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

Siamak Solat, Maria Potop-Butucaru

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

Siamak Solat and 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 the transactions in the system maintained by all miners as a distributed shared register. 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 resource consuming proof-of-work (cryptographic puzzle). The reward is a motivation for mining process but also could be an incentive for attacks such as *selfish mining*. In this paper we propose a solution for one of the major problems in Bitcoin : *selfish mining* or *block-withholding* attack. This attack is conducted by adversarial or *selfish* miners in order to either earn undue rewards or waste the computational power of *honest* miners. Contrary to recent solutions, our solution, *ZeroBlock*, prevents *block-withholding* using a technique free of timestamp that can be forged. Moreover, we show that our solution is compliant with nodes churn.

Keywords: *Bitcoin*; *ZeroBlock*; *block-withholding*; *selfish-mining*; *intentional fork*; *cryptocurrency*

1 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 towards creating a system where transactions between individuals can escape the strict control of the banks and more generally of the financial markets.

Bitcoin was introduced as a pure peer-to-peer [1] electronic currency or crypto-currency to aim at fully decentralizing electronic payment transactions. Bitcoin allows to perform online payment 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 ownership [1] and

[1] Capitalized "Bitcoin" refers to the protocol, while lower case "bitcoin" refers to the coin.

employs Blockchain in order to prevent double-spending attacks. Blockchain is broadcast via a peer-to-peer network to achieve a consensus about the history of the transactions in the system via a proof-of-work (cryptographic puzzle) [13,2,12] performed by *honest* parties (miners that follow the protocol).

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 introduce a new solution, ZeroBlock, which prevents *block withholding* or *selfish* mining without using timestamp contrary to the recent solution proposed by [23] that uses a forgeable timestamp based idea. In our solution, if a *selfish* miner keeps a block private more than a *mat* interval, its block will be rejected by *honest* miners.

Paper Roadmap The rest of this paper is organized as follows: Section 2 presents an overview of Bitcoin, Section 2.5 presents how to generate a block and proof-of-work process, in Section 2.6 we present *block withholding* attack or *selfish* mining and then we describe the related efforts on this problem. In Section 4.1 we describe the important notations that we use in this paper and in Section 4 we present our ZeroBlock solution altogether with its correctness proof. Finally, Section 6 concludes the paper.

2 Bitcoin Overview

Bitcoin is an electronic coin which works as a chain of digital signatures where each owner transfers bitcoin 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 next owner (i.e. the receiver). As a result, the final receiver is able to verify the bitcoin ownership by verifying the signatures [1].

The key entity in Bitcoin is a public log that is a sequence of blocks, named Blockchain, that maintains the history of transactions. The safety of Blockchain is provided 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 network two kinds of information are disseminated. The first one are *2.1 Transactions* : used to transfer coins in the network. The second one are *2.2 Blocks*. Blocks are inserted into a "*public ledger*" called *2.3 Blockchain*. Blockchains are employed to synchronize the state among all miners [3]. In the sequel we detail the key notions in Bitcoin.

2.1 Transactions

A *transaction* transfers the coins from one or multiple source account address to destination account address, where each account address has a unique key. Each address derives from the related public key and is employed to recognize the related account. For transferring a bitcoin between two accounts, a *transaction* must be generated including the destination account address and it must be signed by the private key of the source account.

To calculate the balance of account addresses, 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 broadcast into the network, the "public ledger" will be updated locally in each node. This copy eventually might be different in distinct miners which may lead to inconsistencies. For example, consider a miner receives a *transaction* that shows transferring some bitcoins from *account "A"*, whereas this miner has not received the related *transaction* that shows these coins are "available" for *account "A"*. In another situation, several transactions might try to transfer the same coins more than one time. Such subversive behavior is known as "*double spending*" [3].

