SMARTPOOL: Practical Decentralized Pooled Mining

(Draft v0.1, 10 Jan 2017)

Loi Luu*, Yaron Velner†, Jason Teutsch‡, Prateek Saxena*
*School of Computing, National University of Singapore
{loiluu, prateeks}@comp.nus.edu.sg
†The Hebrew University of Jerusalem
yaron.welner@mail.huji.ac.il
‡The University of Alabama at Birmingham
teutsch@uab.edu

***Abstract*—Many blockchain-based cryptocurrencies such as Bitcoin and Ethereum use Nakamoto consensus protocol to reach agreement on the blockchain state between a network of participant nodes. The Nakamoto consensus protocol probabilistically selects a leader via a mining process which rewards network participants (or miners) to solve computational puzzles. Finding solutions for such puzzles requires an enormous amount of computation. Thus, miners often aggregate resources into *pools* and share rewards amongst all pool members via *pooled mining* protocol. Pooled mining helps reduce the variance of miners' payoffs significantly and is widely adopted in popular cryptocurrencies. For example, as of this writing, more than 95% of mining power in Bitcoin emanates from 10 mining pools.**

Although pooled mining benefits miners, it severely degrades decentralization, since a centralized pool manager administers the pooling protocol. Furthermore, pooled mining increases the transaction censorship significantly since pool managers decide which transactions are included in blocks. Due to this widely recognized threat, the Bitcoin community has proposed an alternative called P2Pool which decentralizes the operations of the pool manager. However, P2Pool is inefficient, increases the variance of miners' rewards, requires much more computation and bandwidth from miners, and has not gained wide adoption.

In this work, we propose a new protocol design for a decentralized mining pool. Our protocol called SMARTPOOL shows how one can leverage *smart contracts*, which are autonomous agents themselves running on decentralized blockchains, to decentralize cryptocurrency mining. SMARTPOOL guarantees high security, low reward's variance for miners and is cost-efficient. We implemented a prototype of SMARTPOOL as an Ethereum smart contract working as a decentralized mining pool for Bitcoin. We have deployed it on the Ethereum testnet and our experiments confirm that SMARTPOOL is efficient and ready for practical use.

I. INTRODUCTION

Bitcoin and emerging cryptocurrencies like Ethereum are popular since they offer trustless platforms for users to transact and run decentralized applications. For example, unlike traditional centralized systems, Bitcoin lacks a central authority to issue fiat currency. Instead, Bitcoin maintains a peer-to-peer distributed *ledger* of prior transactions that demonstrates who owns what. Network participants run a consensus protocol namely Nakamoto consensus to agree on the state of the ledger [1]. In every epoch, Nakamoto consensus probabilistically selects a leader which demonstrates a solution to a computational puzzle (or a "proof-of-work" puzzle) [1]. The leader broadcasts a "block", which includes set of new transactions to be appended to the ledger. If such a block satisfies several predefined validity conditions, such as not having any double spending transactions, all network participants will update their ledger with the new transaction block and the protocol moves on to the next epoch. The leader (or block finder) receives a block reward which includes block subsidy (12.5 Bitcoin, or $9,000$ USD at present) and fees of all included transactions.

Problem. Nakamoto-based cryptocurrencies, such as Bitcoin and Ethereum, utilize massive computational resources for their mining. Finding a valid solution to a proof-of-work is a probabilistic process, whereby miners with modest computational power can have extremely high variance. A desktop CPU would mine 1 Bitcoin block in over a thousand years, for instance. To reduce variance, miners join *mining pools* to mine blocks and share reward together. In a mining pool, a designated pool *operator* is responsible for distributing computation tasks to miners which have moderate difficulty, much lower than difficulty in solving the full PoW puzzle for a block. Each solution to the task has a probability of yielding a solution to the full PoW block puzzle — so if enough miners solve tasks, then some of these solutions are likely to yield blocks. When a miner's submitted solution yields a valid block, the pool operator submits it to the network and obtains the block reward. The reward is fairly divided among all pool members proportional to their contributed computation power. The problem, though, is that the pool operator is centralized (which maybe running on centralized infrastructure), and the pool operator is in control of massive computational power of its participants. At the time of writing, at least 95% of the current computing power of the Bitcoin network is from 10 mining pools, making the Bitcoin network highly centralized. Previous works also show that Bitcoin is not as decentralized as it was designed to be [2], [3].

By design, cryptocurrencies distribute network moderation over trustless, decentralized populations. The security of such distributed systems relies on the assumption that the majority of participants do not cheat. The common practice of pooled mining undermines this security assumption by delegating network moderation power to centralized authorities, called pool operators, who then posses undue network influence. Several times a single mining pool has commandeered more

than half of a cryptocurrency's hash rate (*e.g.*, `DwaftPool` [1] in Ethereum and `GHash.io` [2] in Bitcoin), and in such cases only the pool operator's goodwill prevented the operator from perpetrating a 51% attack against the entire network. Further, mining pools currently can dictate which transactions get included in the blockchain, thus increasing the threat of transaction censorship significantly [4]. Although some Bitcoin pools allow miners to choose their own transactions (still with some rules enforced by the pools) via the getblocktemplate protocol [5], the vast majority of mining power (*i.e.*, more than 80%) in Bitcoin cannot propose their own transactions [3]. Things are worse in the Ethereum cryptocurrency where it is not even technically possible yet for centralized pools to allow miners to include their own transaction sets. Thus, centralized pools not only decrease decentralization, but also make cryptocurrencies more vulnerable to transaction censorship. For example, recently Ethereum network encountered many empty blocks (*i.e.*, blocks which do not include any transactions). Perhaps the 5 largest pools wanted to broadcast their blocks faster in order to earn more rewards [4].

Solution. In this work, we design a new decentralized pooled mining protocol for existing cryptocurrencies like Bitcoin and Ethereum using smart contracts. *Smart contracts*, introduced in 1994 by Szabo [6] and realized in the Ethereum [7] crytpocurrency in 2015, are uncensorable programs that live on Ethereum's blockchain and have their own executable code and internal states, including storage for variable values, and ether currency balance. Smart contracts are executed on a blockchain using a consensus protocol. The smart contract's code, its input and output are all agreed between all the network participants by the consensus protocol in the underlying blockchain. Thus, all network participants agree on the updated state after each time the contract is triggered.

We use Ethereum smart contracts to build a decentralized pooled mining protocol for Bitcoin called SMARTPOOL. Our solution implicitly replaces the centralized pool operator by network participants who run the Ethereum network. Thus, our work shows how we can leverage one expressive cryptocurrency network to build pooled mining for another. In concept, one could build a pooled mining protocol for Ethereum that runs on Ethereum itself — however, we restrict our study here to support pooled mining for Bitcoin as a representative of many Nakamoto- based cryptocurrencies. Applying our solution to support Ethereum is somewhat straightforward as we discuss in Appendix B. SMARTPOOL does not directly make centralized pooled mining in Bitcoin impossible, nor does SMARTPOOL disincentivize it (as done in Miller *et al*. [8]). SMARTPOOL simply offers a practical alternative to miners to move away from centralized pools without degrading any functionality, usability or security. SMARTPOOL takes no cuts or fees [5], unlike centralized pools, and disburses all Bitcoin block rewards to pool participants in their entirety. Most importantly, SMARTPOOL allows miners to freely select which transaction set they want to include in a block. Thus, SMARTPOOL makes cryptocurrencies much more censorship-resistant.

Technical Challenges. While previous efforts towards P2P pools have been proposed, they have suffered from technical drawbacks and have not gained adoption [9]. In designing SMARTPOOL, we have overcome several of these practical challenges (see Section III-B for details). First,everyone must agree on who contributes what to the pool without any centralized operator. Second, the protocol needs to guarantee that no one is able to cheat, or over-claim their contribution to the pool. Finally, SMARTPOOL needs to guarantee feasible costs (*e.g.*, bandwidth and messages used) for processing millions of task solutions for each Bitcoin block. SMARTPOOL's operating costs on Ethereum must be low enough to incentivize miners to join.

SMARTPOOL includes several novel data structures and design choices which make its protocol secure and efficient. Specifically, we devise a new mechanism to verify and record miners' contributions to the pool without centralized operators. SMARTPOOL's efficient probabilistic verification drastically reduces both the number of messages and the costs to run the pool for miners. Using a novel data structure called the *augmented Merkle tree*, SMARTPOOL's batched share submission and efficient payment scheme can detect and efficiently discourage cheating in the pool. In order to evaluate our design, we implemented a prototype of SMARTPOOL and deployed it on Ethereum's testnet. We measured the costs for Bitcoin miners when submitting their contributions to the pool. Our experiments show that these operating costs are negligible compared to projected income from block rewards (*e.g.*, less than 1% of net) for both Bitcoin and Ethereum. Furthermore, each miner only has to broadcast a few messages per day to SMARTPOOL. Finally, like centralized mining pools, SMARTPOOL offers the advantage of low variance payouts.

Previous works have proposed different applications of smart contracts, ranging from outsourced computation [10], smart contracts for criminal activities [11] to a decentralized venture fund [12]. Here we propose the a new use case of smart contracts, which decentralizes the pooled mining process of cryptocurrencies and directly strengthens the underlying security of the network. The security of smart contract systems — protection against history-revision, public (or open) execution, and integrity-protected computation — directly benefits SMARTPOOL. With SMARTPOOL, we demonstrate that it is feasible to build new cryptocurrencies where the "pooled mining" (or SMARTPOOL) is baked into its scripting logic, thereby making solo-mining as incentivized as pooled mining. In the long run, we hope SMARTPOOL obviates the need for centralized pooled mining, which has posed several kinds of threats to pool members in cryptocurrencies [13], [14].

[1] https://forum.ethereum.org/discussion/5244/dwarfpool-is-now-50-5

[2] https://www.cryptocoinsnews.com/warning-ghash-io-nearing-51-leave-pool/

[3] See GBT column in https://en.bitcoin.it/wiki/Comparison_of_mining_pools. In this url, BTCC and Eligius are the only two major pools which support getblocktemplate.

