

ZeroBlock: Timestamp-Free Prevention of Block-Withholding Attack in Bitcoin

Siamak Solat

UPMC-CNRS, Sorbonne Universités,
LIP6, UMR 7606, Paris, France
firstname.lastname@lip6.fr

Maria Potop-Butucaru

UPMC-CNRS, Sorbonne Universités,
LIP6, UMR 7606, Paris, France
firstname.lastname@lip6.fr

Abstract—Bitcoin was recently introduced as a peer-to-peer electronic currency in order to facilitate transactions outside the traditional financial system. The core of Bitcoin, the Blockchain, is the history of all transactions committed by the system. This distributed ledger is similar to a distributed shared register where miners write and read blocks. New blocks in the Blockchain contain the last transactions in the system and are added by miners after a block mining process that consists in solving a difficult cryptographic puzzle. Although, the reward is the main motivation for the mining process in Bitcoin, it also may be an incentive for attacks such as *selfish mining*. In this paper we propose and theoretically analyze a solution for one of the major problems in Bitcoin : *selfish mining* or *block-withholding* attack. This attack is conducted by adversarial miners in order to either earn undue rewards or waste the computational power of *honest* miners. Contrary to the best to date solution for preventing *block-withholding*, [23], our solution, *ZeroBlock*, prevents this attack by using a novel timestamp-free technique that exploits the Poisson nature of the proof-of-work and the current knowledge on the propagation of information in Bitcoin [3]. Note that previous solutions are vulnerable to forgeable time-stamps. Additionally, our solution is compliant with miners churn.

I. INTRODUCTION

In the last few years crypto-currencies [17], [19], [20], [18], [1], [8], [15] are in the center of the research ranging from financial, political and social to computer science and pure mathematics. Bitcoin [1] was one of the starters of this concentration of forces. It targeted the creation of a system where transactions between individuals can escape the strict control of the banks and financial markets.

Bitcoin was introduced as a pure peer-to-peer electronic currency or crypto-currency. It aims at fully decentralization of electronic transactions. Bitcoin allows to perform online transactions directly from one party to another one “without” the interference of a financial institution as a “trusted third party” [1]. It uses digital signatures to verify the bitcoin ownership¹ and employs Blockchain in order to prevent double-spending attacks. In this attack the same bitcoin can be spent several times by a dishonest party. Blocks in the blockchain are created via a proof-of-work (cryptographic puzzle) [13], [2], [12], [42] performed by *honest* parties (miners that follow the protocol). Blockchain is further broadcasted via a peer-to-

peer overlay in order to agree on a common history of the transactions in the system.

Bitcoin is still vulnerable to various attacks including double-spending [24], *selfish* mining [9], Goldfinger [25], 51% attack [25] etc. In this paper we focus the *selfish* mining attack. Recently, [4] provided a full description of incentives to withhold or *selfish* mine in Bitcoin. That is, to force *honest* miners to waste their computational power such that their public blocks become useless (as *orphan* block), whereas the private chain of the *selfish* miners is accepted as a part of the Blockchain. To this end, the *selfish* miners reveal selectively their private blocks to make useless the blocks made by *honest* miners.

Our contribution. In this paper, we present and prove correct a new solution, ZeroBlock, which prevents *block withholding* or *selfish* mining. ZeroBlock scheme, contrary to the recent solution proposed by [23], does not use forgeable timestamps. Our solution builds on the following simple idea: if a *selfish* miner keeps a block private more than a fixed interval of time, its block will be rejected by all the *honest* miners. Zeroblock scheme strives to reduce the probability of *intentional* forks that are result of block-withholding attacks. With ZeroBlock scheme a selfish mining pool cannot achieve more than its expected reward. Only with a low probability, selfish mining pool may create intentionally an *unprofitable* fork. We accentuate “unprofitable”, because this fork does not lead to more reward for selfish mining pool, but also reduces selfish pool’s likelihood to earn unexpected reward regardless of to its mining power. Thus, selfish mining pool is not incentivized to create such fork if its purpose is to achieve more reward. Furthermore, we prove that the maximum probability of such *intentional* fork is very low (≈ 0.04) when selfish pool uses its maximum hashing power. We further extend ZeroBlock in order to be tolerant to miners churn.

Paper Roadmap. The rest of this paper is organized as follows: Section II presents an overview of Bitcoin. Section III presents *block withholding* attack or *selfish* mining and briefly discuss the differences between an intentional and an accidental attack. Section IV presents our ZeroBlock timestamp free algorithm. Section V, proves that when ZeroBlock scheme is used a selfish mining pool cannot achieve more than its expected reward. Only with a low probability, selfish mining pool may create intentionally an *unprofitable* fork.

¹Capitalized “Bitcoin” refers to the protocol, while lower case “bitcoin” refers to the coin.

Section VI extends ZeroBlock algorithm in order to make it compliant to miners churn. Finally, Section VII concludes the paper.

II. BITCOIN OVERVIEW

Bitcoin is an electronic coin which works as a chain of digital signatures where each owner transfers bitcoins to the next party after adding his signature (generated with his private key) along with the hash of previous transactions and the public key of the receiver. As a result, the final receiver is able to verify the bitcoins ownership by verifying the signatures [1].

The key data structure in Bitcoin, called Blockchain, is a public log that is a sequence of blocks that maintain a linearized history of transactions. The safety of Blockchain is ensured by a cryptographic puzzle, named proof-of-work (PoW), solved by several nodes, named “miners”. Miners who can solve PoW are permitted to generate a new block to record transactions and receive some bitcoins as a reward. This reward is used to further motivate miners to share their power resource with the network and continue to mine.

In Bitcoin system builds on top of two key concepts. The first one are *Transactions* : used to transfer coins in the network. The second one are *Blocks*. Blocks are inserted into the *public ledger Blockchain*. Blockchains are further employed to synchronize the state among all miners [3] and to provide to each participant in the system a common view on the history of the transactions in the system. In the sequel we detail these two key notions.

Transactions. A *transaction* transfers bitcoins from one or multiple source account addresses to destination account address. Each account address has a unique key. For transferring a bitcoin between two accounts, a *transaction* is generated. This transaction includes the destination account address and it must be signed by the private key of the source account. To calculate the balance of the accounts, the public ledger calculates each *transaction* output (i.e. a numeric value of coins). Each *transaction* is recognized by using the hash of its serialized context.

When *transactions* are broadcasted into the network, the public ledger will be updated locally by each miner. Without additional care, this copy may be different at distinct miners which may lead to inconsistencies. For example, consider that a miner receives a *transaction* that shows the transfer of some bitcoins from some *account* “A”, while the coins are not “available” for this particular account. In a different situation, several transactions may try to transfer the same coins more than once. Such subversive behavior is known as “*double spending*” [3]. In order to avoid that transactions appear into an inconsistent order at different miners in the network they are encapsulated in blocks that are further chained in a linearizable Blockchain.

