determines whether a plot can produce a winning block, irrespective of what other plots exist.

3.5 No-Slowdown of Chia and other Constructions

Recall that the no slowdown property requires that no adversary can slow down the block arrival time by participating.

3.5.1 No-Slowdown in Bitcoin

In Bitcoin no slowdown holds as whenever an honest miner finds a block, all the honest miners will switch to a heavier chain. An adversary can still kick out this block and replace it with one of his own (and that's what selfish mining is exploiting), but not slow down the growth. Of course here we assume a reliable network, a network level attacker who can increase the latency or even split the network can of course delay chain growth.

3.5.2 A Non-Example, the G-Greedy-Rule

One might assume that the no-slowdown property would be achieved by any "natural" longest-chain blockchain even if based on efficient proof systems. Unfortunately this intuition is wrong. A design for which no-slowdown fails to hold is the proof of stake based chain of Fan and Zhou [FZ17]. Their chain mimics Bitcoin's Nakamoto consensus using proofs of stake, but to harden the design against (what in this writeup is called) double dipping attacks [FZ17] suggest the miners not only extend the longest chain, but instead follow the "g-greedy rule": a miner should try to extend all forks they see which are at most g blocks shorter than the longest chain they've seen so far. The rationale behind this rule is that by letting the honest miners do double-dipping to some extent, the advantage an adversary can get by double dipping shrinks.[5] As shown in [BDK+19], this design has some serious issues as an adversary with relatively small resources can with high probability prevent the chain reaching consensus by strategically releasing blocks and this way keep two forks alive for a long time. An illustration of their attack is in Figure 5. Interestingly (citing [BDK+19]) *"..the efficacy (of the attack) is primarily achieved by slowing down the growth rate of the honest strategy."*

[5] A similar proposal, where the honest parties try to extend the first k blocks they see at every depth was proposed in an early proposal for Chia (with $k = 3$) [CP19]. This variant achieves the no-slowdown property.

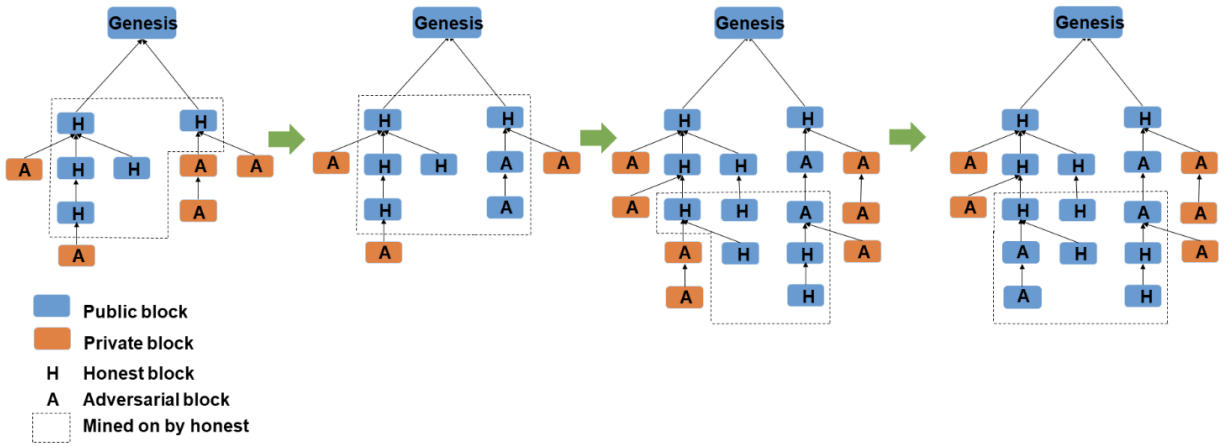

Fig. 6: Snapshot of balance attack in progress on g-greedy protocol with $g = 2$. The goal of the adversary is to balance the lengths of two chains each tracing the genesis block as their ancestor, consequently creating a deep fork.

Figure 5: Illustration of the balancing attack against [FZ17] taken from [BDK+19]

3.5.3 Examples of No-Slowdown.

The lesson from the example above is that the no-slowdown property is not easy to achieve in longest-chain protocols using efficient proof systems. Moreover the absence of this property can lead to various security issues, not just selfish mining opportunities, but even prevent consensus almost indefinitely as in the example above. Similar attacks were also proposed for BFT type protocol, most notably Ethereum [SNM+21].

Naïve Emulation of Bitcoin

There are longest-chain blockchains from efficient proof systems which do have the no-slowdown property. The simplest is to just emulate Bitcoin by replacing PoW with PoSpace or PoStake. While satisfying no-slowdown, this basic construction has all the security issues discussed in §2. Fixing bootstrapping and grinding as outlined in §2 will preserve the no-slowdown property. The challenge is to find a good countermeasure to double-dipping without losing the no-slowdown property and introducing new attack vectors.

D-Distance Greedy

Bagaria et al. [BDK+19] not only prove that g-greedy does not have the no-slowdown property, but also suggest a different rule of a similar flavour they call "D-distance greedy", for which the no-slowdown property does hold [BDK+19 Lemma 12]. This rule reduces the double-dipping advantage factor towards 1 as D increases, but already for moderately large D it becomes computationally infeasible for the miners to even determine which chain to follow.

Old Chia

The first Chia greenpaper [CP19] has a very simple rule where honest farmers try to extend the first $k > 1$ ($k = 3$ was suggested) chain of any given length they become aware of. For this simple construction the advantage factor of double-dipping goes to 1 as k increases while it does achieve no-slowdown [CP19 Lemma 4].

Chia

While the deployed Chia blockchain has many advantages over the old [CP19] proposal, the no-slowdown property is a much tricker issue in the new design. In particular, we do not yet have an analogue of [CP19 Lemma 4] which basically states that even an unbounded adversary (unlimited space, unlimited number of arbitrary fast VDFs) cannot slow down chain growth.

When analyzing the no-slowdown property, it is useful to distinguish the specification of the chain (i.e., what constitutes a valid chain) and its chain selection rule (aka. fork choice rule), which tells the farmers and time lords on which chains to work should competing forks exist.

For example the difference in the g-greedy and the D-distance greedy protocols discussed above (only the latter having the no-slowdown property) is only in the chain selection rule, the specification what constitutes an valid chain is the same.

Unlike the chain specification, which can only be changed by a hard fork once the chain is deployed, the chain selection rule can easily be adapted by the farmers and/or time lords even after the launch. Finding a chain-selection rule for the Chia chain which provably achieves no-slowdown is an interesting open problem.

