chia consensus

Purpose of this document: explain the 1.1 version of the Chia consensus algorithm

Target audience: a technical audience familiar with blockchain but not with Proofs of Space (PoS), Proofs of Time/Verifiable Delay Functions (VDF) and Chia.

If you are new to Bitcoin/blockchain, read this textbook first: Bitcoin and Cryptocurrency Technologies.

Please ask questions on keybase so we can improve this doc!

Motivation

The Chia Consensus algorithm aims to create an environmentally friendly, secure, and decentralized alternative to proof of work and proof of stake.

Proof of work (PoW) cryptocurrencies burn huge amounts of electricity. Furthermore, they tend to become centralized due to the concentration of hardware manufacturing and ownership and concentration of cheap energy, making PoW inaccessible to ordinary users and susceptible to various attacks.

Proof of stake has many forms, each with its own pros and cons. Some common weaknesses are: concentrated control of funds by exchanges; concentration of delegation; reliance on checkpoints and subjectivity (a requirement to be online periodically); inaccessibility to regular users; slashing risk; clock synchronization assumptions, networking assumptions, and other security assumptions.

Introduction

Decentralized consensus algorithms require Sybil resistance with a resource that is cryptographically verifiable and scarce (not infinite). In previous blockchain systems the scarce resources have been computing power and stake. Proof of space is an alternative that comes much closer to Bitcoin's original ideal of "one cpu one vote" by using storage capacity as the scarce resource. For example, someone storing 500GiB has 5 "votes," someone storing 100GiB has 1 "vote", where a vote refers to a chance to win and validate a block, not an actual vote on-chain. Using only storage capacity however, is not secure. One other cryptographic puzzle piece is used to secure this system: namely a verifiable delay function, which is a cryptographic proof that real time has passed. A fair system can be created by combining proofs of space and time. In such a system, users store random-looking data on their hard drives for periods of time and their chance to win Chia is proportional to their allocated space. Furthermore, such a system scales to billions of participants in a similar way to the proof of work lottery. No funds, special hardware, registration, or permission is required to join, only a hard drive. And the system is completely transparent and deterministic -- anyone can efficiently and objectively verify which chain is the canonical one.

Proofs of space

A proof of space protocol is one in which:

1. a Verifier can send a challenge to a Prover, and
2. the Prover can demonstrate to the verifier that the Prover is reserving a specific amount of storage space at that precise time.

The proof of space protocol has three components: plotting, proving/farming, and verifying. Details here.

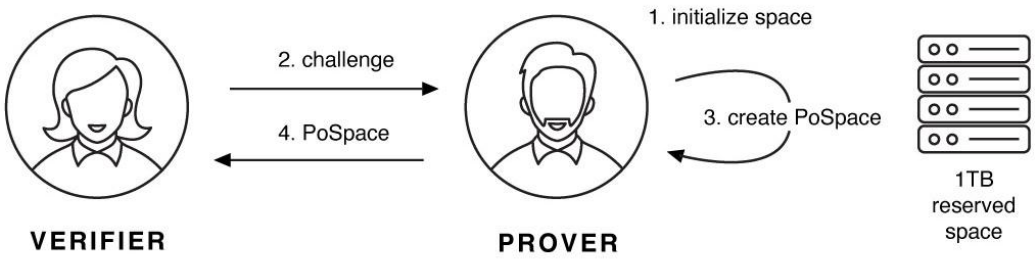

Figure 1: First, the prover “plots" or allocates a portion of disk space (1). Then the prover “farms” by responding to challenges with proofs of space (2,3,4). The verifier checks that the proof is valid for that challenge.

Plotting is the process by which a prover, who we refer to as a farmer, initializes a certain amount of space. A farmer can be any person who has at least 100 GiB available to reserve on their laptop, or an enterprise prepared to allocate a large volume of unused storage space. There is no upper limit. Plotting takes on the order of hours or days, and is performed only once. The initialized space is occupied by a file called a plot. Plot sizes are determined by a k parameter, where space = `780 * k * pow(2, k - 10)`, with a minimum k of 32 (101.4 GiB). As of Chia 1.0, a k32 plot can be created in around six hours with a fast commodity machine, and 24 hours with a slow machine using one CPU core and a few GB of memory. There are opportunities for huge speedups. The PosSpace construction is based on Beyond Hellman [8], but is nested 6 times and contains other heuristics to make it practical.

The result is a plot file that can be, for example, 100 GiB. The file contains seven tables with random-looking data. Each table has 2^k entries. Each entry in table i contains two pointers to table i-1 (the previous table). Finally, each table 1 entry contains a pair of integers between 0 and 2^k, called “x-values.” A proof of space is a collection of 64 x-values that have a certain mathematical relationship.

In the diagram above, once the Prover has initialized 100 GiB, they are ready to receive a challenge and create a proof. One attractive property of this scheme is that it is non-interactive: no registration or online connection is required to create a plot. Nothing hits the blockchain until a reward is won, similar to PoW.

Farming is the process by which a farmer receives a sequence of challenges to prove that they have legitimately put aside a defined amount of storage. In response to each challenge the farmer checks their plots, generates a proof and submits any winning proofs to the network for verification.

Each iteration of this process is a table lookup. A lookup takes a 256 bit challenge as input and outputs a proof. The farmer responds to a challenge by reading a pair of values in table 7. These point to two entries in table 6, etc. Finally, the farmer fetches the whole tree of x-values. This requires one read for table 7, two for table 6, four for table 5, etc. The whole process would take approximately 640ms, assuming a slow HDD with a 10ms seek time. The amount of data read is small and is independent of plot size.

Since most proofs generated by this process are not good enough (as discussed later) to be submitted to the network for verification, we can optimize this process by only checking one branch of the tree, which results in two x-values, depending on the challenge. We then hash the x-values generated in this way into a 256 bit string to determine whether the proof is good. Hashing these x-values gives us the quality string, a 256 bit random value. This is combined with the difficulty and the plot size to generate the required_iterations. If the required_iterations is less than a certain number (we can get into the blockchain), then we look up the whole PoSpace. Looking up one branch requires only around 7 disk seeks and reads or about 70ms on a slow hard drive.

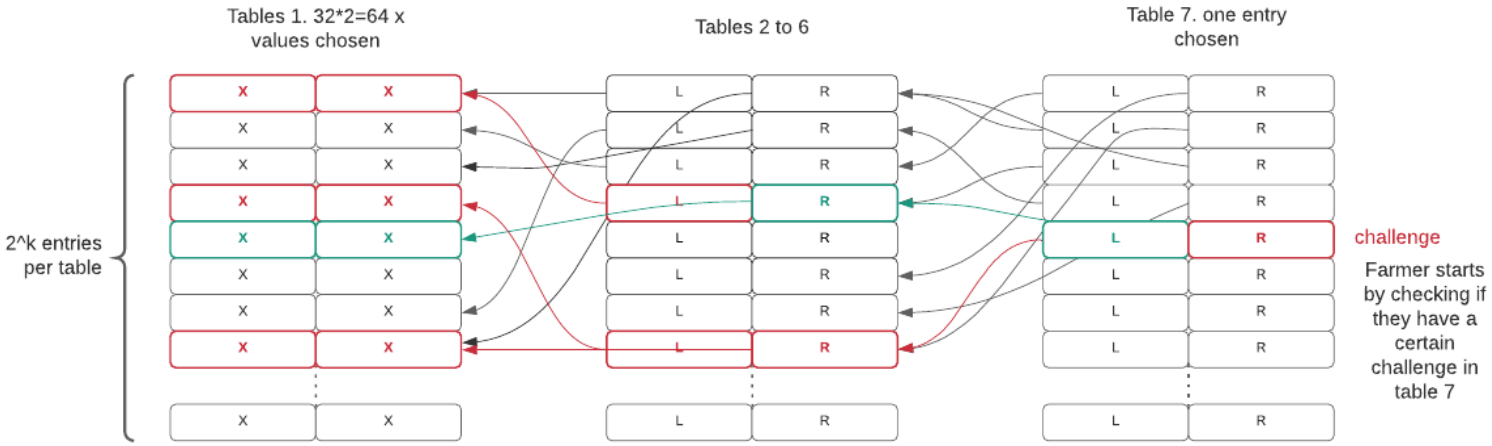

Figure 2: Structure of a plot file. The 64 red x- values represent the proof, the 2 green x- values represent the quality.

A further optimization is to disqualify a certain proportion (for example 511/512) plots from eligibility for each challenge. This is referred to as the **plot filter**. For example, requiring that the hash of the challenge and the plot_id starts with 9 zeros. This hurts everyone equally (except for replotting attackers), and is therefore fair. This makes farming require almost no resources, and very few disk reads every few minutes. Chia users have successfully been farming multiple PiB of storage on a single Raspberry Pi. We assume that farmers always use HDDs since they are cheap and there is no reason to use SSDs since the speed is not relevant for farming. SSDs / RAM can be used for faster plotting, however.

The plot key is a private key that is stored in the plot file. The plot id is generated by hashing the plot public key and the pool public key. Creating a block with a proof of space requires signing with both the plot key and the pool key. Therefore the pool may not be changed after creating the plot. In practice, the plot key is a 2/2 BLS aggregate public key between a local key stored in the plot and a key stored by the farmer software. For security and efficiency a farmer may run a centralized server using this key and signature scheme. The server may be connected to many harvester machines that store plots. Farming requires the farmer key and the local key, but does not require the pool key, since the pool's signature can be cached and reused for many blocks.