Blocks. When a miner receives a transaction it starts a mining process in order to generate the block that will encapsulate the transaction. Bitcoin network uses Hashcash [2] proof-of-work system for block generation such that a block is accepted by the network if miners perform proof-of-work properly and

successfully. A proof-of-work (PoW) is a cryptographic puzzle that is difficult to solve but easy to verify. The difficulty of PoW is adjustable regarding to the hashing power of the network. Currently, the block generation rate is set to one block per 10 minutes [32]. Note that since the success of solving proof-of-work by a miner has very low probability, it is almost impossible to predict which miner or mining pool will generate the next block.

When a new block is generated, it is broadcasted in the entire network. If this new block is accepted as head of Blockchain, other miners start to work on this new block to extend the Blockchain. A competition between two new blocks may occur when their preceding block is equal and they are broadcasted simultaneously. At this point Blockchain may fork. Since broadcasting a block takes only a few seconds but the average time to discover a new block is around 10 minutes, thus, *accidental fork* occurs almost every 60 blocks [4], [3]. In such situation, miners choose the first block which they receive and as a result, the second block will be ignored by the network as an *orphan* block. Consequently, miners that worked on the second block wasted their computational power with no reward.

Blockchain and Forks in Bitcoin. A *Blockchain* is a chain of *blocks* in “chronological order”. Each *block* has a *parent block*. The *genesis block* is the root. The most distant *block* from *genesis block* is called the *head of Blockchain*.

If two miners create two blocks with the same preceding block, Blockchain is *forked* into two branches. This fork may occur *accidentally* or *intentionally*.

Note that an *accidental* fork occurs due to the nature and functionality of proof-of-work and it is not related to some particular attack as block-withholding or selfish mining. Since proof-of-work is a Poisson process, two blocks may be discovered by two mining pools. The probability of an accidental fork is ≈ 1.69 [3] .

In case of an *intentional* fork, a *selfish* miner generates and keeps private a block until it estimates opportunistic to reveal it. When a *honest* miner generates a new block with the same preceding block as the one generated by the *selfish* miner, the latter propagates its private block to create an “intentional fork”. The *selfish* miner can generate more than one private block in order to take the control of the Blockchain since the longest chain will be the only one commonly accepted by the honest miners in the network.

III. BLOCK-WITHHOLDING ATTACK

Block withholding attack was introduced as “*Selfish mining attack*” in [4] and also as “*Block Discarding Attack*” in [26]. This attack relies on “block concealing” and revealing only at a special time selected by *selfish* miners or *selfish* mining pool. According to [4], these *selfish* miners can earn revenues superior to a fair situation [27]. That is, the main purpose of block-withholding by selfish mining pool is achieving more rewards in comparison with its hashing power in the network. Thus, selfish mining pool’s reward oversteps its mining power in the network and it can increase its expected mining reward.

In block-withholding strategy, a *selfish* miner after solving proof-of-work and finding a new block does not broadcasts until a specific time.

In [28] authors extend the *selfish* mining strategy and provide an algorithm to find optimal policies for *selfish* miners. [16] shows that expanding the Blockchain by adding the newest block creates a simple model of “weak and non-unique” Nash equilibrium [28]. [28] shows that these optimal policies compute the threshold such that honest mining would be a “strict and unique” Nash equilibrium.

According to [4], [23], the *selfish* mining in Bitcoin network occurs as follows: In case of generation of a new block by the *honest* miners, (1) if the size of *honest* branch is longer than the selfish branch, then the selfish cartel tries to set its private branch equal to the public branch. (2) If the selfish branch is one block more than the public branch, then selfish miners publish their private chain completely (3) If the selfish branch is more than one block longer than the public branch, then the selfish miners publish only the head of their private branch. In case of generation of a new block by selfish miners, they keep this new block private and in case of a competition with the *honest* miners, they publish their private branch to win the competition. According to [4], the success of selfish miners in this competition is contingent on the parameters α (i.e. hashing power of selfish miners) and γ (i.e. the hashing power of the *honest* miners who work on the selfish branch). According to equation 1, if $\gamma = 0$, the threshold for success of block-withholding behavior is $\alpha \geq 33\%$ and if $\gamma = 0.99$, the threshold is $\alpha \geq 0.009$ [23].

$$\frac{1 - \gamma}{3 - 2\gamma} < \alpha < \frac{1}{2} \quad (1)$$

Eyal and Sirer [4] suggest a solution according to which γ is fixed to 0.5 and consequently the threshold of successful block-withholding is $\alpha \geq 0.25$. Heilman [23] introduces an approach named “Freshness Preferred” (FP). Using random beacons and timestamps, *honest* miners select more fresh blocks and the threshold becomes $\alpha \geq 0.32$.

The rest of the paper is organized as follows. In Section IV we detail our ZeroBlock scheme that targets to reduce the probability of *intentional* forks that may result after block-withholding attacks appear in the network. In Section V we prove that using ZeroBlock idea a selfish mining pool cannot achieve more than its expected reward. Only with a low probability, selfish mining pool may create intentionally an *unprofitable* fork. We accentuate “unprofitable”, because this fork does not lead to more reward for selfish mining pool. Thus, selfish mining pool is not incentivized to create such fork. Furthermore, we prove that the *maximum* probability of such *intentional* fork is very low (maximum ≈ 0.04) when selfish pool uses its maximum hashing power. Furthermore, we extend the ZeroBlock in order to make it resilient to miners churn.

IV. ZEROBLOCK ALGORITHM

In this section we introduce a new solution, ZeroBlock (see Algorithm 1) to prevent *block-withholding* or *selfish* mining in Bitcoin. The key idea of our solution is that each block must be generated and received by the network within a *maximum acceptable time for receiving a new block* interval, *mat* (see equation 7 below). Within a *mat* interval a *honest* miner receives or discovers a new block. Otherwise, it generates a dummy block. The computation of each *mat* interval is done locally by each miner based on the following Bitcoin parameters: the *expected delay for a block mining* and the *information propagation time* in the Bitcoin network. The former parameter is discussed in details below while the latter has been extensively studied in [3] where the authors shown that a published block is received by the whole network within 60 seconds (see Figure 5 in the appendix).