Under the additional assumption that an adversary does not control VDFs which are *faster* than the fastest honest time lord, a very simple chain selection rule achieving no-slowdown exists: always follow the chain with accumulated most VDF steps. Of course this rule would be terrible in practice as security completely breaks if the adversary has an even slightly faster VDF than the fastest honest time lord. For example, such an adversary could create an "empty" chain by refusing to infuse any blocks.

A more sensible rule is to simply follow the *heaviest* fork like in Bitcoin. Unfortunately, unlike in Bitcoin, in Chia the heaviest fork is not necessarily the fork which will be heaviest in the future assuming all honest parties adapt it: a fork A might have one more block infused than some fork B, but if B is way ahead in the VDF computation extending B might give a better chain (in expectation) in the future. Thus, when using this rule, by releasing B an adversary might slow down the chain. The currently deployed chain selection rule for farmers and time lords is basically to follow the heaviest fork, but with some heuristics to avoid clear cases where switching to a heavier chain is slowing down growth.

4 - Hash and VDF chains

A key ingredient in longest-chain blockchains are hash-chains as discussed in §4.1 below. While Chia also uses a hash-chain (for the foliage chain FC), for Chia we use a new chaining structure called a VDF chain defined in §4.2 below.

4.1 Hash chains

For this writeup, a *hash chain* is a sequence $b_0, b_1, b_2 \ldots$ of blocks, where each block $b_i = \{h_i, x_i\}$ contains some data value x_i (possibly empty) and (with the exception of b_0) a hash value of the current data and the previous block.

$$h_i := \mathsf{H}(b_{i-1}, x_i)$$

Security from hash chains.

A hash chain is immutable in the following sense:

Proposition 2 (immutability of hash chains). *If* H *is a collision-resistant hash function, then it is computationally infeasible to find two distinct hash chains* $H = b_0, \ldots b_i$ *and* $H' = b'_0, \ldots, b'_j$ *where* $h_i = h'_j$ *and no chain is a prefix of the other (which holds if they start with the same* $x_0 = x'_0$*).*

4.2 VDF chains

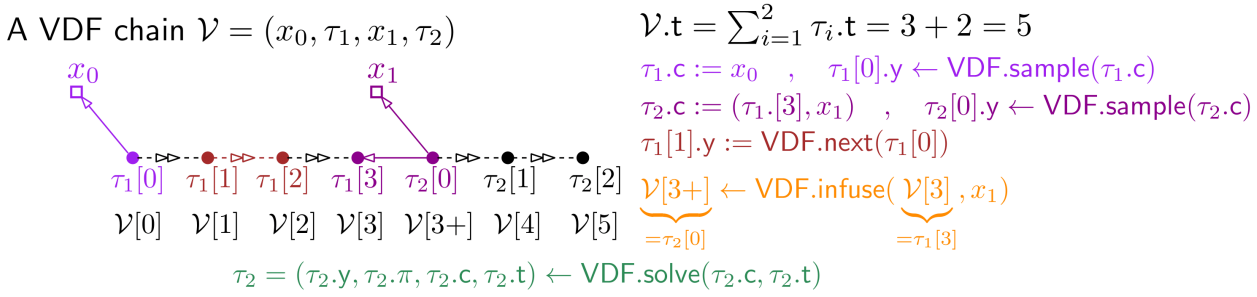

Figure 6: Illustration of a VDF chain.

A VDF chain is a sequence

$$V = z_0, \tau_1, z_1, \tau_2, z_2, \ldots, \tau_\ell \tag{5}$$

alternating data values $z_i \in \{0,1\}^*$ and VDF values $\tau_i = (\tau_i.\mathsf{y}, \tau_i.\pi, \tau_i.\mathsf{c}, \tau_i.\mathsf{t})$ (as described in §A.3). The chain is valid if all VDF proofs are correct

$$\mathsf{VDF.verify}(\tau_i) = \mathsf{accept}$$

and the challenge for the ith VDF is derived from the previous VDF output (except for $i = 1$) and data value

$$\tau_1.c := \mathsf{VDF.sample}(z_0) \quad \text{and} \quad \forall i > 1 : \tau_i.\mathsf{c} := \mathsf{VDF.sample}(\tau_{i-1}.\mathsf{y}, z_{i-1})$$

where we use the convention that $\tau_0.\mathsf{y}$ is the empty string.

4.2.1 Notation for VDF chains

We naturally extend the notion for VDFs as described in §A.3 to VDF chains. The *total number of VDF steps in a VDF chain* as in eq.(5) is simply the sum of the steps in its VDFs

$$V.\mathsf{t} \stackrel{\mathsf{def}}{=} \sum_{i=1}^{\ell} \tau_i.\mathsf{t}$$

Security from VDF chains.

VDF chains give two basic security guarantees, the first is immutability analogous to hash chains, but also sequentiality inherited from the underlying VDF, we discuss them shortly in more detail.

Proposition 3 (immutability and sequentiality of VDF chains). *Like a hash chain, a VDF chain is* immutable* in the sense that it's computationally infeasible to come up with two different VDF chains*

$$V = z_0, \tau_1, z_1, \tau_2, z_2, \ldots, \tau_\ell \qquad V' = z'_0, \tau'_1, z'_1, \tau'_2, z'_2, \ldots, \tau'_{\ell'}$$

where the last VDF outputs collide, i.e., $\tau_\ell.\mathsf{y} = \tau'_{\ell'}.\mathsf{y}$. Here different means that either they have different length $\ell \neq \ell'$ and neither is a prefix of the other. Or (if $\ell = \ell'$) there exists an i s.t. either $z_i \neq z'_i$ or $\tau_i.\mathsf{y} \neq \tau'_i.\mathsf{y}$ or $\tau.\mathsf{t} \neq \tau'.\mathsf{t}$. Note that we ignore the proofs $\tau.\pi$ when comparing chains (we just use them to determine whether the chain is valid) as they must not be unique.

Moreover a VDF chain is *sequential*, meaning that not only the individual VDFs must be computed sequentially (which follows from the security definition of VDFs), but also the VDFs in the chain were computed sequentially. I.e., computing a chain V as above requires $\sum_{i=1}^{\ell} \tau_i.\mathsf{t}$ sequential steps.

---

5 - The Chia Blockchain

In this section we finally outline the design of the Chia blockchain as illustrated in Figure 1 from its basic building blocks PoSpace, VDFs and Signatures. These primitives are specified in §A. We'll use greek letters σ to denote PoSpace, τ for VDFs and μ for Signatures.

