Succinct, Non-Interactive Share Proofs

Kirat Singh[1]

[1]GetBlok.io

kirat@getblok.io

Abstract. *Proof-of-work (PoW) based consensus protocols rely on the process of "mining" to secure their blockchain. Although mining is a crucial process for the security of such blockchains, there are few ways to trustlessly and efficiently calculate the amount of work a miner has performed. Such an ability would allow for the creation of new decentralized applications which can integrate the process of mining into their protocols. In this work, we construct a new primitive known as a Succinct Non-Interactive Share Proof (SNISP) which allows for efficient calculation of a miner's work. Our construction functions using a worst-case constant space complexity, regardless of the total number of shares of work that a miner submits. We pull heavy inspiration from previous works on Non-Interactive Proofs of Proof of Work (NiPoPoWs).*
Using SNISPs, one can prove that a miner performed work using smart contracts on programmable blockchains such as Ergo and Alephium. Moreover, we show that our construction is blockchain agnostic, meaning it may even be used to prove work was performed on non-programmable PoW blockchains such as Bitcoin and Flux. We later show how our construction may be applied to different decentralized applications, especially the creation of efficient decentralized mining pools.

1. Introduction

PoW-based blockchains such as Bitcoin and Ergo use the process of mining to ensure security, as well as being the point of creation for new tokens to become liquid on their ledgers. Mining is the constant application of hash functions to block candidates in order to randomly find new block solutions which may then be placed onto the blockchain. Miners who place valid blocks onto the chain are then rewarded with some amount of the blockchains native asset as an incentive to further secure the blockchain. The random nature of mining means that miners with less powerful hardware are unable to use it alone as a steady source of income.

To combat the instability of miner rewards, especially for miners with less powerful hardware, mining pools have been made to concentrate the hashing power of multiple miners in order to find blocks and their rewards quicker. Most mining pools however, are centralized in nature, and require clear trust in the operators of such pools. Moreover, mining pools represent a point of centralization within the blockchain, and allow for operators to potentially attack the protocol for their own benefit. In fact, there exist multiple instances of mining pools using their concentrated hash power to perform 51% attacks and jeopardize the security of the chain.

Decentralized mining pools offer a solution to the above problems. Decentralized mining pools allow for trustless distribution of miner rewards to the miners participating

in the pool. Some examples include P2Pool, which uses it's own sidechain to record the amount of work each miner has performed. P2Pool however, can be shown to be insecure due to its vulnerability to 51% attacks. Its security is therefore limited by the total hashrate present on the pool itself.

Another example of a decentralized mining pool is SmartPool, a protocol using smart contracts on the Ethereum blockchain to provide more security than a sidechain like P2Pool. By leveraging smart contracts, SmartPool is able to inherit the security of the Ethereum blockchain itself. The problem with SmartPool is its inability to scale, specifically that its protocol is interactive in nature. Ethereum itself has had many instances of significantly high gas fees, meaning that an interactive protocol would require miners to submit multiple transactions and therefore cannibalize their own rewards due to the transaction fees that must be paid.

Our motivation for research into SNISPs is to create an efficient and secure decentralized mining pool, which is able to trustlessly verify the amount of work a miner has performed. By using SNISPs, we eliminate the need for trusted operators which may be able to steal miner rewards or potentially perform 51% attacks on their blockchain.

We wish to target three key properties pertaining to our SNISP construction, to ensure its efficiency when used in the context of a smart-contract based decentralized mining pool:

1. SNISPs should be sufficiently succinct. Specifically, we require a worst-case space complexity that is constant, so as to allow easy evaluation of proofs on the blockchain.
2. Proofs must be submitted and evaluated in a non-interactive protocol, so as to minimize the number of transactions needed to be sent by miners.
3. Block-to-block variability in rewards should be minimized, and two miners with the same hashrate should receive similar rewards.

2. Background

Before delving into SNISPs, we would like to briefly explain the processes behind PoW mining, along with how mining pools work, and common terms used in this area of research. Later, we explain algorithms used by traditional blockchains to evaluate work via share submissions.

To begin, we start with the PoW process that is used by many blockchains today. PoW blockchains define a certain target T, a hash function H and some value w which is derived from a potential block candidate. Valid solutions must satisfy the inequality $H(w) < T$ in order to be accepted onto the blockchain. To efficiently satisfy this inequality, miners must constantly apply H to different values of w until a solution is found. It should be noted that the target may sometimes be represented as *difficulty*, which is just the average number of hashes needed to find a block at a certain target.