Verifying: After the farmer has successfully created a proof of space, the proof can be verified by performing a few hashes and making comparisons between the x-values in the proof. Recall that the proof is a list of 64 x-values, where each x-value is k bits long. For a k32 this is 256 bytes, and is therefore very compact. Verification is very fast, but not quite fast enough to be verified in solidity on ethereum (something that would enable trustless transfers between chains), since it requires blake3 and chacha8 operations.

Proofs of time

A proof of time or a **V**erifiable **D**elay **F**unction, is a proof that a sequential function was executed a certain number of times.

Verifiable: this means that after performing the computation (which takes time), the prover can create a very small proof in a very short time, and the verifier can verify this proof without having to redo the whole computation.

Delay: this means that the prover actually spent a real amount of time (although we don't know exactly how much) to compute the function.

Function: this means it's deterministic: computing a VDF on an input x always yields the same result y.

The key word here is "sequential", like hashing a number many times: hash(hash(hash(a))), etc. This means the prover cannot just buy more machines to go faster, unlike Bitcoin/proof of work. Therefore we can assume that computing a VDF requires real (wall-clock) time. The construction that we use is repeated squaring. The prover must square a challenge x T times. This requires time Θ(T). The prover also must create a proof that this was performed properly.

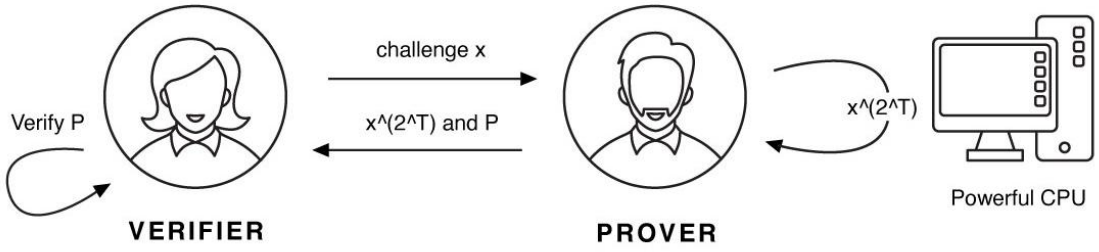

Figure 3: Verifier (blockchain) sends a challenge to a prover (timelord) and prover computes output and proof.

Although the following details are not very important for understanding the consensus algorithm, the choice of what VDF to use is relevant, because if an attacker succeeds in obtaining a much faster machine, some attacks are possible.

The VDF used by Chia is repeated squaring in a class group of unknown order. There are two main ways to generate a large group that has an unknown order. The first is to use an RSA modulus, and use the integers mod N as a group. The order of the group is unknown if you can generate your modulus with many participating parties using an MPC ceremony. An easier approach is to use classgroups with a large prime discriminant, which are groups of unknown order. This does not require any complex or trusted setup, so we chose this option for Chia. To create one of these groups, one just needs a large random prime number. The drawback is that classgroup code is less tested in real life, and optimizations are less well known than in RSA groups. We use the same initial element for the squaring (a=2, b=1 classgroup element), and instead use the challenge to generate a new random prime number for each VDF, which is used as the discriminant. The discriminant has a size of 1024 bits, which means the proof sizes are around 1024 bits. We use the Wesolowski scheme split into n (1<=n<=64) phases so that creating the proofs is very fast. Since the n-wesolowski proofs can be large, we replace them with 1-wesolowski proofs as soon as they are available, since these are smaller, but require more time to make. The proofs themselves are not committed to on chain, so they are replaceable.

Infusion

As a recap, VDFs take in an input, called the challenge, and produce an output together with a proof that certifies that the function was evaluated correctly.

Infusion of a value into a VDF means that that value is combined with an output of a VDF, to generate a new value, which is used as the input/challenge for the next VDF. Therefore, we are chaining VDFs but committing to a new value (block) in between. This is used so that we have a linear progression of blocks, alternating proofs of space with proofs of time.

Consensus Algorithm

BLS Signatures

Whenever signatures are referred to in this document, it is assumed that a deterministic BLS signature is used, following the IETF specification with the Augmented scheme. The private keys performing these digital signatures are controlled and stored by the farmers, and a unique private key is used for each plot.

Roles of Nodes

Farmers

Farmers are nodes which participate in the consensus algorithm by storing plots and checking them for proofs of space. They communicate with a Full Node (usually on the same machine.) Farmers also communicate with one or more Harvesters which is a service that resides on the machine where plots are stored and looks up proofs of space on behalf of the Farmer process.

Timelord

Timelords are nodes which participate in the consensus algorithm by creating proofs of time and infusing blocks into their VDFs.

Full Nodes

Full nodes can be timelords or farmers, or they can just perform the roles of a full node. This entails broadcasting proofs of space and time, creating blocks, maintaining a mempool of pending transactions, storing the historical blockchain, and uploading blocks to other full nodes as well as wallets (light clients).

Challenges

The Chia consensus algorithm relies on running VDFs for periods of time called sub-slots, which are adjusted periodically to add up to about 10 minutes. Periodically challenges are released, which starts a sort of mini lottery where farmers check their plots for proofs of space. When farmers find a proof of space that qualifies, they broadcast it to the network. The difficulty changes to target 32 winning proofs for the entire network in each sub-slot. These proofs are infused into the VDF at different times within the sub-slot. Farmers follow the heaviest chain, which is the chain with the most cumulative difficulty on it (usually the chain with the most blocks).

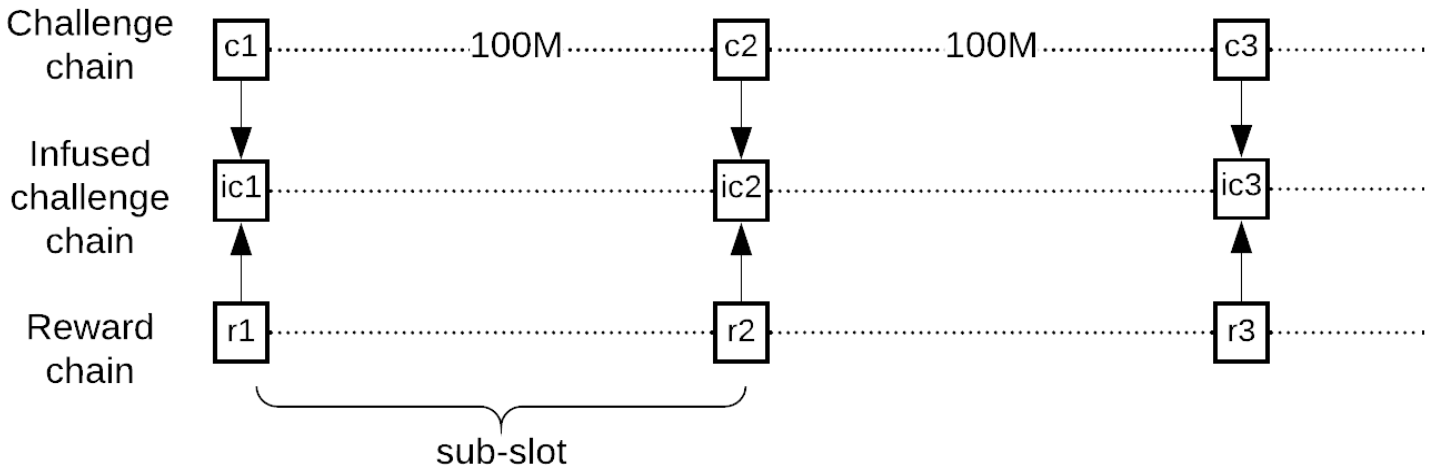

Figure 4: Three sub-slots. The x axis represents time. Dotted lines represent VDF execution, advancing in time from left to right. Arrows represent hash dependencies (an object which points to another object includes the hash of the second object).

In figure 4, we can see three challenge points, c1, c2, and c3. At the points c1, c2, and c3 timelords create challenges (256 bit hashes) which are provided as input to VDFs. Timelords take these hashes, and start computing a VDF on this challenge, for the specified number of iterations. In this example, each slot is 100,000,000 iterations. When the VDF is finished, the timelord publishes the new challenge and the proof of the VDF. An infusion of end-of-slot information happens at the end of each sub-slot.

Sub-slot: a segment of a fixed number of VDF iterations, subject to work difficulty adjustment, always adjusting to a target fixed amount of time (i.e. 10 mins).

Sub-slot iterations: a constant which is periodically adjusted which determines how many VDF iterations each sub-slot must have.

Challenge: sha256 output string which is used as proof of space challenges for farmers' plots, as well as for the challenge chain VDF. This is also referred to as *challenge hash*.

As you can see in Figure 4, there are three VDFs being executed concurrently, each which serve a different purpose. They are explained in the following sections.