[4] https://www.reddit.com/r/ethereum/comments/57c1yn/why_dwarfpool_mines_mostly_empty_blocks_and_only/

[5] The caveat here is that Bitcoin miners will pay in Ether gas to execute SMARTPOOL distributively

In the short run, we hope SMARTPOOL will offer a direct competition to centralized pools as it takes almost "no cuts" and distributes a greater share of block rewards to miners.

Contributions. This paper makes the following contributions.

- We introduce a new and efficient decentralized pooled mining protocol for cryptocurrencies. Our protocol SMARTPOOL leverages smart contracts in existing cryptocurrencies, coupling with our data structures and efficient verification mechanism, provides security and efficiency to miners.
- We implemented a prototype smart contract of SMARTPOOL which run as a decentralized Bitcoin mining pool. Our experiments with our deployed contract in the Ethereum testnet demonstrate that SMARTPOOL is practical and efficient.
- We discuss how to use SMARTPOOL to build a new line of cryptocurrencies where mining is fully decentralized, thus avoid threats related to centralized mining pools.

II. BACKGROUND

We give a brief introduction to mining and pooled mining in cryptocurrency. We then provide background on smart contracts and their execution model.

A. Pooled mining

Mining. In cryptocurrencies like Bitcoin and Ethereum, the history of transactions between users is stored in a public ledger namely *blockchain* which has a special data structure: one block of transactions after another. In order to agree on the state of the ledger, the network participants run a consensus protocol, namely Nakamoto consensus, between themselves to periodically and probabilistically elect a new leader among them. The elected leader then proposes a new block which includes a set of new transactions to modify the state of the ledger. Such election is done via the mining process, where network participants (or miners) are asked to solve computationally hard puzzles (or proof-of-work puzzles) [1], [15], [16]. The miners who find the solutions for the puzzles first are rewarded with newly minted Bitcoin to incentivize them to keep solving next puzzles and strengthen the network.

Typically, Bitcoin miners compete to search for a `nonce` value that makes the following condition satisfied

$$\mathsf{H}(\texttt{PrevBlock} \,||\, \texttt{NewTXSet} \,||\, \texttt{nonce}) \leq D \qquad (1)$$

in which H is some preimage-resistant cryptographic hash function (*e.g.*, SHA-256), `NewTXSet` represents the new set of transactions that the miner wants to include to the ledger and D is a global parameter which determines how hard it is to solve the puzzle on average. For example, as of this writing, D in the Bitcoin network is set so that the valid hash must have at least 80 leading zeros. The expected amount of SHA-256 hashes to find a valid hash is 2^{80}. One can easily compute that a normal Desktop CPU which can do a million SHA-256 hashes per second would take tens of thousands of years to find a valid `nonce`.

```
contract Ownership{
  address public owner;
  uint public price;

  function Ownership(uint256 _price){
    price = _price;
    owner = msg.sender;
  }

  function buy(uint nextPrice){
    if (msg.value >= price){ //send enough money;
      owner.send(price);
      msg.sender.send(msg.value-price);
      price=nextPrice;
      owner=msg.sender;
    }
    else
      thow;
}}
```

Figure 1: A contract that allow users to purchase its ownership.

Pooled Mining. Finding solutions for PoW puzzles in Bitcoin or Ethereum is probabilistic and requires enormous resources. Thus, miners who solve the puzzles separately (or solo-miners) would have to wait for long time to receive their first reward. Worse, solo-miners with limited computation power will suffer from very high variance in their payoffs. To sidestep this problem, miners combine their power to solve the PoW puzzles together and split the reward according to each's contribution. This approach is called *pooled mining* in which miners are asked to solve much easier pool-puzzles. Specifically, each pool-puzzle requires pool miners to find `nonce` so that the hash satisfies some smaller difficulty d. A solution for a pool-puzzle is called a *share* which has some probability of being a valid solution for the main PoW puzzle. For example, d may be set so that each share must have at least 50 leading zeros, hence a share has a probability 2^{-30} of being a valid block.

Once a miner in the pool finds a valid block, the reward (12.5 Bitcoin and transaction fees, as of this writing) is split between all pool miners proportional to their contributions, which are measured based on the number of shares they have submitted [14]. The pooled mining protocol guarantees that the miner cannot claim the reward of the block to himself or any other miners.

B. Smart Contracts in Ethereum

Smart contract. A *smart contract* (or contract for short) is a special account on the Ethereum blockchain. A normal account would have its address and the balance (in Ether). However, a smart contract, in addition, has its code and its private persistent storage (*e.g.*, a mapping between variables and values). The contract's code is akin to normal program, in which the program manipulates its variables. To invoke a contract (*e.g.*, execute its code) at address α, users send a transaction to α with appropriate payload, *i.e.*, payment for the execution (in Ether) and/ or input data for the invocation. Each invocation starts execution in a contract state σ and results in a new state σ' of the contract.

A simple example of a contract is in Figure 1 which allows users to pay to existing owner of the contract and become the next owner. After initialization, any user in the

Ethereum network can just send the required amount of Ether to the contract and claim the ownership of the contract. The execution of the contract is guaranteed to be correct (*i.e.*, always get the ownership after paying enough Ether) as long as a majority of the miners in Ethereum are honest.

Gas system. In order to compensate the miners for executing smart contracts, each operation in Ethereum smart contract costs some specific *gas* amount, which can be directly converted to Ether. However, careful readers may notice that a well-financed adversary can ask the miners to compute expensive computation in some smart contract, thus conducting DoS attack to the whole network [10]. In Ethereum, there is *gas_limit* value in each block which dictates how much computation one can ask the network to compute in the block. Although this *gas_limit* prevents the Dos attack on the miners, it rules out running several computation-intensive applications on the Ethereum blockchain.

III. Problem and Challenges

A. Problem Definition

In this work we consider the problem of building an efficient decentralized pooled mining protocol for cryptocurrencies. Such a protocol must satisfy the following ideal properties.

- *Decentralization.* There is no centralized operator who manages the pool. The pool is collectively maintained and run by all miners in the network. There is also no requirement for joining the pool, *i.e.*, anyone with sufficient mining power can freely participate in and contribute to the pool.
- *Efficiency.* The pool should give miners the same reward and guarantee low variance as if they were mining with centralized pools. Further, the number of messages, the amount of bandwidth, local computation and other costs consumed by miners must be reasonably small.
- *Security.* The pool protocol protects miners from attackers who might steal rewards or prevent others from entering the pool.
- *Fairness.* Miners receive rewards in proportion to their share contributions to the pool. Big miners and small miners are treated equally.

Threat model and security assumptions. Cryptocurrencies like Bitcoin and Ethereum allow users to use pseudo anonymous identities in the network. Users do not have any inherent identities and there is no PKI in the network. Here we do not violate any of these properties in our solutions.

We consider a threat model where miners are *rational*, which means they can deviate arbitrarily from the honest protocol to gain more reward. An alternative is a *malicious* model where the attacker does anything just to harm other miners. In this work we are not interested in the malicious model here since i) such sustained attacks in cryptocurrencies often require huge capital, and ii) existing centralized pools are not secure in such a model either [13], [17], [18]. We also assume that the adversary controls less than 50% of the computation power in the network. This assumption guarantees that the consensus protocols in Bitcoin and Ethereum networks perform correctly.

On the other hand, we do not make any assumption on the centralization or trusted setup in our solution apart from what have been made in existing cryptocurrencies [6].

B. Challenges and Existing Solutions

Challenges. There are several security and performance challenges in building a decentralized mining pool.

- *C1.* First, pool miners must agree on how much each miner contributes to the pool. This essentially requires running a consensus protocol among all miners in the pool on top of Bitcoin's underlying consensus protocol. However, running an additional consensus protocol between pool miners makes the security of the pool dependent on how much computation power the pool has (*i.e.*, how well the pool is adopted). Specifically, any adversary who controls more than half of the computation power in the pool is able to subvert the consensus protocol in the pool.
- *C2.* In decentralized mining pools, messages exchanged between miners in the pool are in plaintext, thus any network adversary can observe other miners' shares and either steal or resubmit the shares. This challenge does not exist in centralized pools where miners can establish secure and private connections to the pool, thus one cannot know when and which shares a miner submits to the pool. In decentralized settings, such secure connections are not immediate since i) there is no centralized operator who can initiate secure connections to miners, and ii) there is no PKI between miners in the pool. Thus, a good design for a mining pool must prevent the adversary from stealing others' shares. Similarly, the pool should prevent miners from over-claiming their contribution by either resubmitting previous shares or submitting invalid shares. Centralized pools can efficiently guarantee this since the pool manager can check every submission from miners.
- *C3.* The number of shares in the pool may be too large, thus increasing the number of messages exchanged between miners in the pool. For example, let us consider the scenario when there are $1,000,000$ shares on average to get a valid block. A naïve solution may require miners to create $1,000,000$ messages and broadcast to other miners in the pool to submit their shares. With the current capacity of existing agreement protocols in open and decentralized environments, no network can process that many messages within the course of a few minutes [19], [20]. On the other hand, reducing the number of shares per block by increasing the share difficulty will increase the variance in reward for miners, thus damaging the sole advantage of pooled mining. Figure 2 demonstrates how adjusting the difficulty of shares affects the variance of miners' reward and the amount of resource (both bandwidth and computation) consumed per miner in a decentralized pool.

[6]Cryptocurrencies like Bitcoin and Ethereum have the trusted setup where the first block in these networks are constructed and provided by Satoshi Nakamoto (for Bitcoin) and Ethereum Foundation (for Ethereum).

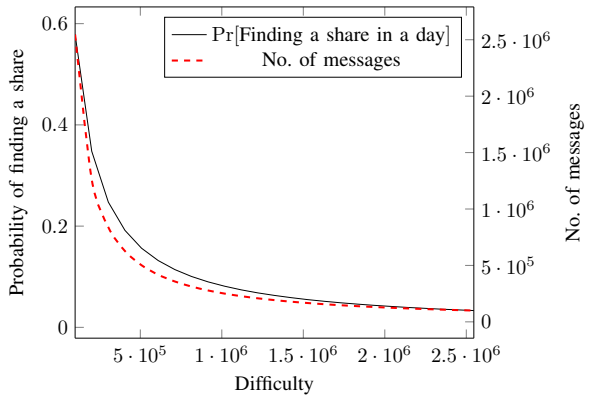

Figure 2: The relationship between the share's difficulty and i) the probability of find a share within a day (black line) as per [14]; ii) resource (*i.e.*, number of messages) consumed by a miner (dashed line); in a decentralized mining pool (*e.g.*, P2POOL).

