Smart contract-based decentralized mining pools for Proof-of-Work blockchains

A.M. Papathanasiou*, C.D. Nassar Kyriakidou*, I. Pittaras*, G.C. Polyzos*†
*Mobile Multimedia Laboratory, Department of Informatics,
School of Information Sciences and Technology, Athens University of Economics and Business, AUEB, Greece
†School of Data Science, Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen), China
{sissypapathanasiou, dnassar, pittaras}@aueb.gr, polyzos@acm.org

2024 IEEE International Conference on Blockchain (Blockchain) | 979-8-3503-5159-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BLOCKCHAIN62396.2024.00037

Abstract—**Mining pools have gained wide attention allowing individual miners, who contribute their computational resources to collectively mine blocks to be rewarded more predictably. Although (traditional) mining pools make the mining process more predictable and incentivize small miners to participate, they introduce centralization and miners need to trust the pool manager that the revenues would be fairly shared among members. Efforts to create decentralized mining pools have been reported in the literature. Nonetheless, the aforementioned schemes have not been widely adopted and are not currently in use, mostly due to their scalability issues and the probabilistic nature of their share validation algorithms, which may result in potential security problems, as only a random subset of shares is chosen for validation. Our solution aims to create an efficient scheme for decentralized mining pools for Proof-of-Work (PoW) blockchains by leveraging Ethereum smart contracts to share mining rewards accordingly, while also utilizing the InterPlanetary File System (IPFS) to minimize gas fees by storing only the necessary information in the smart contract.**

Index Terms—**Ethereum, InterPlanetary File System (IPFS), Merkle Trees, Incentives, Security.**

1. Introduction

Individual miners can combine their mining power in an attempt to earn a block reward and share the revenues proportionally to the computational resources they utilized. In particular, miners in a mining pool, co-operate so as to solve the puzzle that Proof-of-Work (PoW) blockchains introduce, which increases their chances of mining (building) a valid block and earning the reward. As a result, the concept of mining pools allows all pool participants, even those with low mining power, to attempt to mine a block. The revenues generated by the pool are distributed to the members by the pool manager and each member receives rewards (typically) based on the amount of work they have performed. More precisely, miners submit to the pool manager hashes of "near" blocks that they computed, which are named shares. In this way, miners are rewarded according to the shares they submitted.

The emergence and proliferation of mining pools is a result of the benefits they offer to individual miners. Initially, by joining a mining pool, miners can expect to receive more regular payouts, even if their individual mining efforts are not successful in finding new blocks. This can be especially beneficial for small-scale miners, who might not have the resources to mine blocks individually. Moreover, mining pools allow individual miners to join their computational power to mine blocks more effectively. This increases the chances of finding valid blocks and earning rewards compared to individual mining. Finally, mining rewards can be highly variable due to fluctuations in mining difficulty, network hash rate, and block rewards. Nonetheless, mining pools aim to reduce this variance by providing a more consistent stream of rewards.

Despite the several benefits that mining pools introduce, they introduce centralization [1], [2], [3], [4], in the sense that revenues generated by the mining process are sent to a key owned by the pool manager, who is responsible for distributing the rewards. In this way, mining pools violate the decentralization principle of blockchains, as miners should trust the pool manager that the rewards would be fairly shared among members. If a single mining pool operator controls a significant portion of the network's hash rate, he could potentially launch a 51% attack on the network [5]. Centralization in mining pools can also lead to a lack of transparency in the distribution of rewards. In some cases, mining pools may not disclose their payout methods or algorithms, which can make it difficult for individual miners to verify the fairness of the pool. Furthermore, mining pools charge a fee for their services, which can result in reducing the profits of individual miners, while some pools may also require a minimum payout threshold, which can delay the receipt of rewards.

Due to the aforementioned reasons, in this paper, we propose a solution that leverages smart contracts in order to fairly distribute the revenues from mining a block among the members of a mining pool. By leveraging Ethereum smart contracts, trust is promoted among the pool members, as they can be sure that the revenues generated by the pool will be shared according to the number of shares they submitted. Ethereum smart contracts are immutable, meaning that a miner cannot alter the shares of other members and transparent, as they define the rules of how the rewards will be distributed, which are visible to all participants.

There exist other solutions in the literature that aim to address the centralization problem that mining pools introduce, but these schemes raise some security and scalability concerns. For this purpose, our proposed solution will utilize the InterPlanetary File System (IPFS) in order to store only the necessary attributes within the smart contract, thus minimizing gas fees, i.e., cost. Also, we will not base our solution on probabilistic validation algorithms, but instead verify all the shares that members have computed in order to ensure that rewards will be shared accordingly.