Signage points and infusion points

Each sub-slot in the challenge and reward chains is divided into 64, smaller, VDFs, and between each of these small VDFs is a point called a **signage point**. Timelords publish the VDF output and proof when they reach each signage point. Note that both the challenge chain and the reward chains have signage points (but not the infused challenge chain). The number of iterations between each signage point is **sp interval iterations**, which is equal to sub slot iterations / 64.

The challenge at the start of the sub-slot is also a valid signage point. As each of the 64 signage points is reached, they are broadcast through the network by timelords and nodes. Farmers receive these signage points and compute a plot filter based on the signage point, their plot id, and the sub-slot challenge. If the plot filter bits start with 9 zeros, that plot passes the filter for that signage point, and can proceed. This disqualifies around 511/512 of all plot files in the network, for that signage point.

$$plot\ filter\ bits = sha256(plot\ id + sub\ slot\ challenge + signage\ point)$$

The proof of space challenge is computed as the hash of the plot filter bits:

$$pos\ challenge = sha256(plot\ filter\ bits)$$

Using this challenge, the farmers fetch quality strings for each plot that made it past the filter from disk. Recall that this process is almost instant, and that the signage point is a hash derived from part of the proof of space (but the whole proof of space is not retrieved yet).

The farmer computes the **required iterations** for each proof of space. If the required iterations < sp interval iterations, the proof of space is eligible for inclusion into the blockchain, so the farmer fetches the entire proof of space from disk (which takes longer than only fetching the quality), creates an unfinished sub block, and broadcasts it to the network. Note that the vast majority of required iterations will be way too high, since on average 32 will qualify for the whole network for each sub slot. This is a random process so it's possible for a large number of proofs to qualify, but very unlikely. The **signage point iterations** is the number of iterations from the start of the sub-slot to the signage point.

The **infusion iterations** is the number of iterations from the start of the sub slot at which the block with the quality above can be included into the blockchain. This is calculated as:

$$infusion\ iterations = (signage\ point\ iterations + 3 * sp\ interval\ iterations + required\ iterations)\ \%\ sub\ slot\ iterations$$

Therefore, the infusion iterations will be between 3 and 4 signage points *after* the signage point. Farmers must submit their proofs and blocks before the infusion point is reached. The modulus is there to allow overflows into the next sub-slot, if the signage point is near the end of the sub-slot. This is expanded on later.

At the infusion point, the farmer's block gets combined with the infusion point VDF output to create a new input for the VDF from that point on, i.e. we infuse the farmer's block into the VDF. The block is only fully valid after the infusion iterations has been reached, and the VDF proof has been attached to the block.

For the b1 block to be valid/finished, two VDF proofs must be included: one from r1 to the signage point and one from r1 to b1. (actually it's more since there are three VDF chains, explained later). In Figure 5, the farmer creates at the time of the signage point, (let's call it B1'). However, B1' is not finished yet, since it needs the infusion point VDF. Once the infusion iterations VDF is released, it is added to B1' to form the finished block at B1.

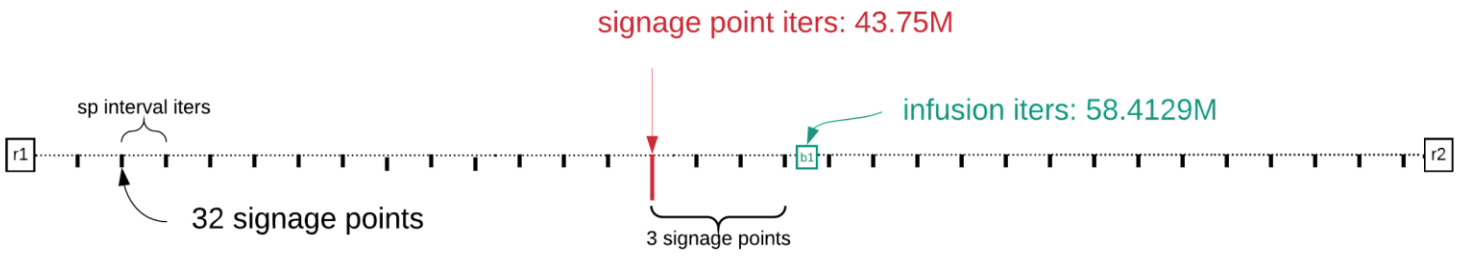

Figure 5: timelords create proofs for both the signage point and the infusion point. But they only infuse (change the VDF classgroup) for the latter. Squared symbolize infusions, where a new VDF is started. Sp_iterval_iterrs = 3.125M. UPDATE TO 64 SP

Let's consider the example in figure 5. The sub-slot iterations is 200M, and the sp interval iterations is 3.125M. Let's say a farmer has a total of 1000 plots.
:w
For each of the 64 signage points, as they are released to the network every 9 seconds, or every 3.125M iterations, the farmer computes the plot filter and sees how many plots pass. For each of the plots that pass the filter for each signage point, the farmer computes the required iterations. In this example, the farmer only gets required_iterations < 3.125M one time in the whole sub-slot (let's say it's 2.2879M). In Figure 5, this is at the 14th signage point. The infusion iterations is computed as:

$$\begin{aligned} infusion\ \ iterations &= signage\ \ point\ \ iterations + 3 * sp\ \ interval\ \ iterations \\ &+ required\ \ iterations \\ &= 14 * 3.125M + 3 * 3.125M + 2.2879M \\ &= \mathbf{55.4129M} \end{aligned}$$

After realizing they have won (at the 14th infusion point), the farmer fetches the whole proof of space, makes a block, optionally including transactions, and broadcasts this to the network. They have a few seconds (up to the infusion iterations), to reach timelords, who will infuse the block, creating the infusion point VDFs. With these VDFs, the block can be finished and added to the blockchain.

Sp interval iterations: Defined as floor(sub-slot iterations / 64).

Signage points: 64 intermediary points in time within a sub-slot in the challenge and reward chains, for which VDFs are periodically released. At each signage point, a VDF output is created and broadcast through the network. The first signage point in the sub-slot is the challenge itself. Each block has a signage point such that the proof of space in the block must be eligible for that signage point.

Required iterations: A number computed using the quality string, used to choose proofs of space which are eligible to make blocks. The vast majority of proofs of space will have required iterations which are too high, and thus not eligible for inclusion into the chain. This number is used to compute the infusion point.

Infusion point: the point in time at *infusion iterations* from the challenge point, for a proof of space with a certain challenge and *infusion iterations*. At this point, the farmer's block gets infused into the reward chain VDF. The infusion point of a block is always between 3 and 4 signage points after the signage point of that block. Computed as signage point iterations + 3 * sp interval iterations + required iterations.

The delay between the signage point and infusion point has many benefits, including defense against orphaning and selfish farming, decreased forks, and no VDF pauses. This delay of around 30 seconds is given so that farmers have enough time to sign without delaying the slot VDF. Well behaving farmers sign only one signage point with each proof of space, meaning that attackers cannot easily reorg the chain.

Multiple blocks

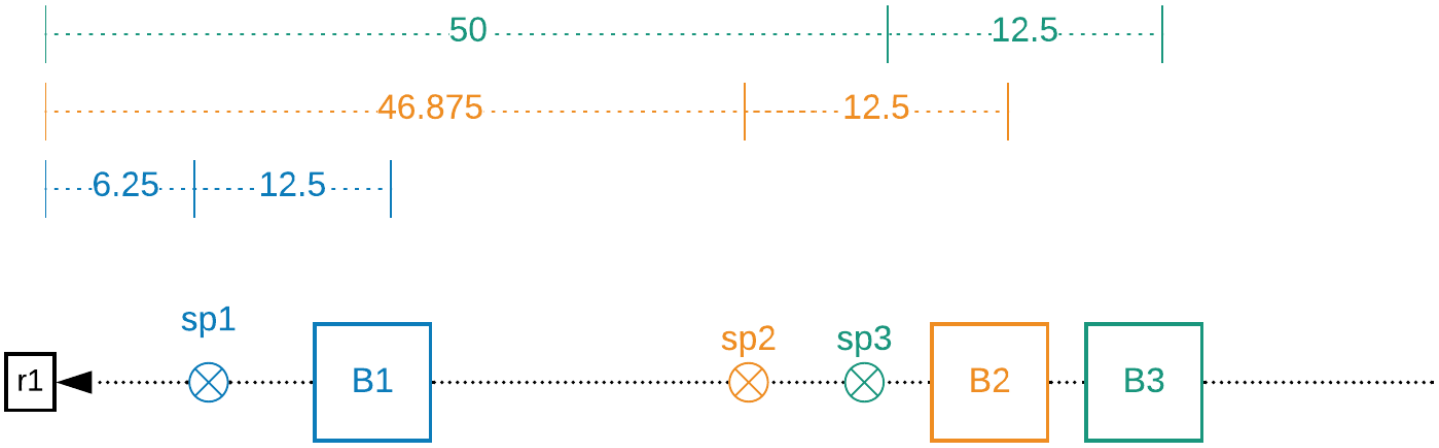

Figure 7: multiple blocks. Sp1 = signage points 1

As you can see in figure 7, multiple blocks can get infused into the same sub-slot. Chia's system targets 32 blocks per sub-slot, and this is adjusted through the work difficulty algorithm. VDFs go from the previous infusion point to the current signage point and from the previous infusion point to the current infusion point. Note that the VDF proofs required for each block can overlap. For example, B2 contains a

VDF proof from B1 to sp2, and from B1 to B2. B3 contains a proof from B1 to sp3, and from B2 to B3. B2 does not depend at all on B3, but B3 depends on B2, since its VDF is from B2's infusion point. Again, the blocks get created at the signage points, but they are missing the infusion point VDF; once this VDF is added, the block is finished, and forms part of the blockchain. There are no signatures at the infusion point; the only things added at the infusion point are the VDFs.