2.2 Blocks

Achieving a "consensus" in a distributed system is not trivial. Bitcoin uses an uncertain commitment of *transactions* and then tries to synchronize them at some time intervals. It is done using *Blocks* that have been generated and broadcast in the entire network [3]. Each *block* consists of several *transactions*. Each miner when receives a new *block*, updates its local chain and then restores the uncertain committed *transactions* since the last block. In this last block, all miners have a consensus on the *transactions*.

2.3 Blockchain

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* [3].

2.4 Fork

If two miners create two blocks with the same preceding block, the chain is *forked* into two branches. It might occur **accidentally** or **intentionally**. Since the proof-of-work is a Poisson process, two blocks might be discovered by two mining pools in a few seconds that causes an accidental fork. The probability of an accidental fork is ≈ 1.69 [3] (almost every 60 blocks). Note that an accidental fork occurs due to the nature and functionality of proof-of-work and it is not related to block-withholding or selfish mining that is the target of our solution. In case of **intentional** fork, a *selfish* miner previously has generated and kept

a block privately and when another miner generates a new block over the same preceding block and broadcasts to the network, then the *selfish* miner immediately propagates the private block to create an "intentional fork". In case the *selfish* miner can generate and keep more than one private block, it always will be winner in the race of branches, because the longest chain will be accepted by the network.

In other words, there is a *Blockchain fork*, when in a *Blockchain* there are more than one *head*. In this situation, the miners do not have a *consensus* on a unique *head*.

2.5 Block Generation and Proof-of-Work

A proof-of-work (PoW) is a cryptographic puzzle that is difficult to solve but easy to verify. 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. 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 broadcast 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 occurs when their preceding block is equal and also they are broadcast simultaneously. At this point Blockchain is forked. 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] (see Section 2.4). 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. Moreover, this leads to a fork on the Blockchain which divides the Blockchain into two branches. When one of miners' groups discovers a new block, their branch will become a part of the Blockchain and the other one will be ignored as *orphan* block.

2.6 Block-Withholding and Related Works

Block withholding attack was introduced as *"Selfish mining"* 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 some miners called *selfish* miners or *selfish* mining pool. According to [4], these *selfish* miners can earn revenues superior to a fair situation [27].

In block-withholding strategy, a *selfish* miner after solving proof-of-work and finding a new block does not broadcast it in the entire network till a specific time such that this leads to *intentional* forking the Blockchain. (see Section 2.4)

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 [29] miners are not motivated enough to propagate transactions. Authors in [31] introduce a cooperative game theory analysis relying on interactive mining pools.

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].

$$\frac{1-\gamma}{3-2\gamma} < \alpha < \frac{1}{2} \tag{1}$$

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 a random beacons and timestamp, *honest* miners select more fresh blocks and the threshold becomes $\alpha \geq 0.32$.

3 Motivation

The key purpose in our idea is reducing the probability of **intentional** forks as much as possible that is the result of block-withholding. 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 Section 5, we describe all possible events and show that using ZeroBlock idea a selfish mining pool cannot achieve more than its expected reward. Only with a low probability, selfish mining pool might 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 for earning expected reward regarding to its mining power in the network. Thus, selfish mining pool is not incentive to create such fork if its purpose is achieving more reward. Note that we also show that the **maximum** probability of such **intentional** fork is very low (maximum ≈ 0.04) when selfish pool has its maximum hashing power in the network. Thus, ZeroBlock idea can disappoint selfish mining pool significantly when it indents to perform block-withholding for achieving more reward.

Note that **accidental** fork is an inherit result of proof-of-work because of its nature functionality that is a Poisson process and thus it is not the result of block-withholding. Thus, for preventing accidental fork, proof-of-work might be replaced by another alternative method with different functionality. Since proof-of-work is a Poisson process, thus with a probability two blocks might be discovered by two different pools at similar times. In Bitcoin network the probability of an **accidental** fork is $\approx 1.69\%$ [3]. Thus, we distinguish *intentional* fork and *accidental* fork due to their different factors causing, "block-withholding" and the "proof-of-work functionality", respectively. We also specify that the purpose of ZeroBlock idea is preventing block-withholding which leads to **intentional** fork.