2. Background

Our scheme leverages IPFS for storage for transactions, while our protocol utilizes mining pool methods in order to fairly distribute the revenues among the members of the pool. For this purpose, we briefly discuss the concept of IPFS and introduce the typical mining pool methods.

2.1. Mining pool methods

We will first refer to the most common mining pool methods, which define how to distribute the block reward. In our solution, we have implemented the *Pay Per Last N Shares* (PPLNS) method, but our scheme can be easily modified to incorporate other methods:

- Pay-per-share (PPS): In this method, miners are paid a fixed amount for each share they contribute to the pool, regardless of whether a block is found. This provides a more predictable stream of earnings for miners, but also introduces greater risk for the pool operator, since he is responsible for paying out the rewards even when no blocks are found.
- Proportional: In this approach, rewards are distributed among miners based on the proportion of shares they contribute to the pool. Once a block is found, the reward is split among all miners who contributed shares to the pool, proportionally to the number of shares they contributed.
- Pay Per Last N Shares (PPLNS): This method is similar to proportional, but takes into account the number of shares contributed over a certain period of time. More specifically, miners, who contribute shares within the last N rounds are eligible to receive rewards when a block is found.

2.2. IPFS

We utilize the IPFS in order to store data related to the mining pool and keep only the necessary information in the smart contract, so as to achieve gas minimization. The IPFS is a *Peer-to-Peer* (P2P) file system that provides decentralized data storing and file distribution. One of its main components is a *Distributed Hash Table* (DHT), which is responsible for the data lookup and access processes. More specifically, the IPFS leverages a variant of the Kademlia DHT [6] to achieve scalability. Also, IPFS utilizes the BitSwap data exchanging protocol in order to achieve efficient content discovery and retrieval. An analysis of the IPFS and its BitSwap protocol is outlined in [7]. Finally, it leverages a Merkle *Directed Acyclic Graph* (DAG), which is a combination of a Merkle Tree and a DAG, to assure the uniqueness of data exchanges. A more detailed analysis of Merkle Trees is presented in [8], in which the authors were the first to introduce a technique for Merkle tree traversal that required only logarithmic space and time.

3. Related Work

Several research efforts have analyzed the security of mining pools, while others focus on the design and implementation aspects. For instance, Meni Rosenfeld [9] conducted an analysis of Bitcoin pooled mining reward systems and proposed a new reward system called proportional reward, which aims to provide a fair distribution of rewards to miners. Also, he discussed the concept of pool hopping, in which miners switch between different mining pools based on current profitability levels. Another study by Li et al. [10] presented a design and implementation of a Bitcoin mining pool with distributed resource allocation, which uses a two-phase protocol for efficient and secure allocation of resources among miners.

One of the key challenges of mining pools is how to distribute rewards fairly among participating miners. Bonneau et al. [11] studied the economics of mining pools and identified various factors that affect the distribution of rewards, including pool size, payout mechanisms, and mining difficulty. Moreover, Han et al. [12] proposed a method for enhancing pool mining reward fairness in cryptocurrency networks by introducing a dynamic pricing mechanism that adjusts the reward distribution based on individual miners' contributions. Miller et al. [13] discuss various mining puzzles and protocols in an attempt to render pooled mining infeasible or discourage its practice. In our solution, we aim to create an efficient and secure smart contract that will encourage mining pool participation and will be practical for real-world applications, without introducing complexity or pool fees.

There have also been some existing solutions that attempt to address the centralization issue that mining pools introduce. Existing decentralized mining pool solutions, namely P2Pool [14], SmartPool [15], and PoolParty [16], have encountered issues hindering their widespread adoption and long-term sustainability. P2Pool, while pioneering decentralized mining pools, suffered from high performance overhead due to extensive message exchanges, especially under low share difficulty, leading to resource consumption issues and scalability challenges. In particular, the quantity of exchanged messages among miners is directly proportional to the number of shares within the pool. In case which the share difficulty is low, miners should consume significant resources, such as bandwidth and local computational power. As a result, in order for the solution to be practical, high share difficulty is required.