Three VDF chains

If we only used one VDF (for the reward chain), the inclusion or exclusion of blocks would allow control of the challenge for the next slot. This means that an attacker could try many different combinations and choose the challenge that suits them best. These types of attacks are called grinding attacks, and they are one of the main difficulties of changing from Proof of Work to Proof of Space or PoStake. More detail is provided in the "Attacks and countermeasures" section.

To mitigate this, the challenges will be based only on the first block to be infused in a slot.

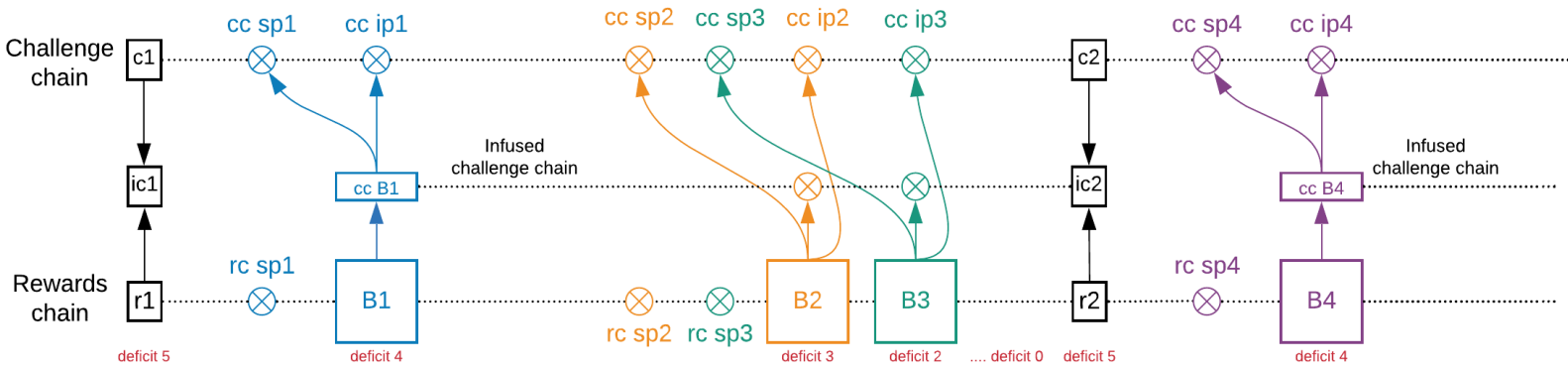

Figure 8: three VDF chains. An attacker can manipulate the reward chain results but this has no effect on c2, and therefore has no effect on the PoSpace lottery. cc = challenge chain, rc=reward chain, sp=signage point. B=block.

There is a lot going on in this diagram. First of all, as you can see, there are 4 **blocks:** B1, B2, B3, B4, these are blocks created by the farmers, which contain all the data that they point to. We assume that more than 5 blocks have been created in that sub-slot, but we don't draw all of them due to space constraints.

Also, both the challenge chain and reward chain create 64 signage points. Blocks must include the signage point VDFs for both chains. Blocks must also include the infusion point VDFs for all three chains.

As you can see, the challenge chain executes the VDF from the start of the sub-slot to the end with nothing infused into it (the circles are VDF proofs but they do not interrupt the VDF). The reward chain infuses every block that is included. The chain in the middle is called the **infused challenge chain**, and it starts at the first infused block for each challenge, and goes on until the end of the slot.

A **slot** is the list of sub-slots which contain at least 16 reward-chain blocks based on the challenge of the first sub-slot, or later sub-slots. For example, we may have only 10 blocks in a sub-slot, and then 3 and

then 7, which means those three sub-slots form one slot. Usually each sub-slot is also a slot, since many more than 16 blocks are included on average. The **deficit** is the number of blocks still necessary to end the slot: this is described later in more detail.

At the end of the slot, the challenge chain is combined with the infused challenge chain to generate the new challenge c2, which is used to start the challenge chain for the next sub-slot.

The only block which affects the challenge chain is the first block, which here is B1, and only a deterministic part of B1, cc B1, which only depends on challenge chain data. An attacker who wants to grind cannot change the challenge by withholding B2, B3, or any other block apart from the first one.

Assuming the attacker has the fastest block (B1), they have three options: withhold it, delay it, or release it. In order to know whether the new challenge will benefit them, they will need to execute the VDF all the way up to c2. By that time, their chance to get included in the reward chain is gone, since honest farmers sign only one block per proof of space. Withholding B1 does not provide much benefit to the attacker, since they must release it before sp2 in order to get the farmers on their chain. Farmers will choose the heaviest chain, which is the one with the most (heaviest) reward chain blocks.

Why do we commit to any blocks at all in the challenge chain? Well, if we did not, an attacker could look ahead with a faster VDF, since they would not need the help of honest participants in order to compute the challenge chain into the future. The challenge chain would be totally deterministic. This would enable some advantage by replotting. Furthermore, the challenge chain can be used to probabilistically prove the weight of the reward chain to light clients, without sharing all reward chain blocks (since the challenge chain depends on the “best” block in the slot, you can calculate the number of reward chain blocks).

Challenge chain: The VDF chain based on each challenge for each sub-slot, which does not infuse anything in the middle of each sub-slot. The challenges are also used for the proofs of space. The signage points in this chain are used for the SP filter.

Reward chain: The VDF chain that contains infusions of all blocks. This chain pulls in the challenge chain and optionally the infused challenge chain at the end of each sub-slot.

Infused challenge chain: A VDF chain which starts at the first block infused in a slot (which is not based on the previous slot’s challenge, this is called the challenge block) and ends at the end of the slot.

Slot: the list of sub-slots which contain at least 16 reward-chain blocks based on the challenge of the first sub-slot, or later sub-slots. At the end of the slot, the infused challenge chain stops, the challenge chain pulls in the result of the infused challenge chain, and the deficit is reset to 16.

Block: a block is a collection of data infused into the rewards chain which contains: a proof of space for a challenge hash with less iterations than the slot iterations, sp and ip VDFs for both chains, optional ip VDF for the infused challenge chain, and a rewards address. Some blocks are also transaction blocks. There is a maximum of 128 blocks per slot.

Transaction Block: A block that is eligible to create transactions, along with an associated list of transactions.

Challenge block: The first block to be infused in each slot, which is not based on a previous slot's challenge. The challenge block always has a deficit of 15, and always starts off the infused challenge chain.

Peak: The peak of the blockchain as seen by a node is the block with the greatest weight. Weight is the sum of the difficulty of a block and all its ancestors, which is similar to height, but a shorter chain can have heavier weight, due to difficulty adjustments.

For a block to be considered valid, it has to provide VDFs for the challenge chain and reward chain, and optionally for the infused challenge chain if it is present. Forcing all VDFs to be included means that all three chains are guaranteed to move forward at the same rate.

Overflow blocks

For a farmer to create a block, their required_iterations must be less than 3.125M, or the sub-slot iterations / 64, as described above. This means that infusion iterations might be greater than the sub-slot iterations, and therefore the infusion must happen in the next sub-slot.

Overflow block: a block whose infusion point is in a different sub-slot than its signage point.

Current-slot challenge: With respect to a certain block B, B's current-slot challenges include all challenges starting at the first challenge in the slot, and ending at the end of the slot (non-inclusive). This is relevant because sometimes a slot spans multiple sub-slots, and thus multiple challenges.

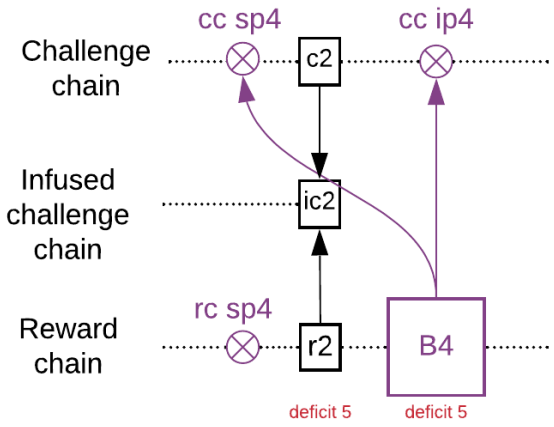

Figure 9: B4 in this diagram is an overflow block, since the infusion is in the next slot. B4 is not based on a current-slot challenge, and thus does not decrease the deficit or make a challenge block. TODO: diagrams should be 16 not 5.

Overflow blocks cannot exist in the first sub-slot of the epoch (since the sub-slot iterations change).

Also, overflow blocks do not change the deficit unless they are based on a current-slot challenge, since overflow blocks are responses to the previous sub-slot's challenge. Overflow blocks are not challenge blocks unless they are based on a current-slot challenge. Note that it is rare for overflow blocks to

decrease the deficit, since the deficit will almost always be decreased to zero, and a new slot will be started on every sub-slot.