Existing solutions. The most prominent pooled mining protocol is a centralized one, where there exists a pool manager who distributes the work to miners, records how much work they have submitted and splits the reward proportionally. Apart from operating in centralized environments and increasing the threat of transaction censorship, other disadvantage of this model is that the pool managers either charge miners some fee, or take all transaction fees included in the block for profit and/ or to compensate for the cost of running the pool. Thus, miners often receive less reward than they should.

P2POOL is the first and only attempt we are aware of which decentralizes pooled mining [9]. At a high level, P2POOL solves the agreement problem (*i.e.*, challenge *C1*) by running an additional Nakamoto Consensus protocol to build a share-chain between all miners in the pool. The share-chain includes all shares submitted to the pool, one after another (just like the normal Bitcoin blockchain, but each block is a share). To guarantee that every share is submitted and credited once (*i.e.*, challenge *C2*), P2POOL leverages the coinbase transaction (a special transaction in a block which pays the block reward to miners, discussed more in Section IV-B). Specifically, a share is valid and belongs to P2POOL and a) it extends the latest share in the share-chain; b) it satisfies some predefined difficulty; c) it pays the rewards to miners correctly based on the state of the share-chain.

P2POOL satisfies almost all ideal properties of a decentralized pool (defined in Section III-A) but the *efficiency* and *security* properties. Specifically, P2POOL does not solve challenge *C3* since the number of messages exchanged between miners in P2POOL is linearly dependent on the number of shares in the pool. If it were easy to find a share in P2POOL, the number of messages to be broadcast would increase and miners would need to spend much more resources (*e.g.*, bandwidth, local computation) to download, verify all the shares from others (see Figure 2). Thus, P2POOL requires high share difficulty in order to reduce the number of transmitted messages. Therefore, often miners get much higher variance of their reward than mining with centralized pools. As discussed in previous work [14], high variance in the reward (*i.e.*, the supply of money) decreases miners' utility, makes it harder for miners to predict their income and verify that their systems are working correctly. As a result, as of writing, P2POOL attracts only few miners and controls a negligible fraction of the mining power in the Bitcoin network (the last block mined by P2POOL was almost a month ago [9]).

In addition, P2POOL, although being decentralized, does not provide much security guarantee because of challenge *C1*. The security of the share-chain in the pool depends on how much computation power in the pool (*i.e.*, just like the security of Bitcoin blockchain). As of writing, P2POOL accounts for less than 0.1% of the Bitcoin mining power, thus all miners in P2POOL are vulnerable to any adversary who controls even only 0.1% of the mining power in the network. Hence, it is arguable if miners in P2POOL enjoy better security guarantee than miners in centralized pools.

C. Our Solution

Our solution for a decentralized pooled mining leverages Ethereum smart contracts which are decentralized autonomous agents running on the blockchain itself. At a high level, we replace the pool manager by a smart contract, which is collectively maintained and run by miners in Ethereum network. The smart contract acts like the bookkeeper for the pool, by storing all shares submitted by miners. When a new share is submitted, the contract verifies the validity of the share, checks that no previous record of the share exists, and then updates the corresponding miner's record. We borrow a technique from P2POOL which allows miners to locally generate the block template of the pool based on the state of the contract (discussed more in Section IV-B). If a miner finds a share which is a valid block, it will broadcast the block to the Bitcoin network and submit the block header to the pool to update the miners' records. Our protocol guarantees that the reward for the block is distributed fairly to other miners in the pool.

Apart from the challenges described in Section III-B, there are additional challenges in building such a smart contract for a mining pool. We illustrate them by considering a straw-man solution in Figure 3 which implements P2POOL in a smart contract. The solution works by having a smart contract which receives all the shares submitted by miners, verifies each of them and records how many shares one has submitted. The contract has a designated address so that miners in the pool can send the block reward to the address (*i.e.*, in the "coinbase" transaction, which pays the newly minted coins as the block reward to miners). A share is valid if it uses only the contract address as the output address in the coinbase transaction and satisfies the predefined difficulty (*e.g.*, Line 7). This guarantees that the pool will receive the reward of the block mined by pool miners. On every share submission, the pool verifies the share and updates the contribution statistics of the pool members (Line 14). If a miner finds a valid block, the smart contract distributes the reward to miners in the pool proportional to their contribution so far, based merely on the amounts of shares they submit (Line 17). A solution like the one in Figure 3, apart from *C1, C2, C3*, has the following additional challenges.

```
contract StrawmanPool{
  mapping (uint256 => boolean) mSubmittedShares;
  mapping (uint256 => int) mContribution;

  function submitShare(someShare) returns (boolean){
    // check validity
    if !isValid(someShare)
      return false;
    // check if the share has been submitted
    if mSubmittedShares[someShare.hash]
      return false;
    mSubmittedShares[someShare.hash] = true;
    // update miner's contribution
    mContribution[msg.owner] += 1;
    // distribute reward if is a valid block
    if isValidBlock(someShare)
      distributeReward(mContribution);
    return true;
  }
}
```

Figure 3: Pseudo-code of a straw-man solution which implements a mining pool in a smart contract

- *C4*. A valid share earns miners a small amount of reward, but miners may have to pay much more for fee (in Ether, the underlying currency of Ethereum) when submitting their shares to the pool. This fee is to compensate for any storage and computation required when verifying the share and update the contract state (see Section II). Thus, a poor design of the pool's like in Figure 3 may render negative incomes for miners when the fee paid to submit a share outweighs the reward earned by the share itself.
- *C5*. A smart contract in Ethereum running a Bitcoin mining pool must guarantee correct payments on Bitcoin'a network. This is tricky because Bitcoin miners expect to receive rewards in Bitcoin, but Ethereum contracts can only send and receive Ether.

Next, we briefly describe how we address these challenges to achieve several ideal properties in SMARTPOOL.

- SMARTPOOL guarantees the *decentralization* property by implementing the pool as a smart contract. Like any smart contract, SMARTPOOL is operated by all miners in the Ethereum network, yet SMARTPOOL can be used to secure different network (*e.g.*, Bitcoin) or even the underlying network (*e.g.*, Ethereum) itself. SMARTPOOL relies on the consensus protocol running in Ethereum network to allow miners to agree on the state of the pool (*i.e.*, challenge *C1*). In addition, SMARTPOOL's security depends directly on the underlying network (*i.e.*, Ethereum) which runs the pool, not on how well the pool is adopted.
- SMARTPOOL's *efficiency* comes from allowing miners to claim their shares in batches, *e.g.*, one transaction to the SMARTPOOL contract can claim, say, 1 million shares. Furthermore, miners do not have to submit data of all shares but only a few for the verification purpose, hence the transaction fee per share is negligible. As a result, the number of transactions required to send to SMARTPOOL is 5 – 6 orders of magnitude less than the number of shares (*i.e.*, the number of messages in P2POOL.)
- We propose a novel and efficient probabilistic method for verifying share submission from miners. Our probabilistic verification, coupling with a simple but powerful payoff scheme, allows us to achieve the same outcome as running a full verification for each submission. Specifically, we guarantee that miners will receive their expected reward based on their contributions even when other miners turn malicious and submit invalid shares. This immediately guarantees *fairness*.
- We devise a novel data structure to prevent miners from submitting duplicated shares or resubmitting shares in different batches. We achieve this fairness property without requiring miners to communicate the bulk of their submitted shares data.
- SMARTPOOL employs similar techniques from P2POOL for checking which shares are valid and belong to the pool before paying the miners for the shares. These techniques also guarantee that miners cannot steal others' shares, and that they can mine directly in their target currency (*i.e.*, Bitcoin) without trusting a third party to proxy the payment. Nevertheless, miners still need to acquire ether to pay for the gas when they send transactions to claim their shares to the pool. Such costs are less than 1% of miners' reward as we show in our experiments with SMARTPOOL's deployment in Ethereum testnet.

IV. DESIGN

In this section, we detail the design of SMARTPOOL. SMARTPOOL's design can be used to implement a decentralized mining pool for many existing target cryptocurrencies, but for clarity of exposition we fix Bitcoin as the target. We remark that one could build a cryptocurrency with completely decentralized by employing SMARTPOOL as the unique mining protocol in that system.

A. *Overview of* SMARTPOOL

SMARTPOOL is a smart contract which implements a decentralized mining pool for Bitcoin, running on the Ethereum network. SMARTPOOL maintains two main lists in its contract state — a claim list `claimList` and a verified claim list `verClaimList`. When a miner submits a set of shares as claim for the current Bitcoin block, it is added to the `claimList`. Each claim specifies the number of shares the miner is claiming to have found, and it has a particular structure that aids verification. SMARTPOOL then verifies the validity of the claim, and once verified, it moves it to the verified share list (denoted as `verClaimList`). As discussed later in Section IV-B, each claim allows miners to submit a batch of (say, 1 million) shares. Submitted claims include sufficient meta-data for the verification purpose.

In Section IV-C we will discuss our verification protocol, a key contribution of this work which enables efficiency. The goal of the verification process is to prevent miners from both submitting invalid shares and over-claiming the number of shares they have found. SMARTPOOL pay claimants proportional to the number of shares claimed, only if the

Field Size (bytes)	Name	Data type
4	version	int32_t
32	prevBlock	char[32]
32	TxMerkleRoot	char[32]
4	timestamp	uint32_t
4	bits	uint32_t
4	nonce	uint32_t

Table I: Header of a Bitcoin block. This is also used as the header for shares in pooled mining.

```
1 Input: Empty
2 Output:
3 Value: 12500000
4 scriptPubKey: OP_DUP OP_HASH160 404371705
      ↪ fa9bd789a2fcd52d2c580b65d35549d
5 OP_EQUALVERIFY OP_CHECKSIG
6
7 Value: 12500000
8 scriptPubKey: OP_DUP OP_HASH160 08578
      ↪ d1ac2b4bb797f6f133933c3ea8fbc418746
9 OP_EQUALVERIFY OP_CHECKSIG
10 ...
```

Figure 4: An example of a coinbase transaction in SMARTPOOL. The first output pays to the owner of the share.

verification succeeds, otherwise nothing. The key guarantee here is that of fairness — SMARTPOOL does not advantage miners who cheat by claiming invalid or excessive shares. If miners are rational, their payoff from cheating is the same or worse than that as honestly reporting their shares.