SmartPool, which utilized Ethereum smart contracts, is the most promising solution regarding efficiency and secu-

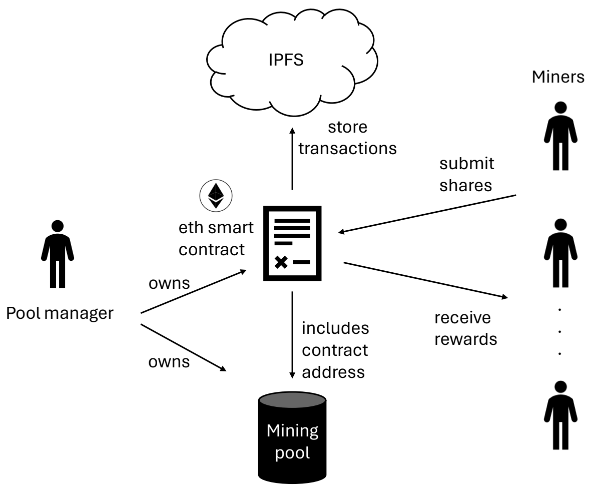

Figure 1. Overview of our system's architecture.

rity concerns. Authors have compared their solution with P2Pool and they showed that they offer stronger security guarantees. Nonetheless, a significant concern in SmartPool is the introduction of probabilistic share verification via Augmented Merkle Trees, resulting in potential security vulnerabilities and scalability concerns as computational and storage requirements increased. More specifically, not all shares are validated and thus revenues may not be fairly distributed. Also, Augmented Merkle Trees require storage and computational overhead and as a result they may introduce scalability concerns. The last decentralized mining pool scheme, namely the PoolParty, while blockchain-agnostic, requires second-layer payment protocols, while not implementing the smart contracts described in the corresponding whitepaper, raising doubts about its practical viability. Due to the aforementioned issues, all these solutions have been abandoned. In particular, P2Pool is inactive since September 2018 and SmartPool by December 2017.

To address these challenges, our proposed solution aims to streamline the process with a single smart contract that will perform the necessary checks and validations, minimizing gas costs and complexity, at the same time. Moreover, by utilizing the IPFS for transaction storage, our scheme offers potential scalability enhancements. Therefore, our approach aims to overcome the limitations of existing solutions and establish a more efficient, secure, and decentralized mining pool ecosystem.

4. Pool Design

In the following section, we outline the design of our system by introducing its architecture, the involved protocols, and the main smart contract actions. A more detailed explanation of the core functionality of our system is analyzed in the following section, in which the contract functions are thoroughly discussed.

4.1. Architecture

The overview of our system's architecture is depicted in Figure 1. Our system comprises of the following entities; the pool manager, who owns the mining pool along with its corresponding Ethereum smart contract, the miners, who wish to join the pool, and the IPFS, which serves as a distributed data storage for the smart contract's information. The pool manager is responsible for setting the share threshold and the number of miners, who will be rewarded according to PPLNS method. These parameters can be viewed by all pool members in the smart contract and cannot be modified. Also, instead of placing the address of the pool manager in the mining pool, we suggest replacing it with the contract's address. This can also be verified by any miner, who participates in the pool. Subsequently, miners attempt to find the next valid block in a PoW blockchain and keep their shares for later validation. In PoW blockchains, miners initially utilize transactions from the network to construct a block header containing information, such as the previous block hash and the Merkle root. They continuously iterate through nonce values in the block header, computing the hash of the header until they find one that meets the network's difficulty target. Once a miner has calculated the next valid block, the corresponding smart contract function is called in order to validate his calculations and offer him half of the block reward. The rest of the earnings are stored in the smart contract and miners submit their shares for validation. When the verification process is performed, miners are rewarded according to the PPLNS method. Price split and mining pool method are specific for our implementation, but they can be easily modified to incorporate other reward sharing approaches. In order to reduce gas consumption, which is a major concern in smart contracts implementations [17], [18], we utilize the IPFS, in order to store only the required information for verification. In particular, each transaction is assigned a unique number, while the list of transactions and their corresponding numbers are stored on IPFS. As a result, a miner should only submit a sequence of numbers for validation instead of the whole list of transactions.

4.2. Protocols

As depicted in Figure 2, from a high-level perspective, the proposed solution involves the following protocols.

Join pool. Initially, each member joins the mining pool, which is performed in two ways; first by joining the mining process in the corresponding mining pool and second by calling the corresponding smart contract in order to join the pool. The smart contract keeps track of the members of the pool by assigning a unique identifier in each member's address. Subsequently, all members try to mine a block in a PoW blockchain and instead of depositing the rewards from mining a block to the pool manager's public key, the amount is deposited in the smart contract's address. In this way, our solution guarantees that the collected money will be fairly shared among the pool members without relying on a trusted third party.

