Smart Contracts Make Bitcoin Mining Pools Vulnerable

Yaron Velner[1], Jason Teutsch[2], and Loi Luu[3]

[1] The Hebrew University of Jerusalem
[2] The University of Alabama at Birmingham
[3] School of Computing, National University of Singapore

Abstract. Despite their incentive structure flaws, mining pools account for more than 95% of Bitcoin's computation power. This paper introduces an attack against mining pools in which a malicious party pays pool members to withhold their solutions from their pool operator. We show that an adversary with a tiny amount of computing power and capital can execute this attack. Smart contracts enforce the malicious party's payments, and therefore miners need neither trust the attacker's intentions nor his ability to pay. Assuming pool members are rational, an adversary with a single mining ASIC can, in theory, destroy all big mining pools without losing any money (and even make some profit).

1 Introduction

Bitcoin and emerging cryptocurrencies offer trustless platforms for users to transact and run decentralized applications. Each cryptocurrency maintains a peer-to-peer distributed *ledger* of prior transactions that records all activities in the network. Network participants run a consensus protocol called Nakamoto consensus to agree on the state of the ledger [1]. In every epoch, Nakamoto consensus probabilistically elects a leader who demonstrates a solution to a computational puzzle (or a "proof-of- work" puzzle) [1]. The leader proposes and broadcasts a "block" which includes set of new transactions to be appended to the ledger. The leader (or block finder) receives a reward (around 12.5 Bitcoin, or $12,000$ USD at present) if his block is valid and accepted by the network.

Pooled mining. Finding a valid solution to a proof-of-work puzzle (or *mining*) is a probabilistic process and requires massive computational resources. Normal miners with modest computational power can have extremely high variance. For example, even a state of the art AntMiner S9 mining hardware [4] would mine one Bitcoin block per year on average. To reduce variance, miners often 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 the 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 puzzle. As a result, 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.

[4] https://www.bitmaintech.com/productDetail.htm?pid=00020160529072433755 30DcJIoK0654

Pools reward model vulnerability. Pools are susceptible to the classical block withholding attack [2], where a miner sends only partial proof-of-work to the pool manager and discards full proof-of-work. As the structure of the block header is determined by the pool operator, an attacker cannot claim the block reward for himself. On the surface, block withholding attacks might not seem profitable, however, miners outside the victim pool may benefit from block withholding. Dropped blocks increase outside miners' computation power relative to the rest of the network [3], and in the long run, outside miners will mine more blocks (see formal analysis in Section 3).

The attack. Smart contracts are unstoppable programs that live on the blockchains (e.g. Bitcoin, Ethereum [4]) and have their own executable code and internal states, including storage for variable values, and currency balance. In this paper we introduce smart contracts that reward pool miners who withhold their blocks. We analyze the outcome of such an attack under the assumption that miners are rational and their behavior aim to maximize their short-term profit (we analyze the incentives of the miners in Section 3). We show that when the attack is targeted towards big mining pools who employ the *pay per share* scheme, the attack is profitable even for an attacker running a single hardware unit. Moreover, such an attacker could in theory drain all of revenues and profit from a big pool. We note that in practice, pool operators that witness significant decrease in their revenues may have to close their operation before being drained out all of revenues and profit. Hence, a successful deployment of our attack would undermine the entire pooled mining model.

The use of smart contracts is crucial in order for the attack to be successful. Indeed, it is unlikely that miners would collaborate with such an attack unless their payment is guaranteed. Moreover, rewarding via smart contracts makes it possible for the attackers to remain anonymous, and prevent other parties from targeting the attacker (e.g., with a denial of service attack) and shutting him down.

Contributions. The contributions of our paper are as follows:

- We show how to mathematically prove block withholding and implement an Ethereum smart contract that rewards block withholding (Section 4).
- We show that under mild assumptions the smart contract could be implemented with Bitcoin transactions (Section A) and we show how one can use an Ethereum smart contract to enforce these assumptions (Section B). Bitcoin contracts are more desirable as they save the need to run a full Ethereum node.
- We show how the attacker can form a pool of block withholders in order to reduce the withholders variance, and analyze the incentives for withholders to withhold a block withholding proof (Section 5).

Comparison with classical block withholding attacks. Block withholding attacks are known almost from the beginning of Bitcoin [2]. In recent years it has become apparent that miners can profit from mining for two pools while withholding their full solutions in one of them [5,6,3]. However, the profit from such an attack is relatively small and an attacker would have to control big computation power, e.g., over 1%, of Bitcoin's computation power, in order to make significant losses (e.g., over 5% decrease in revenues) for large pools (e.g., see [3]).

In this work we propose to pay other miners to withhold blocks. In Section 3 we show that an attacker with only 0.0000002% of Bitcoin's computation power can reduce the revenue of a big pool to zero without any financial losses on his side. In fact the theoretical outcome of our attack (if miners are fully rational) is equivalent to a classical block withholding attack in which a miner rents Bitcoin's entire hash power and withholds all the blocks that he finds.

Other cryptocurrencies. In this paper we focus on attacks on Bitcoin mining pools. Nevertheless, in principle, smart contracts undermine the pooled mining model of all cryptocurrencies. However some cryptocurrencies, e.g., Ethereum, might be currently resilient to such an attack due to some technical issues that we describe in Section 4.