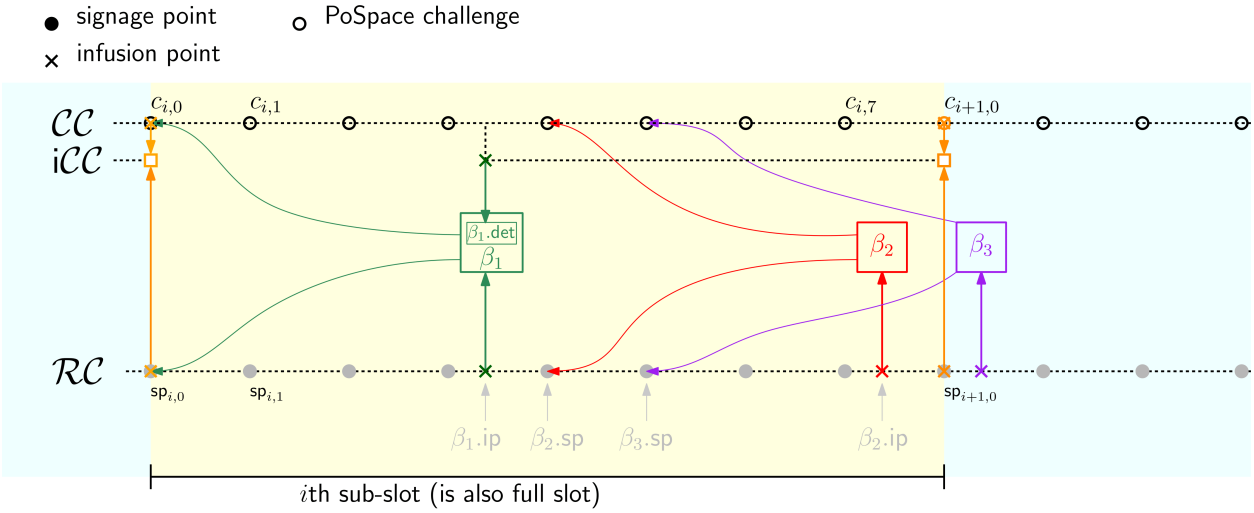

Figure 7: The reward, challenge and infused challenge chains. For illustration we only use 8 not 64 signage points per sub-slot.

5.1 Additional Variables and Notation for this Section

5.1.1 Variables

Variable	Definition
T_i	Time parameter of i-th slot (# of VDF steps per sub-slot). Recalibrated once per day for 10 minutes per sub-slot target.
spi_i	$\mathsf{spi}_i \stackrel{\text{def}}{=} \frac{T_i}{64}$
D_i	Difficulty parameter of i-th slot. Recalibrated once per day for 32 blocks per slot target

5.1.2 Step to Epoch

κ_i :Number of sub-slots in ith slot. Typically $\kappa_i = 1$ but can be larger integer to enforce a 16 block minimum.

5.1.3 Notation for Points of Interest

To describe the Chia chains it will be convenient to introduce some extra notation. Recall that for a VDF τ or VDF chain V we denote with $\tau[t]$ or $V[t]$ the point t steps into the computation. $\tau.\mathsf{t}, V.\mathsf{t}$ is the total number of steps. Sometimes we overload notation and consider τ, V to denote the point at the end of the computation rather than the entire VDF or VDF chain, i.e., $\tau = \tau[\tau.\mathsf{t}], V = V[V.\mathsf{t}]$.

The VDF chains we'll consider (RC and CC) will be split into slots where the starting point of a new slot will always be an infusion point. For a point $\mathsf{point} = V[t]$ on such a chain we denote with

Point	Definition
$\mathsf{point.D}$	the total depth, i.e., the number of steps of this point since genesis
$\mathsf{point.d}$	the depth of this point in the current slot
$\mathsf{point.t}$	the depth of this point in its VDF
$\mathsf{point.x}$	the value of the VDF chain at this point
$\mathsf{point.}\pi$	a proof certifying the VDF computation up to this point
point^+	If point is an infusion point where some value v gets infused, then we denote with point the point before infusion, and with $$\mathsf{point}^+ = \mathsf{VDF.sample}(\mathsf{point.x}, v)$$ the point after infusion

The following points on the Chia VDF chains will be defined

Point	Definition
$\mathsf{cc\_sp}_{i,j}$	The jth **challenge chain signage point** in the ith slot (eq.(6))
$\mathsf{rc\_sp}_{i,j}$	The jth **reward chain signage point** in the ith slot (eq.(9))
$\mathsf{cc\_sp}(\beta)$	The $\mathsf{cc\_sp}_{i,j}$ used as challenge to compute the PoSpace σ in block β
$\mathsf{rc\_sp}(\beta)$	The $\mathsf{rc\_sp}_{i,j}$ whose signature $\mu_{\mathsf{rc\_sp}}$ is in β
$\mathsf{rc\_ip}(\beta)$	The infusion point of β into RC (eq.(10)

The *signage point interval* is the number of VDF steps between signage points, for the ith slot it's

$$\mathsf{spi}_i \stackrel{\mathsf{def}}{=} T_i/64$$

so e.g., the depth of the jth signage point in the ith slot is $\mathsf{rc\_sp}_{i,j}.\mathsf{d} = \mathsf{cc\_sp}_{i,j}.\mathsf{d} = j \cdot \mathsf{spi}_i$. A block in the ith slot will be infused 3 to 4 signage point intervals past its signage point.

$$3 \cdot \mathsf{spi}_i \;<\; \mathsf{rc\_ip}(\beta_T).\mathsf{d} \;-\; \mathsf{rc\_sp}(\beta_T).\mathsf{d} \;<\; 4 \cdot \mathsf{spi}_i$$

The first signage point in a slot is the last signage point after infusion

$$\mathsf{rc\_sp}_{i+1,0} = \mathsf{rc\_sp}^{+}_{i,\tau_i \cdot 64}$$

The ith slot starts at total depth

$$\mathsf{rc\_sp}_{i,0}.\mathsf{D} = \sum_{j=1}^{i-1} T_j \cdot \kappa_j$$

5.2 The Challenge Chain

The challenge chain CC is a VDF chain whose data and VDF values we'll denote as

$$CC = \mathsf{ic}_0, \tau_1^{CC}, \mathsf{ic}_1, \tau_2^{CC}, \mathsf{ic}_2, ...$$

Here τ_i^{CC} is the VDF computation for the ith slot. Usually its number of VDF steps is the current time parameter T_i (and should take 10 minutes to compute), but in exceptional cases it can be an integer multiple $\kappa_i \in \mathbb{N}$ of that as we enforce a 16 block minimum per slot

$$\tau_i^{CC}.\mathsf{t} = \kappa_i \cdot T_i \qquad \text{typically } \kappa_i = 1$$

The value ic_i infused at the beginning of slot $i+1$ depends on the first block in slot i, we'll explain how exactly in §5.5.