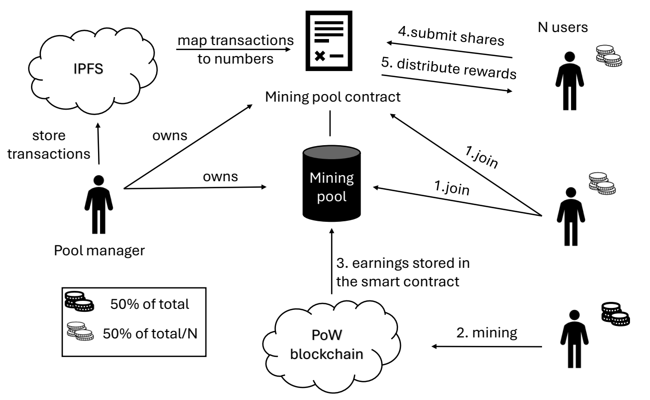

Figure 2. Overview of protocol interactions.

Next block validation. The next block needs to be validated by the smart contract before reward distribution is performed. For this purpose, a smart contract is called with inputs the Merkle root, the timestamp, and the corresponding nonce. The double hash of the header is then computed, by concatenating the fields of version, hash of the previous block, Merkle root, timestamp, difficulty, and nonce. The fields of the header that are not given as an input, are defined by the pool manager, as they are known and can be seen by anyone, who has access to the smart contract. Subsequently, this function checks whether the hash of the header is less than the difficulty and that the user, who called the function is actually a member of the mining pool. Upon successful validation, the miner, who found the next block receives half of the reward.

Mining pool method. We implemented the PPLNS method, which rewards the miners that have calculated the last N shares that are close to the actual hash of the mined block. The threshold up to which miners are rewarded, along with the parameter N are defined by the pool manager and can be viewed by anyone thanks to the smart contract.

Shares validation. Mining pool members, who found the last N shares can get paid by calling the corresponding smart contract function. In particular, miners provide the Merkle root, the timestamp, and the corresponding nonce, while the rest of the header fields are defined by the pool manager as previously discussed. In order to reward the last N shares the following checks need to be made; the hash of the header is less than the difficulty, the one, who called the function is actually a member of the mining pool, the caller has not been already paid and the share is above the threshold defined by the pool manager. After the caller passes these checks, he receives the amount of money left in the contract's balance divided by N.

Merkle Tree Root. Once calculating a valid block, the revenues are stored in the smart contract's balance. The miner who found the next block should then call the corresponding function for block validation, while the other members who wish to submit their shares will call the share validation function. For these functions, the miner should provide a valid list of transactions, meaning one that will yield a valid Merkle Tree root. To minimize gas consumption, each transaction is mapped to a unique increasing number, and as a result a miner provides only the sequence of integers that are used to calculate the Merkle Tree root. When the pool manager receives the sequence, he calculates the Merkle Tree locally and checks whether the Merkle Tree root provided by the user is the same with the one he calculated. If this is the case, the smart contract will perform the necessary checks in order to reward each miner accordingly.

5. Implementation

For our proof of concept implementation, we developed an Ethereum smart contract, called MiningPool,[1] which is written in Solidity. The smart contract is implemented in a way to share rewards only for PoW blockchains, such as Bitcoin. In Bitcoin, the goal in PoW algorithm is to find a suitable nonce, such that the double SHA-256 (hash function) of that nonce is less than a known difficulty. Our solution guarantees that a miner will receive the appropriate rewards, by implementing its core functionalities as smart contract's functions. In particular, we have implemented the functions that were described in the previous section. More specifically, we included methods for validating the next block, the miners' shares, setting the mining pool parameters, and modifiers that perform the necessary checks. We assigned in each pool member a unique identifier, while also mapped each transaction to an integer. The latter is enabled for efficiency purposes, as much less complexity is introduced, if we validate only a sequence of numbers, instead of a list of transactions. In particular, we stored on IPFS the mapping between the hashed transactions and a specific number. In this way, a miner has to provide only the sequence of numbers for share validation and not the whole transaction block, resulting in greater efficiency and reduced gas consumption.

Algorithm 1 validateShares

```
Require: merkleRoot, timestamp, nonce, counter, address
1: concat (version + previousHash + merkleRoot + timestamp + difficulty + nonce)
2: headerHash = hash(concat)
3: doubleHeaderHash = hash(headerHash)
4: require(doubleHeaderHash $>$ difficulty AND isMember(msg.sender) AND doubleHeaderHash $<$ getThreshold() AND isMemberPaid(msg.sender) == false);
5: balance = getBalance()
6: require (counter $<$ getShares());
7: address.transfer(balance/getShares())
```

The main objectives of our solution are share validation and block reward distribution. For this purpose, we provide the pseudocode of these functions in Algorithm 1. This code is executed when a miner wishes to validate his shares and

1. Code available in: https://github.com/sissyp/ADecentralizedMiningPool

receive the corresponding reward. Note that in our implementation, there are separate functions for verification and rewarding, with different inputs, checks, and permissions, but for simplicity, we provide here only the logic of our implementation.

