OCEAN: A Built-In Replacement for Mining Pools

Raymond Chee, Kartik Chitturi, Edouard Dufour-Sans, and Kyle Soska

Carnegie Mellon University
{rchee,kchittur,edufours,ksoska}@andrew.cmu.edu

Abstract. We propose OCEAN, an alternative miner reward system for blockchains that seeks to discourage pooling by providing a pool's variance mitigation functionality as a blockchain built-in. Our proposal relies on Succinct, Non-interactive Arguments of Knowledge (SNARKs), an advanced modern cryptographic tool that enables anyone to prove complex statements with the proof not growing in size with the complexity of the statement. We expect that blockchains that implement our proposal would see less pooling centralization than what is currently observable in deployed cryptocurrencies.

Keywords: Cryptocurrencies · Mining Pools · Risk · SNARKs.

1 Introduction

Blockchains were introduced by Satoshi Nakamoto in the context of the Bitcoin cryptocurrency [8] with the aim of decentralizing electronic payments. Since then, many other applications have been proposed [5, 9, 11] that aim at using blockchains to provide decentralized alternatives to useful services. The picture of the current blockchain ecosystem, however, is bleaker.

1.1 Problem

Centralization is a major concern in today's blockchain ecosystem. Centralization occurs along many axes including governance, development, and currency ownership. In this work, however, we focus on the issue of pooling centralization in Proof of Work blockchains. Mining is heavily centralized through pooling in many major deployed cryptocurrencies.

Mining pools are conglomerates of individual miners who cooperate in order to reach a higher total share of the network and hopefully experience smoother returns than they would on their own. While often those pool participants are simply small miners following their self-interest, they aggregate to large amounts of hashing power that end up under the control of a few pool managers.

This is undesirable for the blockchain ecosystem for a number of reasons. First, as is natural in a market, and despite blockchain enthusiasts' dislike of centralization, a few pools end up dominating. This means that miners who don't want to suffer unreasonable variance in their rewards are forced to deal with one of a limited set of parties, thereby weakening a blockchain ecosystem's claim to openness. The managers of those large pools will also necessarily have large influence in the ecosystem and might be able to influence blockchain governance decisions. That might include blacklisting certain blockchain addresses so as to prevent them from redeeming their coins via, e.g., feather-forking [10]. Second, the centralization of *de facto* control on computing resources means pools have the ability to wreak havoc by misbehaving for a short time, and potentially colluding to carry out 51% attacks. Although misbehaving pools would likely see their miners depart quickly, most miners are not monitoring the blockchain in real time, giving pools a chance to cause meaningful damage.

1.2 Technical overview

Our goal is to provide the same reduction in payment variance as that of mining pools, but with the miners only interacting with the blockchain, rather than with a centralized pool manager. We suggest that miners, instead of presenting near hashes to a mining pool will present them to the blockchain itself. To avoid flooding the blockchain with endless lists of near hashes, the proof will be provided in the form of a SNARK. To present the proof, miners will create a transaction-like claim that they will broadcast on the blockchain network for inclusion in a block. The claim, assuming it is valid, will generate coins (similar to the coinbase transaction in existing cryptocurrencies) which are then spendable by the miner.

1.3 Organization

We start providing the necessary background in Section 2, particularly on Mining Pools and SNARKs. We then discuss issues of miner incentives and how these considerations both affect existing solutions to mining pools and inform the design of our own in Section 3. We then fully lay out the OCEAN proposal in Section 4, including a high-level description of the statement for which SNARKs should be computed in 4.1. We finally list a number of open problems in Section 5.

2 Background

2.1 Mining Pools

Mining pools are groups of miners who combine their computing resources together to mine on the blockchain. They mine blocks which reward a pool manager, and the pool manager then distributes the reward to the miners after taking a cut. Though this reduces a miner's expected profits a little, it benefits them overall by increasing the likelihood they get paid at all. A miner's chances of mining a block scale with their relative computing power in the network, but many individual miners have a minuscule portion of the network's computing power. If a number of them combine their resources into a pool, the pool has a higher chance of mining a block overall, so the miners will get paid more regularly.

Pool managers redistribute the mining rewards to the miners based on their relative computing power. This can be approximated by having miners submit *near hashes* — nonces that result in hashes with slightly fewer zeros than the actual blockchain difficulty.