Expected delay for a block mining in Bitcoin depends mainly on the difficulty of proof-of-work. The major part of proof-of-work consists in discovering a byte string, *nonce*. As pointed out in [3] proof-of-work in Bitcoin is a Poisson process and causes blocks to be discovered randomly and independently. Moreover, as advocated in [32], [33], [34], [35], [36], [37], [38], [39] in Bitcoin, the difficulty of proof-of-work required to discover a block is periodically adjusted such that, on average, *one* block is expected to be discovered every *10 minutes*. Hence, the difficulty of proof-of-work is updated every 2016 blocks. It means that regarding to this adjustment (i.e. one block per 10 minutes) 2016 blocks, on average, is expected to be generated in 14 days. If 2016 blocks are discovered in a shorter time, the difficulty of proof-of-work will be increased and if they are generated in a longer time, difficulty of proof-of-work will be decreased.

In more details, miners for each input of proof-of-work (i.e. a random *nonce*) calculate a hash value. This hash is a number between 0 and a maximum value of a 256-bit number. The miner has discovered the answer of proof-of-work, if and only if this hash is below the *target*.

The proof-of-work works as follows:

$$\text{if } H(pb + nonce) < T \text{ then} \quad (2)$$

proof-of-work succeeded

where *pb* is representing the hash of the previous block, *nonce* is the answer of proof-of-work that must be found by miners, *T* is *target*, ‘+’ is concatenation operation and *H* is the hash function.

Each mining pool can estimate the difficulty of proof-of-work using equation 3.

$$D = \frac{\maxTarget}{T} \quad (3)$$

where *D* is the difficulty of proof-of-work, *T* is current *target* and *maxTarget* is maximum possible value for *target* that is $(2^{16} - 1)2^{208} \approx 2^{224}$. Since the hash function produces

uniformly a random value between 0 and $2^{256} - 1$ thus, the probability that a given *nonce* value would be the answer of proof-of-work is as follows (equation 4):

$$\text{Prob}(\text{nonce is answer}) = \frac{\text{target}}{2^{256}} = \frac{2^{224}}{D \times 2^{256}} \approx \frac{1}{D \times 2^{32}} \quad (4)$$

The number of hashes to discover a block is $D \times 2^{32}$ in expectation. If a mining pool can calculate hashes at a rate *php* (we call this as pool's hashing power), then the expected time (or average time) *avt* in which this pool can discover a block is as follows (equation 5):

$$\text{avt}_{\text{pool}} = \frac{D \times 2^{32}}{\text{php}} \quad (5)$$

When we replace *php* by hashing power of the network, *nethp*, we can use equation 4 for the entire network as follows (equation 6):

$$\text{avt}_{\text{net}} = \frac{D \times 2^{32}}{\text{nethp}} \quad (6)$$

According to the relation between *time*, *difficulty of proof-of-work*, *hashing power of the network* in equation 6, Bitcoin network adjusts *D* such that regarding to hashing power of the network, the average time for block generation rate remains 10 minutes.

To calculate the *the maximum acceptable time for receiving a new block, mat*, we use equation 7 below:

$$\text{mat} = \text{avt}_{\text{net}} + \text{ipt} \quad (7)$$

where *avt_{net}* is given by the equation 6 and *ipt* is the information propagation time in Bitcoin network as estimated in [3].

A. Zeroblock Algorithm parameters and notations

The ZeroBlock algorithm (Algorithm 1) uses the following parameters and definitions:

- *ipt* : information propagation time in Bitcoin network that is an average delay for propagation a block into the network. This average delay has been estimated by simulation in [3] (see also Figure 1 in the Appendix).
- *avt* : block generation rate that has been set by Bitcoin protocol according to which the difficulty of proof-of-work is adjusted regarding to the hashing power of the network using equation 6.
- *mat* : maximum acceptable time for receiving a new block that is computed by equation 7. During a *mat* interval if a miner cannot solve the proof-of-work, it has to generate a dummy Zeroblock.
- *unpermitted block-withholding* : occurs when a *selfish* mining pool discovers a new block and keeps the block private after the end of the current *mat* interval.

- *Dummy Zeroblock* : is generated locally by miners. It includes the index of *mat* interval and the hash of previous block. It is generated by honest miners to prevent *unpermitted block-withholding*. Note that our solution uses *standard Bitcoin blocks* discovered by solving the proof-of-work and *dummy blocks* that are generated by the Zeroblock algorithm for which miners do not need to solve any proof-of-work. The dummy Zeroblocks time generation is therefore ignored when adjusting the difficulty of the proof-of-work.
- *orphan block* : a block that has been discovered but is then rejected by the network.
- *genesis block* : the first block of a Blockchain on which all miners have a consensus.
- *correct chain* : a chain whose blocks have been discovered and inserted correctly according to the described protocol.
- *creative miner* : a miner that in a *mat* interval can solve proof-of-work and then generates a new block.

B. ZeroBlock Algorithm Detailed Description

In this section, we describe the ZeroBlock algorithm shown as Algorithm 1. Each miner, μ , that executes Algorithm 1 performs the following steps:

- (lines 1 to 11) initialization: where *mat₀* and *scounter()* (seconds counter) are set to 0.
- (line 4) miner μ initiates its local chain by Genesis block.
- (line 5) *FlagNewBlock* is set to *False*.
- In (line 12) miner μ starts an infinite loop.
- In (line 13) miner μ checks (*FlagNewBlock = False*) (that means it verifies if there is no new block) and also (*mat_{index} ≠ 0*). For the first time, *mat₀ = 0* and thus the second condition is not satisfied.
- In (line 17) miner μ increases *index* and then in (line 18) invokes *refresh()* function that performs equation 8.

$$\text{mat}_{\text{index}} = \text{mat}_{\text{index}-1} + (\text{avt}_{\text{net}} + \text{ipt}) \quad (8)$$

- (line 19) While miner μ has time to discover and broadcast a new block (*scounter() ≤ mat_{index}*) in (line 20) checks its input to know if there is a new block received from the network.
- (line 21) If yes, in (line 23) miner μ verifies the answer of PoW for the new block.
- (line 24) If PoW has been done correctly, μ replaces the local chain by the new chain and in (line 26) updates the value of *FlagNewBlock = True* and in (line 27) leaves the while loop and goes to (line 13) and then since *FlagNewBlock = True* goes to (line 17).
- If there is no new block in its input and *scounter()* is below *avt_{net}*, then miner μ tries to solve PoW in (line 33).
- If miner can solve PoW, then it generates a new block in (line 35) and broadcasts it in (line 36).