2 Background

2.1 Mining and Pool Mining

Bitcoin and popular cryptocurrencies like Ethereum [4] and Zcash [7] maintain a global ledger between all participants in the networks. The network participants run a consensus protocol called Nakamoto consensus to reach agreement on the state of the shared ledger [8]. At a high level, Nakamoto consensus works by probabilistically electing a leader in every 10 minute epoch. The leader will then propose a set of additions (*e.g.*, transactions) to the ledger; other participants "apply" these additions after verifying that these changes are valid. Then the next epoch begins. As of this writing, the election happens via a "mining" process in which network participants have to solve computationally hard puzzles (i.e., proof-of-work) which probabilistically yields one solution per 10 minutes (epoch time in Bitcoin) on average. Technically, network participants, or miners, have to find a valid `nonce` satisfying the following condition:

$$\mathrm{sha256}(\mathrm{sha256}(\text{Block Template} \,||\, \texttt{Nonce})) \leq D \tag{1}$$

in which "Block Template" includes the miner's proposed changes to the ledger, and D is a global parameter which indicates the difficulty of finding a valid solution.

Solving a PoW puzzle, or finding a valid block, requires an enormous amount of computation. For example, at the time of writing, D is a 256-bit integer with approximately 80 leading zero bits. Thus finding a valid PoW solution requires on average 2^{80} $\mathrm{sha256}$ calculations. A normal workstation which can perform a million $\mathrm{sha256}$ calculations per second will expect to spend millions of years to find a PoW solution. Thus, often miners join forces and form "mining pools" to solve PoW puzzles together. The idea of pooled mining is to ask everyone in the pool to find solutions (or *shares*) to easier PoW puzzles where each share has some probability of being a valid solution for the main PoW puzzle. Specifically, pool members find all `nonce` so that the result of the hash in Equation 1 is less than d, where d is much larger than D. A solution of such puzzles is called a share, and will have a probability D/d being less than D, i.e., being the valid solution for the main puzzle. For example, if d were set to have 60 leading zero bits, then a share would have a probability 2^{-20} of being a valid solution for the main PoW puzzle.

In pooled mining, a pool *operator*, or pool *manager*, keeps track of how many shares each miner submits. If a share is indeed a valid PoW solution, the pool operator broadcasts the block to the network and receives a block reward (12.5 bitcoin and the transaction

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

Table 1: Header of a Bitcoin block

fee as of this writing). This reward is then distributed to miners in the pools based on their contributions (i.e. number of shares). By joining pools, miners receive more frequent and stable reward, thus significantly reducing their income variance compared to mining separately (or *solo* mining). Note that in pooled mining, the pool operator prepares the block template in Equation 1, so even if a miner broadcasts a valid block himself, the reward still goes to the pool.

Formal definitions and notations Bitcoin's block consists of a *block header* and a list of transactions [5]. Table 1 depicts the block header format, which consists of 80 bytes.

A block is said to be a *valid extension* of the blockchain if (i) its difficulty matches the network difficulty, i.e., the $\mathrm{sha256}(\mathrm{sha256}(\text{block header})) < D$; and (ii) the previous block hash field in the block header corresponds to a valid block in the blockchain; and (iii) the transactions of the block are valid. A publicaly known block that is a valid extension but does not reside on the longest chain is called an *orphan block* or *stale block*.

A block is a *full solution* (or *valid solution*) if its header matches the difficulty D. A block is a *partial* solution if its header matches the difficulty of the pool share difficulty d. The *hash power* (or *hash rate*) of a miner (or a group of miners) is the relative fraction of computation power he possesses relative to the entire Bitcoin network.

2.2 Smart Contracts

Bitcoin transactions are deemed valid only if their linked script condition holds. While Bitcoin scripts have limited expressiveness, emerging cryptocurrencies support expressive scripts that have enabled the development of a variety of powerful decentralized applications. Bitcoin's scripts are stateless, that is, they do not maintain any internal states, and their behavior depends only on their input.

The Ethereum cryptocurrency introduced smart contracts in which the contract code is a Turing-complete program [9]. In addition to being more expressive, Ethereum smart contracts can also maintain internal states which are shared among transactions. For example, a smart contract can record the number of different addresses in all transactions sent to its address. Users interact with a contract, i.e. modify the contract state, by sending transactions with payloads (i.e. input data) to the contract address.

[5] https://en.bitcoin.it/wiki/Block_hashing_algorithm

3 Block withholding incentives

In this section, we analyze the incentives for an attacker to pay pool miners for dropping blocks. We recall that the actual block is worthless for the attacker, as the destination of the block rewards is fixed as the pool's address. Hence the attacker only benefits from reducing the effective hash rate of the entire network. In order to maintain a consistent block rate (*e.g.*, one block per 10 minutes in Bitcoin), the network periodically adjusts the difficulty of hashing puzzle based on the number of miners participating. In Bitcoin this adjustment happens once every 2018 blocks.

To formally analyze the incentives, we denote the fraction of the network's hash rate controlled by the attacker as α $(0 \leq \alpha \leq 1)$, the block reward by r, and a miner's reward for submitting a full solution to the pool by $s \cdot r$.