The rate at which H may be applied to w is known as a miner's hashrate. Hashrate is limited by the physical computation power of a miner's hardware, and so cannot be artificially increased. Miners with higher hashrates may perform more computations in the same amount of time compared to miners with lower hashrates. Hashrate is therefore directly related to the chances of a miner satisfying the inequality and submitting a solution onto the blockchain within a certain amount of time.

Due to the random nature of PoW mining, as well as the constant adjustment of T in order to reach stable average block times, miners with lower hashrates are unable to consistently find blocks in a short time span. To solve this problem, mining pools have been created to combine the resources of multiple miners in order to find blocks quicker. Mining pools then distribute these block rewards to miners proportionally according to the amount of work each miner performed while searching for solutions. In traditional mining pools, each miner's work is measured using shares, which are applications of H which do not satisfy the inequality given by the blockchain. Shares instead satisfy the inequality $H(w) < T_p$, where T_p is a special target set by the mining pool such that $T_p > T$.

2.1. Share Processing

Shares are constantly being submitted by miners on pools in order to prove that valid work was performed by them. These shares are then incorporated into different share processing algorithms which output a number which shall now be referred to as the share score. The share score provides a simple scalar value which allows the pool to derive miner payouts by taking the proportion of the miner's share score to the share score of the entire pool.

One common share processing algorithm is known as the Pay-Per-Share algorithm (PPS). The PPS algorithm pays miners proportionally according to the number of shares they submitted divided by the pool target T_p. Specifically, we define the equation $s = N/T_p$ where s represents a miner's share score, N represents the number of valid shares submitted by the miners, and T_p represents the pool target given to the miner by the pool. Although in many cases T_p may be a constant set by the pool, some pools incorporate "variable difficulty" mining to allow for efficient share submissions that lower network load. We have included T_p into the equation to show that miners working on shares with a lower target are given a higher score per share submission.

The PPS algorithm is used by many mining pools today. However, one strict disadvantage of it is that the algorithm is susceptible to pool hopping attacks. To prevent such attacks, alternative algorithms such as Pay-Per-Last-N-Shares (PPLNS) and Pay-Per-Last-N-Timeshares (PPLNT) are used. These algorithms both work using some window W in which submitted shares may be used in processing. For the PPLNS algorithm, W is a constant value representing the last N shares submitted to the pool before a certain block was found. In PPLNT, W is a set period of time before a certain block is found. In both of these algorithms, shares must be within W in order to be processed into a share score by the algorithm. A miner's share score for both of these algorithms can be derived by the equation $s = N_W/T_p$, where N_W represents the miner's shares present within the window.

3. Initial Construction

It is apparent from the above share processing algorithms that share scores are usually some proportion of the number of shares divided by the pool target. Although this works well for traditional mining pools, such an equation would struggle to work in a decentralized setting. For one, data storage is limited on programmable blockchains, having a miner submit the total number of shares created while mining is unviable. High target

shares could easily become hundreds of thousands of bytes of data to parse. On the other hand, submitting a small amount of low target shares can mean large variability in profits, which would defeat the purpose of the mining pool itself.

To create functional and secure SNISPs, we must look to other places to draw inspiration from. Specifically, let us analyze NiPoPoWs as proofs that achieve similar levels of succinctness and security in PoW blockchains.

3.1. NiPoPoW Background

Non-interactive Proofs-of-Proof-of-Work represent succinct proofs that may be generated from certain PoW blockchains, and which may prove various predicates about the blockchain itself. NiPoPoWs function by using blocks with especially low targets (called *super-blocks*) as checkpoints to show that work was done on the blockchain. Specifically, these proofs focus on the property of *goodness*, which is the tendency of block hashes on an honestly-mined blockchain to follow a Binomial Distribution. Generally, the distribution of super-blocks must be sufficiently good for a NiPoPoW to be considered valid. In the case of a malicious adversary, "certificates of badness" may be used to justify why a certain NiPoPoW may lack goodness.