Algorithm 1 ZeroBlock algorithm

```
1:  $index \leftarrow 0$  ▷ index of  $mat$ 
2:  $mat[index] \leftarrow 0$  ▷  $mat$  at the beginning is set to zero
3:  $avt_{net} \leftarrow$  block generation average time ▷ according to equation (6)
4:  $localChain \leftarrow$  Genesis
5:  $FlagNewBlock \leftarrow$  False
6:  $nonce \leftarrow 0$ 
7:  $HPrB \leftarrow 0$  ▷ hash of previous block
8:  $T \leftarrow$  target
9:  $newChain \leftarrow$  Null
10:  $ansPoW \leftarrow 0$  ▷ answer of PoW
11:  $scounter() \leftarrow 0$  ▷  $scounter()$  is a seconds counter
12: while (True) do
13:   if ( $FlagNewBlock = False$ ) AND ( $mat[index] \neq 0$ ) then
14:      $dummy\ Zeroblock \leftarrow SHF(getHead(localChain)) + SHF("FixedStringZB") + index$ 
15:      $localChain \leftarrow join(dummy\ Zeroblock, localChain)$ 
16:   end if
17:    $index \leftarrow index + 1$ 
18:    $refresh(mat[index])$ 
19:   while ( $scounter() \leq mat[index]$ ) do
20:      $newChain \leftarrow checkInput()$ 
21:     if ( $newChain \neq Null$ ) then
22:        $HPrB \leftarrow SHF(getHead(localChain))$ 
23:       if ( $FHF(HPrB, newChain.ansPoW) \leq T$ ) then ▷ proof-of-work is done
24:          $localChain \leftarrow newChain$ 
25:          $newChain \leftarrow Null$ 
26:          $FlagNewBlock \leftarrow True$ 
27:         break
28:       end if
29:     end if
30:     if ( $scounter() < avt_{net}$ ) then
31:       if ( $FlagNewBlock = False$ ) then
32:          $HPrB \leftarrow SHF(getHead(localChain))$ 
33:         if ( $FHF(HPrB, nonce) \leq T$ ) then ▷ proof-of-work succeeded
34:            $ansPoW \leftarrow nonce$ 
35:            $localChain \leftarrow join(GenerateBlock(), localChain)$ 
36:            $BroadcastBlock(localChain, ansPoW)$ 
37:            $FlagNewBlock \leftarrow True$ 
38:            $nonce \leftarrow 0$ 
39:           break
40:         end if
41:        $nonce \leftarrow nonce + 1$ 
42:     end if
43:   end if
44: end while
45: end while
```

- In case the given *nonce* is not the answer of PoW (the condition of (line 33) is incorrect), miner μ increases the *nonce* (in line 41) and goes back to (line 19).
- If *scounter()* is more than mat_{index} , immediately miner μ generates a dummy Zeroblock in (line 14) and then adds the dummy Zeroblock to its local chain (line 15).
- Then, miner μ refreshes index and value of *mat* in (lines 17 and 18).
- If a dummy Zeroblock is generated, miner μ rejects a *selfish* block in (line 23) because the answer of proof-of-work for selfish block is not the correct *nonce*, since it does not include the hash of the dummy Zeroblock. Then miner μ goes to (line 30) and repeats the algorithm as described above.

V. ZEROBLOCK ALGORITHM RESILIENCE TO BLOCK-WITHHOLDING ATTACK

First we prove that honest miners never accept chains infected with unpermitted block withholding. Then we prove that selfish miners are not incentivized to withhold blocks or not follows the Zeroblock algorithm.

Lemma. *Honest miners, regardless of their percentage in the network, never accept chains infected with unpermitted block-withholding.*

Proof. In a mat_i interval, a *honest* miner either generates or receives a new block b_i , otherwise it generates a dummy Zeroblock ζb_i .

An *unpermitted block-withholding* occurs if a *selfish* miner discovers a new block b_i^s but keeps the block private after end of mat_i . At this point, to prevent *unpermitted block-withholding*, *honest* miners generate a dummy Zeroblock ζb_i including the hash of previous block, since they have not received new block b_i^s which is discovered by a *selfish* miner in mat_i interval.

When mat_i interval is finished (line 19), *honest* miners leave the *while* loop and go to the (line 12). Since two conditions in (line 12) are satisfied, they generate a Zeroblock ζb_i . This is due to the fact that a new block b_i has been received (thus *FlagNewBlock* = *False*) and $mat_i \neq 0$ (because, in (line 12), mat_i has been updated according to equation 8). A *selfish* miner (which here is a *creative miner*) has kept a new block b_i^s after the end of mat_i (see Figure 1 that shows a similar situation in which b_3^s that is discovered in mat_3 by a selfish mining pool, *sm*, has been kept private after the end of mat_3). Thus, the block b_i^s will not be accepted by *honest* miners, because they have generated a dummy Zeroblock ζb_i at the end of mat_i . Therefore, proof-of-work must be restarted from beginning for discovering the next block, i.e. b_{i+1} , such that its proof-of-work includes the recent ζb_i . At this point, b_i^s is not acceptable since its proof-of-work has not been solved based on ζb_i .

In other words,

$$\forall mat_{\Delta} :$$

$$\begin{aligned} & \text{If } (\Delta \neq 0) \text{ then} \\ & \quad \exists (b_{\delta} \vee \zeta b_{\delta}), \delta \geq \Delta \\ & \text{else} \\ & \quad \exists gb \end{aligned}$$

Thus,

$$\forall mat_{\Delta} : \\ b_{\delta} \in X, \text{ If } (\delta < \Delta)$$

where Δ is the index of a *mat*, δ is the index of blocks, X is the set of *selfish* blocks, b is a standard blocks, ζb is a dummy Zeroblock, and gb is the *genesis* block. As shown in Figure 1, where block b_3^s belongs to selfish mining pool in mat_4 , it is rejected by honest miners, since $\delta < \Delta$, where $\delta = 3$ and $\Delta = 4$. □

In the following, we prove that selfish miners are not incentivized to withhold blocks or to not follow correctly the dummy Zeroblocks generation. In the following, we describe all possible events in a *maximum acceptable time for receiving a new block* interval. To simplify, we consider that the network consists of two mining pool types, including: two honest mining pools (hmp_1 and hmp_2) and a selfish mining pool (*sm*). We also assume that at time $t = 0$ all mining pools have a consensus on the first block, *genesis* block.

a) *Event 1 - In a mat_i interval, neither honest nor selfish pools discover a new block:* In this situation, a dummy Zeroblock ζb_i will be generated at the end of mat_i interval by all honest pools. We recall that dummy Zeroblock generation time is negligible. Consequently, the hash of this dummy Zeroblock will be used for discovering the next block. It means that the answer of proof-of-work (*nonce*) for discovering the next block depends on this dummy Zeroblock (since hash of previous blocks is used in proof-of-work and thus affects its answer). As a result, if a selfish mining pool does not generate this dummy Zeroblock, then it will not be able to find the correct answer of proof-of-work for next block (since all honest mining pools at time of verifying answer of proof-of-work will reject any *nonce* that has not been earned from hash of this dummy Zeroblock.) (see Figure 1, Part (a).)