We first calculate the attacker's expected net revenue increase from purchasing β fraction of the blocks. In the absence of an attack, the attacker's expected revenue is $\alpha \cdot r \cdot$ per block epoch. When β fraction of the network's valid blocks are discarded, the attacker's effective hash rate is $a = \alpha/(1-\beta)$, and hence his expected revenue is $a \cdot r$. Thus the attacker's extra revenue from purchasing the blocks is

$$a \cdot r - \alpha \cdot r = \frac{\alpha\beta \cdot r}{1-\beta}. \tag{2}$$

This quantity represents the attacker's block purchasing budget.

On the other hand, in order to incentivize a pool member to withhold a block, the attacker would have to offer at least the equivalent of the member's reward for finding a full solution. As β fraction the miners do not submit their blocks to the network, in the long run network difficulty decreases by a multiplicative factor of $(1-\beta)$. Hence per block epoch those miners would collectively expect to find $\beta/(1-\beta)$ valid blocks and would expect to be paid

$$\frac{\beta \cdot s \cdot r}{1-\beta} \tag{3}$$

for this work by the pool manager. Comparing the quantities (2) and (3), we see that the attacker and participating pool members both profit when

$$\alpha > s. \tag{4}$$

We now analyze the share rewards of miners. The two most popular share rewards schemes are the *pay per share* (PPS) and *pay per last N shares* (PPLNS) [10]. In addition, some pools offer bonus payments for miners who submit full solutions.

PPS. In the pay-per-share scheme, every pool miner receives a reward for every share (whether it constitutes a block or only a partial solution) he submits. Initially, the miner sets a share difficulty d and receives $(r \cdot d)/D$ reward for every submitted share, where D is the difficulty level of the Bitcoin network As of November 1, 2016, $D \geq 253,618,246,641$ Gig [6]. A pool member can set his own share difficulty for each of his ASIC hardwares,

[6] https://blockchain.info/charts/difficulty

however the recommended upper bound is currently $d \leq 4,096$ Gig [7] [8]. Hence, in PPS

$$s = \frac{d}{D} \approx 2 \cdot 10^{-8}, \tag{5}$$

and a rational miner would, at the current block reward rate, agree to withhold his blocks for $r \cdot s \approx (12.5 \text{ btc}) \cdot s = 2.5 \cdot 10^{-7}$ btc, which, as of November 1, 2016, is less than 0.02 cents of a USD [9]. In practice, the attacker likely has to pay more than 0.02 cents in order to motivate pool members to divert their standing loyalties away from pool managers and to compensate them for the risk that the pool manager will run out funds and will not be able to pay them for their previously submitted shares. To aid in this overcoming this inertia, we introduce block withholding pools in Section 5. Combining Equations (4) and (5) we find that the attacker could make a profit if his mining power fraction is at least $1/50,000,000$ of the network (0.000002%). This mining power is currently equivalent to 4 TH/s mining power, which is obtainable by modern ASICs [10]. Moreover, a miner with N ASICs could offer a reward that is N times higher and still make a profit. We note that all the large mining pools work in the PPS model [11]. Hence, all of them are potentially vulnerable to such an attack.

PPLNS. In the pay-per-last-N-shares model, at a high level overview, all miners share the mining rewards proportionally to their relative hash power. In this model, block shares and standard shares equally count towards proof-of-work, however withholding a block would lower the total revenues of the pool and inevitably also the rewards of the single miner. Hence, the effective block reward for a pool member is $s = \gamma$ where γ is the miner's hash power divided by the entire pool's hash power [12]. Hence, the attacker would profit only if $\alpha > \gamma$. We speculate that in most common cases, $\gamma \gg 1/50,000,000$ and thus the price of the attack is more expensive in the PPLNS model (see Table 2 as an example). Nevertheless, the attack could still be profitable for big miners who possess a percent or more of the entire network's hash power. An instantiation of our mathematical analysis can be derived from P2Pool publicly available statistics [13], which present the hash power of every miner in the pool. Table 2 shows the damages that an attacker could cause P2Pool, under the assumptions that P2Pool employs a pure PPLNS scheme and its miners are rational.

Finally, some pools try to prevent block withholding by giving special bonuses for miners who submit full solutions. These rewards must be limited to a few percent of a block reward, as higher bonuses would significantly increase the variance of payouts to pool members (*e.g.*, in P2Pool the bonus is 0.5%). If a pool offers p fraction of a block reward as a special bonus, then $s = p \cdot r$ and an attack is profitable only if $\alpha > p$, that is, only if the attacker hash power is greater than p.

[7] https://slushpool.com/help/#!/first-aid/troubleshooting
[8] Our analysis is valid even for much larger difficulty levels.
[9] http://www.coindesk.com/price/
[10] https://www.bitmaintech.com/
[11] https://en.bitcoin.it/wiki/Comparison_of_mining_pools
[12] The miner would also get the standard share reward, however, these are typically smaller by a factor of over 10^9.
[13] http://p2pool.org/stats/

Attacker hash power	Pool revenue loses	Orphan blocks daily costs
0.1%	10%	$4
2%	22%	$80
4%	32%	$160
6%	37%	$240
13%	70%	$520

Table 2: The revenue losses that an attacker can cause a pool while making the attack profitable for himself. Attacker power is in percentages of the entire Bitcoin network hash power. Pool loses are in percentages of the total pool revenue. For example, an attack with 0.1% hash power (i.e., 0.001 fraction of entire Bitcoin's hash power) can cause pools' revenues to decrease by 10%. The results are based on miner's hash power distribution in P2Pool. The third column describe the daily costs that an attacker would have to bare if he also pays for orphan blocks, under the assumption of $12,000 block reward (see Section 4.3 for more details).