In order to generate valid shares, miners query the `verClaimList` in the contract which records the contributed shares by different miners to date. Thus, if a miner finds a fraction f of the shares in SMARTPOOL, he gets paid proportional to f in the reward that SMARTPOOL's miners get. Further, to enable efficient verification checks, SMARTPOOL forces miners to search for blocks with a particular structure and dictates a particular template for claim submissions, which we discuss in Section IV-B. Unlike P2POOL, miners in SMARTPOOL do not have to run any additional consensus protocol to agree on the state of the SMARTPOOL. Instead, miners rely on the underlying Ethereum network in which SMARTPOOL is deployed to maintain the consistent state information of the pool.

B. Claim Submissions

Miners can submit a large batch of shares in a single claim. To permit this, SMARTPOOL defines a Claim structure which consists of a few pieces of data. First, the miner cryptographically commits to the set of shares he is claiming. The cryptographic commitment goes via a specific data structure we call a augmented Merkle tree, as discussed in Section IV-D. The Merkle root of this data structure is a single cryptographic hash representing all the shares claimed and is included in the Claim as a field called `ShareAugMT`.

After a miner claims several shares in a batch, SMARTPOOL requires the miner to submit proofs to demonstrate that the shares included in the claim are valid. For each claimed share being examined, SMARTPOOL defines a `ShareProof` structure to help validate the share. First, SMARTPOOL requires a Merkle proof, denoted as `AugMkProof`, to attest that the share has been committed to `ShareAugMT`. Furthermore, SMARTPOOL guarantees that if a miner finds a share that is a valid Bitcoin block, then the miner must share its reward with all the pool members in proportion to their previously submitted shares. In Bitcoin, there is a special transaction called a "coinbase" transaction whose output consists of a list of Bitcoin addresses paid and along with their payment amounts. A share in SMARTPOOL is valid if the miner can demonstrate that the share has a valid coinbase transaction (labeled as the field `Coinbase`) in their `ShareProof` paid out to the pool members. The miner cannot selectively choose to omit this transaction; it is required to be the first transaction in the list of transactions (called `TxList`) on which the miner has searched for shares. The claimant must submit a Merkle root as commitment over the set `TxList` he has selected, and a Merkle proof (labeled `CoinProof`) that it contains the coinbase transaction. Second, the `ShareProof` contains an indication of the `verClaimList` based on which the payouts to miners were determined by the claimant. This last field is called a `Snapshot` and is an implementation detail to allow discretizing payouts on an ever-growing `verClaimList`; we refer readers to Section IV-D for details. Figure 5 effectively reports all data fields of our Claim and ShareProof structures.

It is straightforward to see how SMARTPOOL's use of cryptographic commitments prevents certain timing vulnerabilities. SMARTPOOL asks the miners to fix their coinbase transaction before starting to mine shares. Once a share is found, it is not possible to change or eliminate the coinbase transaction. SMARTPOOL credits the first output of the coinbase transaction as the founder of the share. Although miners may use different addresses to submit their claims to the contract, SMARTPOOL credits only a single account per share by fetching the beneficiary account from the coinbase transaction, irrespective of who is the sender of the transaction submitting the claim. This prevents miners from claiming the same share to different Bitcoin addresses (or accounts), forcing a one-to-one mapping between shares found and addresses credited for it. If a network attacker steals someone else's share, it cannot pay itself since the coinbase transaction has already committed to a payee.

Similarly, a miner cannot decide to change the coinbase transaction after he has found a share which is a valid block. This prevents the miner from claiming Bitcoin block rewards and not sharing them with the pool miners. If he modifies the coinbase transaction from the `TxList` *after* discovering a block, he must recompute the hash on a different transaction list which will result in re-doing all the work in finding a valid Bitcoin block.

Shares in SMARTPOOL vs. centralized pools. Shares in SMARTPOOL follow the same data templates as that of a block header in Bitcoin, which is illustrated in Table I. The miner's task is to find a valid `nonce` which results in hash of the specified share difficulty. If the share satisfies the Bitcoin block

difficulty, the miner can submit it to the Bitcoin network and that has the effect of paying out the newly minted Bitcoin to all the SMARTPOOL members as per the coinbase transaction. Figure 4 depicts an example of a coinbase transaction in SMARTPOOL. The first output of the transaction pays to the miner who is mining the block; latter outputs pay to other miners included in the `verClaimList`. The total value of all outputs in the coinbase transaction equals to the block reward. The Bitcoin block reward is 12.5 Bitcoin and the transaction fees of all included transactions, as of this writing. All shares, whether valid Bitcoin blocks or not, can be directly used in SMARTPOOL claims.

In a centralized pool, the pool manager prepares the share's header as in Table I, but without a valid `nonce` and gives it to miners. Once miners find and submit valid shares or blocks, the pool manager will check whether these shares/blocks are actually generated from correct headers given by the pool before accepting them. This is also how the pool operators can dictate which transaction set to be included in a block. In our decentralized setting, such pool operators do not exist. Instead, SMARTPOOL forces a particular structure which miners can later use to prove that their shares and blocks are constructed in a valid way — a technique also used in P2POOL.

C. Batched Submission & Probabilistic Verification

The practicality of SMARTPOOL stems directly from its efficiency in processing a large number of shares claimed. Miners can claim multiple shares to SMARTPOOL in one submission. Each Claim includes less than one hundred bytes which has a cryptographic commitment for the shares, as field called `ShareAugMT`. This cryptographic commitment forces the miner to commit to a set of shares before including them in the claim. Ideally, before accepting any claim of n shares submitted by the miner, we want to verify that

(i) all shares submitted are valid;

(ii) none of the shares is repeated twice in the same claim;

(iii) none of the shares is included in one claim are reused in another.

Probabilistic verification. For efficiency, SMARTPOOL uses a simple but powerful observation: if we *probabilistically verify* the claims of a miner, and pay only if no cheating is detected, then expected payoff to cheating miners is the same or lesser than those of honest miners. In effect, this observation reduces the effort of verifying millions of shares down to probabilistically verifying one or two!

We provide a way to sample shares to verify, outline a detailed procedure for checking validity in Section IV-D, and a full proof in Section V. Here, we begin by explaining this observation intuitively with an example, since it may appear too strong or counter-intuitive at first. Let us consider the case that a cheating miner finds 500 valid shares, but claims that he has found a 1000 valid shares to SMARTPOOL. If SMARTPOOL were able to randomly sample one share from the miner's committed set, and verify its validity, then the odds of having detected the cheating is $500/1000$ (or $1/2$). If the miner is caught cheating, he is paid nothing; if he gets lucky without being detected, he gets rewarded for 1000 shares. Note that the expected payoff for such a miner is still 500, computed as $(0.5 \cdot 1000 + 0.5 \cdot 0) = 500$, which is the same as that of an honest miner that claimed the right amount of valid shares. The argument extends easily to varying amounts of cheating; if the cheater wishes to claim $1,500$ shares, he is detected with with probability $2/3$ and stands to get nothing. The higher his claim away from the true value of found shares, the lower is the chance of a successful payout. By sampling more than once, SMARTPOOL can reduce the success probability of a cheater's payout further, thereby strictly disincentivizing cheating as we show in Section V.

Searching for shares. To enable probabilistic verification, SMARTPOOL prescribes a procedure for mining shares. Each SMARTPOOL miner is expected to search for shares in a monotonic order, starting from a distinct value that it commits to. Specifically, if the set of claimed shared $S = \{s_1, s_2, \ldots, s_n\}$ by a miner, SMARTPOOL requires that the first k (say 20) bits of all $s_i \in S$ form a monotonically increasing sequence. To ensure this, each time a miner find a valid nonce that yields a valid share, they increment the counter by at least 1 and search for the next share. When the miner claims for the set S, this ensures that the set S is lexicographically ordered. If the nonce is long enough, there is expected to be one share per counter value, with overwhelming probability. The miner commits the latest counter in his Claim to this set S, which has at most one share for each counter value. This eliminates any repeats in claimed shares in one claim, and across claims by one miners. In Bitcoin, as we discussed in Section IV-D, we use the share's timestamp to act as the counter of a share.

Note that SMARTPOOL effectively guarantees that shares of miner are distinct from that of others. Each miner has different beneficiary address, so their `Coinbase` transactions and share templates are also different. This ensures that each miner is searching in a distinct sub-space of the search space over time.

Checking Validity of Shares. To check if miners have followed the prescribed procedure, SMARTPOOL randomly samples a share in a submitted Claim, and asks the miner to submit a `ShareProof` (as described in Section IV-B). SMARTPOOL validates the following:

(i) the hash value of the share nonce meets the difficulty criterion;

(ii) the share is constructed on a `TxList` which includes the `Coinbase` transaction;

(iii) the `Coinbase` transaction is constructed correctly based on a valid `verClaimList`;

The check for (i) is straighforward. The check for (ii) can be done using the `TxMerkleRoot`, the `Coinbase` and the Merkle proof `CoinProof` submitted in the `ShareProof`. The check for (iii) confirms that a valid `verClaimList` (identified by the `Snapshot` field) is used in the payouts of the `Coinbase` transaction, by checking the outputs of `Coinbase` transaction.

It remains to discuss (a) how miners cryptographically commit to a batched set of shares in a claim, (b) how does SMARTPOOL verify that the committed set has monotonically increasing counters, and (c) how shares are sampled. For (a)

and (b), one can think of using a standard Merkle tree on all the claimed share set to generate the cryptographic commitment. However, in a normal Merkle tree, verifying the inclusion of a share is efficient, but checking the ordering of the set elements is not efficient. In SMARTPOOL, we devise a new data structure called *augmented Merkle tree* to help us verify inclusion and ordering of shares efficiently. We describe this scheme and implementation of sampling on Ethereum next.