Minimum block requirement

A minimum of 16 current-slot challenge blocks must be infused into the rewards chain in order for a slot to be finished.

The deficit is a number between 0 and 16 that is present at the start of a sub-slot. This is defined as the number of reward chain blocks that we need to infuse in order to finish a slot. It is reset to 16 whenever we start a slot (so there must be at least 16 total blocks per challenge chain infusion). The deficit goes down for each reward chain infusion that is based on a current-slot challenge.

The block with deficit 15 is a challenge block.

The normal case is where the deficit starts at 16, and goes down to zero within the sub-slot, and resets back to 16 as we finish the slot and start a new one. In the case that we don't manage to reduce it to 0 within the end of the slot, the challenge chain and infused challenge chain (if present) continue, and the deficit does not reset to 16. Blocks (including overflow blocks now), keep subtracting from the deficit until we reach 0. When we finish a sub-slot with a zero deficit, the infused challenge chain is included into the challenge chain, and the deficit is reset to 16.

This requirement is added to prevent long range attacks, and is described in detail in the Countermeasures section below. The vast majority of sub-slots will have >=5 blocks, therefore it does not affect normal operation very much.

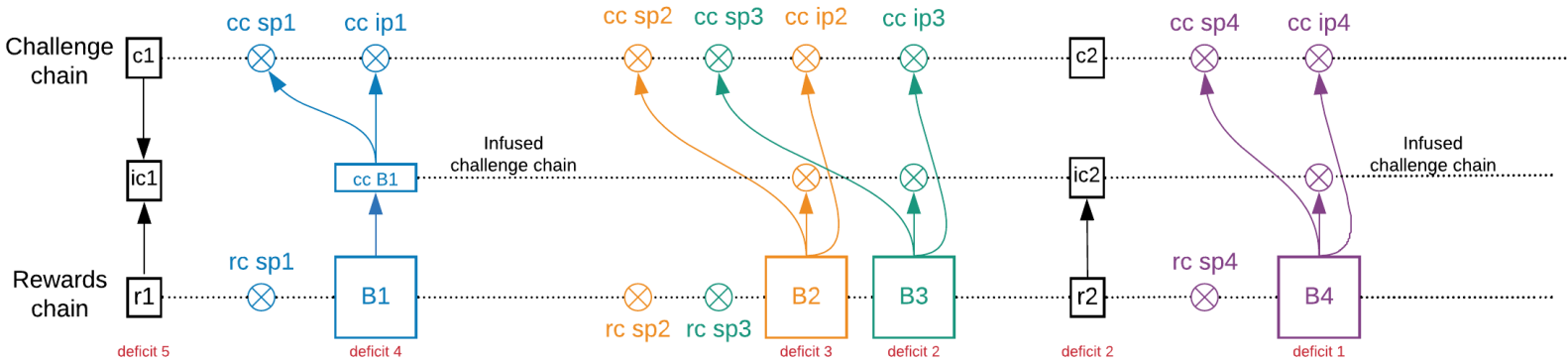

Figure 10: c2 is the end of the sub-slot but not the end of the slot. c2 does NOT point to ic2, since the slot did not end at this sub-slot. Deficit is 2 instead of resetting to 5, and the infused challenge chain continues.

Weight

The **weight** of a block is the sum of the difficulty of this block, plus all previous blocks that are ancestors of this block. Honest full nodes must choose the peak of the blockchain such that the peak is the block with the heaviest weight that they know of. This is a crucial requirement, and is identical to Bitcoin's heaviest chain rule. Due to this rule, an attacker with less than 50% of the space and no VDF advantage will have trouble earning more than their fair share, since they must get lucky and create more reward chain blocks than the honest chain. Furthermore, farmers only farm on the challenges that correspond to the heaviest chain.

Both VDF speed and total amount of space are important for weight, and changes in these can trigger difficulty adjustments. If the amount of space increases, more than 32 blocks per slot will be created, so the difficulty has to be increased. If the network VDF speed increases, more than 32 blocks are created every 10 minutes, and thus the difficulty (and the sub-slot iterations) has to be increased.

A farmer with exclusive access to a slightly faster VDF, however, cannot easily get more rewards than a farmer with the normal speed VDF. If an attacker tries to orphan one of the blocks on the chain, having a faster VDF will not help, since the attacker's chain will have less blocks (and thus a lower weight). Farmers must sign the block which they are building on top of, and they will only build on top of the highest weight chain.

The VDF speed comes into play when the attacker wishes to launch a 51% attack, however. In this case, an attacking farmer can use the VDF to create a completely alternate chain with no honest blocks, and overtake the honest chain.

Foliage

In the above diagrams, there is no place for farmers to specify their rewards, since all blocks are canonical. Farmers have no say in how their block is constructed, since they must use the exact proof of space, VDFs, and signatures that are specified. In order to include farming rewards, as well as transactions in the system, we must introduce an additional component of blocks called foliage. Up to now we have been discussing the "trunk" component.

Trunk: The component of blocks and the blockchain which includes VDFs, proofs of space, PoS signatures, challenges, and previous trunk blocks, and is completely canonical. The trunk never refers to the foliage chain.

Foliage: The component of blocks and the blockchain which includes specification of where rewards should go, which transactions should be included, and what the previous foliage block is. This is up to the farmer to decide and is grindable, so it can never be used as input to the challenges.

Reorg: A reorg (or reorganization) is when a node's view of the peak changes, such that the old view contains a block that is not included in the new view (some block is reversed). Both trunk and foliage reorgs are possible, but should be rare in practice.

In figure 11 below we can see that the foliage is added to blocks to produce an additional chain. This foliage includes a hash of the previous foliage, a reward block hash, and a signature. These foliage pointers are separate from the trunk chain, and not canonical. That is, farmers could theoretically create a foliage reorg where foliage is replaced, but the exact same trunk (proofs of space and time) are used. To prevent this, honest farmers only create one foliage block per block. As soon as one honest farmer has

added a foliage block, the foliage becomes impossible to reorg beyond that height with the same PoSpace, since that farmer will not sign again with the same PoSpace.

Furthermore, blocks like B3 which come parallel with another foliage block (B2) do not have to sign the previous foliage block, since they do not necessarily have enough time to see it. By “coming in parallel”, we mean that the second block’s signage point occurs before the first block infusion point. The red arrows in the diagram represent a foliage pointer that is signed by the plot key for the proof of space in that block. The gray arrows represent a hash pointer which is not signed by the plot key (therefore the gray arrow in B3 can be replaced if B2 changes or is withheld). This prevents attacks where B2 modifies their block and forces B3 to reorg.

Blocks which have red pointers are also eligible to create transactions, and are therefore called transaction blocks. A block is a transaction block if and only if it is the first block whose signage point occurs after the infusion of the previous transaction block. sp3 comes before B2, (a transaction block, and the previous block of B3), so B3 cannot be a transaction block. The red arrows provide security by burying foliage blocks, but the gray arrows do not. The purpose of the gray arrows is to maintain a linked list in the foliage, and to reduce complexity in implementations. However, blocks with gray arrows pointing to them do get buried in the next-next block.

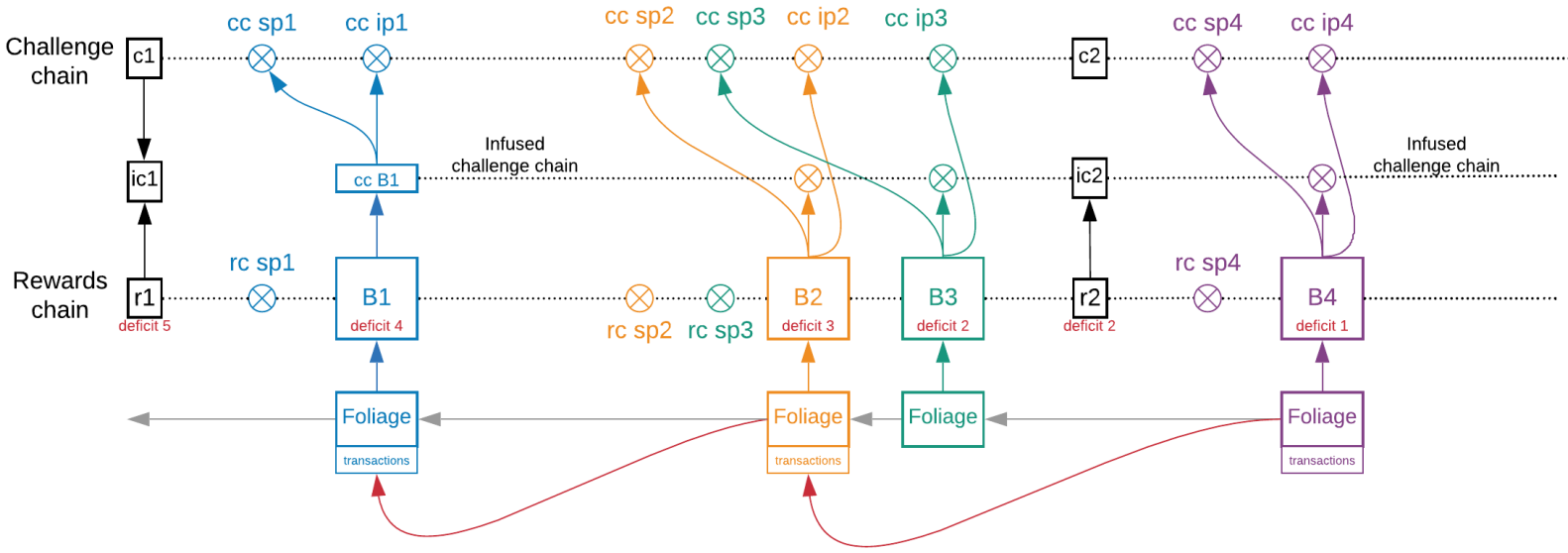

Figure 11: Foliage blocks and blocks. Blocks have transactions and have red pointers (pointers to last block). Note that the start of the sub-slot is also a signage point.