Although the problems that NiPoPoWs attempt to solve may differ from our own goals with SNISPs, it is evident that the properties of PoW that are used in the prior setting can also apply to our own. Specifically, we can see that:

1. PoW blockchains and PoW Mining Pools both have a threshold target for valid hashes.
2. Mining pools operating on a certain blockchain must ensure their miners use the hashing function H in order to create valid block solutions.
3. Applying H to w ideally produces a random value
4. Each iteration of a unique nonce over H can be considered an independent Bernoulli Trial, regardless of the threshold target used.

Given these properties, and in order to ensure the succinctness of our share proofs, we shall focus our attention onto goodness. Specifically, we wish to evaluate a set of shares based on how good the set is, and whether or not it constitutes a valid, target-based binomial distribution.

Ensuring goodness in the context of a decentralized mining pool is actually easier than ensuring it on a blockchain. Attempts to attack goodness on NiPoPoWs all require an adversary who is withholding super-blocks and making the blockchain inherently bad, *i.e*, not good. No such problem exists here, as each miner is submitting their own set of hashes that they independently computed while performing work. Moreover, Lemmas 2 - 5 prove that an honest miner has an overwhelming probability of creating a set of hashes which are good.

With this in mind, we propose a share proof that is evaluating a miner's work based on the goodness of their share set, rather than the number of shares submitted. Moreover, we shall ensure succinctness by extending the concept of super-blocks to *super-shares*, which shall represent shares whose value is sufficiently lower than the threshold target given by the pool. We'll similarly use the idea of *levels* to represent subsets of super-shares.

3.2. Model

For clarity, we will now describe our model for miners performing PoW mining on a decentralized mining pool. To begin, we assume that all actors in our mining pool are *rational*. Miners on the pool are constantly mining, and all miner's are assumed to have stable hashrates with few fluctuations. Everytime a block is found by a member of the pool, all miners in the pool must submit a share proof to a smart contract which then evaluates the proof. Each new block find on the pool constitutes a new *round* in our protocol. Each miner is therefore a *prover*, while our ideal smart contract is a *verifier* which is independently validating the share proofs given to it. If a miner's share proof is considered invalid, they gain no reward for the work they performed.

All miners in the pool are producing shares with a personal maximum target τ. τ can be seen as the threshold that divides hashes from shares, similar to how the blockchain target T divides hashes from blocks. For a hash to be considered a share, it must satisfy the inequality $H(w) < \tau$. Shares created in this way are constantly being placed into some share set S. Upon the start of a new round, miners will submit some share proof P to the verifier in order to receive their mining rewards. The value of τ is set by each miner as they see fit, and is submitted in conjunction with P. In order to ensure consistency of mining rewards, we propose that in an ideal scenario, two miners mining with the same hashrate should each have equal values of τ submitted with their share proofs.

We shall adapt the idea of super-shares so that S_i represents all super-shares at level i in the share set S. In our model, miners will be aiming to prove that they performed the work of creating N shares whose hash value is less than the target τ. In addition, the shares submitted in P must be made within some window of time W before the end of the round. The addition of W ensures that miners may not submit old share proofs that have no relevance to the work currently being performed by the pool.

In order to ensure the succinctness of our share proofs, we will use a security parameter μ, which represents the maximum number of shares which may be present in the share proof. W, μ and N will be global parameters that all miners in the pool must follow. We therefore define a level i to be a subset of shares in S such that for each share s in S_i, the value of s conforms to the inequality $s < \tau \times (i/N)$.

For a share proof P to be considered valid, the number of shares present in the proof must be less than or equal to μ. In addition, P may only be considered valid if all shares present in the proof are at some constant level i. To prevent what can be considered unnecessary information, a proof P at level i must have exactly i shares inside of it. These constraints mean that the maximum size of a proof P is μ shares, and that the maximum target for each share in a proof of length μ is $\tau \times (\mu/N)$. They also show that μ acts not only as a security parameter ensuring succinctness, but also as the maximum level of shares that a miner may submit.

We can now finally say that the goal of each miner on the pool is to mine N shares and place them into the share set S. Upon the end of a round, miners will find the best subset of super-shares in S, and form the proof P. For P to be valid, each super-share s in a proof of length i must satisfy the inequality $s < \tau \times (i/N)$. To ensure succinctness, we constrain i to the domain $1 \leq i \leq \mu$. We can therefore say that miners here must only prove goodness at a single level i in the entire share set S in order to have a valid proof.