D. Detailed Constructions

Augmented Merkle tree. Recall that a *Merkle tree* is a binary tree in which each node is the hash of the concatenation of its children nodes. In general, the leaves of a Merkle tree will collectively contain some data of interest, and the root is a single hash value which acts as a certificate commitment for the leaf values in the following sense. If one knows only the root of a Merkle tree and wants to confirm that some data x sits at one of the leaves, then holder of the original data can provide a path from the root to the leaf containing x together with the children of each node traversed in the Merkle tree. Such a path is difficult to fake because one needs to know the children preimages for each hash in the path, so with high probability the data holder will supply a correct path if and only if x actually sits at one of the leaves.

For the purposes of submitting shares in SMARTPOOL, we not only want to ensure that shares exist in the batch list but also that there are no repeats and ordering of the counters is correct. We therefore introduce an augmented Merkle tree structure which we use to guard against duplicate leaves.

Definition 1 (Augmented Merkle tree)**.** An *augmented Merkle tree* for a set of objects $\mathsf{S} = \{s_1, s_2, ..., s_n\}$ with respect to a integer-valued *counter* function ctr is a tree whose nodes x have the form $(\mathsf{min}(x), \mathsf{hash}(x), \mathsf{max}(x))$ where

(I) $\mathsf{min}(x)$ is the minimum of the children nodes' min (or $ctr(s_i)$, if x is a leaf corresponding to the object s_i),

(II) $\mathsf{hash}(x)$ is the cryptographic hash of the concatenation of the children nodes (or $\mathsf{hash}(s_i)$ if x is a leaf corresponding to the object s_i), and

(III) $\mathsf{max}(x)$ is the maximum of the children nodes' max (or $ctr(s_i)$, if x is a leaf corresponding to the object s_i).

An augmented Merkle tree is called *sorted* if all of its leaves occur in strictly increasing order from left to right with respect to its counter function.

SMARTPOOL expects claims of submitted shares to be ordered by their counters. Thus, for our purposes, each object s_i will be a share, and the ordering function $ctr(x)$ will return the timestamp of x. In Appendix A, we discuss alternative candidates for the ordering function ctr with backward compatibility to Bitcoin.

Figure 6 gives an example of an augmented Merkle tree based on four submitted shares with timestamps as $1, 2, 3, 4$ respectively. To prove that the share c has been committed, a miner has to submit two nodes d and e to SMARTPOOL. SMARTPOOL can reconstruct other nodes on the path from c to the root (*i.e.*, b and a sequentially) and accepts the proof if the computed root is the same as the committed one.

Batch Submission with augmented Merkle trees. After collecting a list of shares, the miner locally constructs an augmented Merkle tree for all the shares in the list. It then submits the data of the root node of the tree along with a number indicating how many shares it finds to SMARTPOOL. For example, the miner in Figure 6 submits the node a as the cryptographic commitment, which has min and max as 1 and 4 respectively. We use this committed data to i) verify that the sampled shares are found before the miner submits the claim; ii) efficiently check if a share is duplicated in a claim. Verifying i) is straightforward as aforementioned. We compute the probability that we detect duplicated shares in a claim in Section V. Basically, any duplicated shares in a claim will yield a sorting error in at least one path of the augmented Merkle tree. Thus, by sampling the tree in a constant number of places and checking the corresponding paths, with some probability we will detect a sorting error in the augmented Merkle tree if there is one.

Verifying with Samples. Our augmented Merkle tree allows us to detect if miners over claim shares or submit invalid shares in a claim. However, it does not help us guarantee that miners do not submit the same shares in two different claims, *i.e.*, over-claiming shares across claims. Our solution to prevent this problem is to track the counters of the shares in every claim, or the timestamp in our current implementation. We observe that, for a single miner, the timestamps are different for different shares, and often increasing over time. Thus for any two different claims, the maximum timestamp of the shares in the earlier claim is always smaller than the minimum timestamp of the shares in the later one. This observation enables a simple duplication check on the shares submitted in two different claims. Specifically, we use timestamps as the counters for the shares, and require miners to submit their claims in chronological order of timestamps. We use an additional variable $\mathsf{last\_max}$ in our smart contract to keep track of the maximum timestamp (*i.e.*, max value of the root node in the augmented Merkle tree) from the last claim. We only accept a new claim if the min value of the root node is greater than $\mathsf{last\_max}$, and update $\mathsf{last\_max}$ properly if the new claim is valid.

Payment scheme. Miners are rewarded according to the amount of shares that they submitted to the pool. In centralized pools, the pool manager is able to check every share submitted by miners, thus can apply any payment scheme precisely to pay miners for only the valid shares that they submit. In SMARTPOOL, since we use probabilistic verification, our payment scheme must be designed to not only pay fairly to miners, but also disincentivize miners from submitting invalid shares by penalizing them if they do so. We propose a candidate of such schemes in Definition 2.

Definition 2 (Payment Scheme)**.** In SMARTPOOL, the payment scheme for a claim of n submitted shares is as following:

$$\begin{cases} \text{Pay all } n \text{ shares if invalid share was not detected;} \\ \text{Pay } 0 \text{ otherwise.} \end{cases}$$

Notations

- Let `NSize`, `NSample` denote the number of shares included in a claim and the number of random samples SMARTPOOL will verify in each claim respectively.
- Let `claimList`$[x]$ store all unverified claims submitted by the miner at address x.
- Let `verClaimList`$[x][y]$ store all verified and unpaid claims submitted by the miner at address x at block y.
- Let `maxCounter`$[x]$ store the maximum counter of the miner at address x.
- We denote d as the minimum difficulty of a share.

Data structures. The Claim structure has the following fields.

1) the number `NSize` of claimed shares;
2) the `ShareAugMT` commitment of the set of claimed shares.

The ShareProof structure for a share s_i has the following fields.

1) the header of the share s_i (as in Table I) located at the i-th leaf in the augmented Merkle tree;
2) the `AugMkProof`, attesting that s_i is committed to the `ShareAugMT`;
3) the `Coinbase` transaction;
4) the `CoinProof`, attesting that the coinbase transaction is included in the `TxList` of s_i; and
5) the `Snapshot` of `verClaimList` that the `Coinbase` is computed on.

Main executions in SMARTPOOL

- **Accept a claim.** Accept a claim $\mathcal{C}$ which has the Claim structure and includes `NSize` shares from a miner x. Add $\mathcal{C}$ to `claimList`$[x]$ and update `maxCounter`$[x]$.
- **Verify a claim.** Receive a proof p which has ShareProof structure for a share s_i included in a claim $\mathcal{C}$ from miner x. SMARTPOOL verifies the following.
  1) if i is the supposed position that we want to sample based on the intended block hash;
  2) if s_i's hash is included in the claim $\mathcal{C}$ by verifying amkp_{s_i};
  3) if s_i meets the minimum difficulty d;
  4) if s_i's counter is greater than the last `maxCounter`$[x]$;
  5) if `Coinbase` is included in s_i based on `CoinProof`;
  6) if `Coinbase` is correctly constructed with respect to `Snapshot` of `verClaimList`.

  We reject the claim $\mathcal{C}$ if any of the above checks fail. If everything is correct and we have verified `NSamples` from $\mathcal{C}$, update `verClaimList`$[x]$. Otherwise, wait for more proofs from miner x.
- **Get a new valid block.** If a new block is mined by SMARTPOOL, update `verClaimList`.

For miners

- **Fetch coinbase transaction.** Fetch `verClaimList` from SMARTPOOL and build the coinbase transaction locally.
- **Find valid shares.** Simply search for valid `nonce` which yields valid shares.
- **Submit a claim.** If have found enough `NSize` shares, build an augmented Merkle tree and submit a claim $\mathcal{C}$ to SMARTPOOL to claim these `NSize` shares.
- **Submit proofs.** Wait until $\mathcal{C}$ is accepted then construct and submit `NSamples` proofs $\mathsf{p}_i, (i = 1, 2, \ldots, \texttt{NSamples})$ to SMARTPOOL.

Figure 5: Summary of how SMARTPOOL protocol works for both the pool and miners.

In Section V, we prove that our payment scheme disincentivizes rational miners from submitting wrong solutions. Our detailed analysis shows that we need to randomly sample only 1 share in each claim to make expected payoffs from cheating equal to that of honest mining in SMARTPOOL.

Randomly sampling shares. In order to randomly sample shares, we need a random seed. A practical random seed can be the hash of a future block. To reduce the amount of bias that any adversary can introduce to the block hash, one can take several samples based on several consecutive block hashes. For example, let us consider a scenario where a miner submits a claim of 1 million shares at block 1, and we wish to sample 5 random shares for our probabilistic verification. The miner is required to submit the data of 5 shares which are corresponding to hashes of blocks $1, 2, 3, 4$ and 5 (*e.g.*, the hash values modulo 10^6) to SMARTPOOL for the verification. If the miner fails to submit any of these determined shares after, say, 20 blocks, the share is considered invalid.

Putting everything together, we summarize the entire SMARTPOOL protocol in Figure 5.

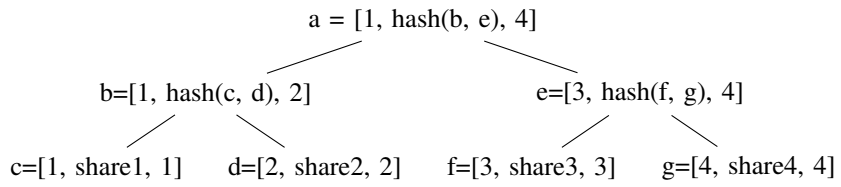

Figure 6: A sorted augmented Merkle tree for a list of shares with timestamp values from 1 to 4.

V. Analysis

We analyse the security guarantee that SmartPool provides with our probabilistic verification and payoff scheme in Definition 2. In SmartPool, we rely on the block hash in Ethereum to randomly sample the shares to verify claims from miners. Thus, our analysis considers two different scenarios in which the adversary can and cannot drop blocks in Ethereum to bias the random seeds.