Remark 1. Our calculations hold also in the extreme case where $\beta = 1 - \alpha$, i.e., when all the network but the attacker are withholding their blocks. However, in reality, if the attacker would manage to attract non-negligible fraction of miners, then PPS pools will go bankrupt and PPLNS pools will suffer from massive abandonment rate, as it would become more profitable to mine solo. Hence, the plausible outcome of a successful attack is a change in the pool mining model (e.g., shift towards solo mining or private pools).

4 Proving Block Withholding

The attacker in Section 3 pays a *withholder* to refrain from broadcasting a valid block to the blockchain. In order to convince the attacker that a block has been withheld, the withholder has to prove that (i) he found a valid block; and (ii) he (or his pool operator) did not submit it to the rest of the network. We observe that even for the task for block verification, namely, to verify that 80 bytes data consists of a valid block header, one would have to store the entire blockchain inside a smart contract, and ask for a 1 MB block's transactions data as a witness for the validity of the block. This approach is infeasible as e.g., in Ethereum it would costs $76,000 to store 1 GB of data [14] (Bitcoin's blockchain size currently exceeds 100 GB [15]).

Thus, we relax the requirement for block withholding proof and ask for a *proof-of-stale-work*. Proof-of stale-work proves that a miner is performing $\mathrm{sha256}$ operations over some data without an intention of submitting full solutions to the blockchain. When the withholder allocates his mining equipment for stale work, the effective hash power of the network is reduced (see Section 4).

In the next two subsections we present two different approaches for proving stale work. The non-interactive approach requires only a single submission from the withholder whereas the interactive approach requires the attacker to respond to the withholder submission. The non-interactive scheme makes use of Ethereum's expressive scripting language. Under the non-interactive scheme the withholder can, in a single step, submit

[14] http://ethereum.stackexchange.com/questions/872/what-is-the-cost-to-store-1kb-10kb-100kb-worth-of-data-into-the-ethereum-block

[15] https://blockchain.info/charts/blocks-size

his proof-of-stale-work to an Ethereum smart contract and get paid for it in ether without trusting the attacker or vice versa. The more complex interactive scheme, while implementable in Bitcoin's limited scripting language, leaves the attacker more vulnerable to the withholders.

In Section 4.3 we discuss how to mitigate the submission of orphan blocks.

4.1 Non-interactive proof

A *non-interactive proof-of-stale-work* is a tuple (b_1, b_2, b_2', b_3), where:

- b_1, b_2, b_2', b_3 are block headers; and
- b_2 and b_2' both extend b_1; and
- b_3 extends only b_2.

Intuitively, b_2' is the withheld block, and the fact that b_3 extends b_2 implies that b_2 is in the blockchain. Formally, in order to prove stale work, we consider two distinct cases:

- In the first case the miner who found b_2' never intended to submit it to the blockchain. In this case, the proof trivially follows.
- In the second case, the miner did submit it to the blockchain. In this case, the network had no incentive to find an extension for b_2 and therefore the withholder would have to spend effort in computing b_3 [16]. In this case, the withholder did stale work to find b_3, and the proof follows.

While it is possible to implement this scheme as an Ethereum smart contract (see Figure 1), one cannot implement it in the current Bitcoin script language as Bitcoin's parsing functionality is currently disabled [17]. Indeed Bitcoin transactions cannot even extract the previous block hash out of a block header.

Remark 2. It is possible to target the attack towards a specific pool. The block header contains some information on the destination account of the block reward and it is possible to extract it if the withholder provides the leftmost branch of the block transaction Merkle tree. The connection between the account and the pool operator is typically public information. Hence the attacker could reward only blocks that are associated with certain accounts.

4.2 Interactive proof

Let us recall that the header for a valid block, after two composed invocations of $\mathrm{sha256}$, has many leading zeros. The data B in the first step below corresponds to a single $\mathrm{sha256}(b'')$ for some valid block header b''.

- Initially, the withholder submits a 32-byte of data B with $\mathrm{sha256}(\mathsf{B})$ that matches Bitcoin difficulty level (i.e., has enough leading zeros).
- The attacker has time period T to find a block header b' such that $\mathrm{sha256}(b') = \mathsf{B}$.
- The attacker pays if and only if he did not find b' after time period T.

[16] To prevent cases where it would be profitable to find b_3 for the purposes of selfish mining, we could ask for a chain of blocks that extend b_2 rather than only a single block.

[17] https://en.bitcoin.it/wiki/Script

```
function verifyProofOfStaleWork( bytes b1, bytes b2, bytes b2_, bytes b3 ) returns(bool) {
        uint prevB2    = 0;
        uint prevB2_   = 0;
        uint prevB3    = 0;
        for( uint index = 4 ; index < 32 ; index++ ) {
            prevB2 = b2[ index ] | ( prevB2 * 256 );
            prevB2Prime = b2_[ index ] | ( prevB2_ * 256 );
            prevB3 = b3[ index ] | ( prevB3 * 256 );
        }
        if( prevB2 != prevB2_ ) return false;
        if( prevB2 != sha256(sha256(b1))) return false;
        if( prevB3 != sha256(sha256(b2))) return false;
        if( b2 == b2_ ) return false;
        uint lowestDifficulty = sha256(sha256(b1))  | sha256(sha256(b2)) |
                                sha256(sha256(b2_)) | sha256(sha256(b3));
        if( lowestDifficulty > difficulty ) return false;

      if( prevSubmissions[b2_] ) return false;
      prevSubmissions[b2_] = true;

        return true;
    }
```

Fig. 1: Solidity code that verifies proof-of-stale-work.