Finally, we add the constraint that each share s must have been made within a window of time W before the end of the round.

3.3. Evaluation

From the above construction, we can see that goodness of the share set is evaluated by having a miner submit a set of sufficiently low target super-shares which are all existing at some level i where $1 \leq i \leq \mu$. It is also apparent that our share proofs have a worst-case constant space complexity, due to the limit on data size that μ is enforcing. It is clear then that this model is able to achieve both succinctness and proper evaluation of goodness. As one final addition, we shall create a method to convert our share proofs into share scores which may be used to award miners proportions of the block reward. The equation to find a miner's share score given a valid share proof is simply $2^{256}/\tau$. The constant 2^{256} is derived from the maximum value of a 256-bit hash (which blockchains such as Bitcoin and Ergo employ in their PoW algorithms). Keen observers may notice that this in fact the equation to calculate *difficulty*, which can be considered a different representation than the target. We use difficulty here for it's cleaner representation of the target. We omit N unlike previous share score equations, due to N being a protocol-wide constant.

As shown above, the most important variable affecting a miner's reward is τ. Although this initial model correctly evaluates whether or not an honest miner is performing work with a share target threshold at τ, there is an issue here which jeopardizes the consistency of payouts. Because τ is submitted in addition to P, adversarial miners may simply use their lowest target hash to represent P. Accordingly, these adversaries may then adjust τ so as to satisfy the above goodness constraints, while still earning more than honest miners working at the same hashrate as them. In this way, dynamic adjustment of τ after the end of a round allows two miners with the same hashrate to obtain wildly different payouts. We will henceforth refer to attacks of this nature as *target adjustment attacks*.

4. Optimizing τ

In this section, we aim to append our initial construction so as to make it resistant to adversaries aiming to maximize profits via target adjustment attacks. Specifically, we can add a simple change which prevents adjustment of τ after a round ends. To achieve this goal, we now require that all miners on the pool must make an initial commitment to τ, such that the τ submitted with P must remain constant for some minimum number of rounds r. Each time a miner wishes to change their value of τ, they must wait at least one round for their update to take affect. In addition, every update to τ requires that a miner must wait r rounds before submitting a new commitment to their τ value. In the context of a smart-contract verifier, miners will be making a cryptographic commitment to their value of τ via a pool-wide authenticated dictionary, such as an AVL+ tree.

The affects of this change are plain to see, miners can no longer freely adjust τ to maximize rewards based on their lowest target shares. Instead, miners now receive share scores based on their commitment for at least $r + 1$ rounds before being able to change their τ value. Let us now analyze how our appended construction functions when a miner with some arbitrary, stable hashrate commits to different values of τ.

4.1. Idealized τ

We first propose that for any hashrate a miner may have, there exists a corresponding value of τ which will on average produce N shares within the window of time W. This ties back into our previous statement that two miners with the same hashrate will ideally have equal values of τ. For a miner who has committed to their ideal τ, we theorize that the miner is exceedingly likely to produce a valid proof. We base this claim using the fact that an honest miner's share set is overwhelmingly likely to follow a binomial distribution.

4.2. Inflated τ

We define an inflated τ value to be any value of τ which is deviating above a miner's ideal τ. Miners who commit to inflated τ values are, on average, producing more than N shares every round. We argue that there is no incentive for rational miners to commit to an inflated τ value. Given that share scores are calculated via the equation $2^{256}/\tau$, any miner who commits to an inflated τ value is receiving a smaller share score than other miners in the pool who have the same hashrate. It should also be known that as τ deviates higher from its ideal value, a miner is increasingly likely to create a good and valid share proof, even more so than miners using their ideal τ.

4.3. Deflated τ

We define a deflated τ value to be any value of τ which is deviating below a miner's ideal τ. Miners who commit to deflated τ values are, on average, producing less than N shares every round. Commitment to such a τ value increases a miners expected share score, however, it also increases the likelihood that a miner is unable to produce a valid share proof for a given round. We base this claim again on the fact that S represents a binomial distribution, and that there is a real hardware-based limit on the number of hashes a miner may produce within the window.