2.2 SNARKs

Before we introduce SNARKs, we will first define another cryptographic tool known as SNARGs. A SNARG allows a person to prove to others that some statement is true. The first SNARG system was introduced by Micali in 2000 [6], then under the name "Computationally-Sound Proofs". The SNARG creator is referred to as the prover and those interested in checking the argument output by the SNARG are referred to as verifiers. SNARG is an acronym for Succinct, Non-interactive Argument. We define these terms below.

– **Argument**: Arguments are similar to proofs in that they convince the verifier that some statement is true. However, they differ in that an argument assumes that the prover is computationally limited while a proof makes no such assumptions. In other words, in an argument system, a

prover with unbounded computational power would be able to construct a convincing argument of a false statement. However, in a proof system, this prover would not be able to construct a convincing proof of a false statement. Although proofs are more theoretically secure, it is currently unknown if there is a succinct, non-interactive proof system. For the purposes of our proposal, we can practically assume that an attacker has a limited amount of computation power, which is already an assumption in the blockchain design we build on.
- **Succinct**: A succinct argument is an argument requiring at most polylogarithmic space with respect to the runtime of the program which verifies the argument. This property is important because it places an upper bound on the size of an individual argument, making it efficient to communicate over the network.
- **Non-interactive**: As opposed to interactive arguments which require provers and verifiers to exchange messages to complete the argument, non-interactive arguments require only a single message to make the argument. Such arguments do not require verifiers to respond in order to convince them of the truthfulness of the argument. This construct allows arguments to be asynchronously checked, and multiple verifiers can independently check an argument after it is transmitted. Non-interactive arguments are therefore very scalable since it doesn't require the prover to stay online to convince multiple verifiers of the truth to its argument.

One might be tempted to assume that if a prover were able to prove a statement, they must know a value w that is a witness for it. But the properties we naturally associate with SNARGs provide no such guarantee. To illustrate why that might matter, consider the following, somewhat artificial, example. Suppose that, for some polynomial $p(x)$, Victor wanted to know that $p(x)$ has a real root, and that Peggy had provided a SNARG proof of that fact. Victor might be tempted to believe that this means Peggy (or whoever created the proof) knows one such root. However, Peggy could have proven this existence without actually knowing any roots (say using Descartes' rule of signs), and for some SNARGs, that might be enough for Peggy to create a SNARG proof.

To rectify this, we may use SNARKs: Succinct, Non-interactive Arguments of Knowledge [1]. These are SNARGs with the additional property that in order for a prover to create an argument of knowledge, they must know the witness. In our example above, if Peggy were to supply an argument of knowledge to Victor (as opposed to a plain argument), Victor would know (with high probability) that Peggy knows a real root.

To be more formal, a SNARK tries to prove $x \in \mathcal{L}$ for a NP language $\mathcal{L}$ with a Turing machine M. x has witness w that x is in $\mathcal{L}$. A simplified definition of a SNARK is a tuple of three algorithms: (Setup, Prove, Verify) where

- $\textsc{Setup}(1^\lambda) \to (\sigma, \tau)$: On input of the security parameter λ, the setup algorithm outputs a reference string σ and verification state τ;
- $\textsc{Prove}(\sigma, y, w) \to \pi$: On input of the reference string σ, statement $y = (M, x)$ and witness w, outputs a proof π;
- $\textsc{Verify}(\tau, y, \pi) \to \{0, 1\}$: On input of the verification state τ, statement y, and proof π, outputs 1 if it believes the proof is correct and 0 otherwise.

For a SNARK to be *complete*, the verifier must always output 1 on correct proofs. For a SNARK to be *sound*, the verifier may only output 1 on incorrect proofs with probability negligible in the security parameter λ. We say that the SNARK is *efficient* if the proof size and verification time is

polynomial in y and λ and is polylogarithmic in w.

Finally, in order to satisfy the argument of knowledge requirement (as opposed to the basic argument requirement), we need to define *extractability*. For any prover P, we require there exist an algorithm $\mathcal{E}_p$ known as the *extractor*. $\mathcal{E}_p$ takes in y, σ, and any auxiliary input that P used as inputs and must output a candidate witness w^*. We then require the probability that the proof π is accepted by the verifier and w^* is not a NP-witness for y is negligible. We know that $\mathcal{E}_p$ gets the same inputs as the prover. Thus the existence of such an extractor shows that it is possible to efficiently find a witness w using only information that the prover was provided.