(i.e., $\mathrm{sha256}(\mathrm{sha256}(b''))$ matches the difficulty level). If the valid block b'' was submitted to the blockchain [18], then the attacker could easily come up with $b' = b''$ say after $T = 1$ *day*. Otherwise, finding the pre-image of $\mathrm{sha256}$ is computationally infeasible, and the attacker would not be able to find b' in the time period T.

Formally, we first claim that finding B requires roughly the same amount of work as finding a valid block header. Indeed, although technically only half of the $\mathrm{sha256}$ operations are required in order to find B (as in block mining one would have to compute the $\mathrm{sha256}$ function twice for every candidate byte stream), we conjecture that using the existing mining ASICs it is faster to find a block header and take B as its $\mathrm{sha256}$, rather than specifically looking only for B. Given the claim and the impossibility of finding a $\mathrm{sha256}$ pre-image, it is straightforward that if the withholder did the stale work, then he will get paid, and his work was not stale, then he will not get paid.

The interactive scheme requires a script language that can (i) compute $\mathrm{sha256}$; (ii) make 32-byte integer comparison; and (iii) store state (to store the withholder submission). Out of the three, only the first is possible with Bitcoin script language. In Appendix A we show how to perform this scheme over Bitcoin with several off-chain operations, and in Appendix B we show how to use Ethereum smart contracts to force correctness of the off-chain operations.

4.3 Mitigating orphan blocks

The two approaches we described above are not resilient to submission of orphan blocks. In theory, an orphan block and a withheld block are not different. Hence, in this section we focus on the practical implications (e.g., attacker's losses) that orphan blocks introduce, and suggest practical ways to mitigate them.

[18] We make the assumption that orphan blocks are also publicly visible, e.g., see https://blockchain.info/orphaned-blocks.

We first focus on the expected losses of the attacker due to orphan blocks. In the 365 days between March 2016 and March 2017, 129 orphan blocks were recorded [19]. Hence, in our analysis we assume 0.35 orphan blocks to occur every day (on average). Hence, ignoring orphan blocks will cost an attacker $0.35sr$ per day. The third column in Table 2 illustrates the daily loses of an attacker who attacks P2Pool. Our analysis suggests that an attacker who could afford to pay for orphan blocks will bare loses of \$4 per day, while decreasing the victim pool's revenue by 10% (which might be enough to make all miners leave the pool). However, the costs could rise up to $\$520$ per day, if the attacker wishes to reduce the victim pool's revenue by 70%. In order to evaluate the total costs of an attack, one would have to speculate on the number of days a pool can successfully survive such an attack. We leave this empirical evaluation to future research. We note that in networks with lower block intervals like Ethereum who operates with the GHOST [11] protocol, the rate of stale blocks is much higher. However, in Ethereum, stale blocks are rewarded and are also included in the blockchain (as so called uncle blocks). Hence, our schemes should be adjusted to verify that the blockchain does not contain the submitted block as an uncle block. Finally, we conjecture that Bitcoin's low orphan block rate might be the result of highly centralized miners network, and if the network were truly decentralized more orphans would occur.

We now suggest practical ways to mitigate the losses of the attacker.

The non-interactive scheme of Section 4.1 could reject orphan blocks by requiring that the timestamps of b_2 and b'_2 to differ by at least one minute. The publicly available orphaned blocks statistic in [20] for the period of January till March 2017 suggests that in practice Sybil blocks (i.e., the orphan block and the accepted block) timestamp differ by at most 40 seconds. Hence, our restriction will prevent the submission of orphan blocks, but might deter withholders to withhold their blocks, as the contract will not accepted it if a real block is mined in the following minute. For this purpose the attacker should increase the offered reward sr by a factor of $q = e^{0.1} \approx 1.105$. Now the withholder will receive qsr provided that no additional block was mined in the following minute (and the probability that 0 blocks are mined in a single minute is $e^{-0.1}$), and will receive nothing if additional block was mined. Hence, the expected reward is still sr, and our theoretical mathematical analysis from Section 3 still holds. A further empirical study is needed to evaluate the motivation of big pools to skew their timestamps to mitigate our attack and the reaction of withholders to the increase in the variance of their reward.

The interactive scheme of Section 4.2 can mitigate orphaned blocks either by assuming that the attacker is always aware of orphaned blocks (e.g., via public blockchain explorers like blockchain.info or by becoming a peer of all major pools), or by giving incentives to the rest of the network to report that a submitted block is an orphaned block. The latter solution would require the submitter to deposit some collateral along with his submission of B. If a preimage of B is submitted by a peer in the network then half of the collateral is given to this peer (and other half is slashed). An empirical experiment is needed to evaluate the number of orphaned blocks that are not presented in blockchain.info and the affect of collateral (and collateral size) on the willingness of withholders to participate and of peers to report on a preimage.