We now describe the functionality of these two processes. Initially, we create the header by concatenating the corresponding values, such as the header of the previous block, the timestamp, and the nonce. In order to perform actions with string variables, we utilized the String smart contract from the OpenZeppelin library.[2] Subsequently, we calculated the double hash of the header and we performed the following checks. First, we needed to assure that the hashed header is less than the share threshold and exceeds the known difficulty. Second, we validated that the miner, who called the smart contract is indeed a member of this mining pool and finally, we made sure that he has not been already paid. For member inclusion, we stored each miner's address in an array during the join protocol. If these checks are correct, we retrieve the balance from the smart contract. We also have a counter to measure the number of miners, who called the function and we require this counter to be less than the number of shares, which will be rewarded according to the PPLNS method. If the miner passes these checks, rewards are transferred from the contract's balance to his address and the validation process is successful.

Regarding the storage requirements, the pool manager stores on IPFS the mapping between transactions and unique numbers. In this way, a miner should only provide the sequence of numbers that correspond to the transactions.

The Merkle Tree root is calculated locally by the pool manager by first mapping each number that a miner submitted in a transaction fetched from IPFS. Subsequently, the pool manager constructs the Merkle Tree by continuously hashing pairs of transactions. Once the Merkle Tree root is calculated, it is provided as input in the validation functions, which in turn call the payment functions that perform the necessary checks to distribute the revenues accordingly. By storing only the numbers in the smart contract, while the list of transactions is uploaded on IPFS, gas fees are reduced.

Parameters In our implementation we designed the system for pay per last N shares method. This can be easily changed for other methods, such as proportional, but not for methods, which reward members even if the next block was not calculated by a member of the pool. As for the share threshold and N, these are set by the pool manager, so depending on the mining pool, these parameters may vary, but all members can view them from the smart contract. Finally, in our implementation, we reward the member, who found the next block with half of the amount of money and the remaining amount divided by N is split among the members with the last N shares.

Modifiers Finally, it is important to mention the two modifiers that were used; onlyOwner and onlyOwnerOf. The former ensures that only the owner of the contract, meaning the pool manager, can call the corresponding functions, while the latter makes sure that only the member with the corresponding identifier will be able to call these functions. This is useful in many cases, as for instance only the owner of the contract should define the threshold up to which shares will be rewarded.

2. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Strings.sol

6. Performance Evaluation

In Table 1, we have measured the gas prices of our core functions in the Sepolia [3] test network of the Ethereum blockchain. More specifically, we measured the gas required for validating the new block and distributing the corresponding reward to the miner, who calculated the block, as well as the gas needed for share validation and rewarding a miner according to the shares he submitted. Moreover, the third column refers to the percentage of the reward that gas fees represent. In particular, we assumed a Bitcoin block, whose reward is equal to $\$520,825.18$. In our protocol, we reward the miner, who computed a valid block with half of the earnings and the rest of the reward is equally distributed to the miners, who calculated the last N shares. Note that share validation is performed for each share separately, while block validation is performed only once. As observed from the results, minor gas fees are required compared to the earnings of the block reward. For share validation we assumed 1000 shares and as depicted in Table 1, gas fees comprise only approximately 0.185% of the share reward.

In block validation function, we measured the gas required to compute the block header, performing a double hash and checking that is less than the known difficulty. Upon successful validation, we calculated the gas needed to transfer half of the block reward to the miner, who computed it, after performing the necessary checks, such as ensuring that he is part of the mining pool and he has not been paid yet. Moreover, in share validation we calculated the corresponding gas for constructing the block header and checking that its double hash is greater than the difficulty, but does not exceed the share threshold. Finally, for rewarding the shares, we estimated the required gas for verifying that the miner has not been already paid for this particular share.

Furthermore, the two last columns refer to the gas fees and the percentage of the reward that gas prices represent in SmartPool. The first two rows are marked as not applicable (N/A) for the SmartPool design, since the gas fees for these actions are not provided. Mining pools that are based on the SmartPool contract have mined a total of 105 blocks in Ethereum and Ethereum Classic networks. In terms of gas fees, as a percentage of the block reward, SmartPool required 0.61% from individual miners, while our solution requires apprroximately 0.185%. These results occur by adding all the rows in the ”Percentage of reward” column for SmartPool and our protocol respectively.