4 ZeroBlock

In this section we introduce a new solution, ZeroBlock[2], to prevent *block-withholding* or *selfish* mining in Bitcoin. The key of our solution is that each block must be generated and received by the network within a *mat* interval (i.e. maximum acceptable time for receiving a new block). Within a *mati* interval an *honest* miner either receives or discovers a new block b_i. Otherwise, it generates a specific block ζb_i called Zeroblock. In the sequel we propose a detailed description of the algorithm and the proof of its correctness.

The proof-of-work is a Poisson process [3] and causes blocks to be discovered randomly and independently. In our idea, we consider a key concept in Bitcoin network: "In Bitcoin protocol, difficulty of proof-of-work required to discover a block is adjusted regarding to hashing power of the entire network such that on average, ***one*** block is expected to be discovered every ***10 minutes*** [32,33,34,35,36,37,38,39]." According to this fact and also the fact that proof-of-work is a Poisson process [3], if a mining pool generates block b_i at the first minute of a mat_i and keeps the block private until the last minute of mat_i and then broadcasts the block, we call such behavior as "permitted" block-withholding. Since the mining pool broadcasts the block at the last minute of mat_i, it does not lead to additional delay for block generation rate. The expected delay for block discovering rate is set to: "one block per 10 minutes, on average [32,33,34,35,36,37,38,39]." We use this delay setting for computing a mat_i. In the rest of the paper, we show that even this type of block-withholding is not

[2] "ZeroBlock" refers to the algorithm, while "Zeroblock" refers to a specific block.

in favor of a selfish mining pool and it decreases the probability of receiving the logical and expected reward of mining pool regarding to its hashing power in the network such that this behavior is harmful for selfish mining pool if its purpose is receiving more reward. However, we show that the maximum probability of such block-withholding is at most ≈ 0.04. Thus, our idea does not lead to any additional delay, since our solution is based on the Bitcoin protocol configuration for block generation rate (i.e. that "one block per 10 minutes, on average.")

The major part of proof-of-work is discovering a byte string, *nonce*, as the answer of proof-of-work which is merged with block header. It results in a hash value including a given number of leading zero, called as *target*. This *target* is in proportion to the difficulty of proof-of-work such that a smaller value for *target* increases the difficulty.

In Bitcoin protocol, according to hashing of the network, difficulty of proof-of-work is adjusted such that on average, one block per 10 minutes is expected to be discovered in whole of the network. In Bitcoin protocol, 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, such that if 2016 blocks are discovered in a shorter time, 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. This results in fixing block generation rate i.e. one block per 10 minutes.

Note: In this paper, when we say "block" b_i, it means a "standard block" that is discovered by solving the proof-of-work, whereas for generation of a Zeroblock ζb_i miners do not need to solve proof-of-work and so its generation does not take a significant time period. Thus, in our algorithm, Zeroblocks are not counted at time of adjusting difficulty *target*.

4.1 Key notations

The ZeroBlock algorithm (Algorithm 1) uses the following notations:

- *ipt* : information propagation time in Bitcoin network that is an average delay for propagation a block into the network. This average delay has been simulated before in [3].

- *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 5.

- *mat* : maximum acceptable time for receiving a new block that is computed by equation 6. During a mat_i if a miner cannot solve the proof-of-work, it has to generate a Zeroblock ζb_i.

- *intentional fork* : it occurs due to block-withholding when a selfish pool keeps its discovered blocks private instead of publishing block in the network.

- *accidental fork* : since the proof-of-work is a Poisson process, two blocks might be discovered by two mining pools in a few seconds that causes an accidental fork. The probability of an accidental fork is ≈ 1.69 [3] (almost every 60 blocks). Note that an accidental fork occurs due to the nature and functionality of proof-of-work and it is not related to block-withholding or selfish mining that is the target of our solution.

- *unpermitted block-withholding* : it occurs when a *selfish* mining pool discovers a new block b_i^s and keeps the block private after end of mat_i.