[19] https://blockchain.info/charts/n-orphaned-blocks?timespan=1year
[20] https://blockchain.info/orphaned-blocks

5 Block Withholders Pool

We discuss two factors that could deter a miner from participating in the withholding scheme from Section 4.

- *Ethical and long-run considerations*. Participating as a withholder might violate some agreements with the pool operator. In addition, if such attacks were to become common, miners might face the risk that all pools will cease to operate. In the absence of pools, miners' income variance would become undesirably high.
- *Complicated setup for a rare chance to profit*. On the one hand, collaborating with the attacker requires the miner to install a special patch for his mining software. On the other hand, a miner could only withhold a block after he finds at least one valid block, which is not even a once in a lifetime event for most small miners. Even if the attacker offers high reward, most small miners would likely not be willing to make the effort and update their software for an event that is unlikely to happen.

In this section, we describe how to mitigate the second issue by forming a pool of block withholders, which reduce the variance of the reward and incentives small miners to participate in the attack. Intuitively, in a *withholders pool* miners submit proof-of-work shares to demonstrate their hash power and share attacker rewards for withholding blocks. The withholders pool distributes block withholding rewards among miners in proportion to their relative hash power.

In order to make the scheme profitable, the block withholder pool's proof-of-work should correlate with the work the miners do for their legitimate pool operator. Otherwise the additional proof-of-work would lead to financial losses. In classical mining pool models, shares must include the pool operator's data in order to ensure that profit from valid blocks gets distributed among pool members. By analogy, in order to ensure fair reward distribution in a withholder pool, the attacker,who distributes all withholding rewards, must serve as the operator. To be precise, the attacker should form exactly one withholding pool and declare that all rewards are routed via that pool. As the attacker may not be trusted, he should form a smart contract that collects proof-of-work and distributes the reward once a proof of block withholding is submitted (as described in Section 4).

Ironically, such withholder pool is vulnerable to a *block-withholding-withholding attack*, where miners could avoid submitting the withholding proof to the attacker and instead submit them to their legitimate pool operator. Thus the attacker's payments to withholder pool members must suffice not only to convince miners to install the mining software patch for dropping valid blocks but must also directly and fairly compensate pool members who actually perform withholding. The payment to the member who performs a withholding must exceed what he would have received for submitting his same share to the legitimate pool operator instead. Moreover, the attacker must distribute additional funds to the remaining pool members via some PPLNS scheme in order to motivate miners to install the software patch which includes instructions for diverting valid blocks. In short, while running a withholding pool increases chances that miners will participate in an attack, it also increases the attacker's execution costs.

6 Related Work

Recent literature has pointed incentive structure flaws in pooled mining [12,3] as well as Nakamoto consensus itself [13,14,15]. In many of these instances, as in this work, the

attacker benefits by withholding publication of a live block. References [5,6,3] showed that a miner who mines in multiple pools simultaneously and withholds publication in one of them can, on average, increase his expected net mining profit while decreasing the revenues of the attacked pool. However, in all of these works the attacker must have substantial mining power in order to make a significant attack. For example, a miner who wishes to attack a big pool like F2Pool (for example), which currently posses 20% for the network hash power [21], should initially possess 6.7% [3] hash power in order to not lose money during the attack. In our work, in theory, a mining power of 0.000002% is enough to cause losses to the victim pool. On the other hand, our attack requires cooperation of other parties and guarantees success only if the other parties are rational with respect to their short-term revenues.

Recently Luu *et al*. propose a new efficient decentralized pooled mining protocol using Ethereum smart contracts [16]. Such a protocol, if is deployed at a cryptocurrency's protocol level to build the only mining pool in the network, can prevent block withholding attack to pooled miners.

Teutsch, Jain, and Saxena recently proposed to attack blockchain miners by paying them to use their mining equipment for non-mining purposes (e.g., to solve non-blockchain PoW puzzles) [15]. Bonneau suggested to bribe miners [17] or equivalently rent their equipment, and instruct them how to mine. In both of these options, the attacker can benefit by working on a private chain that will eventually exceed the public chain length, and thus could collect all block mining rewards. However the initial mining power to make the first such attack profitable in Bitcoin is 38.2% [15], while the latter attack relies on exotic rationality assumptions. Our attack is inspired by these attacks as it conceptually pays pool miners to perform certain work (i.e., stale work). It exploits the fact that the pool operator pays for most of the miners work, and thus we can make a profitable attack with very small initial hash power. On the other hand, the above attacks directly affect the core blockchain protocol and demonstrate vulnerabilites in Nakamoto consensus itself. Our attack affects only the pool mining protocol, which is not part of Nakamoto consensus.

Sometimes by sheer luck a miner who controls a significant portion of the network's mining power can win two or more blocks in rapid succession. In this case the miner can, on average, increase his profit by withholding a block as the basis for a longer private chain and mining on top of it. Just before the public chain catches up to the private one, the attacker releases his block, making the private chain both public and valid, and wasting the efforts of other miners who were mining on top of the former public chain. This attack is known as selfish mining [13], and has been recently optimized [18] and combined [19] with the network-layer eclipse attack [20].

Finally, we note that not all exploitable incentive structure flaws found in Nakamoto consensus necessarily manifest themselves as block withholding attacks. Miners who benefited from the recent denial-of-service attacks in Ethereum [21] made use of a "verifier's dilemma" [14] to waste others' time, while publishing their own blocks quickly.