b) *Event 2 - In a mat_i , the first pool which discovers a new block is the honest mining pool:* In this case, it immediately broadcasts the discovered block to the entire network and then starts to discover the next block. Then, other pools receive this new block within *ipt* time interval (i.e. *information propagation time in Bitcoin network* which is simulated and estimated in [3]). Thus, other mining pools after receiving and verifying this new block starts to discover the next block. As a result, the block creator will begin to discover the next block a little sooner than the rest of the network. The maximum of this time is *ipt*. This might be considered as a time reward for the block creator that increases the miners' motivation to be the first one who discovers a new block. This time advantage dedicated to block creator is not unfair since creator mining pool is the first one who discovered the new block and for this it is fair to receive a time reward. However,

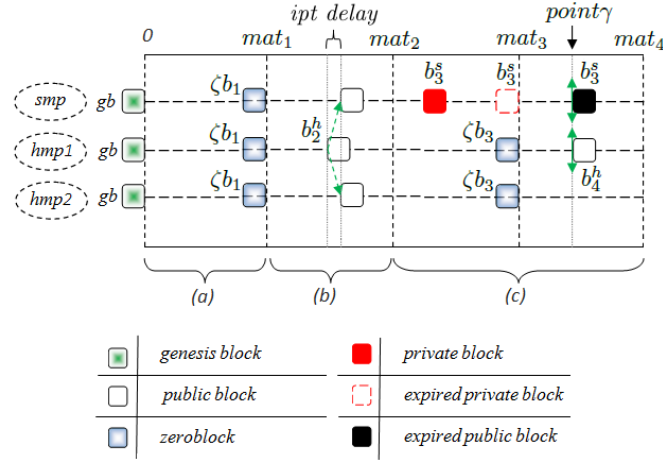


Figure 1. *smp*: selfish mining pool, *hmp1*: first honest mining pool, *hmp2*: second honest mining pool. Part (a) represents event 1: In mat_1 , neither honest nor selfish pools discover a new block. Part (b) represents event 2: In mat_2 , the first pool which discovers a new block is the honest mining pool. Part (c) represents event 3: In mat_3 , the first pool which discovers a new block is selfish mining pool, then at point γ when honest pool discovers block b_4^h selfish pool broadcasts b_3^s to create an intentional fork, but thanks to dummy Zeroblock ζb_3 it will be rejected by honest pools since it does not include the hash of dummy Zeroblock ζb_3 .

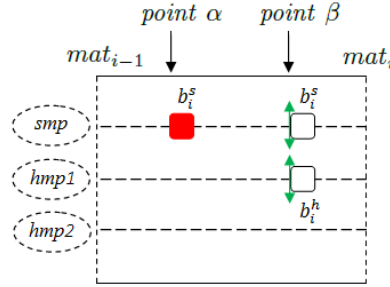


Figure 2. The selfish pool, *smp*, discovers a new block b_i^s at the point α and keeps the block privately till point β when an honest pool, *hmp1*, generates block b_i^h . At this point, selfish pool broadcasts b_i^s to create an intentional fork. The maximum probability of this event is (≈ 0.04).

this time difference according to simulations in [3] is less than one minute (see Figure 1, Part (b)).

c) *Event 3 - In a mat_i , the first pool which discovers a new block is selfish mining pool*: In this case, selfish pool keeps the block private until the end of mat_i interval and does not broadcast its block. We assume that by this time, honest mining pools could not discover a new block. Thus, honest mining pools generate a dummy Zeroblock ζb_i and restart to find the answer of proof-of-work based on hash of ζb_i . Consequently, the next new block is acceptable by honest pools if its proof-of-work has been solved base on ζb_i . As a result, this selfish block will be rejected by honest pools. (see Figure 1, Part (c), point γ .)

d) *Event 4 - In a mat_i , the first pool which discovers a new block is the selfish mining pool (b_i^s at the point α in Figure 2)*: In this case, selfish pool decides to keep the block private until the end of mat_i . At the point β (see Figure 2) an honest mining pool discovers a new block, b_i^h . As soon as honest pool broadcasts the block b_i^h , selfish pool broadcasts block b_i^s to create an **accidental** fork. Thus, a part of network receives b_i^s and works on this block and another part works

on b_i^h . As a result, with a probability one of b_i^s or b_i^h will be winner block and another one is ignored as *orphan* block and eventually the winner block creator (*smp* or *hmp1*) will receive the respective reward. Whereas, if selfish pool at point α had broadcast its block, b_i^s , it had received the reward as the first block creator without any rival with probability of 100%. Consequently, selfish pool with delay in broadcasting its block, b_i^s , caused to reducing probability of earning the reward. An action which is in contrast to the main purpose of block-withholding that is achieving more reward. We called this event as *unprofitable block-withholding*. As a result, the selfish mining pool is not incentivized to reduce its chance for receiving the reward.

In the following we calculate the *maximum* probability of event 4, when the selfish mining pool has maximum possible percentage of hashing power of the network. Regarding to 51% attack, the selfish pool can have at most 49% of total hashing power of the network, because otherwise selfish mining pool can get control of the network.

Recall that the proof-of-work is a Poisson process and also

the difficulty of proof-of-work is adjusted regarding to hashing power of the network such that the expected time to discover the next block is 10 minutes. Thus, the probability that ρ blocks to be discovered in a time interval in which we expect the network discovers λ blocks, is given by equation 8.

$$Prob(\rho|\lambda) = \frac{e^{-\lambda} \cdot \lambda^\rho}{\rho!} \quad (9)$$

For example, the probability of discovering one, two, three, four and five blocks in 10 minutes, in descending order, is as follows:

$$Prob(\rho = 1|1) = e^{-1} \approx 0.3679$$

$$Prob(\rho = 2|1) = \frac{e^{-1}}{2!} \approx 0.1839$$

$$Prob(\rho = 3|1) = \frac{e^{-1}}{3!} \approx 0.0613$$

$$Prob(\rho = 4|1) = \frac{e^{-1}}{4!} \approx 0.0153$$

$$Prob(\rho = 5|1) = \frac{e^{-1}}{5!} \approx 0$$

where: the expected number of blocks to be discovered in 10 minutes is $\lambda = 1$.