A. Security Analysis

We first analyze the scenario where the adversary cannot drop Ethereum blocks to introduce bias on our random seed, so the sample blocks in our probabilistic scheme are randomly selected. We argue that it suffices for the SmartPool contract to check a single, randomly chosen path through a submitted augmented Merkle tree in order to pay fairly for shares, on average. If all submitted shares are valid and there are no duplicates, then SmartPool pays for all shares with probability 1. The following facts will be useful.

Lemma 3. *For any node x In any augmented Merkle tree,*

(I) $\mathsf{min}(x)$ *is the minimum of all nodes below x, and*

(II) $\mathsf{max}(x)$ *is the maximum of all nodes below x.*

Proof. We will prove (I), and (II) follows by symmetry. Let y be any node below x, and trace a path from x to y in the given augmented Merkle tree. The min of x's immediate children along this path is, by definition of augmented Merkle tree, no greater than $\mathsf{min}(x)$. Similarly for the next children down, and so on, down to y. Therefore $\mathsf{min}(x) \leq y$. □

Proposition 4. *Let A be an augmented Merkle tree. The following are equivalent:*

(I) *A is sorted.*

(II) *For every node x, the* max *of x's left child is less than the* min *of x's right child.*

Proof. We argue by induction. Assume (I), and further assume than (II) holds restricted to the first n levels above the leaves (the leaves are at the ground (*i.e.*, zero) level). Consider a node x at depth $n+1$. By the inductive hypothesis, the max of x's left child is less than the min of the next right child down, which is less than the min of the next right child down and so on, all the way down to some leaf y. By a symmetrical argument, the min of x's left child is greater than some leaf z which happens to be to the right of y. Since A is sorted, it follows that $\mathsf{min}(x) < y < z < \mathsf{max}(x)$.

Next assume (II), and let y and z be any two leaves. Let x be the lowest node (farthest from the root) which is an ancestor of both y and z. By Lemma 3, y is less than or equal to the max of x's left child, and z is is greater than or equal to the min of x's right child. Now $y < z$ follows from the assumption, hence A is sorted. □

Definition 5. A node in an augmented Merkle tree which satisfies condition (II) of Proposition 4 is called *valid*. Furthermore, we say that a path from a root to a leaf is *valid* if all its constituent nodes are valid. A path which is not valid is *invalid*.

Theorem 6. *Let A be an augmented Merkle tree. If A is sorted, then all paths in A are valid. If A is not sorted, then there are at least as many invalid paths in A as sorting there are sorting errors among the leaves. In particular, there are at least as many invalid paths as there are duplicate values among the leaves.*

Proof. If A is sorted then all its nodes are valid by Proposition 4, hence all paths in A are valid. Now suppose A is not sorted, and consider the highest node x in the tree (farthest from the root) which is is an ancestor of two distinct leaves y and z where y is left of z but $z \leq y$. Now x is not valid, because by Lemma 3 the max of x's left child is at least y and the min of x's right child is no more than z. It follows that neither the path from root to y nor the path from root to z is valid because both pass through x. □

The theorem above shows that miners who submit valid, sorted shares will receive their proper reward. It remains to demonstrate that sampling and checking a single one branch in the augmented Merkle tree suffices to discourage miners from submitting duplicate shares.

Corollary 7. *Under the payment scheme in Definition 2, if* SmartPool *checks one random branch in the augmented Merkle tree of a claim, the expected reward when submit invalid or duplicated shares is the same as the expected reward when submit only valid shares.*

Proof. Suppose that in a claim of an adversary, there are k shares which are either invalid or duplicated. Since we randomly pick a path, by Theorem 6, an invalid share is detected with probability at least k/n. Hence the expected profit from the payment scheme in Definition 2 is

$$((n-k)/n) \cdot n + 0 = n - k,$$

in which $((n-k)/n) \cdot n$ is the reward when the cheating is successful, and 0 is the reward when SmartPool detects

invalid or duplicated shares. On the other hand, one expects to obtain this much profit by submitting only the $n-k$ valid shares. Thus, on average, it is not profitable to submit invalid shares to SMARTPOOL if we employ the payment scheme in Definition 2 and check one random path from the augmented Merkle tree. □

In summary, SMARTPOOL can efficiently probabilistically check that an augmented Merkle tree is sorted. Since it's difficult to construct a fake path through an augmented Merkle tree, we can assume that if the path looks real, then so is the augmented Merkle tree that contains it.

B. Analysis of Biasing Seed Selection

We next consider the scenario in which the the adversary is able to drop Ethereum blocks to bias the random seed. Thus, the sample blocks in our probabilistic verification are not randomly selected, *i.e.*, the adversary can drop the blocks which sample invalid shares from his claim. We show that, even in the extreme case where the adversary controls up to 50% of Ethereum mining power (*i.e.*, can drop 50% of the blocks), SMARTPOOL contract can check only two randomly chosen paths through a submitted augmented Merkle tree to discourage the adversary from cheating. We note that this analysis is for the completeness of the paper. In practice, such dropping block attacks rarely happen since the loss from dropping an Ethereum block is much more than the reward gained by a claim itself, as we show in Section VI.

Theorem 8. *If an adversary controls less than 50% of Ethereum hash power, then it suffices to sample only two branches of the augmented Merkle tree based on two Ethereum consecutive blocks to pay miners fairly, on average.*

Proof. We call an Ethereum block a *good* block for the adversary $\mathcal{A}$ if its hash samples a valid share in the adversary's claim; otherwise the block is a *bad* block. Suppose that in the adversary's claim, γ fraction of the shares are invalid ($0 \leq \gamma \leq 1$). By Theorem 6, at least γ fraction of the paths in the corresponding augmented Merkle tree are invalid. Hence, on average $1-\gamma$ fraction of the blocks are good blocks, since each block hash is a random number. The probability that the adversary's claim is still valid after two samples is the probability that two consecutive blocks in Ethereum are good blocks. We aim to compute this latter probability.

Let us assume that the choices of the two sample shares are drawn based on the hash of a single block hash, and let us assume that attacker controls p fraction of the network's mining power. The attacker's strategy is to successively drop blocks until he finds one that favorably samples his claim submission. We estimate his probability of success. The probability that he succeeds in exactly one round, regardless of who mined the block, is $(1-\gamma)^2$, that is, if the samples drawn are favorable. The chances that the attacker wins in exactly two rounds is the probability that the first block gave unfavorable sampling, but the attacker managed to mine it, and the next sample was favorable. The probability that all three of these independent events occurs is $[1-(1-\gamma)^2]\cdot p\cdot(1-\gamma)^2$. In general, the chance that the attacker succeeds in exactly k rounds is

$$f(k) = \left(1-(1-\gamma)^2\right)^{k-1}\cdot p^{k-1}\cdot(1-\gamma)^2.$$

Summing over all possible game lengths k, we find that the chance that the attacker wins is exactly

$$\sum_{k=1}^{\infty} f(k) = (1-\gamma)^2\cdot\sum_{k=0}^{\infty}\left[\left(1-(1-\gamma)^2\right)\cdot p\right]^k.$$

Since the right-hand side is a geometric series in which the magnitude of the common ratio is less than 1, we obtain

$$\sum_{k=1}^{\infty} f(k) = \frac{1}{1-(1-(1-\gamma)^2)\cdot p} = \frac{1}{1+(\gamma^2-2\gamma)p}.$$

The block withholding strategy is profitable if and only if this probability exceeds the attackers chances of success without block withholding, namely $1-\gamma$. That is, the values p for which block withholding is advantageous satisfy

$$\frac{1}{1+(\gamma^2-2\gamma)p} > 1-\gamma. \tag{2}$$

We complete the analysis by inspecting the cases where p is greater than or less than the threshhold $1/(2\gamma-\gamma^2)$. In the first case it follows that $p \geq 1/2$, since this threshhold is always at least $1/2$ when $0 < \gamma \leq 1$, and if $\gamma = 0$ then the attacker has no incentive for dropping blocks. In the second case, the left hand side of (2) is negative, and so the inuequality in (2) fails in this case. □

Our result in Theorem 8 also applies to the scenario in Section V-A where the adversary cannot drop Ethereum blocks. By checking only two samples in each claim, SMARTPOOL disincentivizes miners to submit invalid shares, and still pays fairly to honest miners, on average.

VI. IMPLEMENTATION AND EVALUATION

In this section, we describe how we implemented SMARTPOOL in our prototype and present experimental evaluation of the expected fees after deploying our prototype in Ethereum testnet.

A. Implementation

We implement SMARTPOOL protocol (as described in Figure 5) in an Ethereum smart contract. Our implementation consists of three modules, namely, *claim submission, verify submission* and *block submission*.

Claim submission. This module allows miners to submit their shares in batch. A miner submits a set of shares by calling `submitClaim()` with the parameters: (i) augmented Merkle root of the corresponding augmented Merkle tree for the shares; (ii) number of shares in the tree; (iii) timestamp interval of the shares. A submission is accepted only if the initial timestamp is greater than the previous submission final timestamp.

Verify submission. A miner submits a proof for the validity of his last submitted batch of shares by calling `verifyClaim()`

with a branch in the augmented Merkle tree that corresponds to the next block hash. In our current implementation, we only sample one branch, or `NSample` $= 1$. We allow different claims to include different amounts of shares, *i.e.*, `NShare` are different between claims. If the verification fails, then the claim is discarded, and the miner will not be able to submit all the shares (or a subset of them) again (forced by validating timestamp in `submitClaim`()). If the verification is successful, then the claim is added to the to the `verifiedClaim` list.

Construct and verify coinbase transaction. Recall that the payment to the miners is done via the coinbase transaction of a mined block. As per Figure 5, SMARTPOOL allows miners to fetch the `verifiedClaim` list and build the coinbase transaction locally. This approach, however, has a technical challenge regarding the transaction size when we implement SMARTPOOL in the current Ethereum network. Specifically, a single coinbase transaction should be able to pay to hundreds or thousands of miners, thus it will have as many outputs. As a result, the size of the coinbase transaction could be in the order of 10KB (e.g., P2POOL's coinbase transactions is of size 10KB [7]). Hence, it is expensive to submit a coinbase transaction of that size to an Ethereum contract. Thus, in SMARTPOOL implementation we could not ask miners to submit the coinbase transaction as the input for `verifyClaim`() function.