As the VDF τ_i^{CC} of the ith slot is computed by a time lord, they release equidistant points of this computation called the *challenge chain signage points*, one every $\mathsf{spi}_i = T_i/64$ VDF steps or around $9.375 = 600/64$ seconds

$$\mathsf{cc\_sp}_{i,0}, \mathsf{cc\_sp}_{i,1}, \dots, \mathsf{cc\_sp}_{i,\tau_i \cdot 64} \quad \text{where} \quad \mathsf{cc\_sp}_{i,j} \overset{\mathsf{def}}{=} \tau_i^{CC}[j \cdot \mathsf{spi}_i] \tag{6}$$

The point $\mathsf{cc\_sp}_{i,\tau_i \cdot 64}$ at the end of the slot must also be broadcast as it's required to verify the VDF, but it's not used as a challenge as it's at the same depth as the first signage point $\mathsf{cc\_sp}_{i+1,0} \leftarrow \mathsf{VDF.sample}(\mathsf{cc\_sp}_{i,\tau_i \cdot 64}, \mathsf{ic}_i)$ of the next slot.

Design Choice 1: A Continuous Flow of Challenges
To get the security gains of correlated randomness, we let our PoSpace challenges depend on only one block (out of around 32) per slot, so there's a fresh challenge every 10 minutes. At the same time, we want a smooth continuous block arrival time (target is 18.75 seconds) and the challenge for each block should be revealed around 30 seconds before the block is

infused (not much less to avoid orphan blocks, not much more to limit selfish mining and bribing opportunities). Deriving challenges $\mathsf{cc\_sp}_{i,1}, \mathsf{cc\_sp}_{i,2}, \ldots$ deterministically from one initial challenge $\mathsf{cc\_sp}_{i,0}$ using a delay function as outline above achieves exactly that.

The reward chain RC is a VDF chain that the time lords evaluate in parallel to CC and also has signage points at the same depth as CC, i.e., $\mathsf{rc\_sp}_{i,j}.\mathsf{d} = \mathsf{cc\_sp}_{i,j}.\mathsf{d}$. Before we can define RC we first need to explain the content of blocks.

5.3 Trunk Blocks

Whenever a farmer receives new signage points $\mathsf{cc\_sp}_{i,j}, \mathsf{rc\_sp}_{i,j}$ they first check whether this points lie on a heaviest chain (cf. the discussion in §1.5) and their VDF proofs verify. If the this is the case, the farmer checks they can create a winning PoSpace proof. This process will, for a subset of the plots, produce a PoSpace σ and some additional value $\sigma.\mathsf{required\_iterations}$. Whether this PoSpace is a winning proof is now determined by the time parameter T_i as

$$\text{winning condition} : \sigma.\mathsf{required\_iterations} < \mathsf{spi}_i \quad (= T_i/64) \tag{7}$$

Design Choice 2: Why 32 Blocks in Expectation and not Exactly?
With our winning condition we have 32 blocks per slot *in expectation* depending on a challenge. We could have used a different design to enforce *exactly* 32 challenges, but then it would be impossible to achieve our Objective 1.(c), which asks that whether a plot wins must depend solely on the challenge.

If a farmer has a winning PoSpace σ they can produce a block $\beta = (\beta_T, \beta_F)$ which contains the foliage block β_F and the trunk block β_T. The actual Chia blocks are more sophisticated than our description below, but in this writeup we focus on the entries which are absolutely necessary for functionality and security of the chain and ignore entries which are there for efficiency like weight proofs for light clients or pooling. They key entries in a valid trunk block

$$\beta_T = (\sigma, \mu_{\mathsf{rc\_sp}})$$

are

$\sigma \leftarrow \mathsf{PoSpace.prove}(S, \mathsf{cc\_sp}_{i,j})$, a proof of space for some plot S on challenge $\mathsf{cc\_sp}_{i,j}$ where the proof σ satisfies the winning condition from eq.(7).

$\mu_{\mathsf{rc\_sp}} \leftarrow \mathsf{Sig.sign}(S.sk, \mathsf{rc\_sp}_{i,j})$, a signature using the secret key of the plot S (so it verifies under the public-key $\sigma.pk$ in the PoSpace) of the signage point in the rewards chain discussed in the next section.

5.4 The Reward Chain

The reward chain RC is a VDF chain that time lords compute in parallel to CC. Like CC, RC can be spilt in a sequence of slots.

$$RC = RC_1, RC_2, ...$$

While in CC the ith slot just contains a VDF τ_i^{CC} and the value ic_i infused at the end, each slot RC_i of the RC chain

$$RC_i = \tau_{i,1}^{RC}, \beta_1, \tau_{i,2}^{RC}, \beta_2 \ldots, \beta_{b_i} \tau_{i,b_i+1}^{RC}, (\mathsf{ic}_i, \tau_i^{CC}.\mathsf{y}) \tag{8}$$

is a VDF chain with typically around 33 infused values: around 32 blocks b_i and at the end of the slot also the CC and iCC points at the same depth. The RC signage points are

$$\mathsf{rc\_sp}_{i,j} \stackrel{\mathsf{def}}{=} RC_i[j \cdot \mathsf{spi}_i] \tag{9}$$

Where do Blocks get Infused.

Let $\beta_T = (\sigma, \mu_{\mathsf{rc\_sp}})$ be some valid block for challenge $\mathsf{cc\_sp}(\beta_T)$, its reward chain infusion point $\mathsf{rc\_ip}(\beta_T)$ is then at depth

$$\mathsf{rc\_ip}(\beta_T).\mathsf{d} = \mathsf{rc\_sp}(\beta_T).\mathsf{d} + 3 \cdot \mathsf{spi}_i + \sigma.\mathsf{required\_iterations} \tag{10}$$

As $\sigma.\mathsf{required\_iterations}$ is at most spi_i the infusion point is somewhere between 3 and 4 signage points past the signage point it refers to. That means we have somewhere from 28.125 to 37.5 seconds for a round trip from the time lord who gossips the signage point, to a farmer who computes and gossips a block, back to the time lord who then infuses the block.

5.5 The Infused Challenge Chain

Recall that the challenge chain CC is used to create PoSpace challenges, and we want these challenges to only depend on one block per slot. For this, at the end of the ith slot we infuse the PoSpace σ from the first trunk block β_T whose signage point is in the ith slot into CC. We don't simply infuse σ, but to delay revealing the challenge for as long as possible and make sure it's buried deep in the chain when revealed, we run a VDF on top of σ to get the infused challenge value ic_i to be infused as defined in §5.2.

Concretely, the infused challenge of the ith slot is the output of a VDF computation

$$\mathsf{ic}_i \stackrel{\mathsf{def}}{=} \tau.\mathsf{y} \qquad \tau = \mathsf{VDF.solve}(x, t)$$

on some challenge x and time t which are defined as follows.