- *unprofitable block-withholding* : it occurs when a *selfish* mining pool discovers a new block b_i^s and keeps the block private until discovering another new block b_i^h by an honest miner before end of mat_i. Then, selfish miner broadcasts b_i^s to create an intentional fork. We demonstrate that in this event, selfish miner reduces its likelihood to receive the respective reward according to its hashing power in the network. We also show that such event, in best case for selfish miner, has a low probability (maximum ≈ 0.04).

- *Zeroblock* : is a dummy block including the index of mat_i and the hash of previous block. It is generated by honest miners to prevent *unpermitted block-withholding*. (see Algorithm 1)

- *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_i can solve proof-of-work and then discover a new block b_i.

4.2 Relation: Time, Difficulty, Hash rate

The proof-of-work in Bitcoin network has a difficulty level that is a measure of how difficult it is to discover a block. The proof-of-work is a cryptographic puzzle to find a hash value below a given *target*. In fact, in Bitcoin protocol the difficulty of proof-of-work is adjusted by choosing this *target* that is a 256-bit number. More precisely, 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 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*. For achieving more stability,

the proof-of-work is adjusted such that the network produces, on average, one block every 10 minutes. This is the adjustment which we use in ZeroBlock idea.

The proof-of-work works as follows:

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

$$proof\text{-}of\text{-}work\ succeeded$$

where pb is representing hash of 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.

$$D = \frac{maxTarget}{T} \tag{3}$$

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):

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

Thus, the number of hashes to discover a block is $D \times 2^{32}$, in expectation. So, a mining pool which 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):

$$avt_{pool} = \frac{D \times 2^{32}}{php} \tag{5}$$

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

$$avt_{net} = \frac{D \times 2^{32}}{nethp} \tag{6}$$

According to this relation between triple parameters (*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 [32,33,34,35,36,37,38,39] (see Figure 1 [40]). **Thus, in this case we set avt_{net} to 10 minutes or 600 seconds**.

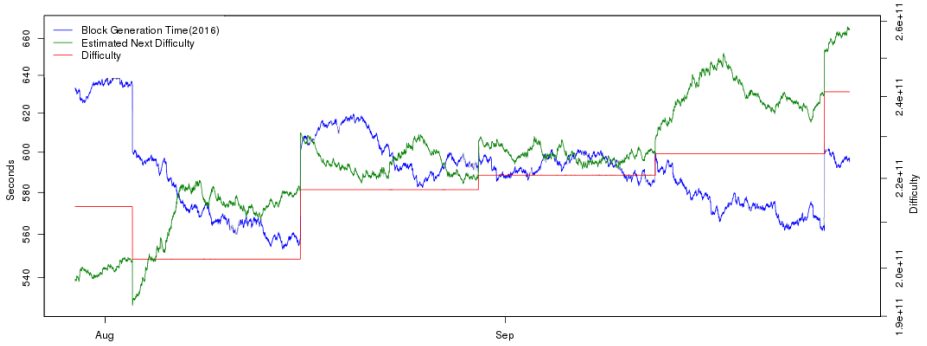

Fig. 1. This figure represents the relation between block generation time and difficulty of proof-of work such that using equation 6 when 2016 blocks are generated in less than 600 seconds (10 minutes), the difficulty is increased and if they are discovered in more than 10 minutes, then difficulty is decreased.

Acceptable Time for Receiving a New Block. Bitcoin network employs a multi-hop broadcast to propagate a discovered block in the entire network. This leads to a few delay till the whole of network receives a new block. This delay time recently has been simulated and estimated by Decker and Wattenhofer [3]. Figure 2 shows their result for block propagation delay in the entire Bitcoin network. To see this simulation in more details we invite reader to discover [3]. 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*).

To calculate a *mat* (i.e. *the maximum acceptable time for receiving a new block*) we add this delay to avt_{net} as follows: (equation 7)

$$mat = avt_{net} + ipt \tag{7}$$

4.3 ZeroBlock Algorithm

In this section, we describe the ZeroBlock algorithm line by line according to the Algorithm 1. More in details, each miner, μ, that executes Algorithm 1 performs the following steps:

- (*lines* 1 to 11) initialization: where mat_0 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*.