To address the challenge, we modify SMARTPOOL protocol slightly. Instead of asking miners to construct and submit the whole coinbase transaction, we ask them to work on only a small part of it. Specifically, we observe that we can fix the postfix of the coinbase transaction by using the pay per share scheme. Recall that the block reward consists of the block subsidy (12.5 Bitcoin) and the transaction fees. Thus, in our implementation, we pay the transaction fee to the miner who finds the block in the first output of the coinbase transaction. The rest 12.5 Bitcoin (the block subsidy) is paid to, say, the next 1 million shares in `verifiedClaim`. This distribution is encoded in all the latter outputs. Thus, we can fix all the outputs but the first one in the coinbase transaction, since the next 1 million shares in `verifiedClaim` are the same for all miners. This allows us to maintain the postfix of the coinbase transaction in SMARTPOOL and only ask miners to submit the prefix (the first output) when they verify a share. Our approach significantly reduces both the gas fees paid for `verifyClaim`() and also the amount of bandwidth that miners have to send for verification.

Block submission. The block submission module allows any user to submit a witness for a new valid block in the Bitcoin blockchain so that SMARTPOOL can have the latest state of the blockchain. If the block is mined by miners in SMARTPOOL, SMARTPOOL updates the `verifiedClaim` list to remove the paid shares from the list. This also reduces the amount of persistent storage required in the contract since we do not need to store all verified claims in SMARTPOOL.

There are other technical subtleties in block submission and constructing coinbase transaction, we discuss these in the Appendix A.

[7] http://tinyurl.com/zrp3dod

Function	Gas	Price	Tx size
`submitFullBlock`()	297550	0.07	1925
`submitClaim`()	38757	0.008	68
`verifyClaim`()			
2^{10} shares	395587	0.09	1956
2^{20} shares	441256	0.10	2956
2^{30} shares	452127	0.10	3236
2^{40} shares	486862	0.11	3876
2^{50} shares	608207	0.13	4516

Table II: Ethereum fee of contract operations. Prices are in USD based on the price of 11 USD per Ether (as of November 2016.) Tx (transaction) size is in bytes.

B. Experimental results

In this section we present experimental results to evaluate the ethereum fees that our protocol entails. The results are presented in Table II. In our experiments Bitcoin blocks contain 2048 transactions (roughly the maximum number of transactions a block can have [8]), and coinbase transactions have 300 outputs (*i.e.*, 300 miners are paid whenever a block is found). The contract contains 450 lines of Solidty code and we deployed it in Ethereum testnet network [9]. The deployment of the contract consumed 3223680 gas (0.72 USD). The contract source code is anonymously available at [10]. The transactions we used for the evaluation are [11] [12] [13] [14] [15] [16] [17].

Execution Costs. To compare our fees with standard pool fees we calculate the expected fees for every submitted share. The expected fee depends on three values:

- Bitcoin's block reward.
- The value (difficulty) of a share.
- Size of a claim (*i.e.*, number of shares).

Bitcoin's block reward is currently 12.5 BTC or more than $8,000$ USD. A share difficulty depends on the miner hash power. It is recommended that miners should set the difficulty such that a share is submitted 20 times per minute [18]. Hence, a single ASIC miner with 4Th/s mining power, which has only third of the hash rate with comparison to the most modern mining ASICs [19], should set his share difficulty to 4,096 [20]. Hence, on average, each of his shares should be rewarded

$$8,000 \cdot \frac{4,096}{254,620,187,304} = 0.00012869 \text{ USD} \qquad (3)$$

Finally, we set the size of a claim to $100,000$. We note that in 20 share per minute rate, a batch submission should occur

[8] https://blockchain.info/charts/n-transactions-per-block
[9] http://tinyurl.com/j4g54gr
[10] http://tinyurl.com/zmlae5y
[11] http://tinyurl.com/hcuy8xn
[12] http://tinyurl.com/z62mxpz
[13] http://tinyurl.com/zfemq7l
[14] http://tinyurl.com/zftnr65
[15] http://tinyurl.com/zvncv7y
[16] http://tinyurl.com/gp77d5z
[17] http://tinyurl.com/zq8f66c
[18] https://slushpool.com/help/#!/manual/terminology#vardiff
[19] https://www.hobbymining.com/mining-hardware/
[20] https://slushpool.com/help/#!/first-aid/troubleshooting

every 3.5 days. One can always adjust the share difficulty to be able to submit their claim with higher frequency.

In our pool, the effective block reward is slightly smaller, as 0.07 USD of the block reward should be given to the miner who submitted the witness for a block that was found (see Table II). In addition, every claim submission also have fees. For a batch of size 100,000, the total fees for the submission are 0.108 USD. Hence, the expected reward per share is

$$(8,000-0.07)\cdot\frac{4,096}{254,620,187,304}-\frac{0.108}{100,000}=0.0001276 \text{ USD} \quad (4)$$

From (3) and (4), our pool fees, as a fraction of the share reward, are

$$\frac{0.00012869-0.0001276}{0.00012869}=0.0084$$

Namely, our fees are less than 1%, which are less than fees in most of the centralized pools [21].

Costs of non-probabilistic verification approach. To demonstrate the usefulness of the probabilistic verification, we also deployed a DUMBPOOL contract [21] which verifies each share In this contract every share is submitted to `verifyShare` function in order to check its validity and claim a payment. In this approach, a single call to `verifyShare` consumes 320056 gas, costs 0.07 USD and requires 1,124 bytes of data [22]. We note that the fee of the submission exceeds the share reward by two orders of magnitude. Hence, such an approach is infeasible with Ethereum contracts.

VII. APPLICATIONS

We discuss several applications that can be built based on SMARTPOOL. One straightforward application is to build decentralized mining pools for cryptocurrencies as we have established. Apart from requiring low costs, guaranteeing low variance in rewards to miners than the only related solution P2POOL, SMARTPOOL is also more secure. Specifically, one must compromise the entire Ethereum network (*e.g.*, having more than 50% of Ethereum network) in order to compromise SMARTPOOL. On the other hand, the adversary only needs to acquire 51% of P2POOL's mining power in order to build the longest share-chain in P2POOL and rule out other miners' contributions.

The second application is a new cryptocurrency based on SMARTPOOL in which mining is fully decentralized. Typically, we enforce the consensus rule such that only the blocks generated by SMARTPOOL is accepted as valid blocks. Thus, if we are able to run a pool's smart contract based on SMARTPOOL in the same cryptocurrency network, miners have no incentives to go for centralized pools. In such cryptocurrencies, miners can solo mine and still enjoy low variance in reward, better security guarantee and pay no fee. Unfortunately, it is not feasible to deploy the above idea in existing cryptocurrencies without a major change in their design. As shown in previous Sections, we run SMARTPOOL as a Bitcoin mining pool in Ethereum network (but not a Bitcoin mining pool in the Bitcoin network, or an Ethereum mining pool in the Ethereum network) is because of two reasons. First, its not technically possible to run SMARTPOOL in the Bitcoin network yet since Bitcoin's script is very strict and not expressive enough to implement all logics in SMARTPOOL. Second, running SMARTPOOL as an Ethereum mining pool requires additional complications (see Appendix B) since efficiently verifing Ethereum's proof of work within a smart contract is hard [23].

Technically, one can easily build a SMARTPOOL-based cryptocurrency by replacing the proof of work in Ethereum by the Bitcoin's proof of work and adding the aforementioned consensus rule which dictates that only SMARTPOOL can produce new valid blocks. Such cryptocurrencies can offer several good properties to the network that existing cryptocurrencies cannot. First, mining is fully decentralized, yet miners still enjoy low variance in reward. This improves the security of the underlying network as a whole significantly. Second, miners are not susceptible to several attacks targeting to pooled mining. For example, in [13], [17], [18] the authors demonstrate that if a malicious miner withholds blocks from a victim pool and mines privately in other pool, the miner can earn more profits from the loss of miners in the victim pool. Such block withholding attack does not work in SMARTPOOL-based cryptocurrencies since there is only one pool in the network.

VIII. RELATED WORK

P2POOL. The work which most directly relates to SMARTPOOL is P2POOL [9]. At a high level, P2POOL maintains a share-chain among the miners in the pool: once miners find a share they broadcast it to everyone so others can extend the share-chain further. Thus, miners in P2POOL run a second consensus protocol on top of the main Bitcoin consensus protocol to agree on the list of shares that each miner contributes to the pool. As discussed in Section II, P2POOL consumes much more resource (both computation and network bandwidth), and the variance of reward is still much higher than in centralized pools. SMARTPOOL solves these problems in P2POOL by i) relying on the smart contracts which are executed in a decentralized manner; ii) use probabilistic verification and novel data structure to reduce verification costs significantly; iii) apply appropriate payment scheme to discourage miners from cheating the pool. As a result, SMARTPOOL is the first decentralized pooled mining protocol which has low costs, guarantees low variance of reward to miners. Further, SMARTPOOL is more secure than P2POOL since any miner who has more than 50% of the mining power in P2POOL can fork and create a longer share-chain. On the other hand, the adversary has to obtain more than 50% of computation power in Ethereum network to compromise SMARTPOOL.

Pooled mining research. Several previous works have analysed the security of pooled mining in Bitcoin [3], [13], [14], [17], [18]. In [13], [17], [18], the authors study the block

[21] http://tinyurl.com/hzlcxl2
[22] http://tinyurl.com/z8oszxr
[23] ethereum.stackexchange.com/questions/2328/is-it-possible-to-verify-ethash-pow-in-a-contract

withholding attack to mining pools and show that the attack is profitable when conducted properly. In [14] Rosenfeld *et al*. discuss (i) "pool hopping" in which miners hop across different pools utilizing a weakness of an old payoff scheme, and (ii) "lie in wait" attacks where the miner strategically calculates the time to submit the found block. These attacks also apply to SMARTPOOL when SMARTPOOL is used as a decentralized mining pool in existing network. However, in SMARTPOOL-based cryptocurrencies where there is only one mining pool which follows SMARTPOOL design, these attacks no longer work.