Let $\beta_T = (\sigma, \mu_{\mathsf{rc\_sp}})$ be the first trunk block infused into the ith slot RC_i past the 3rd signage point,

using notion as in eq.(10)

$$\beta_T = \beta_j \text{ where } j = \min\{k \ : \ \beta_k.\mathsf{d} > 3 \cdot \mathsf{spi}\}$$

now the challenge x is derived from the PoSpace in this block and the value of CC at the depth of its infusion point

$$x \leftarrow \mathsf{VDF.sample}(\sigma, CC[\mathsf{rc\_ip}(\beta_T).\mathsf{D}].\mathsf{y})$$

the number of steps t is the the remaining number of VDF steps in the slot, so the value ic_i will be available at the end of the slot when it's required, but not earlier

$$t = \mathsf{cc\_ip}_{i,0}.\mathsf{D} - \mathsf{rc\_ip}(\beta).\mathsf{D}$$

Security Notice 1: Why iCC depends only on σ
We only use σ, not the entire trunk block $\beta_T = (\sigma, \mu_{\mathsf{rc\_sp}})$, to compute the infused challenge ic_i. This is crucial to ensure that the challenges depend only on a single challenge per slot. Had we infused the entire β_T (as we do into RC), the challenges would depend on all blocks (as $\mu_{\mathsf{rc\_sp}}$ depends on RC which infuses all blocks) and we would not get security against double dipping.

Design Choice 3: Why Infusing the First Block?
Recall that by our Objective 1.(a) we want challenges to be only revealed when necessary and (b) to be immutable once revealed.
While (b) suggest to infuse the first possible block so it's buried once revealed, for (a) it would be better to use the latest possible block. We go with the first block to achieve (b), and by running a VDF on top of the block we also achieve objective (a).

Design Choice 4: Upper and Lower Bounds on Blocks per Slot
The target number of blocks per slot is 32, and there's an upper bound of 64 and lower bound of 16. These bounds make some attacks more difficult. In normal deployment the number of blocks will be close to its expectation, so these bounds should basically never be reached. The lower bound is required to bound the efficacy of double-dipping as sketched in §2.2, while the upper bound is necessary to prevent replotting attacks as explained in §1.8.3.

Objective 2: The Trunk is (almost) Ungrindeable
To prevent grinding, the only decision that influences the trunk should be whether to add a block or not. We need one exception to this rule: the time-stamps (in blocks in the foliage) are used to recalibrate the time parameter which determines the numbers of VDFs steps per slot in the trunk. This gives a minor grinding opportunity, to further limit the usability of

this for attacks, the window used to calibrate the time parameter is not simply the previous epoch, but shifted XXX back so the relevant time-stamp is already buried deep in the chain.

Objective 3: CC (almost) only Depends on the First Block
To limit the impact of double-dipping we use correlated randomness [BDK+19]). For this the challenge chain CC should only depend on the first block in every slot. If this was the case, this would allow for long-range replotting attacks. For this reason, once every sub-epoch (approx 2h) the rewards chain is infused to the challenge chain.

5.6 The Foliage

Whenever a farmer finds a winning PoSpace they can create a block $\beta = \{\beta_F, \beta_T\}$ which contains the trunk block β_T as discussed above, and a foliage block

$$\beta_F = \{\mathsf{data}, \mu_F\}$$

which contains the payload of the block data (transactions, a time-stamp) and a signature (using the key $S.sk$ as for $mu_{\mathsf{rc\_sp}}$, cf.§5.3})

$$\mu_F \leftarrow \mathsf{Sig.sign}(S.sk, (\beta_T, \beta'_F))$$

that links this foliage block to the chain. It signs the (hashes of the) current trunk block β_T as well as the foliage block β'_F from the last transaction block as discussed below.

While *every* valid trunk block will typically be infused into RC (unless the time lord is malicious, the block arrives too late to be infused or the slot is already at its 64 block upper limit), only a subset of the foliage blocks are included in the foliage chain FC for reasons outlined below:

Objective 4: Block Arrival vs. Creation Time
We want blocks to arrive at a rather high frequency (9.375 seconds on average) to achieve fast confirmation. At the same time we want to give sufficient time (28.125 to 37.5 seconds between signage and infusion points of a block) for block creation to prevent oprhan blocks.

Objective 5: Sequential Transaction Blocks
Every block of transactions added must refer to the previous block of transactions. This way we avoid having to deal with transactions that are invalid due to previous transactions that were added but were not known to the creator of the current transaction block.

Design Choice 5: Foliage
To achieve the above objectives, we let farmers who found a winning PoSpace and can create a block always create a foliage block referring to the last transaction block before the signage

point of their block. The time lord will add the foliage of a block – and thus make this block a *transaction block* – if no other transaction block was infused between the signage and infusion point of that block.

5.7 Fraction of Transaction Blocks

With the above rule, we expect one transaction block every $\left(\frac{1}{e^{0.5}-1}+4\right) \approx 5.54$ slots, that's one every 51.95 seconds and corresponds to 36% of all blocks.

For the interested reader, let us shortly outline how the above is determined. The signage points of consecutive transactions blocks must be at least 4 points apart as a block with signage point $\mathsf{rc\_sp}_i$ is infused somewhere between $\mathsf{rc\_sp}_{i+3}$ and $\mathsf{rc\_sp}_{i+4}$. The gap can be bigger than 4 points if no block is found in response to $\mathsf{rc\_sp}_{i+4}$ and potentially more points after that. The expected number of blocks found for each slot is 0.5 (32 block target for 64 points). The number of blocks found for each slot is Poisson distributed with expectation 0.5 (32 block target for 64 points). With this distribution, the expected number of consecutive points with no blocks is $\frac{1}{e^{0.5}-1}$.

6 - Recovering from 51% Attacks and Dynamic Availability

In this Section we have a look at two closely related security properties of longest-chain blockchains, namely recovery from malicious majority (aka. 51% attacks) and security under dynamic availability. We'll discuss proofs of work, stake and space, for the latter two also looking at how adding VDFs changes the picture.

We discussed in §2 the main security issues of a PoSpace based longest-chain blockchain arise from the fact that PoSpace is an efficient proof system. PoSpace shares those security challenges with PoStake, and all three countermeasures summarized in Figure 3 (namely splitting the chain to prevent grinding, correlated randomness to prevent double-dipping and using VDFs to prevent bootstrapping) can readily be applied in the stake setting, correlated randomness was even originally proposed for stake [BDK+19]. But as we'll discuss below, when it comes to security under dynamic availability or 51% attacks there are fundamental differences between space and stake. In particular, using proofs of space in combination with VDFs one can handle both attacks basically as well as with proofs of work, while proofs of stake cannot, even in combination with VDFs.