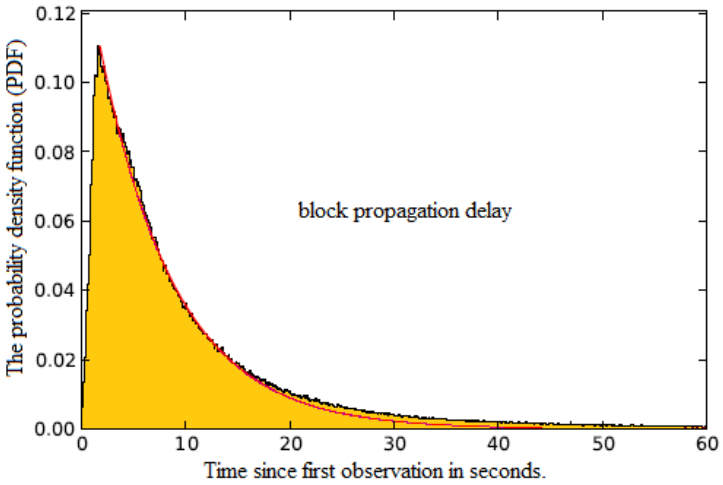

Fig. 2. Decker and Wattenhofer's simulation [3] to estimate block propagation delay in the entire Bitcoin network shows that before 60 seconds the whole of network receives a published discovered block. We call this delay *ipt* (*information propagation time*)

- In (*line* 12) miner μ starts an infinite loop.
- In (*line* 13) miner μ checks ($FlagNewBlock = False$) (that means verifying if there is not a new block) and also ($mat_{index} \neq 0$). For the first time, $mat_0 = 0$ 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.

$$mat_{index} = mat_{index-1} + (avt_{net} + ipt) \tag{8}$$

- (*line* 19) While miner μ has time to discover and broadcast a new block ($scounter() \leq mat_{index}$) 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).

- In case the given *nonce* is not 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 Zeroblock in (*line* 14) and then adds Zeroblock to its local chain (*line* 15).

- Then, miner μ refreshes index and value of *mat* in (*lines* 17 and 18).

- If a 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 Zeroblock. then miner μ goes to (*line* 30) and repeat the algorithm as described above.

4.4 Correctness of Algorithm 1

Theorem 1. *Honest miners, regardless of their percentage in the network, never accept chains infected with unpermitted block-withholding.*

Proof. In a mat_i interval, an *honest* miner either generates or receives a new block b_i, otherwise it generates a 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 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 end of mat_i (see Figure 3 that shows a similar situation in which b_3^s that is discovered in mat_3 by a selfish mining pool, *smp*, has been kept private after end of mat_3). Thus, the block b_i^s will not be accepted by *honest* miners, because they have generated a 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 word,

Algorithm 1 ZeroBlock algorithm

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

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

Thus,

$$\begin{aligned}&\forall\ mat_{\Delta}:\\ &\quad b_{\delta} \in X\ ,\ \textit{If}\ (\delta < \Delta)\end{aligned}$$

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

Theorem 2. *In a dynamic network, if at least* $\left[\frac{n+2}{2}\right]$ *miners are honest, then when an honest miner joins the network, it can distinguish the correct chain which belongs to honest miners, when n is size of the network.*