Acknowledgments. We thank our shepherd, Iddo Bentov, for useful discussions and the anonymous reviewers of an earlier draft of this paper for helpful feedback.

[21] https://en.bitcoin.it/wiki/Comparison_of_mining_pools

References

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.
2. Meni Rosenfeld. Analysis of Bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011.
3. 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 *2015 IEEE 28th Computer Security Foundations Symposium*, pages 397–411, July 2015.
4. Ethereum Foundation. Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.
5. Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in Bitcoin digital currency. *CoRR*, abs/1402.1718, 2014.
6. I. Eyal. The miner's dilemma. In *SP*, 2015.
7. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, 2014.
8. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.
9. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf, 2014.
10. Bitcoin Wiki. Pool mining's payout schemes. https://en.bitcoin.it/wiki/Comparison_of_mining_pools.
11. Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 507–527, 2015.
12. Ittay Eyal. The miner's dilemma. In *IEEE Symposium on Security and Privacy (SP 2015)*, pages 89–103, May 2015.
13. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer Berlin Heidelberg, 2014.
14. 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 2015)*, pages 706–719, New York, NY, USA, 2015. ACM.
15. Jason Teutsch, Sanjay Jain, and Prateek Saxena. When cryptocurrencies mine their own business. (to appear in *Financial Cryptography and Data Security (FC 2016)*).
16. Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. Smart pool : Practical decentralized pooled mining. Cryptology ePrint Archive, Report 2017/019, 2017. http://eprint.iacr.org/2017/019.
17. Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, pages 19–26, 2016.
18. Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. (to appear in *Financial Cryptography and Data Security (FC 2016)*).
19. Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 305–320, March 2016.
20. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX 2015)*, pages 129–144, Washington, D.C., August 2015. USENIX Association.
21. https://www.reddit.com/r/ethereum/comments/55xh2w/i_thikn_the_attacker_is_this_miner_today_he_made/.

22. Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012:248, 2012.
23. blockcypher.com. Confidence factor. http://dev.blockcypher.com/#confidence-factor.
24. Bitcoin Wiki. Transaction malleability. https://en.bitcoin.it/wiki/Transaction_Malleability.

A Bitcoin Implementation

In this section, we refine the interactive protocol from Section 4.2 for use in Bitcoin. The security of our Bitcoin protocol relies on the following two assumptions which we will later relax in Section B:

- The attacker always wants to attack. That is, he is always willing to pay a predefined amount for a valid proof of block withholding.
- The withholder is willing to withhold the block in return for a Bitcoin zero-confirmation payment.

The first assumption is reasonable as the attack is profitable. However, it is not trivial, as malicious parties could dishonestly declare their intentions to make such an attack but never collaborate with the withholder. Such behavior might be expected, e.g., by pool operators who wish to undermine trust between attackers and withholders. The second assumption could be justified as zero confirmation double spending is not trivial to perform [22]. Our protocol would allow the withholder to wait for a short period of time before deciding on his actions. In this period of time the transaction would propagate to the majority of the network, and the odds for double spending could be evaluated and bounded from above, e.g., via [23]. If odds are, for example, less than 50%, then it is enough to double the offered reward in order to incentives the withholder. In Section B we will introduce Ethereum smart contracts that enforce our assumptions. That is, the contracts would compensate the withholder (in ether currency) if the attacker does not collaborate with the protocol or performs double spending.

We are now ready to introduce the protocol.

- Initially, the withholder submits (off-chain) a 32-byte chunk of data b and his Bitcoin public key.
- The attacker computes $\mathrm{sha256}(b)$ and rejects the submission if: (i) the difficulty level is not sufficient; or (ii) $\mathrm{sha256}(b)$ corresponds to a block in the public blockchain; or (iii) b was already submitted in the past.
- (Otherwise) The attacker signs and sends the withholder a Bitcoin transaction t such that:
  - The attacker can redeem t with an input string b' that satisfies $\mathrm{sha256}(b') = b$.
  - The withholder can redeem t after T block epochs (provided that it was not already redeemed).
- The withholder submits t to the network, waits for it to propagate, withholds his block, and redeems t after T block epochs.

The correctness of the scheme follows by our two assumptions and by the arguments of the correctness proof of the protocol in Section 4.2. We note that in the last phase

of the protocol, the withholder cannot afford to wait for a block confirmation. Indeed, a block confirmation occurs only after a new block is mined, and when this happens the withholder's block becomes worthless as he can no longer submit it to his pool operator [22].

An implementation of transaction t as a Bitcoin script is illustrated bellow.

Implementation with bitcoin script. Transaction t locking script is:

```
OP_IF
    OP_HASH256
    <b>
    OP_EQUALVERIFY
    <buyer_public_key>
OP_ELSE
    <time_lock> OP_CHECKLOCKTIMEVERIFY OP_DROP
    <seller_public_key>
OP_ENDIF
OP_CHECKSIG
```

The buyer can redeem t with this unlocking script:

```
<signature>
<b'>
OP_1
```

The seller can redeem t after sufficient enough time with this unlocking script:

```
<signature>
OP_0
```

We note that in order to make these transaction standard we use pay to script hash transactions [23].