	Recovery from 51% Attacks	Security under Dynamic Availability
Proof of **Work** Bitcoin	Yes	Yes
Proof of **Space** Spacemint [PKF+18]	No	No [BP22]
Proof of **Space & Time** Chia	Yes	Yes
Proof of **Stake** Ouroboros Genesis [BGK+18] Sleepy Consensus [PS17]	No	Yes, but requires careful modelling
Proof of **Stake & Time** PoSaT [DKT21]	No	Yes, but requires careful modelling

Figure 8: Table 1: Summary of the ability to heal from malicious majority and provide security under dynamic availability of longest-chain protocols based various proof systems.

6.1 Recovery from 51% Attacks

A key difference between a PoW based longest-chain protocol and a longest- chain protocol based on an efficient proof system like PoStake or PoSpace is the fact that only the PoW based chains is guaranteed to recover security once an adversary that controls a sufficiently large fraction of the resource, even if it's just for a short period. This is sometimes called "a 51% attack" referring to

the fact that in bitcoin an adversary controlling $> 50\%$ of the hashing power can break security in pretty much any way they like (they can double spend, get 100% of the block rewards or censor). We'll stick with this expression even though the fraction of the resource required to control a chain can be lower than 50% (as mentioned in §1.1, in Chia controlling 43% of the space is sufficient).

There's also a key difference between PoStake and PoSpace. By using VDFs in addition to PoSpace as in Chia we get a chain that does have this self- healing property. While we can also augment a PoStake based chain with VDFs [DKT21], the resulting chain will not be self-healing.

6.1.1 Recovering from PoW Majority in Bitcoin

While Bitcoin provides no security if more than half of the hashrate is controlled by an adversary, it is "self-healing" in the sense that once the majority of the hashrate is again controlled by honest parties, Bitcoin regains (after some delay) all its security properties.

A bit more formally, let $\mathsf{PoW}_h(t)$ and $\mathsf{PoW}_a(t)$ denote the hashing power of the honest and adversarial parties at clock time t, respectively. For $t_0 < t_1$ let

$$\mathsf{PoW}_h(t_0, t_1) = \int_{t_0}^{t_1} \mathsf{PoW}_h(t)\, dt \quad , \quad \mathsf{PoW}_a(t_0, t_1) = \int_{t_0}^{t_1} \mathsf{PoW}_a(t)\, dt$$

denote the cumulated hashing power in the time window from t_0 to t_1. If D is the difficulty in this time window,[6] then the expected number of blocks found by the honest parties in this time window is $\mathsf{PoW}_h(t_0, t_1)/D$.

It's instructive to understand when a transaction is trivially insecure: consider a transaction that is contained in a block attached at time t, if one waits for k blocks on top before considering the transaction confirmed (for Bitcoin $k = 6$ is often suggested), then an adversary can fork the chain in order to double spend this transaction with good probability if for some t_0, t_1 with $t_0 \leq t < t_1$ we have

$$\mathsf{PoW}_a(t_0, t_1) \geq \mathsf{PoW}_h(t_0, t_1) \quad \text{and} \quad \mathsf{PoW}_h(t, t_1)/D \geq k \tag{11}$$

If this holds the adversary can simply start at time t_0 to mine a chain in private, and release it at time t_1. By the first inequality the adversaries chain will be heavier than the honest one with probability at least 0.5, and by the 2nd the honest block added at time t will be buried by k blocks with probability 0.5, so both hold and we have a successful double spending attack with probability at least $\approx 0.5^2 = 0.25$ (it can actually be a bit less than that as the two events are negatively correlated).

To be secure it's not sufficient that no t_0, t_1 as in eq.(11) exist, but one needs to be "sufficiently far" from this situation to guarantee that double spending can only happen with some tiny probability. From the standard Chernoff bound it follows that the probability that a fork starting at a block added at time t_0 and being released at time t_1 will be successful (i.e., have higher weight than the

[6] If there's an epoch switch at some $t, t_0 < t < t_1$ where the difficulty switches from D_0 to D_1, let D be the weighted average $D = D_0 \frac{t - t_0}{t_1 - t_0} + D_1 \frac{t_1 - t}{t_1 - t_0}$.

honest chain) is exponentially small in the number of expected honest blocks $\mathsf{PoW}_h(t_0, t_1)/D$ and the square of the honest to adversarial advantage, i.e.,

$$\begin{aligned}&\Pr\left[\text{fork starting at } t_0 \text{ and released at } t_1 \text{ heavier than honest chain}\right]\\ &\leq -\exp\left(\frac{\mathsf{PoW}_h(t_0,t_1)}{D}\cdot\left(\frac{\mathsf{PoW}_h(t_0,t_1)}{\mathsf{PoW}_a(t_0,t_1)}-1\right)^2\right)\end{aligned} \tag{12}$$

6.1.2 Recovering from PoStake Majority

This is in stark contrast to PoStake based longest-chain protocols, where once an adversary gets hold of keys controlling a sufficiently large amount of stake, security cannot be recovered by the honest parties without resorting on some external mechanism. The reason is bootstrapping as discussed in §2.3: an adversary who holds keys which at some point in the chain controlled stake N, can fork at that point and bootstrap a chain to the present that looks as if they had N stake throughout. The issue is aggravated due to "stake-bleeding" [GKR18], which refers to the fact that the fork can amass additional stake through fees and block-rewards.

6.1.3 Recovering from PoSpace Majority

A longest-chain protocol using only PoSpace (like Spacemint [PKF+18]) is basically as bad as PoStake based protocols when it comes to healing after an adversary got control of a large amount of the resource. One difference is that in the PoStake case the bootstrapping is only possible while the adversary holds the space resource, while bootstrapping in PoStake just requires keys that were valid at some point in the past but can be worthless (i.e., not hold any stake in the chain currently considered by the honest parties) now. On the positive side, stake-bleeding is not an issues for PoSpace.

6.1.4 Recovering from Space-Time Majority in Chia

While a pure PoSpace based longest-chain protocol fails to heal from adversarial majority due to bootstrapping, by combining space with time as in Chia we prevent bootstrapping, and get a chain that naturally heals from adversarial majority. Though, what exactly constitutes the resource in a PoST protocol is less obvious than e.g. in the PoW or PoSpace setting. We already shortly touched this issue in §1.1. We'll now recap the notion for PoST resources introduced there, but in a more fine-grained manner using a time parameter to reflect that resources can change over time. Let

Term	Definition
$space_h(t), spa$	denote the disk space (more precisely, the space with initialised plots) available to the honest and adversarial parties at time t, respectively
$vdf_h(t)$	denote the speed of the (three) VDFs available to the fastest honest and online time lord at time t
$vdf_a(t)$	denote the speed of the VDFs available to the adversary, the number of VDFs available to the adversary is unbounded.