We also computed the time required to upload and fetch the transaction list from IPFS. For this purpose, we assumed

3. https://sepolia.etherscan.io/

Functions	Gas fees	Percentage of reward	SmartPool gas	SmartPool percentage of reward
Block validation	53,376	0.00006527148	N/A	N/A
Reward of next block	4,748	0.000005758	N/A	N/A
Share validation	52,752	0.0652414	79,903	0.01
Reward of shares	4,652	0.00575507	2,872,693	0.6

TABLE 1. GAS PRICES OF MAIN SMART CONTRACT FUNCTIONS

the average number of Bitcoin transactions that are included in a block is approximately 2000-3000 [4]. In either case, the transaction block should not exceed 1MB. On average, for 2500 transactions of 1MB length, approximately 43 seconds were required to be uploaded on IPFS and 41 seconds in order for the list to be retrieved.

7. Security Analysis

We analyzed the security of our protocol by introducing its requirements and designing the corresponding threat model. We also define our penalty scheme, which aims to incentivize miners to behave honestly. The security of our solution mostly relies on the smart contracts' and Merkle Tree's security, which fulfill our security requirements. In particular, we define the following requirements for our protocol:

- Miners do not benefit by submitting false or double shares.
- Shares are fairly shared among pool members according to the PPLNS method.
- Our protocol assures that all shares are correctly validated.

In SmartPool, on average, it is not profitable to submit invalid shares based on their payment scheme and in case one random path from the Augmented Merkle tree is checked. This may apply on the average case, but stronger security guarantees apply when validating all shares and not just a random subset of them. Compared to SmartPool's security protocol, our scheme validates all shares so as to avoid earning any rewards from submitting invalid shares.

7.1. Threat model

We assume that miners act rationally, meaning that they aim to maximize their rewards and thus, they will behave according to that objective. We do not require miners to be honest, meaning submitting their actual shares, but instead, the system will decentivize them from submitting false or double ones. Also, we do not take into account an irrational model, in which miners attempt to cause harm to other miners or the pool manager, since this is also the case in centralized mining pools [19], [20]. Furthermore, we assume that there is no miner controlling over 50% of the total network. In this way, our system is secure against double-spending with 51% attacks [5].

4. https://ycharts.com/indicators/bitcoin_average_transactions_per_block

In other words, our system relies on the security of the underlying consensus algorithm, in our case PoW, so as to prevent manipulation of the blockchain. We also rely on the transparency, immutability, and auditability of the smart contracts, meaning that each miner can view the rules of the pool, while being sure that these rules will be respected and cannot be modified, by anyone.

7.2. Penalty scheme.

In our solution the last N shares are rewarded according to the PPLNS method. In centralized mining pools, the pool manager has to verify each share and distribute the rewards accordingly. As a result, in this scenario, miners have to trust the pool manager that he will fairly share the revenues. Instead, in our scheme the smart contract is responsible for performing the necessary checks and thus, we do not rely on any trusted parties. For this purpose, we define the following penalty scheme, which incentivizes miners to act honestly and submit their actual shares only once:

$$p(x) = \begin{cases} \text{pay } x \text{ out of } N \text{ shares} & \text{if shares not false/double} \\ 0 & \text{otherwise} \end{cases}$$

We will later show that in our protocol, miners do not have any motivation to submit false or double shares, as not only they will not receive any reward, but they will also be charged with the gas fees.

7.3. Merkle Trees security

Merkle Tree Root construction. The Merkle Tree root is one of the main components in the block header. A Merkle Tree is constructed by iteratively hashing pairs of transactions to form a single hash, namely the root. In particular, each transaction within the Tree is individually hashed with SHA-256, in case of Bitcoin, and these hashes are then paired, concatenated, and hashed iteratively, until the root is reached. The Merkle root facilitates verification of block contents, as the integrity of a specific transaction can be confirmed by examining a logarithmic number of hashes, rather than the entire block.

The tamper resistant property of Merkle Trees is important for our verification protocol. More specifically, a modification of a single transaction within the Tree would result in ultimately altering the Merkle root. If a malicious miner provides a false Merkle root, the verification process will be unsuccessful and he will not participate in the mining process. Only providing a valid list of transactions will allow him to continue participating in the protocol.

Share Validation. We prove that our penalty scheme is correct, in the sense that only miners who submit valid shares will be rewarded according to the PPLNS method. In particular, with valid shares we define values that are below the share threshold, but exceed the known difficulty. We also wish to avoid submitting double shares and get paid multiple times for the same share. For this purpose, first, recall that we assumed rational miners, who aim to maximize their profit. Also, we rely on the tamper resistant property of the Merkle Trees. Now suppose that a miner submits a false share. In this case, the provided list of transactions will not yield a valid Merkle Tree root and thus the verification process will not continue. Furthermore, submitted shares are stored within the contract and therefore a miner cannot submit double ones. In either case, the miner falls into the second category, which means he will have zero earnings. Besides, he also pays the gas fees for calling the corresponding smart contract function, while not receiving any reward.