The block hash is a hash of the entire foliage and trunk block. Reorgs work on block hashes. Even if we see a chain with the same proofs of space and time, as long as the foliages are different, the blocks are different. Note that both farmers (B2 and B3) might have a chance to create the block, so they must both provide the signed pointer and transactions. However, any transaction block can be included as a normal block as well, and since B2 and B3 are in parallel, only one of them can make a transaction block.

While all blocks still choose the puzzle hashes of where their rewards go, those transactions do not get included into the blockchain until the next transaction block.

For the chia mainnet, there will be 32 blocks every 600 seconds, for an average block time of 18.75 seconds. There will be 64 signage points, so the minimum time between blocks is 3*600/64 = 28.125 seconds. This puts the average transaction block time at 46.875 seconds.

Epochs & difficulty adjustment

Sub-epoch: Sub-epoch N starts when sub-epoch $N-1$ ends (except for 0th sub-epoch), and it ends at the end of the first slot where $384 * (N+1)$ blocks have been included since genesis.

Epoch: Epoch N starts when epoch N-1 ends (except for 0th epoch), and it ends at the end of the first slot where $4608 * (N+1)$ blocks have been included since genesis.

Difficulty: A constant that scales the number of iterations for a given proof of space. Iterations are computed as difficulty / quality.

Every 4608 blocks, the difficulty adjustment kicks in. This modifies two parameters: The slot_iterations parameter, and the difficulty parameter.

The sub_slot_iterations parameter is reset so a 300 second slot requires close to slot_iterations many iterations. The reset is done using the values from the last epoch to approximate the iterations per second ration, concretely.

For an epoch let epoch* denote the slightly shifted period where epoch* starts with the last block that was infused before the epoch starts, and ends with the last block that was infused in an epoch. The values t1,i1 and w1 denote the timestamp, iterations since genesing, and weight since genesing at the beginning of epoch*, (t2,i2,w2) are the values at the end of epoch*.

$$\begin{aligned} iterations\ per\ second &= floor\left(\frac{\#\ iterations\ in\ last\ epoch}{duration\ of\ last\ epoch}\right) \\ &= floor\left(\frac{i_2 - i_1}{t_2 - t_1}\right) \end{aligned}$$

That is, the delta in total iterations from the start to the end of the epoch, divided by the delta in timestamps, i2, is the total iterations of the infusion point of the last block in the epoch. i1 is the total iterations of the infusion point of the last block in the previous epoch. Sub-slot iterations is the total number of iterations per sub-slot.

$$\begin{aligned} sub\ slot\ iterations &= iterations\ per\ second * 300 \\ sp\ interval\ iterations &= floor\left(sub\ slot\ iterations / 64\right) \end{aligned}$$

Note that we don't take the iterations and time exactly at the end of an epoch, but at the last infusion point of a block in an epoch, the reason being simply that we only have timestamps available when blocks are infused.

$$weight/sec\ of\ last\ epoch = \frac{(new\ weight - old\ weight)}{duration\ of\ last\ epoch} = \frac{w_2 - w_1}{t_2 - t_1}$$

$$new\ difficulty = \frac{weight/sec * target\ seconds}{target\ number\ of\ blocks} = \frac{w_2 - w_1}{t_2 - t_1} * \frac{(4608/64) * 300}{4608} = \frac{w_2 - w_1}{t_2 - t_1} * 4.6875$$

This can be rearranged to use only one floor division:

$$new\ difficulty = floor\left(\frac{75 * (w_2 - w_1)}{16 * (t_2 - t_1)}\right)$$

The sub-slot iterations are adjusted such that each slot lasts around 600 seconds. The difficulty is adjusted such that every challenge gets 32 blocks on average with less iterations than the slot_iterations. It is important to note that the VDF iterations per slot is not material to the weight. That is, if there were two identical worlds where VDF speeds were equal and space was equal, but the sub-slot iterations parameter was 2 times higher in one world, then the blockchain with the higher sub-slot iterations would get twice as many blocks included per slot, but each slot would take twice as long, so the weight per second added to the chain is the same in both cases. Another way to look at it is that increasing sub-slot iterations increases the number of blocks per slot, but it also makes slots last longer, and thus has no effect on weight / second.

Sub-epochs

As described earlier, the challenge chain is completely separate and does not refer to anything in the rewards chain. If these chains stayed separate forever, an attacker with a faster VDF would be able to look into the far future and predict challenges. The attacker can create one block per slot, with limited space, thus creating a whole challenge chain. This would allow them to create plots and instantly create proofs of space for these plots that will win in the future, and then delete the plots (a long range replotting attack). This way, they can fill their reward chain and increase their weight.

The solution to this is to periodically (every 384 blocks, which is an average of 2 hours) infuse the reward chain end of slot into the challenge chain. This means that the attacker can only perform the replotting attack for a few hours into the future. Plotting itself takes a few hours, but even if the attacker could replot

instantly, the cost of a replotting attack will outweigh the benefits. We infuse not the current reward chain output, but the reward chain output of the end of the previous sub-epoch (2 hours ago).

The cost of creating a plot includes the electricity to calculate all of the tables, the RAM necessary while creating this plot, and the fixed infrastructure costs (space, power, cooling, etc). Assuming the worst case scenario of a super fast VDF, and instant ASIC plotting - the benefits would be equivalent to the benefits of storing that plot on a HDD for a few hours. It is clear that this attack is not worthwhile, and that storing the plots is much cheaper (analysis below).

The above explains why the sub-epoch interval should be kept relatively low. But why can't we further reduce it to lower than 2 hours to further disincentivize replotting attacks? The reason is that whenever non-canonical data is infused into the challenge chain, an opportunity for grinding occurs. This means an attacker can possibly choose to include or exclude blocks to manipulate what the challenge will be 2 hours into the future. If this time is too short, they can gain a small space advantage by doing this more often.

The second purpose for sub-epochs is to act as checkpoints in a flyclient-like protocol explained below, to increase efficiency of light clients.

Light client verification

Light client support is another benefit of proof of space when compared with proof of stake, since all proofs can be objectively verified cryptographically, and require controlling an actual resource at a certain point in time.

For light clients who want to sync up quickly to the chain (for example mobile wallets), a full node can create a small sized proof that can convince the light client that the weight of a chain is close to some value. This is called a **proof of weight**. Naively, the light client can download every single block and all the required proofs and verify them, but with such a large number of blocks, this would require a lot of bandwidth and CPU.

A more efficient method relies on a protocol similar to Flyclient[4]. The node (prover) sends all the sub epoch summaries from the fork point, which include difficulty resets, to the light client. There is only one every 384 blocks, so this can only reach a few MB of data. The node also deterministically samples several sub-epochs based on the challenge of the last block. Sub-epochs have a chance to be chosen proportional to the difficulty during that sub-epoch. For the chosen sub-epoch, the light client downloads one of the challenge chain blocks (which are approximately 1/32 of all blocks), and computes the average infusion iterations of all challenge blocks in that sub-epoch. Based on this time, the light client can extrapolate how many blocks the reward chain contains. For example, if the challenge blocks all occur with very small iterations (close to the beginning of the slot), there are likely many blocks in that slot. Conversely, if the iterations are close to the middle of the slot, there is likely only one block per slot. This allows the light client to only download 1/32 of the blocks in each slot, but still get a good estimate of the total weight.

Furthermore, the last few sub-epochs should be downloaded in full for the light client. This adds a small amount of data, but prevents attackers from creating small forks at the end of the chain. The main difference between this protocol and flyclient is that blocks are not committed to using a merkle mountain range, but instead the light client downloads the entire list of sub-epoch hashes from genesis,

guaranteeing that the queried sub-epochs are included in the chain. Another difference is that entire sections are downloaded, as opposed to individual blocks.

More analysis needs to be done on how many sub-epochs should be downloaded and what the bounds are for what the proof of weight implies.

Pooling

Pooling in Chia is designed to be both extremely simple, and more decentralized than pooling in Bitcoin/ethereum. In Chia, the pool public key is embedded into the plots, to prevent the farmer from stealing rewards from the pool by participating in more than one pool. The farmer downloads the pool's address, along with their signature. A farmer periodically sends partials for proofs of space that have less than T iterations, where T is chosen by the pool.