The maximum probability that in 10 minutes two blocks are discovered such that the first one is generated by selfish mining pool with the hashing power of $sp = 0.49$ and the second one is generated by honest mining pool with the hashing power of $hp = 0.51$ is as follows:

$$\begin{aligned} \text{Maximum Possible Probability (Event 4)} &= \\ &= (sp \cdot e^{-sp})(hp \cdot e^{-hp}) \times sp \\ &\approx 0.3 \times 0.3 \times 0.49 \approx 0.04 \end{aligned}$$

Note that as we mentioned above if *event 4* occurs, selfish pool reduces its likelihood for receiving the respective reward. Even if the purpose of selfish pool is only the network sabotage, it has to deplete its limited computational resource for achieving an event that in the *best case for the selfish pool* has a low probability (at most ≈ 0.04). This leads to reduce selfish pool's motivation to perform such behavior. Note that this probability is for the case the selfish pool has its *maximum possible* hashing power (i.e. 49% of total hashing power of the network) regarding to 51% attack.

VI. ZEROBLOCK ALGORITHM EXTENSION TO DYNAMIC MINERS

In the following we show how Algorithm 1 can be extended to environments where miners can join at any time. First, we describe a simple mechanism that assumes that a new miner that joins the system has the ability to communicate with all the other miners in the network. When a new miner joins the network it broadcasts a message including its address to announce its entrance. The other miners respond with the last

η	η percentage	ψ	ψ percentage	P
4750	95%	250	5%	$\approx 0.9994\%$
4250	85%	750	15%	$\approx 0.9785\%$
3750	75%	1250	25%	$\approx 0.8862\%$
3250	65%	1750	35%	$\approx 0.7063\%$

Table I

THE PROBABILITY THAT AT LEAST HALF PLUS ONE NODES FROM THE SET OF SELECTED NODS, σ ARE HONEST. IN EACH EXAMPLE, THE NETWORK INCLUDES η *honest* NODS AND ψ *selfish (adversarial)* NODS. WE USE PRACTICAL VALUES FOR THE SIZE OF THE NETWORK, n , AND THE NUMBER OF SELECTED NODES FOR CONNECTION, σ . WE THUS SET n AND σ TO 5000 [41] AND 8 [3], RESPECTIVELY.

version of their local chain. Then, the new miner compares the received chains and selects the chain belonging to the majority. Thus, if half plus one miners of the network are *honest*, the *honest* chain will be chosen by the new miner and it will further mine the *correct chain*. Then, the new miner, similar to other miners, will perform the protocol as described in Algorithm 1. According to this process, each miner is able to leave and re-join the network infinitely.

Note that the above describe process uses as hypothesis that a new miner connects to the entire network. However, in current Bitcoin protocol a new node connects to only a subset of nodes in the network (usually 8 randomly selected nodes). If we assume a situation in which a new node connects to a subset of nodes, then the new node achieves the *correct chain* with probability P as follows:

$$\begin{aligned} P(h \geq [\sigma/2] + 1) &= \sum_{i=1}^{[\sigma/2]} P(h = [\sigma/2] + i) \\ &= \sum_{i=1}^{[\sigma/2]} \frac{\binom{\eta}{[\sigma/2]+i} \binom{\psi}{\sigma - ([\sigma/2]+i)}}{\binom{n}{\sigma}} \end{aligned}$$

where n is the network size, σ is the number of selected nodes to connect, η is the number of *honest* nodes in the entire network, ψ is the number of *selfish* nodes in the entire network and h is the number of *honest* nodes in the set of selected nodes.

Table I shows some numerical examples in different situations, each of which includes η *honest* nodes and ψ *selfish (adversarial)* nodes. We use practical values for the size of the network, n , and the number of selected nodes for connection, σ . We thus set n and σ to 5000 [41] and 8 [3], respectively.

In the following, we say two chains C_1 and C_2 are *homogeneous*, if and only if both of them belong to the *honest* pools or both of them belong to the *adversarial* pools. Otherwise chains are called *inhomogeneous*.

Note that since all honest chains are equal to each other, thus the new node is able to distinguish a set of *homogeneous* chains. We propose if a new node receives a set of *inhomogeneous* nodes, then it must *retry* to connect to σ nodes till the new node receives a set of *homogeneous* nodes. In this case, the new node eventually achieves a set of chains that belong to the adversary if and only if all σ chains are not correct chain. Thus, the adversarial cartels size needs to be

increased significantly such that with a good probability all σ chains would not be correct chains. Assume a situation in which only one chain is correct chain and the rest are not. Then, the new node will retry to connect again to σ nodes, whilst if the new node consider the majority chains, then it will accept the adversarial chain.

Thus, we calculate the probability that all σ chains are homogeneous and *correct* and then the probability that all σ chains are homogeneous and are *not correct chain*, respectively, as follows:

$$P_{(hcorr)} = P(h = \sigma) = \frac{\binom{\eta}{\sigma}}{\binom{n}{\sigma}}$$

$$P_{(hnotc)} = P(h = 0) = \frac{\binom{\psi}{\sigma}}{\binom{n}{\sigma}}$$

and then, we calculate the probability that all σ chains are homogeneous regardless of chains type as follows:

$$P_{(hom)} = P(h = \sigma \text{ or } h = 0) = \frac{\binom{\eta}{\sigma} + \binom{\psi}{\sigma}}{\binom{n}{\sigma}}$$

We define R as the number of trials for connection to σ nodes to achieve a set of homogeneous chains. Thus, the probability that after m trials ² the new node achieves a set of *homogeneous* and *correct chains* is as follows:

$$P(R = m \mid \forall c_i \in SC, c_i \text{ is correct chain}) = \left(1 - P_{(hom)}\right)^m \times P_{(hcorr)}$$

where, SC is the set of received chains, c_i is a chain $\in SC$ and $i \in \mathbb{N}$ such that $1 \leq i \leq \sigma$.

VII. CONCLUSION AND DISCUSSIONS

In this paper, we introduced a new timestamp-free solution to prevent block-withholding or *selfish* mining. The key idea of our solution, Zeroblock algorithm, is that each block must be generated and received by the network within a *maximum acceptable time for receiving a new block* interval that is locally estimated by each miner. Within this interval, *mat*, a *honest* miner has to receive or to discover a new block. Otherwise, it generates a dummy block that does not need any proof-of-work computation. The computation of each *mat* interval is done locally by each miner based on the following Bitcoin parameters: the *expected delay for a block mining* (estimated based on the Poisson nature of the proof-of-work) and the *information propagation time* in the Bitcoin network (that has been previously estimated in [3]). We prove that our Zeroblock algorithm is resilient to withholding attack. That is, we demonstrated that if a *selfish* miner wants to keep its blocks private more than the duration of a *mat* interval, then the selfish block will be rejected by the *honest* miners. Moreover we prove that selfish miners are not incentivized to ignore dummy Zeroblock or to generate too many of them.