First, let us remember that each hash produced by a miner represents an independent Bernoulli trial, and that decreasing the target also decreases the probability that any given hash satisfies the SNISP inequality. Given that the total number of hashes a miner may produce in the window is a constant value, we can see that the only result of a miner deflating their τ is reduced probability of creating a valid share proof in a single round. We therefore argue that there is little incentive for rational miners to deflate their τ value. Mining pools are created in order to deliver consistent payouts to miners. By setting τ lower than its ideal value, miners are needlessly increasing payout variability, which therefore defeats the purpose of pool mining itself.

Although miners have no incentive to commit to significant deviations from their ideal τ, there may be some miners who make small adjustments according to their personal risk-reward ratio. Such changes have little affect on other miners, and can instead be seen as a benefit of the protocol by placing more power into the hands of miners.

5. Concrete Example

From our analysis, we can see that rational miners are likely to be mining at or near their ideal τ value. With this, we have achieved our goals of succinctness, and consistency of miner rewards. It can be argued that non-interactivity has been violated by requiring that miners commit to their value of τ. However, given that miners are operating under

the assumptions of stable hashrates, and that the parameter r limits the number of times τ may be changed, rational miners are unlikely to change their τ value often. We can therefore say that our proofs are non-interactive for all rounds after the initial ideal τ value is committed to. Now, we will go into an example with set parameters to provide a clearer explanation of the protocol.

Let's imagine a decentralized SNISP pool, which is operating with the parameters $N = 10000$, $\mu = 20$, $W = 3600$ seconds, and $r = 10$. A new miner, with a consistent hashrate h of 500 MH/s wishes to join the pool. First, the miner must calculate their ideal τ and submit it to the pool smart contract. We can calculate the ideal τ by finding the number of expected hashes within the window, dividing it by N, and multiplying it's inverse by 2^{256}, which is the maximum value for a 256-bit hash (Used by Bitcoin, Ergo, and other blockchains).

Our miner is creating 5×10^8 hashes every second. Within the window, the miner is expected to create a total of $h \times W$ or 1.8×10^{12} hashes. We can then say that for our miner, $\tau = (2^{256} \times 10000)/(1.8 \times 10^{12})$, or 6.4328938×10^{68}. The share score (which may also be seen as a miner's threshold *difficulty*) received for mining at this τ can simply be expressed as 180M or 1.8×10^8.

As stated in our model, for a share proof of level i, a miner must submit exactly i shares within P such that each share s satisfies the inequality $s < \tau \times (i/N)$. Given these parameters, valid share proofs would include 1 share with value $s < (\tau/10000)$, 2 shares with values $s < (\tau/5000)$, and 3 shares with values $s < (\tau/3333)$. For the maximum size share proof of length μ, 20 shares must satisfy the inequality $s < (\tau/500)$.

6. Empirical Data

7. Applications

The most obvious and discussed application of SNISPs is their usage in decentralized mining pools. For a smart contract based mining pool, SNISPs allow calculation of a miner's work into a single share score, which can then be used to derive each miners reward proportional to the amount of work the miner did. There could be multiple designs for SNISP pools which each have their own benefits and drawbacks. One design of interest may be the usage of an *optimistic rollup*, which could have great scalability benefits. Assuming miners are honest and using *fraud proofs* in order to prove them wrong would allow for less computation on the blockchain, and would ensure that a SNISP-based pool can scale to thousands of users. Such a method would be impossible using past decentralized mining protocols such as SmartPool and P2Pool.

Because SNISPs are based on the standard PoW inequality used by blockchains and mining pools, SNISP pools may be deployed for any standard PoW-based blockchain. One caveat is that although SNISPs can be used for any blockchain, they must be deployed in smart contracts to ensure full security guarantees. SNISP-based pools may therefore be created on blockchains such as Ergo and Alephium, both of which represent programmable PoW blockchains. Pools on these blockchains may then service other PoW blockchains which lack programmability. For example, a SNISP pool on Ergo could potentially serve as a decentralized Bitcoin or Flux mining pool. The integration of the standard PoW inequality ensures that SNISPs remain blockchain agnostic.

Another potential idea may involve the creation of SNISPs which use a different share calculation algorithm. The SNISPs presented in this paper most closely resemble the PPLNT share processing algorithm, however application of SNISPs into algorithms like PPS may also be interesting.

Other usages of SNISPs may involve smart-contract based lending of hashrate and direct work-based governance on decentralized applications. In reality, any application which provides the right incentive structures and incorporates work can use SNISPs within their their protocol.

References