3 Miner incentives

Academic works on incentives in blockchains, particularly in the context of attacks, tend to focus on risk-neutral, expectation maximizing miners [2] [3] [4]. We argue that the predominance of pools in real-world operating blockchains suggests this approach is flawed, because manager fees make pool mining an irrational behavior for participants who focus on maximizing their expected return. Miners choose to join pools precisely because they are risk-averse and are willing to pay the manager's fee and temporarily entrust the manager with their rewards for the benefit of steadier income.

3.1 Existing Solution to Pooling Centralization

In 2015, Miller *et al.* [7] proposed Nonoutsourceable Puzzles as a blockchain design tool that aims doing away with pool mining. Nonoutsourceable Puzzles change the details of the proof of work mechanism that is used to reward miners on a blockchain in such a way that pooling becomes impractical: the pool manager, if they don't want to entrust every pool participant with the private key that holds the funds collected by the pool, needs to participate in every hashing attempt by a miner in the pool. Such a strategy scales poorly and essentially lacks the core properties that make pool mining practical. The use of Nonoutsourceable Puzzles would presumably see mining pools gone for good in the blockchain that would introduce them. To the best of the author's knowledge, Nonoutsourceable puzzles have not been deployed.

3.2 Problems with Nonoutsourceable Puzzles

We argue that the direct removal of mining pools via cryptographic means that make them impractical would be undesirable. Mining pools emerge to address a need small miners have to reduce their variance. While mining may be profitable in expected value, the unpredictability of mining outcomes makes it a risky venture for miners that cannot reach a hash rate high enough that they have a steady income stream. The risk associated with mining may discourage small miners from joining the blockchain ecosystem, thus resulting in increased, not decreased, centralization.

3.3 An incentive-aware approach to Pool removal

We believe it is better to do away with mining pools through game theoretic means. The need of miners to reduce their risk is legitimate, and the centralization we observe is only an unfortunate

consequence of the fact that pooling is the only available option for risk-reduction. If we could provide other means of smoothing miner rewards, for instance, as a blockchain built-in, then the default risk might be low enough that only the most risk-averse of miners would still be willing to pay fees to further lower their risk.

4 OCEAN Proposal

The OCEAN proposal builds on and resembles the Bitcoin protocol. To mine a block, a miner will concatenate a set of transactions, a hash of the previous block, and a nonce. As with Bitcoin, if the hash of the nonce is less than a certain number (the current difficulty of mining), the miner gets a block reward and their block is accepted by other miners. Additionally, they receive the transaction fees for the transactions they include in their block. What differentiates OCEAN from Bitcoin is that miners may also obtain a reward if the hash of their nonce is not less the current difficulty.

Specifically, suppose the current block reward is B and the current difficulty is k. If a miner publishes a block with nonce n where $\text{Hash}(n)$ has at least k leading zeros, then the miner earns B plus transaction fees and their block is published in the chain. On the other hand, if $\text{Hash}(n)$ has t leading zeros where $t < k$, then the miner can include this block in a SNARK. We call such nonces, *near hashes* and a block containing a near-hash a *near-hashed block*. The miner will earn $f(n, B)$ for this near hash where $f(t, B) = B \cdot \left(2^{k-t}\right)^{-1}$. Note that this function only applies to near hashes. While the function implies that hashes that have more leading zeros than required will be rewarded greater than B, we cap the reward at B. The SNARK proves that the miner should earn $\sum_i f(n_i, B)$ for all the near-hashed blocks they included in the SNARK. However, near hashes do not earn any transaction fees. Transaction fees are only awarded to "fully-hashed" blocks that are actually included in the blockchain. At some point a miner will publish their SNARK in order to claim the rewards for their near-hashes. Other miners will then include this SNARK in their set of transactions to include in blocks. As with regular transactions, the miner may choose to apportion part of their reward as a transaction fee in order to coax other miners to include their SNARK.