²When $m = 1$ it means that the new tried to connect to σ nodes previously but it has not achieved a set of *homogeneous* chains.

Furthermore, we demonstrated that our solution is compliant to nodes churn. That is, nodes that freshly enter the system are able to retrieve the *correct chain* provided that a majority of nodes are *honest*.

Note that Zeroblock scheme is not a solution to *accidental* forks since these forks are not the result of a block-withholding attack. Contrary to *intentional* forks, *accidental* ones are the result of the Poisson nature of proof-of-work in Bitcoin. Therefore, with non zero probability two blocks may be discovered by two different pools at almost same times. In Bitcoin network the probability of an *accidental* fork is $\approx 1.69\%$ [3]. In order to solve this problem an alternative to proof-of-work should be find.

A criticism to our Zeroblock algorithm can be the fact that the blockchain may include too many sequences of dummy Zeroblocks. This can be easily solved as follows. Each honest miner after receiving and accepting a new standard block b_i removes all Zeroblocks between b_i and previous standard block (see Figure 3).

Zeroblock solution is a step further in solving one of the major problems in Bitcoin and can be used also as an *altcoin*, (a term refers to a cryptocurrency based on the Blockchain technology [30]) or in conjunction with other cryptocurrencies.

REFERENCES

- [1] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Consulted 1.2012 (2008): 28
- [2] Back, Adam. "Hashcash-a denial of service counter-measure." (2002)
- [3] Decker, Christian, and Roger Wattenhofer. "Information propagation in the bitcoin network." Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on. IEEE, 2013
- [4] Eyal, Ittay, and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable." Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2014. 436-454
- [5] Garay, Juan, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol: Analysis and applications." Advances in Cryptology-EUROCRYPT 2015. Springer Berlin Heidelberg, 2015. 281-310
- [6] Courtois, Nicolas T., and Lear Bahack. "On subversive miner strategies and block withholding attack in bitcoin digital currency." arXiv preprint arXiv:1402.1718 (2014)
- [7] Miers, Ian, et al. "Zerocoin: Anonymous distributed e-cash from bitcoin." Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013
- [8] Bonneau, Joseph, et al. "Mixcoin: Anonymity for Bitcoin with accountable mixes." Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2014. 486-504
- [9] Eyal, Ittay. "The miner's dilemma." Security and Privacy (SP), 2015 IEEE Symposium on. IEEE, 2015
- [10] Bastiaan, Martijn. "Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin." Available at <http://referaat.cs.utwente.nl/conference/22/paper/7473/preventingthe-51-attack-a-stochastic-analysis-of-two-phase-proof-of-work-in-bitcoin.pdf>. 2015
- [11] Gervais, Arthur, et al. "Is Bitcoin a decentralized currency?." IACR Cryptology ePrint Archive 2013 (2013): 829
- [12] Reid, Fergal, and Martin Harrigan. An analysis of anonymity in the bitcoin system. Springer New York, 2013
- [13] Dwork, Cynthia, and Moni Naor. "Pricing via processing or combatting junk mail." Advances in Cryptology—CRYPTO'92. Springer Berlin Heidelberg, 1993
- [14] Androulaki, Elli, et al. "Evaluating user privacy in bitcoin." Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2013. 34-51
- [15] Miers, Ian, et al. "Zerocoin: Anonymous distributed e-cash from bitcoin." Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013

*Zeroblocks between two standard blocks will
be removed after accepting b_i by honest miner.*

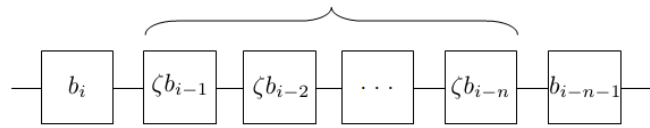


Figure 3. This Figure demonstrates a situation when between standard blocks b_i and b_{i-n-1} there are some Zeroblocks from ζb_{i-1} to ζb_{i-n} that will be removed after accepting standard block b_i by honest node. Note that even after removing Zeroblocks from the Blockchain, the standard block b_i still includes the hash of Zeroblocks ($\zeta b_{i-1}, \dots, \zeta b_{i-n}$) and thus in the future, all of honest nodes are able to understand that between b_i and b_{i-n-1} there have been Zeroblocks.

- [16] Kroll, Joshua A., Ian C. Davey, and Edward W. Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries." Proceedings of WEIS. Vol. 2013. 2013
- [17] Fugger, Ryan. Money as IOUs in social trust networks and a proposal for a decentralized currency network protocol. Hypertext document. Available electronically at [www.ripple.sourceforge.net\(2004\)](http://www.ripple.sourceforge.net(2004))
- [18] Yang, Beverly, and Hector Garcia-Molina. "PPay: micropayments for peer-to-peer systems." Proceedings of the 10th ACM conference on Computer and communications security. ACM, 2003
- [19] Saito, Kenji. i-WAT: the internet WAT system—an architecture for maintaining trust and facilitating peer-to-peer barter relationships. Diss. PhD thesis, Graduate School of Media and Governance, Keio University, 2006
- [20] Vishnumurthy, Vivek, Sangeeth Chandrakumar, and Emin Gun Sirer. "Karma: A secure economic framework for peer-to-peer resource sharing." Workshop on Economics of Peer-to-Peer Systems. Vol. 35. 2003
- [21] Douceur, John R. "The sybil attack." Peer-to-peer Systems. Springer Berlin Heidelberg, 2002. 251-260
- [22] Ben Sasson, Eli, et al. "Zerocash: Decentralized anonymous payments from Bitcoin." Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014
- [23] Heilman, Ethan. "One weird trick to stop selfish miners: Fresh bitcoins, a solution for the *honest* miner." (2014)
- [24] Decker, Christian and Seider, Jochen and Wattenhofer, Roger. "Bitcoin meets strong consistency." Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, 2016.
- [25] Kroll, Joshua A., Ian C. Davey, and Edward W. Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries." Proceedings of WEIS. Vol. 2013. 2013
- [26] Bahack, Lear. "Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft)." arXiv preprint arXiv:1312.7013 (2013)
- [27] Luu, Loi, et al. "On power splitting games in distributed computation: The case of bitcoin pooled mining." Computer Security Foundations Symposium (CSF), 2015 IEEE 28th. IEEE, 2015
- [28] Sapirshstein, Ayelet, Yonatan Sompolinsky, and Aviv Zohar. "Optimal selfish mining strategies in Bitcoin." arXiv preprint arXiv:1507.06183 (2015)
- [29] Babaioff, Moshe, et al. "On bitcoin and red balloons." Proceedings of the 13th ACM conference on electronic commerce. ACM, 2012
- [30] Bonneau, Joseph, et al. "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies." 2015 IEEE Symposium on Security and Privacy. IEEE, 2015
- [31] Lewenberg, Yoad, et al. "Bitcoin mining pools: A cooperative game theoretic analysis." Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2015
- [32] Eyal, Ittay, et al. "Bitcoin-NG: A scalable Blockchain protocol." 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). 2016.
- [33] ODwyer, Karl J., and David Malone. "Bitcoin mining and its energy footprint." Irish Signals and Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CICT 2014). 25th IET. IET, 2013.
- [34] Taylor, Michael Bedford. "Bitcoin and the age of bespoke silicon." Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems. IEEE Press, 2013.
- [35] Kraft, Daniel. "Difficulty control for Blockchain-based consensus systems." Peer-to-Peer Networking and Applications 9.2 (2016): 397-413.
- [36] Möser, Malte, and Rainer Böhme. "Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees." International Conference on Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2015.
- [37] Hayes, Adam. "What factors give cryptocurrencies their value: An empirical analysis." Available at SSRN 2579445 (2015).
- [38] Alqassem, Israa, and Davor Svetinovic. "Towards reference architecture for cryptocurrencies: Bitcoin architectural analysis." Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCoM), IEEE. IEEE, 2014.
- [39] Wang, Luqin, and Yong Liu. "Exploring Miner Evolution in Bitcoin Network." International Conference on Passive and Active Network Measurement. Springer International Publishing, 2015.
- [40] <https://bitcoinwisdom.com/assets/difficulty/bitcoin-difficulty.png?1476276622>
- [41] <https://bitnodes.21.co/>
- [42] Solat, Siamak. "Security of Electronic Payment Systems: A Comprehensive Survey." arXiv preprint arXiv:1701.04556 (2017)