When the farmer wins a block, they submit the farmer signature and the pool signature. The transaction fees, along with ⅛ of the block reward go to the farmer, while ⅞ of the block rewards go to the pool. The reason for giving part of the reward to the farmer is to disincentivize attacks where one pool attacks another one by "pooling" for them, but not actually submitting the winning proofs. This is an attack that can make the other pool go out of business.

This is simpler because the pool does not need to do anything apart from posting their signature once on a website, collecting partials, and periodically making payouts. It is more decentralized because the blocks are made by the farmers, so large centralized pools have little control over the network and that increases resistance to transaction censorship.

A second more complicated pooling protocol will allow you to specify a singleton smart contract in which to store the pool address. The plots would then include the puzzle hash of the smart contract, allowing farmers to switch pools at any time, with a delay. The drawback of this pooling protocol is that an on-chain transaction is required to start farming, and thus it is not strictly better than the first pooling protocol.

Timelord algorithm

A timelord keeps track of the current peak which includes an infused block at a certain height, and signage points from the peak onwards. A timelord might receive new blocks to infuse, new peaks (blocks which are already infused), or new signage points.

How does a timelord decide which challenges to create proofs of time on, given a limited number of available processors? While ASICs are likely to develop in the future, at the moment the fastest classgroup VDF implementations are on general purpose hardware as it appears that the classgroup VDF is FPGA hard. Furthermore, even after the development of ASICs, it's important that any user with a CPU can be a timelord, to provide fallbacks in the case that the ASIC timelords go down, or become malicious, etc.

In general, timelords work on the heaviest chain. They create proofs of time at the signage points, and broadcast these to the network as they reach them. They also infuse blocks as often as they can. When the timelord receives an infused block which has a greater weight than the current peak, they switch to it immediately.

Timelords also run the three VDF chains in parallel. Therefore at least 3 fast CPU cores are necessary to advance the blockchain at an efficient rate. Extra CPU cores will be necessary to create proofs at an efficient rate, but they do not have to be as fast.

If the timelord receives a challenge with less weight than their current peak, they ignore it.
If the timelord receives a challenge point later in the current chain, the safe thing to do is to ignore it. The reason is that by switching to a point further in the future, the timelord might be skipping infusal of blocks, and thus orphaning valid blocks.

If the timelord receives a block for infusion which is late (we have already reached the challenge point at which the block should have been infused), we ignore this, since switching to it would allow attackers to withhold blocks [TODO expand]. Therefore the main operation of the timelord involves keeping a cache of future blocks to infuse, broadcasting challenge points when they are reached, and infusing blocks when we reach their challenge points.

If the timelord receives a challenge with equal weight as the current peak, they choose the unfinished block which they saw first (that is, the block that has not been infused yet), as opposed to choosing the infused block (peak) which they saw first. This also disincentivizes withholding of blocks.

Relevant attacks and countermeasures

51% (46%) attack:

A 51% attack involves creating an alternate chain which eventually reaches a higher weight than the honest chain, and forces users to reorg. The classic long range attack which is also present in proof of work systems is the 51% attack. In the 51% attack, the attacker with 51% of the network space creates an alternate chain and eventually catches up. There are two main differences between Chia consensus and Proof of work: the first is that the attacker can extend and farm on many chains simultaneously. The second is that if the attacker has the fastest VDF, they can get an additional space advantage/boost.

Extending many chains

If an attacker is making their own private chain, they can choose which block gets infused into the challenge chain, and can therefore try many different infusions such that they get the best possible chain. Due to the average of 32 blocks with the same challenge, the attacker can only try about 32 different combinations (which block to include in the challenge chain), and exponentially branching of trying each of these would give a small boost in space for the attacker (Having 5 PiB they can pretend to have 6 or 7, etc). This is because the alternative chains being tried are inferior and less likely to overtake the longest one. This has been analyzed in [1].

The actual amount of space required to perform this attack (for the attacker to get a heavier chain than the rest of the network combined) is 46.3%, due to the ability for an attacker to "try" different combinations of blocks, for example omitting or not omitting the first block. If there was a new proof of space challenge for every single block, the attacker can multiply their space by a factor of e=2.718, where only 27% is required to overtake the network. Setting the number of blocks to 32, increases the attacker's required space to 46%.

The reason for not increasing this further than 32 is the following: if we increased the number of blocks per 10 minute slot to something like 200, then the ability for someone with a slightly faster VDF to orphan others would increase. This is because the time between blocks would get very small. With 32 blocks, the time between blocks is around 15-25 seconds, and a much faster VDF is required to orphan.

Furthermore, the Stanford paper [Tse et. al, 1] shows that increasing the number of blocks per challenge increases security at a very slow rate, so increasing this number slightly does not provide much benefit.

If the attacker were to manipulate the difficulty, they can change it so that they get less reward blocks per slot. Then they can either include or exclude each block, and exponentially extend all chains simultaneously, and they would be able to multiply their space by a small factor [1]. It is not clear whether this attack gains very much, since the attacker must change the difficulty, which requires sacrificing some weight. However, to prevent this attack, there is a requirement that at least 16 reward chain blocks must be created for a challenge block to be included. This brings the required attacker space in the worst case scenario from 27% up to 42%.

Faster VDF and 46% of space

The 46% attack gets worse if the attacker's VDF is faster. Let's assume the attacker's VDF is 2x faster. Then their chain will be able to create challenges and blocks at 2x the rate of the rest of the network, which means they can create a "heavier" chain with the same amount of space.

This required space drops from 46% to approximately 30% of the total network space. 0.46/0.54 = 2x/(1-x). x=0.30. If the attacker does not have access to the fastest VDF, they will not be able to get a space advantage.

Chia space / global hard drive space

There is a concern that if the Chia system does not have a significant amount of space compared to the available free space of hard drive manufacturers or large companies that it will be vulnerable to 51% attacks. Therefore the more space taken by the Chia system, the more secure the network is. One plausible scenario is that a lot of space comes on, making the rewards per TB quite low, and not significant enough to justify buying drives or deleting business data. Furthermore, creating a plot requires a fixed amount of upfront time and money (from current calculations in beta17, about 1kWh for a k32, or about 10 cents, which is $1 per terabyte).

100% attack

If difficulty adjustment was triggered every X VDF slots, as opposed to every X blocks, this would allow for a 100% attack, where all farmers collude to constantly decrease or increase the difficulty. In normal operation, there are 32 blocks per slot. Under a 100% attack, the difficulty is manipulated such that difficulty goes down by 2, so there are 64 blocks per slot, and then goes up by 4, so there are 16 blocks per slot, alternating forever. This would allow farmers to earn on average 64+16/2 = 36 block rewards per slot. This is the reason for making difficulty adjustment based on the number of blocks.

Short range replotting attack

Plotting usually takes several hours (8 hours for a k32 in beta 14 with one core), but it is very parallelizable, so attackers might find ways to create plots after a challenge is released, and then delete the plot, in effect being able to farm without storing the space continuously. This will likely require

expensive specialized hardware with fast memory, since the plot must be created in time for the infusion (less than 30 seconds).

If we assume the worst case scenario of a farmer being able to create a plot instantly, the question becomes, what is the cost and what is the benefit of the attack? The cost is the electricity, memory, hardware and infrastructure cost of creating that plot. The cost of creating 1TB is currently on the order of $1 in electricity costs. The benefit would be the same benefit as storing that plot for 80 minutes (the signage point interval times the plot filter constant). This is because the attacker can choose a plot that passes the plot filter. Assuming $5 per year value per terabyte, the value of a 1TB plot for 80 minutes is $0.00094. Therefore with current plotting software and hardware, it is significantly cheaper to store the plots as opposed to recreating them.

The plot filter constant is very useful to reduce the amount of disk lookups farmers must do. With a plot filter of 512, Instead of 7 disk reads per plot every 9 seconds, farmers only need to do about 7 reads for every 80 minutes. The plot filter constant provides a multiplier of replotting benefit to the attacker, so it must not be set too high. With a plot filter constant of 512, 1/512 plots are valid for every challenge. The attacker can then only create plots that pass the filter, therefore not needing to create the other 511/512ths. Setting it to 512 provides a 512x multiplier, etc.

Faster VDF (but not 51% attack)

With the fastest VDF in the system, an attacker can more effectively perform a 51% attack: i.e expand their space, when farming in a private chain. If the attacker does not reach a total of 51% of space (with the VDF boosting and extending many chains as above), the usefulness of the faster VDF decreases substantially. This is because inclusion and exclusion of blocks does not depend on how fast you can perform the VDF, but instead depends on whether it's less than the sub-slot iterations. Furthermore, an attacker needs the space of the rest of the network in order to advance, and therefore must release the challenges to the network.

In certain cases where blocks come very close together, having a faster VDF can allow an attacker to orphan certain blocks, although this does not increase rewards in the short term, and has a risk of undermining the network in the long term. TODO: expand: bram

Selfish Farming

Selfish farming is an attack where the attacker farms blocks in private, and only releases them when they are at risk of being surpassed by the honest chain. In Nakamoto PoW this provides significant gains, because at any point at which the miner is ahead of the rest of the network, the rest of the network is wasting their hashpower on a chain that will not win. In Chia consensus this is different, due to the 30-40 second delay and the fact orphaning other farmers' blocks does not increase rewards. (??)TODO: expand: bram

Farmer bribe trunk attack

An interesting attack explored by [10] is the bribing attack which takes advantage of the predictability of the elected "leaders" in each slot. The authors analyze a proof of stake chain, and argue that when participants know that they are going to win in advance, there is a potential bribing attack. If participants knew in advance which plots would win, each user can notify an attacker that they are willing to participate in the attack, and if they reach a certain threshold, they can completely reorg the chain (or

orphan those who do not participate, censor transactions, etc). This attack does NOT require the majority of the space in the network to participate; only the winners in that short time period. Furthermore, it is undetectable, since the attacker can make a normal looking chain.