In [8], Miller *et al*. study different puzzles and protocols which either make pooled mining impossible and/ or disincentivize it. Out work is different from [8] in several aspects. First, we aim to provide an efficient and practical decentralized pooled mining protocol so miners have an option to move away from centralized mining pools. Second, SMARTPOOL is compatible with current Bitcoin and Ethereum networks as we do not require any changes in the design of these cryptocurrencies. In [8], the solutions are designed for new and future cryptocurrencies which have different design from existing ones.

In [2], [3], the authors study the decentralization of the Bitcoin network. Gervais *et al*. in [2] showed that Bitcoin is not as decentralized as it was design to be in terms of services, mining and protocol development. On the other hand, Bonneau *et al*. provided an excellent survey on Bitcoin which also covered the security concerns of pooled mining [3].

Smart contract applications. Previous works have proposed several applications which are built on top of smart contracts [11], [12], [22]. For example, in [11], Juels *et al*. study how smart contracts support criminal activities, *e.g.*, money laundering, illicit marketplaces, and ransomware due to the anonymity and the elimination of trust in the platform. Such applications are built separately from the underlying consensus protocol of the network. In this work, we propose a new application of smart contract that enhances the security of the underlying network by supporting decentralized mining pools with low variance of reward, high efficiency and security.

IX. CONCLUSION

In this paper, we present a new protocol design for an efficient decentralized mining pool in existing cryptocurrencies. Our protocol, namely SMARTPOOL, resolves the centralized mining problem in Bitcoin and Ethereum by enabling a platform where mining is fully decentralized, yet miners still enjoy low variance in reward and better security. Our experiments in the Ethereum testnet show that SMARTPOOL is efficient and ready for deployment in real networks.

REFERENCES

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.

[2] Arthur Gervais, Ghassan O. Karame, Vedran Capkun, and Srdjan Capkun. Is bitcoin a decentralized currency? *IEEE Security and Privacy*, 12(3):54–60, 2014.

[3] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Naryanan, Joshua A. Kroll, and Edward W. Felten. SoK: Bitcoin and second-generation cryptocurrencies. In *IEEE Security and Privacy 2015*, May 2015.

[4] The problem of censorship. https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/, June 2015.

[5] Bitcoin Wiki. getblocktemplate mining protocol. https://en.bitcoin.it/wiki/Getblocktemplate, November 2015.

[6] N. Szabo. The idea of smart contracts. http://szabo.best.vwh.net/smart_contracts_idea.html, 1997.

[7] Ethereum Foundation. Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[8] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 680–691, New York, NY, USA, 2015. ACM.

[9] P2pool: Decentralized bitcoin mining pool. http://p2pool.org/.

[10] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 706–719, New York, NY, USA, 2015. ACM.

[11] Ari Juels, Ahmed Kosba, and Elaine Shi. The ring of gyges: Investigating the future of criminal smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 283–295, New York, NY, USA, 2016. ACM.

[12] Thedao smart contract. https://daohub.org/, 2016.

[13] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. On power splitting games in distributed computation: The case of bitcoin pooled mining. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 397–411, 2015.

[14] Meni Rosenfeld. Analysis of Bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011.

[15] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.

[16] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[17] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in Bitcoin digital currency. *CoRR*, abs/1402.1718, 2014.

[18] Ittay Eyal. The miner's dilemma. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 89–103, Washington, DC, USA, 2015. IEEE Computer Society.

[19] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.

[20] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. *On Scaling Decentralized Blockchains*, pages 106–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[21] Bitcoin Wiki. Comparison of mining pools. https://en.bitcoin.it/wiki/Comparison_of_mining_pools, Accessed on 10 November, 2016.

[22] Jason Teutsch, Loi Luu, and Christian Reitwiessner. Truebit: A verification and storage solution for blockchains. https://medium.com/@chriseth/truebit-c8b6a129d580#.d2txm5usu, 2016.

APPENDIX

A. Implementation Subtleties

In this section we address two technical issues that arise from the design of the protocol. The first issue is the format of a witness for a new valid block, and the second issue is how a miner should decide on his coinbase transaction in the next share he mines.

Witness for a new valid block. Intuitively, a witness for a new block is a block header with sufficient difficulty. However, in Bitcoin network (like in any blockchain based network), some of the mined blocks could be *orphan*, namely, they could be transmitted to the network a short period before or after an *uncle block* (a different block that extends the same previous block) was found. In this case the network will eventually

form a consensus over only one of the blocks, and the other block(s) will become orphan (and will not get any block reward from the network). In our protocol we must update the miners `verClaimList` list only according to non-orphan blocks. For this purpose, as a witness we ask for a chain of six blocks. While in theory, even a chain of six blocks could become orphan, in practice it never happened.

Deciding on the coinbase transaction of the next share. In order for a share to be valid it must have a coinbase transaction that corresponds to a `verClaimList` list. However, the `verClaimList` list is updated by the Ethereum contract. Hence, the contract is only aware of the Ethereum timestamp in the time the list is updated. On the other hand, `verifyClaim()` function supposes to verify the coinbase transaction according to the Bitcoin timestamp of the share. For this purpose we need to synchronize Bitcoin and Ethereum timestamps. The synchronization is done by introducing a new time metric, namely, the number of blocks SMARTPOOL has found. With this new notion of timestamp, we implement the `verClaimList` list in such way that a list of payment claims is maintained for every integer n. The list of n corresponds to the payments that have to be done when SMARTPOOL finds block number n. As new blocks might be reported with some delay, a payment request for a bulk that is verified in time n is added to the payment list of time $n + 20$.

Given this implementation, the miner should construct the coinbase transaction in time n in the following way: As long as a new block is not found, the coinbase should correspond to list n. Once a new block is added to Bitcoin's blockchain, the miner should immediately start working on list $n + 1$ (which already exists, as it was constructed at time $n - 19$), even before the new block is submitted to the contract. If the new block becomes orphan, the miner should switch back to list n. Otherwise, after six blocks he should submit a witness for block n.

We note that in this approach the miner might do some stale unrewarded work in case the new block ends as an orphan block. However, such cases are also not rewarded in standard pools.

Other candidates for counter. Careful readers may realize that the timestamp field has only 4 bytes, thus we will run out of values for the counter after 2^{32} shares. In SMARTPOOL, one can have several ways to implement the share's counter. For example, one can embed the counter inside the coinbase transaction of a share. Specifically, Bitcoin allows users to insert 40 random bytes in a transaction output after the `OP_RETURN` opcode [24]. SMARTPOOL can force miners to store the share's counter in these 40 bytes, which can accommodate much more number of shares (*i.e.*, 2^{320}).

B. Verifying Ethereum PoW

The cryptographic hash function that Ethereum is using is Ethash [25]. Ethash is not a native opcode nor a pre-compiled contract in the Ethereum virtual machine (EVM). Hence, to verify that a block header satisfies the required difficulty we have to explicitly implement a code that computes it. Ethash was designed to be ASIC resistance, which is achieved by forcing miners to extract 128 values from pseudo-random positions of a 1 GB dataset. Thus, to explicitly compute Ethash we would have to store 1 GB data in a contract, which costs roughly 33,554 ether (storing 32 bytes of data costs 50,000 gas). Moreover, the Ethereum protocol dictates that the dataset is changed every three days (on average). Hence, one would require a budget of approximately $100,000 per day to maintain the dataset [26].

Luckily, for our purposes, there is no need to compute Ethash. Instead it is enough to verify that result of an Ethash computation. For this purpose it is enough to ask the miner to submit along with every block header the 128 dataset values that are used when computing its Ethash and a *witness for the correctness of the dataset elements*, i.e., that the 128 values correspond to the values of the corresponding positions in the 1 GB dataset. Intuitively, to verify the witness for dataset elements the contract will hold merkle-root of the dataset and a witness for a single element is its merkle-branch. Formally, the pool contract holds the merkle-roots of all the 1 GB datasets that are applicable for the next 10 years. We note that the content of the dataset only depends on block number (i.e., the length of the chain). Hence, it is predictable and the values of all future datasets is already known. Storing one year dataset roots requires storing 122 elements, and would cost 0.122 Ether. Hence, storing 10 years of dataset roots would cost in the order of 1 Ether.

We note that technically, our approach does not provide a mathematical guarantee for the correct computation of Ethash. Instead it guarantees correct computation provided that the correct dataset roots were stored. Hence, it is the miner's responsibility (and best interest) to verify the stored values (at least for the next several months) before joining the pool. As the verification is a purely algorithmic, no trust on the intentions of the contract authors is required.

Further optimizations. Some initial experiments [27] suggests that the computation of Ethash would still require non-trivial amount of gas (in the order of 3M gas). The consequence of an expensive verification is that miners would have to submit big batches in order to keep fee at low level. To reduce the *expected fees* we propose to skip the validation of some of the batches. Denote $p = \frac{1}{N}$ for some integer N. We propose to verify a batch only with probability p, and if the batch is invalid then we seize a reward of N submitted batches. For this purpose we change the protocol in the following way:

- Miner can withdraw the reward for its first N submissions only when he leaves the pool.
- In the first N submissions, every batch is verified.

[24] https://en.bitcoin.it/wiki/OP_RETURN

[25] https://github.com/ethereum/wiki/wiki/Ethash

[26] Technically, one could store a smaller subset of seed elements and calculate the values of the dataset on the fly. Unfortunately, to extract values from seed one would have to compute SHA3_512, which is not a native opcode in the EVM, and would require massive gas usage if employed many times.

[27] http://tinyurl.com/gw83mqs

- After the first N submissions, a batch is verified only with probability p.

We now analyze the economic implications of the changed protocol. On the positive side, in the long run, the expected fees are dropped by a factor of N. On the negative side, a miner would have to pay fees in the order of 0.1 Ether for each of the first N batches (which is reasonable for e.g., $N = 10$). In addition, he will get a possession over his first N batches reward only when he leaves the pool.