Proof. When a new miner joins the network and broadcasts a message including its address to announce its entrance, other miners respond with the last version of their local chain. Then, the new miner compares the received chains and selects the chain belonging to the majority of network. Thus, if half plus one miners of the network are *honest*, the *honest* chain will be chosen by new miner and it works on 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 1. Our hypothesis for theorem 2 is that a new miner connects to the entire network. But in current Bitcoin protocol, the new node connects to only a subset of nodes in the network (normally 8 randomly selected nodes). However, 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:

$$\begin{aligned}&\mathrm{P}(\textit{at least half plus one nodes from set of selected nods are honest})\\ &= \mathrm{P}(h \geq \lceil \sigma/2 \rceil + 1) = \sum_{i=1}^{\lceil \sigma/2 \rceil} \mathrm{P}(h = \lceil \sigma/2 \rceil + i) = \sum_{i=1}^{\lceil \sigma/2 \rceil} \frac{\binom{\eta}{\lceil \sigma/2 \rceil + i}\binom{\psi}{\sigma - (\lceil \sigma/2 \rceil + i)}}{\binom{n}{\sigma}}\end{aligned}$$

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

Table 1 shows some numerical examples in different situations, each of which includes η *honest* nods and ψ *selfish* (*adversarial*) nodes. We try to use a practical value for size of the network, n, and number of selected nodes for connection, σ. We thus set n and σ to 5000 [41] and 8 [3], respectively.

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 1. The probability that at least half plus one nodes from set of selected nods, σ are honest. In each example, the network includes η *honest* nods and ψ *selfish* (*adversarial*) nodes. We try to use a practical value for size of the network, n, and number of selected nodes for connection, σ. We thus set n and σ to 5000 [41] and 8 [3], respectively.

definition 1. We say chains C_1 and C_2 are *homogeneous*, if and only if both of them belongs to the *honest* pools **or** both of them belongs to the *adversarial* pools.

definition 2. We say chains C_1 and C_2 are *inhomogeneous*, if and only if one of them belongs to the *honest* pools **and** the other one belongs to the *adversarial* pools.

Note 2. 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 belongs to 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 chain*. 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 chain* and then the probability that **all** σ chains are homogeneous and are not *correct chain*, respectively, as follows:

$$\mathrm{P}_{(hcorr)} = \mathrm{P}(h = \sigma) = \frac{\binom{\eta}{\sigma}}{\binom{n}{\sigma}} \qquad \mathrm{P}(h = 0) = \mathrm{P}_{(hnotc)} = \frac{\binom{\psi}{\sigma}}{\binom{n}{\sigma}}$$

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

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

Then, we define R as the number of retries for connection to σ nodes to achieve a set of homogeneous chains. Thus, the probability that after m retries[3] the new node achieves a set of *homogeneous* and *correct chains* is as follows:

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

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

5 Eventualities

In this section, we describe all possible events in a mat_i interval (*maximum acceptable time for receiving a new block*). 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 (smp). We also assume that at time t = 0 all mining pools have a consensus on the first block, *genesis* block.

event 1. In a mat_i, neither honest nor selfish pools discover a new block. Thus, in such a situation, a Zeroblock ζb_i will be generated at the end of mat_i interval by all honest pools. We remind that Zeroblock generation has no resource consumption for a miner (that means proof-of-work is not needed to generate a Zeroblock ζb_i and thus its generation time in negligible.) Consequently, hash of this Zeroblock will be used for discovering the next block. It means that answer of proof-of-work (*nonce*) for discovering the next block is depended on this Zeroblock (since hash of previous block is used in proof-of-work and thus affects its answer.) As a result, if a selfish mining pool does not generate this 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 Zeroblock.) (see Figure 3, *Part (a).*)

event 2. In a mat_i, the first pool which discovers a new block is honest mining pool. Thus, it immediately broadcasts discovered block to the entire network and then starts to discovering 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 discovering the next block. As a result, the block creator will begin to discovering the next block a little sooner than the rest of network. The maximum of this time is *ipt*. This is because of a few delay for information propagation in Bitcoin network. 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. Thus, this few advantage for time

[3] When $m = 1$ it means that the new node one time has tried to connect to σ nodes previously but it has not achieved a set of *homogeneous* chains.

dedicated to block creator is not unfair since creator mining pool is the first one who discovered new block and for this it receives a few time reward. However, this time difference according to simulations in [3] is less than one minute. (see Figure 3, *Part (b)*.)