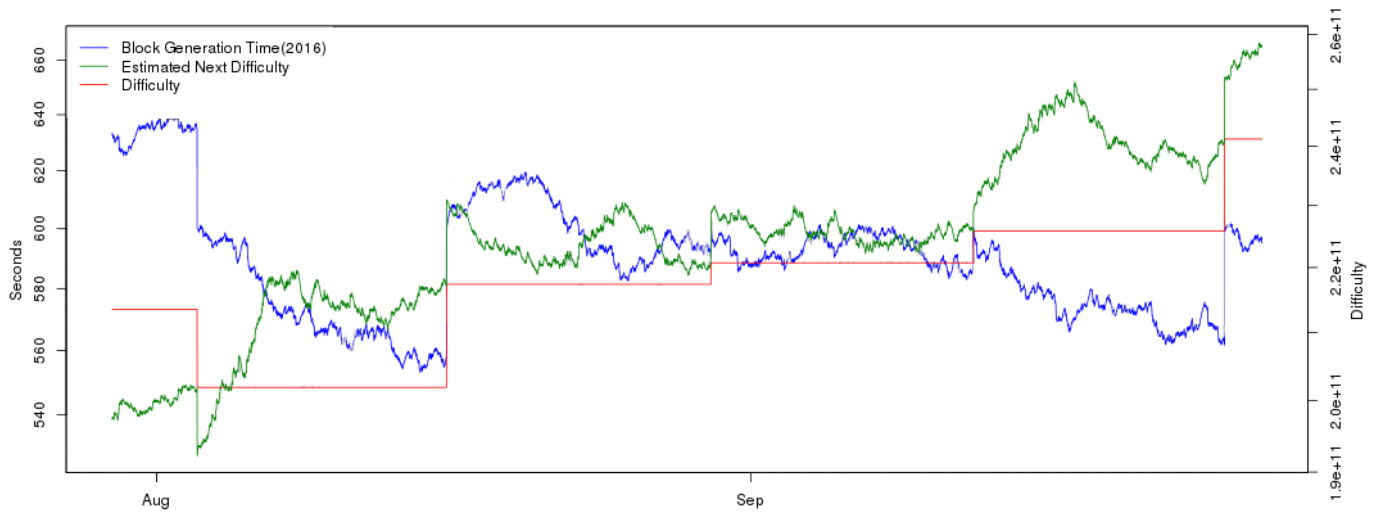


Figure 4. This figure represents the relation between block generation time and difficulty of proof-of work such that using equation 6 when 2016 blocks are generated in less than 600 seconds (10 minutes), the difficulty is increased and if they are discovered in more than 10 minutes, then difficulty is decreased.[40]

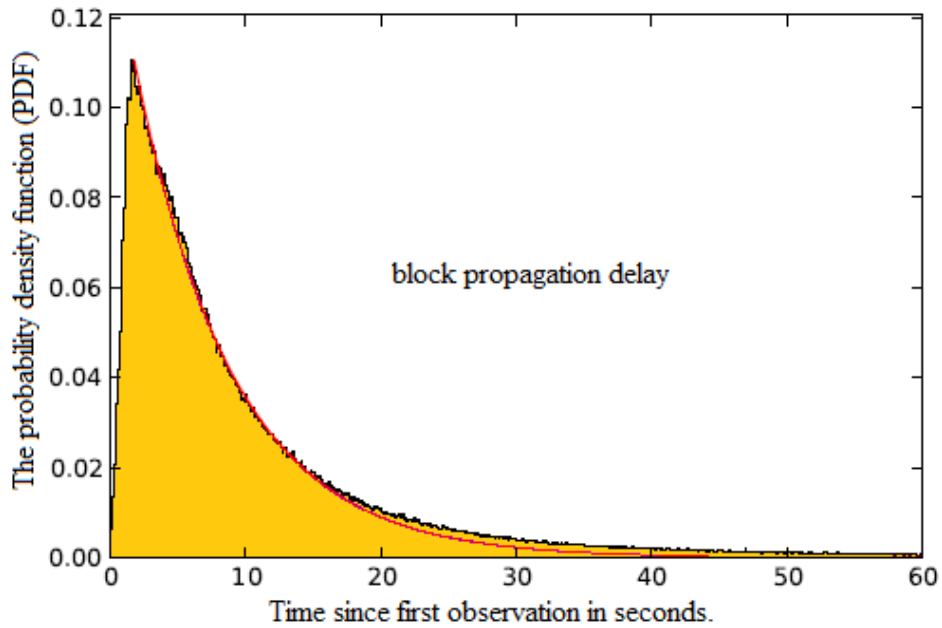


Figure 5. Decker and Wattenhofer's simulation [3] to estimate block propagation delay in the entire Bitcoin network shows that before 60 seconds the whole of network receives a published discovered block. We call this delay *ipt* (information propagation time)