Additionally note that it is useless to the ecosystem for miners to mine on old blocks. However, a miner may include near hashes for multiple blocks in a single SNARK. Therefore we do not have a straightforward requirement that blocks included in a SNARK must have a certain freshness. In fact, at this time, we cannot foresee a reason why a miner would purposefully mine on an old block. Doing so would cost them the opportunity of earning transaction fees. That being said, one could devise a method that prevents miners from purposefully mining old blocks. To do this, we could require a certain distribution on the age of blocks within a SNARK or attenuate the reward by the block age. The exact method for this prevention is left as an open problem.

Finally, note that in Bitcoin, the block reward is halved every four years. Eventually, the block reward will become zero. Note that if the block reward in our blockchain becomes zero, there will be no reward for near hashes. At this point, our blockchain will have no advantage over blockchains with pools. For that reason, we do not reduce the block reward over time. A consequence of this is that there is no cap on the total monetary mass of the cryptocurrency with OCEAN, and thus the cryptocurrency will presumably experience some inflation.

4.1 Details of the SNARK used in OCEAN

In this section, we provide some details of the SNARK used in OCEAN. At a high level, we want our SNARK to output a proof that a miner with public key pk deserves a reward of R due to partial hashes.

To determine how much money the miner is owed, we will need everyone to agree on relevant blockchain parameters, such as the block reward B and the mining difficulty k. Besides, we will need to enforce that a miner should not be able to claim the same near hash twice. A component of our solution to this problem is to require every claim for rewards by a miner to list a pair blocks (identified by their hashes Z_i and Z_j) that were published on the blockchain. Then, we require that all near hashes included in the claim were being mined on top off a block that was a child of the block identified by Z_i and a parent of the block identified by Z_j. Finally, require that subsequent claims by the miner with pair of hashes $Z_{i'}, Z_{j'}$ have the block behind $Z_{i'}$ be a descendant of the block behind Z_j.

An instance x on which the SNARK will operate is defined as a tuple of:

- The claimant's public key pk^*;
- The reward R (expressed in cryptocurrency units);
- The block reward B;
- The mining difficulty k;
- The block interval on which this claim operates, as represented by a pair of hashes Z_i, Z_j.

Given such an instance, a miner will seek to prove a statement $y = (M, x)$, which can be described as follows: there exists a list of blocks $\{X_i\}$ such that the concatenation of the following NP statements holds:

- The $\{X_i\}$ are pairwise distinct;
- The public key pk_i that would have received the block reward on block X_i verifies $pk_i = pk^*$, for all i;
- The hashes of each X_i have at least k_i leading zeros, and $R = \sum_i f(k_i, B)$;
- The blocks $\{X_i\}$ were all mined on blocks that were descendants of a block behind Z_i and parents of a block behind Z_j.

The SNARK will take the following auxiliary information from the miner as the witness w:

- The information for each block X_i that the miner wants to claim near-hashes for. This includes a set of transaction T_i, a hash of a previous block h_i, a nonce n_i, and a public key pk_i. Note that by public key pk_i we denote the public key in the generation transaction;
- The information of other blocks Z_j on the main chain as detailed below.

Note that we assumed above that the difficulty level is fixed for the portion of the main chain on which the claim operates. If this were not true, the instance would need to include a difficulty level for each block. Additionally, the claim would need to specify this difficulty level for each block since a verifier would need to verify the miner used the correct difficulty levels.

Since the output of the SNARK is an argument of knowledge, we know that the miner which submits the argument must know all the information in w.[1] Specifically, they know the nonces $\{n_i\}$ for the near-hashes of blocks $\{X_i\}$ and the SNARK proves that $pk^* = pk_i$ for all i where pk_i is the public key associated with block X_i. Thus we know that the miner with public key pk must have mined the blocks $\{X_i\}$.

In order to prove that the blocks X_i were mined on the main chain, we need a connected portion of the block chain. For example, assume that the blockchain currently looks as such:

$$\begin{array}{ccccccccc} \cdots Z_5 & \to & Z_6 & \to & Z_7 & \to & Z_8 & \to & Z_9 \to \cdots \\ & \searrow & & \searrow & & \searrow & & & \\ & & X_6 & & X_8 & & X_{10} & & \end{array}$$

Where the sequence of Z_i denotes the main blockchain hashes and the X_i denote the miner's near-hashed blocks. To prove that the miner tried to mine on the main blocks, the miner will input the blocks Z_5 through Z_9 into the SNARK. That way, the SNARK can first check that Z_5 through Z_9 are consistent (i.e. Z_{i+1} contains the hash of Z_i) and that X_{i+1} was mined on top of Z_i. The proof that is output does not need to explicitly contain information of all the blocks in this connected portion. It only needs to contain the hashes of Z_5 and Z_9. This makes the proof size constant in the length of the connected portion yet still allows others to know that the miner has not submitted proofs whose intervals intersect with this proof's interval. That is to say, the miner has not previously submitted a proof on block interval (Y_i, Y_j) where $Y_j \geq Z_i$.