7.4. Smart contract security

Unauthorized access. To prevent unauthorized access to certain functions within the mining pool smart contract, we employed the appropriate modifiers. These modifiers ensure that only specific pool members, can access and modify certain parameters. For instance, the pool manager is responsible for defining the number of shares in the PPLNS method. In this way, we assure that no member of the pool can alter this value, but thanks to the smart contract's transparency, it is visible to all members.

Reward Distribution. A major objective of our scheme is to fairly distribute revenues across pool members. For this purpose, the miner, who successfully computed the next block, receives half of the reward, while the N nearest valid blocks are rewarded accordingly with the remaining half. In the mining pool, the pool manager's address is replaced with the address of the smart contract, and as a result each time a miner finds the next valid block, the revenues are stored in the contract's balance. Subsequently, the functions for block and shares validation are being called from the corresponding miner. After performing the necessary checks, rewards are distributed accordingly, while the contract ensures that each miner will not be paid for falsely or double shares.

Transactions Integrity. By associating each transaction with a unique identifier, our contract offers miners the opportunity to submit a single sequence of numbers, rather than a list of transactions. In this way, we ensure the integrity of transactions, as they are unique, while also preventing manipulation or duplication of transaction data during the verification process. Also, we assure that only the appropriate sequence of numbers will yield a valid Merkle Tree root, therefore avoiding transactions' modifications by malicious miners. Note that constructing a Merkle Tree with duplicate transactions will not result in a valid root as through their mapping with integers, transactions in our protocol are unique.

Preventing Double Payment. Our smart contract includes checks to prevent double payment of miners by submitting the same shares multiple times. More specifically, miners are required to be members of the mining pool and must not have already been paid before receiving their share of the mining reward. For this purpose, when miners join the pool and call the corresponding smart contract function, their address is stored in an array, which keeps track of the pool members. Moreover, the N shares that are rewarded are also stored in the smart contract. As a result, this prevents miners from exploiting the system by claiming multiple rewards for performing the same work.

8. Discussion

Efficiency and gas fees A major drawback that previous solutions introduced is the increased complexity that occurred, when the number of miners and shares increases, which has led to scalability issues and as a result, previous solutions were abandoned. In particular, P2Pool relied on a secondary blockchain for recording shares, while SmartPool included computational and storage requirements for maintaining the Augmented Merkle Trees, when the number of shares and miners increased. Finally, the latest scheme that was proposed, namely PoolParty, requires the target blockchain to have a second layer payment protocol for micro-payments, such as Bitcoin's Lightning Network [21], which may limit the compatibility with blockchains that lack such protocols. Our solution aims to create a scheme that will reduce computational and storage overhead by leaving only the necessary information in the smart contract, by leveraging IPFS, as storage for hashed transactions.

Some previous solutions lacked efficiency, as they required high share difficulty, in order to reduce the number of transmitted messages, or they included penalty schemes to punish miners' misbehavior. In contrast, our protocol aims to include a single smart contract to perform share validation, without altering the share difficulty or incorporating game-theoretic elements. Our penalty scheme is quite simple and aims to incentivize miners to submit valid shares, as otherwise, they will not receive any reward.

In our solution, we suggested storing the list of transactions on IPFS and leave only the necessary content in the smart contract, so as to minimize gas consumption. Gas consumption is an important problem when referring to smart contract optimization [22]. It is important not to affect miners participation in the mining pool, so we ensured that gas is minimized as possible, while keeping our solution feasible and secure.

Sharding techniques. An extension of our solution would be to incorporate sharding techniques, so as to improve scalability [23], [24], [25]. In particular, sharding refers to the partitioning of the UTxO into smaller subsets, called shards. By splitting the UTxO set into shards, we allow for parallel processing of transactions and thus reduce the burden on individual nodes. In this way, we can enable faster transaction processing, while also maintaining the decentralization principle of our solution.