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

event 4. In a mat_i, the first pool which discovers a new block is selfish mining pool (b_i^s at the point α in Figure 4). Then, selfish pool decides to keep the block private until end of mat_i. At the point β (see Figure 4) an honest mining pool discovers a new block, b_i^s. 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 a portion of the network 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 hmp_1) 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 call this event as "*unprofitable block-withholding*". As a result, the selfish mining pool is not incentive to reduce its chance for receiving the reward. However, 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 at most can have 49% of total hashing power of the network, because otherwise selfish mining pool can get control of the network.

Maximum Probability of event 4. 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 [32,33,34,35,36,37,38,39]. Thus, the probability that ρ blocks to be discovered in a time interval in which we expect the network discovers λ blocks, is equal to equation 8.

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

For example, if we want to calculate the probability of discovering one, two, three and four blocks in 10 minutes, in descending order, is as follows:

$$Prob(\rho = 1|1) = e^{-1} \approx 0.36$$

$$Prob(\rho = 2|1) = \frac{e^{-1}}{2!} \approx 0.18$$

$$Prob(\rho = 3|1) = \frac{e^{-1}}{3!} \approx 0.06$$

$$Prob(\rho = 4|1) = \frac{e^{-1}}{4!} \approx 0.01$$

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

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

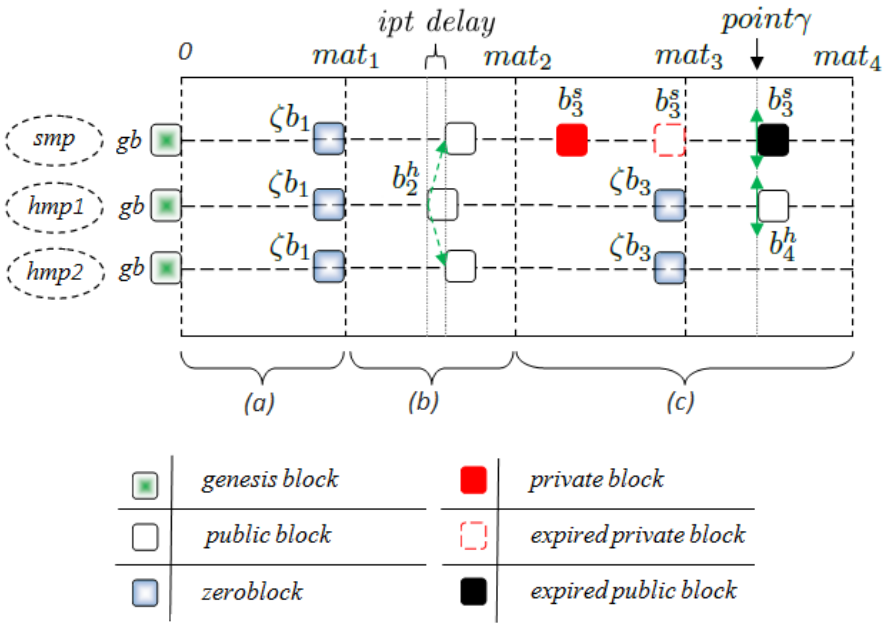

Fig. 3. *smp*: selfish mining pool, hmp_1: first honest mining pool, hmp_2: second honest mining pool. *Part (a)* represents *event 1*: In mat_1, neither honest nor selfish pools discover a new block. *Part (b)* represents *event 2*: In mat_2, the first pool which discovers a new block is honest mining 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 Zeroblock ζb_3 it will be rejected by honest pools since it does not consist of hash of Zeroblock ζb_3.