To summarize, the SNARK will output a proof of the following instance: "A miner with address pk mined near-hashed blocks on top of a subchain starting with Z_i and ending with Z_j, the difficulty level was k and the block reward was B, and they deserve a reward of R." Any person given this SNARK proof would just need to verify that Z_i and Z_j are on the main blockchain and that B and k are correct.

5 Open Problems

The OCEAN Proposal significantly changes the way miners are rewarded in Nakamoto consensus, and these changes have implications for a number of aspects of blockchain design, not all of which were addressed in this proposal. We list a number of open problems:

- In OCEAN, we focused on hashes that have less zeroes than the target difficulty, rewarding them less that full hashes, by a factor depending on how far away they are from a full hash. It would be interesting to consider what happens when hashes that get more zeroes that the target are made to yield a larger reward as they get more and more zeroes.
- OCEAN, as described, requires the blockchain to constantly mint more cryptocurrency to ensure small miners, who don't have access to transaction fees, get rewarded. While an important selling point of Bitcoin was the fact that its total supply is fixed, we believe having a statically predictable amount of currency creation could be a reasonable price to pay if it enables stronger decentralization guarantees. How to make OCEAN work without constantly inflating the currency is an interesting research direction.

[1] Note that this is true since mining is known to be exponentially hard (as a function of the difficulty parameter, in the random oracle model). If mining were easy, the extractor could find the nonces itself in polynomial time and figure out the witness, and extractability would not get us anything.

- A related issue is that of block delays. Implicitly, OCEAN, uses the traditional blockchain mechanism to ensure blocks are somewhat evenly distributed in time, by looking at the timestamps of full blocks. This could expose OCEAN to the type of attacks described in [3]. These attacks have pools sabotaging other pools by participating in them but not submitting full hashes. As OCEAN is in a sense a Blockchain-scale pool, miners might benefit from not submitting some of their full hashes if that lowers the difficulty and increases the amount of cryptocurrency they receive from near hashes. Those attacks have not been observed in practice on traditional blockchains, presumably because they are based on risk-neutral models of miner behavior which do not accurately represent the participants' incentives.
- Finally, an important feature of OCEAN is the addition of a special type of transactions that are SNARG publishing redeem claims. Those claims use some of the block's space, and a key issue for OCEAN will the share of its block space that it will use up, and what that leaves for regular transactions. We stress that we do not believe this will be too significant of an issue as current blockchains already have a large fraction of their space used for the purpose of rewarding pool miners, although it is currently done via standard transactions created by the pool manager.

Acknowledgement The authors wish to thank Prof. Nicolas Christin for interesting discussions on the subject and his encouragements in the pursuit of this work.

References

1. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 326–349. ITCS '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2090236.2090263, http://doi.acm.org/10.1145/2090236.2090263
2. Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 154–167. ACM (2016)
3. Eyal, I.: The miner's dilemma. In: 2015 IEEE Symposium on Security and Privacy. pp. 89–103. IEEE (2015)
4. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM **61**(7), 95–102 (2018)
5. Matsumoto, S., Reischuk, R.M.: Ikp: Turning a pki around with decentralized automated incentives. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 410–426. IEEE (2017)
6. Micali, S.: Computationally sound proofs (2000)
7. Miller, A., Kosba, A., Katz, J., Shi, E.: Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 680–691. ACM (2015)
8. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at metzdowd.com (2008), http://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html
9. Namecoin, https://namecoin.org/, https://namecoin.org/
10. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton University Press (2016)
11. Soska, K., Kwon, A., Christin, N., Devadas, S.: Beaver: A decentralized anonymous marketplace with secure reputation. IACR Cryptology ePrint Archive **2016**, 464 (2016)