Security Notice 2: Unlimited VDFs
vdf_h only considers the fastest honest time lord, as only they matter for the growth of the honest chain. The adversary on the other hand is allowed an unlimited number of VDFs of speed vdf_a. Not putting any bound here makes the security statements stronger, but it might seem to give the adversary an unrealistic advantage. This is not really the case as most of the advantage an adversary can get by using many VDFs (trough double dipping) can already be achieved by using a fairly low amount of VDFs. So it would hardly make a quantitative difference if we put a cap on the number of VDFs, say 100, or simply put no cap at all.

Define the honest and adversarial resource at time time as the product of their space and VDF speed

$$\mathsf{PoST}_h(t) = space_h(t) \cdot vdf_h(t) \quad , \quad \mathsf{PoST}_a(t) = space_a(t) \cdot vdf_a(t)$$

and analogously to the work setting let the cumulative space-time resource in a window from t_0 to t_1 be

$$\mathsf{PoST}_h(t_0, t_1) = \int_{t_0}^{t_1} \mathsf{PoST}_h(t)\, dt \quad , \quad \mathsf{PoST}_a(t_0, t_1) = \int_{t_0}^{t_1} \mathsf{PoST}_a(t)\, dt$$

With these definitions we now get a similar bound on the probability that an adversary can create a fork starting at t_0 and being released at time t_1 as we did for PoW in eq.(12)

$$\begin{aligned} &\Pr\left[\text{fork starting at } t_0 \text{ and released at } t_1 \text{ heavier than honest chain}\right] \\ &\leq -\exp\left(\frac{\mathsf{PoST}_h(t_0, t_1)}{D} \cdot \left(\frac{\mathsf{PoST}_h(t_0, t_1)}{1.47 \cdot \mathsf{PoST}_a(t_0, t_1)} - 1\right)^2\right) \end{aligned} \tag{13}$$

A difference to the PoW setting is the additional 1.47 factor boosting the adversary's resource which is necessary to account for the fact that they can do some bounded double dipping.

Analogously to the PoW case, a block added at time t can be considered secure even in a setting where the adversary can get temporary majority as long as for all $t_0, t_1, t_0 < t < t_1$ where $t_1 - t$ is large enough for the block added at time t to be considered confirmed at time t_1, the probability in eq.(13) is small.

6.2 Dynamic Availability

A blockchain based on some resource is *secure under dynamic availability* if it's security properties hold even if the amount of the resource dedicated towards securing the chain varies over time as long as at any point in time the honest parties control sufficiently more of the resource than a potential adversary.

6.2.1 Dynamic Availability for PoW (Bitcoin)

Using notation from §6.1.1, for a PoW based chain that means that for some $f > 1$ (that captures the advantage of the honest parties) and any time t we have

$$\mathsf{PoW}_a(t) \leq f \cdot \mathsf{PoW}_h(t) \tag{14}$$

To see that Bitcoin is secure under dynamic availability we can reuse our inequality eq.(12) which using $\frac{\mathsf{PoW}_h(t_0,t_1)}{\mathsf{PoW}_a(t_0,t_1)} \geq f$ simplifies to (recall that $\frac{\mathsf{PoW}_h(t_0,t_1)}{D}$ is the expected number of honest blocks in the t_0 to t_1 window)

$$\begin{aligned}&\Pr\left[\text{fork starting at } t_0 \text{ and released at } t_1 \text{ heavier than honest chain}\right]\\ &\leq -\exp\left(\frac{\mathsf{PoW}_h(t_0,t_1)}{D}\cdot(f-1)^2\right)\end{aligned} \tag{15}$$

Which simply means that the probability that an adversary will be able to create any particular a fork decreases exponentially in the length of the fork.

6.2.2 Dynamic Availability for PoST (Chia)

Analogously to PoW just outlined, and using notation from §6.1.4 we can define dynamic availability for PoST as used in Chia by requiring that at any time t

$$\mathsf{PoST}_a(t) \leq f \cdot \mathsf{PoST}_h(t) \tag{16}$$

With this eq.(13) becomes

$$\begin{aligned}&\Pr\left[\text{fork starting at } t_0 \text{ and released at } t_1 \text{ heavier than honest chain}\right]\\ &\leq -\exp\left(\frac{\mathsf{PoST}_h(t_0,t_1)}{D}\cdot\left(\frac{f}{1.47}-1\right)^2\right)\end{aligned} \tag{17}$$

Thus like in Bitcoin, in Chia the probability of a successful fork decreases exponentially fast in the length of the fork.

Unlike for PoW, to guarantee security it's not sufficient that $f > 1$, but we need a more substantial gap in the resources of the honest parties and the adversary to account for double dipping, for parameters as in Chia $f > 1.47$ is sufficient.

6.2.3 Dynamic Availability from PoSpace

While in Chia we achieve security under dynamic availability by using space and time as a resource, it's an intriguing question whether a longest-chain blockchain based on proofs of space alone like Spacemint [PKF+18] could be secure under dynamic availability.

Surprisingly, the answer is a resounding no as shown in [BP22]. They consider a setting where the chain progresses in steps, where a step happens every time a new challenge is picked. The adversary

can change the amount of space available to the honest parties by a factor $1 \pm \epsilon$ with every step, and the space available to them is always a factor $f > 1$ smaller than what the honest parties have. Moreover the space can be replotted in R steps. Their result states that no matter what chain selection rule is used, in this setting a PoSpace based blockchain can always be successfully forked by an adversary with a fork of length at most $R \cdot \epsilon / f^2$ steps. This bound is tight as a (albeit fairly complicated and thus not practical) chain selection rule achieving this bound exists.

6.2.4 Dynamic Availability from PoStake

The impossibility from [BP22] just discussed does not translate to proofs of stake based chain as there's no analogue for replotting in the stake setting. In fact, PoStake based longest-chain protocols secure under dynamic availability do exist [BGK+18]. The Ouroboros genesis chain selection rule from this paper works as follows: given two competing chains, one just compares the chains at a fairly short window right after the fork. Intuitively, the reason such a chain selection rule does not provide security under dynamic availability for space is because an adversary could use replotting to make this short window have large weight, thus create a winning chain even with much less space than the honest party.

References

Identifier	Publication
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, *Advances in Cryptology - ASI- ACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 357–379. Springer, 2017.
BBBF18	Dan Boneh, Joseph Bonneau, Benedikt Bu¨nz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I, volume 10991 of Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.
BBF18	Dan Boneh, Benedikt Bu¨nz, and Ben Fisch. A survey of two verifi- able delay functions. *IACR Cryptol. ePrint Arch.*, page 712, 2018.
BDK+19	Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof- of-stake longest chain protocols: Security vs predictability. 2019.
BGK+18	Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 913–930. ACM, 2018.
BNPW19	Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S. Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In Anna Karlin, Nicole Immorlica, and Ramesh Johari, editors, *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 459–473. ACM, 2019.
BP22	Mirza Ahad Baig and Krzysztof Pietrzak. On the existence of proof of space longest chain protocols (working title), 2022. 2022. Manuscript in preparation.