B Ethereum Contracts as Insurance

In this section, we describe two Ethereum smart contracts that eliminates the need for the assumptions we made in Section A. The contracts provides the following guarantee:

> The withholder is either payed the promised amount in bitcoin or payed disproportional high value in ether currency.

Such guarantee should mitigate any concern from the withholder side, even if he has strong preference towards bitcoin payments. Indeed, either he gets payed with bitcoin or he receive high ether payment that compensates for his bitcoin preference.

We first describe how to mitigate the assumption that the attacker always wants to attack, and then describe an Ethereum insurance contract against Bitcoin double-spending. For the rest of the section we assume that $1,000$ ether (currently worth around $\$10,000$ USD [24]) are enough to compensate for any preference towards Bitcoin payment.

[22] E.g., see line 110 in https://raw.githubusercontent.com/slush0/stratum-mining/38637575c8c253aba18f95dffd25c49ca6d0434b/lib/block_template.py

[23] https://en.bitcoin.it/wiki/Pay_to_script_hash

[24] https://www.coingecko.com/en/price_charts/ethereum/usd

B.1 Forcing the attacker to attack

In this section we describe how an Ethereum contract (published by the attacker) can enforce the attacker to honestly execute his part in the protocol. We recall that the attacker role in the protocol is to publish a signed transaction t when a valid block withholding witness b is submitted, i.e., when a never before submitted block header with sufficient difficulty is submitted. The contract has four functions, namely, **depositCollateral**, **submitWitness**, **submitTx** and **seizeCollateral**.

- **depositCollateral**: in this function the attacker deposits $1,000$ ether.
- **submitWitness**: in this function the withholder submits the witness b and his Bitcoin public key. The function checks that b was never submitted before and its difficulty is sufficient (i.e., $\mathrm{sha256}(b)$ is small enough), and records the current time.
- **submitTx**: in this function the attacker submits a signed tranasction t and the contract verifies that the transaction is properly signed in the format as described in Section A.
- **seizeCollateral**: in this function the withholder can withdraw the $1,000$ ether if the attacker did not respond in time (or responded with invalid transaction).

See Figure 2 for partial implementation of the contract. Intuitively, the contract enforce the

```
1    function submitWitness( bytes b, uint publicKey ) {
2        if( sha256(b) < difficulty ) {
3            witnessSubmissionTime = now;
4            withholderPublicKey = publicKey;
5        }
6    }
7
8    function submitTx( bytes t ) {
9        if( isSigned( t, attackerPublicKey ) && isInRightFormat(t, b, withholderPublicKey ) ) {
10           txSubmitted = true;
11       }
12   }
13
14   function seizeCollateral( ) {
15       if( ! txSubmitted && ( witnessSubmissionTime + T < now ) ) {
16           msg.sender.send(1000 ether);
17       }
18   }
```

Fig. 2: Solidity code that force the attacker to attack.

attacker to post a signed Bitcoin transaction to the Ethereum blockchain (within a given time period T). Once published, the withholder can post it to the Bitcoin network and claim his payment in bitcoin currency. If the attacker decides not to post the transaction, then the withholder collects the collateral that serves as a compensation for the block withholding and for getting paid in ether.

We note that the contract can serve as an insurance only when the balance is sufficient (i.e., when a collateral is deposited). Hence, the withholder should check the balance before participating in the scheme [25].

[25] To mitigate the incentive for the attacker to seize the collaterals and give it to a sibyl identity, we can change the contract so it would give only half of the collateral and the other half would be destroyed (e.g., would be sent to address 0x000...000).

B.2 Insurance against double-spending

In this section, we introduce an Ethereum contract that serves as an insurance against Bitcoin double-spending scenarios. We use it to mitigate the zero-confirmation assumption for our Bitcoin implementation of the block withholding attack. The contract provides an insurance for up to simultaneous N double-spending operations of a single Bitcoin address.

Formally, we say that a transaction tx is double-spent by address a if tx is signed by a and there exists another signed transaction tx' such that tx and tx' share at least one common input and differ by at least one output [26].

The contract is illustrated in Figure 3. In **createInsurance** function the owner of the bitcoin account deposits $1,000$ ether for every insured double-spending operation. In **claimCompensation** function, the victim submits the witness for double-spending and unlocking script for the controversial output, as a witness for being eligible for compensation. To prevent a sibyl attack, where the owner of the insured account claim N compensation units to himself, we could half the compensation and destroy the rest of the 500 ether.

```
1   function createInsurance( uint publicKey, uint N ) {
2       # check if there are enough funds
3       if( msg.value != N * 1000 ether ) throw;
4       insuredAccount = publicKey;
5   }
6
7   function claimCompensation( bytes tx, bytes txPrime, uint inputIndex,
8                               uint outputIndex, bytes unlockScript ) {
9       if( isSigned( tx, insuredAccount ) && isSigned( txPrime, insuredAccount) )
10          if( inputTx(tx, inputIndex) in txPrime )
11              if( outputTx( tx, outputIndex ) not in txPrime )
12                  if( runBTCScript( outputTx( tx, outputIndex ), unlockScript ) )
13                      msg.sender.send( 1000 ether );
14  }
```

Fig. 3: Insurance for bitcoin double-spending.

[26] A naive approach that only search for common inputs and check that $tx \neq tx'$ would fail due to Bitcoin's transaction malleability issue [24].