9. Conclusions and Future Work

Mining pools encourage miners to participate in the mining process, as they can combine their computational resources in order to receive a portion of the block reward. In this way, even miners with low mining power can increase their revenues. Nevertheless, mining pools are controlled by a pool manager, in the sense that the earnings collected from calculating the next block are sent to his address. In our solution, we solve the problems related to centralization and trust in a mining pool by introducing smart contracts to fairly distribute the revenues among the members of a mining pool. We showed that our solution offers transparency, automation, security, and trust, while exploiting the IPFS in order to achieve gas (cost) minimization. By addressing the aforementioned challenges, our solution provides a promising alternative for achieving decentralization in mining pools and may lead to their wider adoption. The proposed scheme could be further extended by introducing a blockchain-agnostic solution, which will support different mining pool methods and payment protocols. In addition, the rationality assumption could also be relaxed by introducing a threat model that assumes rational adversaries with harmful behavior.

Acknowledgement

This work was supported in part by project InterConnect, which has received funding from European Union's Horizon 2020 Research and Innovation Programme under Grant 857237, and by the Research Center of the Athens University of Economics and Business.

References

[1] A. Beikverdi and J. Song, "Trend of centralization in bitcoin's distributed network," in *16th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD), Takamatsu, Japan*, 2015.

[2] P. Chatzigiannis, F. Baldimtsi, I. Griva, and J. Li, "Diversification across mining pools: Optimal mining strategies under pow," *Journal of Cybersecurity*, vol. 8, no. 1, 2022.

[3] M. Romiti, A. Judmayer, A. Zamyatin, and B. Haslhofer, "A deep dive into bitcoin mining pools: An empirical analysis of mining shares," *arXiv preprint arXiv:1905.05999*, 2019.

[4] L. Zeng, Y. Chen, S. Chen, X. Zhang, Z. Guo, W. Xu, and T. Moscibroda, "Characterizing ethereum's mining power decentralization at a deeper level," in *the 2021 IEEE International Conference on Computer Communications (INFOCOM)*, 2021.

[5] F. A. Aponte-Novoa, A. L. S. Orozco, R. Villanueva-Polanco, and P. Wightman, "The 51% attack on blockchains: A mining behavior study," *IEEE access*, vol. 9, pp. 140 549–140 564, 2021.

[6] P. Maymounkov and D. Mazieres, in *the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, MA, USA*, 2002.

[7] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: a storage layer for the decentralized web," in *the 2022 ACM SIGCOMM Conference, Amsterdam, Netherlands*, 2022.

[8] M. Szydlo, "Merkle tree traversal in log space and time," in *Advances in Cryptology-EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland*, 2004.

[9] M. Rosenfeld, "Analysis of bitcoin pooled mining reward systems," *arXiv preprint arXiv:1112.4980*, 2011.

[10] L. Quan-Lin, C. Yan-Xia, and W. Qing, "Performance evaluation, optimization and dynamic decision in blockchain systems: a recent overview," *Blockchain*, vol. 1, no. 1, 2023.

[11] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *the 36th IEEE symposium on security and privacy, San Jose, California*, 2015.

[12] R. Han, Z. Yan, X. Liang, and L. T. Yang, "How can incentive mechanisms and blockchain benefit with each other? a survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–38, 2022.

[13] A. Miller, A. Kosba, J. Katz, and E. Shi, "Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions," in *the 22nd ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA*, 2015.

[14] P2pool: Decentralized bitcoin mining pool. Accessed: 2024-06-25. [Online]. Available: https://p2pool.io/

[15] L. Luu, Y. Velner, J. Teutsch, and P. Saxena, "Smart pool: Practical decentralized pooled mining." in *the 26th USENIX Security Symposium, Vancouver, BC, Canada*, 2017.

[16] N. Dana Troutman and A. Laszka, "Poolparty: Efficient blockchain-agnostic decentralized mining pool," in *the 3rd International Conference on Blockchain Technology*, 2021.

[17] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in *the 24th international conference on software analysis, evolution and reengineering (SANER), Klagenfurt, Austria*, 2017.

[18] T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, X. Xiao, J. Chen, and X. Zhang, "Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1433–1448, 2020.

[19] N. T. Courtois and L. Bahack, "On subversive miner strategies and block withholding attack in bitcoin digital currency," *arXiv preprint arXiv:1402.1718*, 2014.

[20] I. Eyal, "The miner's dilemma," in *the 36th IEEE symposium on security and privacy, San Jose, California*, 2015.

[21] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016, white paper.

[22] G. A. Pierro and H. Rocha, "The influence factors on ethereum transaction fees," in *the 2nd IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Montreal, QC, Canada*, 2019.

[23] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," in *the 1st ACM Conference on Advances in Financial Technologies, Zurich, Switzerland*, 2019.

[24] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *in the 2019 international conference on management of data, Amsterdam, Netherlands*, 2019.

[25] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *the 25th ACM SIGSAC conference on computer and communications security, Toronto, Canada*, 2018.