Identifier	Publication
CKWN16	Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 154–167. ACM, 2016.
CP19	Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. 2019.
DFKP15	Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II, volume 9216 of Lecture Notes in Computer Science*, pages 585–605. Springer, 2015.
DKT21	Soubhik Deb, Sreeram Kannan, and David Tse. Posat: Proof-of-work availability and unpredictability, without the work. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II, volume 12675 of Lecture Notes in Computer Science*, pages 104–128. Springer, 2021.
DW13	Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013*, Proceedings, pages 1–10. IEEE, 2013.
EFKP20	Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154. Springer, 2020.
ES18	Ittay Eyal and Emin Gu¨n Sirer. Majority is not enough: bitcoin mining is vulnerable. *Commun.* ACM, 61(7):95–102, 2018.
FZ17	Lei Fan and Hong-Sheng Zhou. iching: A scalable proof-of-stake blockchain in the open setting (or, how to mimic nakamoto's design via proof-of-stake). *IACR Cryptol. ePrint Arch.*, page 656, 2017.

Identifier	Publication
GKL15	Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
GKR18	Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018*, pages 85–92. IEEE, 2018.
Hel80	Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.
Lew21	Andrew Lewis-Pye. Byzantine generals in the permissionless setting. *CoRR*, abs/2101.07095, 2021.
LR21	Andrew Lewis-Pye and Tim Roughgarden. How does blockchain security dictate blockchain implementation? In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1006–1019. ACM, 2021.
Pie19a	Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of LIPIcs, pages 59:1–59:25. Schloss Dagstuhl - Leibniz- Zentrum für Informatik, 2019.
Pie19b	Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, 1*0th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of LIPIcs, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum fu¨r Informatik, 2019.
PKF+18	Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gazi, Joël Alwen, and Krzysztof Pietrzak. Spacemint: A cryptocurrency based on proofs of space. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Cura cao, February 26 - March 2, 2018, Revised Selected Papers*, volume 10957 of *Lecture Notes in Computer Science*, pages 480–499. Springer, 2018.

Identifier	Publication
PS17	Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017.
SNM+21	Caspar Schwarz-Schilling, Joachim Neu, Barnab´e Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. *IACR Cryptol. ePrint Arch.*, page 1413, 2021.
SSZ15	Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.
Wes20	Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, 2020.

Additional Reading

- How to Store a Permutation Compactly by Bram Cohen and Dan Boneh

A - 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; AAC+17] and verifiable delay functions [Pie19b; BBBF18; Wes20]. The definitions are not fully general, but instead tailored to the particular constructions of PoSpace from [AAC+17] and the VDFs [Pie19b; BBBF18; Wes20] based on sequential squaring.

A.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:

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

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

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

A.2 (Unique) Proofs Of Space

A.2.1 Algorithms for PoSpace

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

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[7]

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

[7] 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 [PKF+18] 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).

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.

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.

PoSpace.verify

on input a proof σ outputs accept or reject

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

We assume perfect completeness

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

A.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, initializing 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.

A.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.

A.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.

A.3 Verifiable Delay Functions

The definition of verifiable delay functions (VDFs) given below is not completely general, but makes some additional properties of VDF we'll need in Chia explicit. In particular, we want a VDF where the sequential computation can start before we know the number of sequential steps for which it

will run, while still being able to output proofs reasonably fast at any point during the sequential computation. This similar to the functionality provided by continuous VDFs [@Ephraim2020], which require that one can provide proofs for intermediate values almost immediately. We can allow some slack, and thus can use "normal" practical VDF constructions. We'll use the following notation to an (ongoing or finished) VDF computation τ

$\tau.c \in \{0,1\}^*$: the challenge (usually one or more unpredictable values) used for this VDF

$\tau.t \in \mathbb{N}$: total number of sequential steps performed

For $i \ : \ 0 \ \leq i \leq \tau.t$ we let $\tau[i]$ denote the state of the VDF after i sequential steps, and

$\tau[i].x \in X$: denotes the value after i steps.

$\tau[i].\pi$: is a proof certifying that $\tau[i].x$ is correctly computed.

We'll denote the value and proof for the last value as

$$\tau.y \stackrel{\mathsf{def}}{=} \tau[\tau.t].x \qquad \tau.\pi \stackrel{\mathsf{def}}{=} \tau[\tau.t].\pi$$

The functions defining a VDF are

VDF.sample

on input a challenge $c \in \{0,1\}^*$ samples the initial value x and outputs a partial VDF value

$$\tau.\mathsf{t} := 0 \ , \ \tau[0].\mathsf{x} := x \ , \ \tau.\mathsf{c} := c$$

VDF.next

$X \to X$ the function doing one step of the sequential computation

VDF.solve

on input a challenge $c \in \{0,1\}^*$ and time parameter $t \in \mathbb{Z}^+$ outputs a proof

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

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.

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 $\log(t)$ [Pie19b] or even constant [Wes20] 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 [Pie19b; Wes20] VDFs discussed below also $\tau.\pi$ is unique.

sequentiality: Informally, sequentiality states that for any t, an adversary A who makes less than t sequential steps will not find an accepting proof on a random challenge. I.e., for some tiny ϵ

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

Let us stress that 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.

A.3.1 The [Pie19b, Wes20] VDFs

The VDFs proposed in [Pie19b; Wes20] (see [BBBF18a] 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 $\mathsf{VDF.solve}(c, t)$ one would first map the challenge c on a group element, say as $x_c := hash(c) \bmod N$, and the output is (y, π) with $y = x_c^{2^t} \bmod N$. This y can be computed by squaring x_c sequentially t times $x_c \to x_c^2 \to x_c^{2^2} \to \cdots \to x_c^{2^t}$, and it is conjectured that there is no shortcut to this computation if one doesn't know the factorization of N.

The VDFs from [Pie19b; Wes20] differ in how the proof π that certifies that $y = x_c^{2^t} \bmod N$ is defined. The proof in [Pie19b] 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. As this sampling is expensive, one would then think of N as a public parameter to be used indefinitely.

Wesolowski [Wes20] suggests 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. On the other hand, it's somewhat tricky to obliviously sample random group elements in class groups (here obliviously means in a way that does not reveal the discrete log of the element). Thus in the class group setting we can let $\mathsf{VDF}(c, t)$ sample a fresh group using the challenge c, and then exponentiate a fixed easy to find group element (concretely the element (a=2, b=1)). This is the approach taken in Chia.