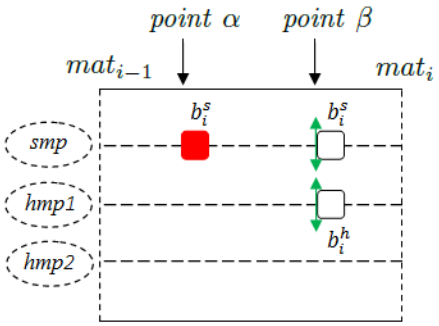

Fig. 4. The selfish pool, *smp*, discovers a new block b_i^s at the point α and keeps the block privately till point β at which an honest pool, hmp_1, generates block b_i^h. at this point, selfish pool broadcasts b_i^s to create an intentional fork. The maximum probability of this event has been calculated above, (≈ 0.04). Also as we described, such block-withholding is unprofitable and reduces the probability of receiving the respective selfish pool's rewards. The result that decreases the motivation of selfish mining pool for such selfish behavior.

And 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:

$$\textit{Maximum Possible Probability}\ (\textit{event 4}\ ) = \left(sp.e^{-sp}\right)\left(hp.e^{-hp}\right) \times sp$$

$$\approx 0.3 \times 0.3 \times 0.49 \approx 0.04$$

Note that as we mentioned above if *event 4* occurs, selfish pool reduces its likelihood for receiving the respective reward. This an "*unprofitable block-withholding*" (see *event 4*.) Even if the purpose of selfish pool is only a sabotage, it has to deplete its limited computational resource for achieving an event that in **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.

5.1 Keynote

- 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 5).
- Note that use of *mat* time interval does not increase the *selfish* miner's motivation to keep the *honest* blocks. For example, if a *selfish* miner keeps

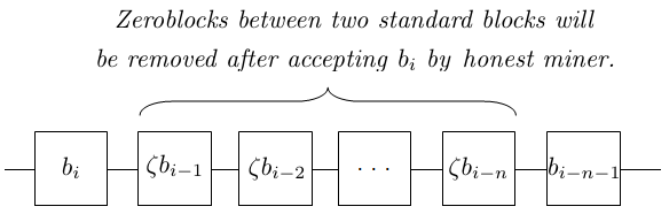

Fig. 5. This Figure demonstrates a situation when between standard blocks b_i and b_{i-n-1} there are some Zeroblocks from ζb_{i-1} to ζb_{i-n} that will be removed after accepting standard block b_i by honest node. Note that even after removing Zeroblocks from the Blockchain, the standard block b_i still includes the hash of Zeroblocks $(\zeta b_{i-1}, ..., \zeta b_{i-n})$ 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.

an honest block to expire it, this is not because of using *mat* time interval. Because, without using *mat* in the algorithm (like standard Bitcoin protocol) a *selfish* miner might do this sabotage behavior (means that keeping new *honest* block "forever") since this miner is *"selfish"* and motivated to prevent publishing the blocks which do not belong to *selfish* mining pool.

- Note that it is not possible to generate a Zeroblock anytime, anywhere and in any number. To understand better how to generate a Zeroblock, we invite reader to discover algorithm 1 line 12, along with Figure 3.
- Since, there is no need for proof-of-work to generate a Zeroblock, thus ζb is not counted for updating the difficulty of proof-of-work. It means that if, for example, the difficulty of proof-of-work is updated every 2016 blocks, in this case only the standard blocks will be counted. As a result, our idea does not affect the difficulty *target*.
- The timestamps are unreliable and forgeable and thus in our algorithm no timestamp is inserted in the blocks and hence our solution is a timestamp-free idea contrary to the recent solution proposed by [23].

6 Conclusion

In this paper, we introduced a new timestamp-free solution to prevent block-withholding or *selfish* mining as one of the most important problems of the Bitcoin network. We demonstrated that if a *selfish* miner wants to keep its block b_i^s private more than mat_i interval, that is the average time in which the whole of network is expected to discover a new block regarding to the hashing power of the network, then selfish block will be rejected by the *honest* miners. Also, since the result of *selfish* mining is **intentional** fork, thus our approach reduces significantly forking the Blockchain by preventing "*unpermitted block-withholding*". 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*. Our idea does not lead to any additional delay, since our solution is based on the Bitcoin protocol configuration for block generation rate that on average is "one block per 10 minutes (i.e. avt_{net}"). 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." Availab le 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
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)
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. Sapirshtein, 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/CIICT 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/