This problem is not present in this revision of the Chia consensus algorithm. This problem is solved by reducing the predictability: each farmer does not know for sure if their proof of space is fully eligible until the signage point. Therefore an attacker must bribe a large majority of the space to pull off this attack.

Farmer bribe foliage reorg attack

Since blocks are signed by PoSpace keys, a farmer can theoretically sign multiple blocks with the same PoSpace, at the same height. The attack requires a malicious party to bribe farmers with a certain amount of funds for them to provide a signature of an alternate chain. If the attacker can convince every single farmer in N blocks to sign, they can revert or reorder any transaction in those N blocks. Potentially fraud proofs could be used, but these were not chosen since they enable other attacks and complicate behaviour.

Instead, the solution is simply to wait longer. After 32 blocks (approximately 10 minutes), the assumption that at least one farmer is following the protocol and not double signing is a reasonable one. If 54% is non-colluding (the assumption for 46% attack resilience), the probability of a reversal after 32 blocks is$1.8 * 10^{-13} = 0.00000000000018$. Furthermore, this attack is detectable so it is not easy to pull off.

Each user can choose their own threshold for which they accept a transaction/block as final. For example, in cases where the total network space drops suddenly, users can be more careful and not consider transactions final, in case there is another existing fork, due to a network split, for example.

Orphaning transaction blocks for transaction fees

Transaction blocks are different from non-transaction blocks, since they contain transaction fees. These may surpass block rewards. At the time of writing (November 2020), in peak defi hype we are seeing 2 eth block rewards with 8 eth fees per block. In Chia this will be more extreme, since not every block contains transactions. This leads to attacks where the 2nd place farmer ignores the 1st place in an attempt to win the transaction block. If the 2nd block comes less than 30 seconds after the 1st, they do not specify the previous block, and thus the 2nd place cannot orphan the 1st. The 3rd place could orphan both, but nobody would follow this chain since it is shorter.

However, if there are no blocks within 30 seconds of the 1st block, the 2nd could orphan the 1st, but they would have to convince the next block to farm on their alternate chain. An easier attack would be if the attacker controlled both the 2nd and 3rd, in which case they could ignore the first and still be longer. These orphaning attacks do not allow the attacker to steal rewards, but rather allow the attacker to slightly lower the difficulty. Since they are very situational and require a lot of space, attempting this attack will likely harm the network more than the potential gain to the attacker.

Orphan Rate

In Chia consensus, two competing blocks around the same time can both be included into the blockchain in parallel, without knowing about each other. (Although at most one can be a block). Since all transaction blocks are also blocks, they are both included into the chain, resulting in a chain with higher weight. This means that the orphan rate in Chia will be essentially zero, assuming low network latency. If network

latency exceeds the infusion delay (30-40 seconds), then orphaning of a block is almost guaranteed, so it is more of a step-function. This is in contrast with Nakamoto-PoW in which the orphan rate is high if there is network delay, and decreases smoothly as network condition improves, but never reaches zero.

Analysis

Safety

The safety is similar to other Nakamoto consensus algorithms like Bitcoin. There is no guaranteed finality, but the more confirmations a transaction has, the safer it is. A transaction needs a certain number of confirmations for a receiver to assume that it cannot be reorged, under the <46%(* vdf advantage) colluding assumption. Since farmers can theoretically sign multiple blocks at the same height, more confirmations should be used in Chia than in Bitcoin. However with a rate of 32 blocks per 10 min, 6 confirmations in Bitcoin is equivalent to 192 in Chia, which is more than enough to be considered safe. As long as one of those 192 farmers is well behaving (not double signing), that transaction will not be reversed.

It's worth noting that there is no requirement of 54% honest farming space, but 54% non colluding. Profit seeking farmers gain very little by deviating from the protocol.

There is the added assumption that at least one fast timelord must be connected to the non-colluding portion of the network, and that the attacker's timelord is not significantly faster.

Liveness

The liveness of the Chia consensus system is one of its greatest strengths. Like Bitcoin, the Chia system continues advancing even when a majority of the space goes offline. Unlike bitcoin though, the system does not slow down significantly when this happens, since not all blocks are transaction blocks. Therefore transactions throughput does not drop by too much if many participants go offline. It will continue even if only 1 farmer is online, although there will be many empty slots, since a transaction block can only be created if it’s below the sub-slot iterations threshold.

Of course, in the event of a long term network split the effects are that one chain must be chosen, so there can be large reorgs in this case. Still, the network chooses the heavier chain, similar to PoW.

Comparison to BFT consensus algorithms

Proof of space could also be used as a Sybil-resistant mechanism in order to bootstrap a Byzantine consensus (k-agreement) system. Filecoin, and many proof of stake systems use aspects of byzantine consensus.
The pros and cons of using Chia Nakamoto Consensus vs Byzantine consensus, which vary from algorithm to algorithm:

- + Much simpler
- + No registration requirement
- + No scalability requirement (scales to millions of farmers)
- + More censorship resistant. As long as a small portion of the farming space does not censor, eventually you can get into the blockchain.
- + No liveness requirements, potentially less network assumptions

- \+ Fully objective (A node can compare chain 1 and chain 2, and immediately know which one is heavier). No need for checkpoints with ⅔ consensus.
- \+ Better light client support [11]
- \- No finality, only probabilistic.
- \- Need to wait longer for transaction confirmations (related to no finality).
- \- Less consistent block times and transaction throughput

Comparison to Nakamoto PoW

- \+ Different resources. PoSpace is ASIC resistant and therefore anyone can participate in farming. Hopefully more decentralized.
- \+ Easy merge farming. Other cryptocurrencies can use the same format, and everyone can share the space. Probably the top one will be the only secure one, since the farmers can attack smaller ones.
- \+ Minimum energy used, since only a few nodes run VDFs, and these are not parallelized. Very low marginal cost to mine.
- \+ More consistent transaction block times (one per ~1 min).
- \+ Less susceptible to selfish mining attacks
- \+ Smaller orphan rates and forks, since blocks can be included in parallel.
- \+ Still advances at the same rate when space decreases, since only 1/16 blocks include transactions. PoW nakamoto consensus slows down.
- \- Drawback of more potential attackers (large companies). Hardware is general purpose, and therefore attackers could switch between farming, attacking, and using for data storage.
- \- Speeding up VDF could give a space advantage for someone attacking the network.
- \- More complexity due to sub slots and VDFs, potentially more cryptographic assumptions

Comparison to Proof of Stake

This consensus algorithm can also be used for proof of stake, where the space farmers are replaced by stakers who own coins in the system. The benefit would be the ability to slash (delete people's stake), and farmers would have "skin in the game", but there are some concerns if proof of stake is used. (+ means benefit for using space vs stake).

- \+ An attacker can transfer their stake to someone else, but fork the chain right before their stake is transferred. In this alternate chain, the attacker still has all of their stake, and can therefore advance the chain. The "nothing at stake" issue is different in PoStake than in PoSpace since creating a PoSpace requires a physical resource (hard drive space), while creating a PoS only requires a key.
- \+ An attacker can guarantee their share of the whole pie, by staking their rewards (the rich get richer), since the total number of coins is limited.
- \+ There might be situations where the attacker can grind on many different ways to transfer stake. Perhaps this can be mitigated by requiring a long period before stake becomes active.
- \+ Registration is required, you cannot participate in proof of stake until you sign up. This reduces privacy and scalability (how many people can stake).
- \+ Higher barrier to entry: security deposits and slashing make it difficult for small users to participate. Slashing can be a huge risk for participants in the network. Centralized custodians lead to a less distributed set of participants.
- \+ Some assumptions [11] are required to perform light client syncs in proof of stake.

- - Skin in the game: with PoStake, the consensus can slash people's stake, and also requires some investment into the system (exposure to price). In Proof of space hard drives can be used for other purposes and there is no ability to "slash" peoples hardware.

References

1. Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, Ofer Zeitouni, Proof of Stake Longest Chain Protocols, Security vs Predictability
2. Aggelos Kiayias, Alexander Russell, Bernardo David, Roman Oliynykov, Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol
3. Bram Cohen and Krzysztof Pietrzak, The Chia Network Blockchain
4. Benedikt Bunz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani, Flyclient
5. Krzysztof Pietrzak, Efficient Verifiable Delay Functions
6. Benjamin Wesolowski, Simple Verifiable Delay Functions
7. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak, Proofs of Space
8. Hamza Abusalah, Joel Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin, Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space
9. Chia Network, Chia Proof of Space Construction
10. Soubhik Deb, Sreeram Kannan, David Tse, PoSAT: Proof-of-Work Availability and Unpredictability, without the Work
11. Alexander Skidanov, Light clients in Proof of Stake Systems